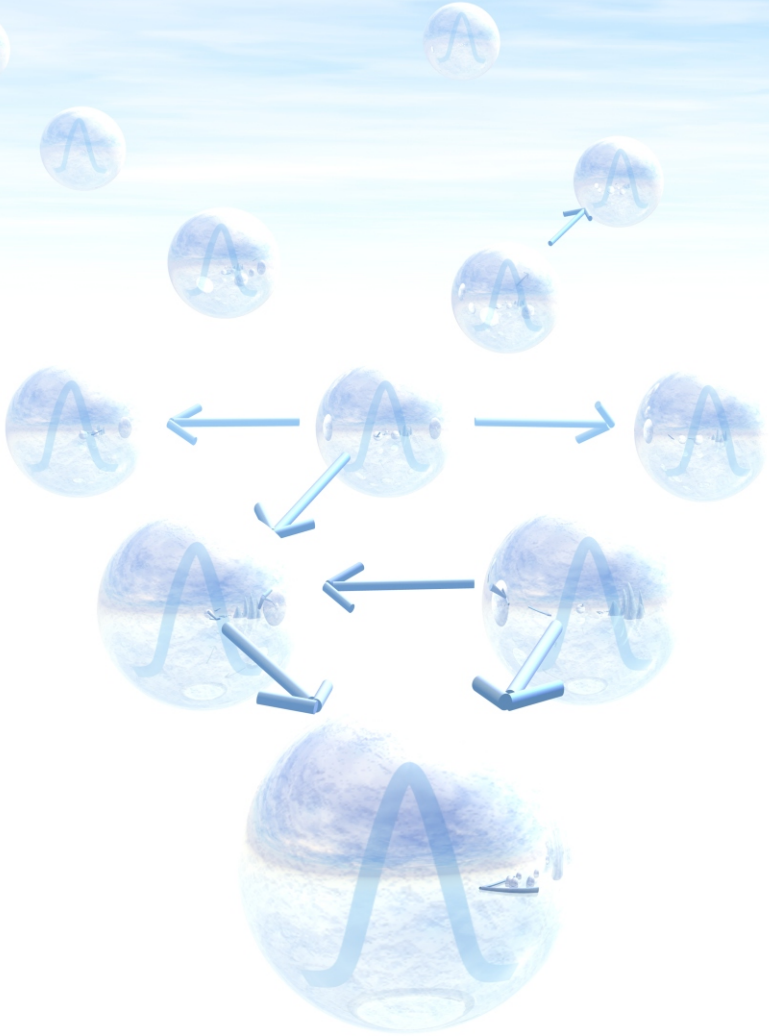
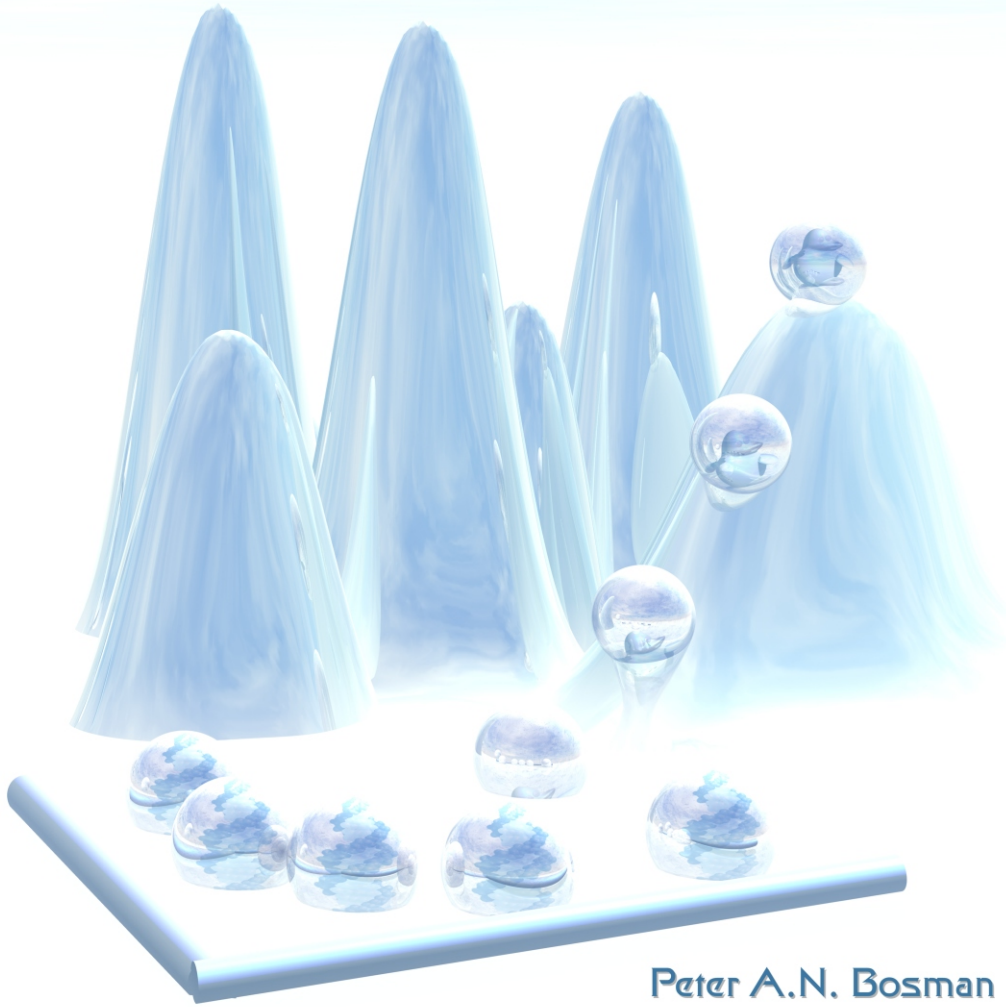


ISBN 90-901-6946-6

Design and Application of Iterated Density-Estimation Evolutionary Algorithms

Peter A.N. Bosman

Design and Application of Iterated Density-Estimation Evolutionary Algorithms



Peter A.N. Bosman

Design and Application of Iterated Density–Estimation Evolutionary Algorithms

Design and Application of Iterated Density–Estimation Evolutionary Algorithms

Ontwerp en Toepassing van Geïtereerde Dichtheid–Schattende Evolutionaire Algoritmen

(met een samenvatting in het Nederlands)

Proefschrift

ter verkrijging van de graad van doctor
aan de Universiteit Utrecht, op gezag van de
Rector Magnificus, Prof. dr. W.H. Gispen,
ingevolge het besluit van het College voor Promoties
in het openbaar te verdedigen op
dinsdag 20 mei 2003 des morgens om 10:30 uur

door

Peter Alexander Nicolaas Bosman

geboren op 25 december 1975 te Utrecht

Promotores	:	Prof. Dr. Ir. L.C. van der Gaag
		Prof. Dr. J. van Leeuwen
Co–Promotor	:	Dr. Ir. D. Thierens

Bosman, Peter Alexander Nicolaas

Design and Application of Iterated Density–Estimation Evolutionary Algorithms
Peter Alexander Nicolaas Bosman
Utrecht, Institute of Information and Computing Sciences
Proefschrift Universiteit Utrecht – met een samenvatting in het Nederlands
ISBN 90–901–6946–6

To my parents
For bestowing upon me the freedom of choice
and for after having made my free choice,
crafting the opportunity for me to one day
write what may be found in these pages

To Tanja
For making such a long, bumpy and winding road
just a simple one to take, simply by being there
every step of all the way, every hour of every day,
never turning for even the slightest moment away

Contents

1 Introduction **1**
- 1.1 Optimization . . . 2
  - 1.1.1 Definitions and notation . . . 2
  - 1.1.2 Local, global, uni–modal and multi–modal optimization . . . 4
- 1.2 Induction . . . 6
- 1.3 Scope and overview . . . 7

2 Concepts from probability theory **11**
- 2.1 Basic concepts and properties . . . 12
  - 2.1.1 Random variables and sample spaces . . . 12
  - 2.1.2 Probability measures and probabilities . . . 12
  - 2.1.3 Probability distributions . . . 12
  - 2.1.4 Marginalization . . . 13
- 2.2 Generalized probability density functions . . . 14
- 2.3 Probabilistic models . . . 15
- 2.4 Factorized probability distributions . . . 15
  - 2.4.1 Multivariately factorized probability distributions . . . 16
  - 2.4.2 Bayesian factorized probability distributions . . . 17
- 2.5 Likelihood, entropy and the Kullback–Leibler divergence . . . 19
  - 2.5.1 Likelihood . . . 19
  - 2.5.2 Entropy . . . 20
  - 2.5.3 Kullback–Leibler divergence . . . 20
- 2.6 Greedy model selection . . . 21
  - 2.6.1 A general framework . . . 22
  - 2.6.2 Choosing between models . . . 22
  - 2.6.3 Greedy multivariate factorization selection . . . 27
  - 2.6.4 Greedy Bayesian factorization selection . . . 29
- 2.7 Mixture probability distributions . . . 32
  - 2.7.1 By means of clustering . . . 34
  - 2.7.2 By means of expectation maximization . . . 38

3 Evolutionary algorithms **41**
- 3.1 Evolutionary algorithms . . . 42
  - 3.1.1 Natural evolution as an optimization process . . . 42

3.1.2 Natural evolution as a metaphor for optimization 43
3.1.3 A general algorithmic framework for EAs 44
3.2 The simple genetic algorithm 46
3.2.1 Algorithmic description 46
3.2.2 Dynamics of the simple GA: schema analysis 50
3.2.3 Difficult problems for simple GAs 55
3.2.4 Competence of the simple GA: mixing analysis 57
3.3 Linkage–respecting genetic algorithms 61
3.4 Discussion 63

4 Iterated density–estimation evolutionary algorithms 65
4.1 Prelude to the IDEA: from crossover to probability distributions 66
4.1.1 Crossover probability distributions 66
4.1.2 Factorized probability distributions 69
4.1.3 Comparing crossover with factorizations 70
4.1.4 From crossover linkage learning to factorization selection . 76
4.2 The IDEA framework 77
4.2.1 Definition of IDEAs 77
4.2.2 Monotonic IDEAs 79
4.3 A literature overview of IDEA instances and related algorithms . 81
4.3.1 Factorized probability distributions 83
4.3.2 Mixture probability distributions 92

5 Numerical optimization 95
5.1 Real–valued probability distributions 96
5.1.1 Generalized probability density functions 96
5.1.2 Mixture probability distributions by clustering 108
5.2 Exploiting local gradient information by hybridization: GLIDE . 111
5.3 Experiments 112
5.3.1 Numerical optimization problems 113
5.3.2 Experiment setup 116
5.3.3 Results 118
5.3.4 Practitioner's summary 125
5.4 Discussion and future research 126

6 Permutation optimization 127
6.1 Evolutionary algorithms and genotypes for permutations 128
6.1.1 Integer permutation genotype 128
6.1.2 Random keys genotype 129
6.2 Hard relative ordering problems for simple permutation EAs . . . 131
6.2.1 Relative order schemata 131
6.2.2 Relative–ordering deceptive problems 132
6.2.3 Decomposable relative–ordering deceptive test problems . 133
6.2.4 Scale–up behavior of simple permutation EAs 135
6.3 Factorizations for random keys random variables 135
6.3.1 Interpreting random keys as real values 137

6.3.2 Interpreting random keys as permutations 137
6.4 More efficient probabilistic processing of random keys: $\mathbb{ICE}$ . . . 152
6.4.1 The $\mathbb{ICE}$ framework 153
6.4.2 Specific $\mathbb{ICE}$ instances based on factorizations 154
6.5 Experiments 155
6.5.1 Permutation optimization problems 156
6.5.2 Experiment setup 156
6.5.3 Results 158
6.5.4 Practitioner's summary 167
6.6 Discussion and future research 169

7 Multi–objective optimization **171**
7.1 Multi–objective optimization definitions 172
7.2 Multi–objective optimization goals 172
7.2.1 Approximation sets, optimality and benchmarking 172
7.2.2 Performance indicators 173
7.3 Multi–objective optimization algorithms 180
7.3.1 Repeated optimization 180
7.3.2 Simultaneous optimization 180
7.4 Multi–objective mixture–based IDEAs 182
7.4.1 Balanced multi–objective truncation selection 183
7.4.2 Stimulating diversity with mixture probability distributions 185
7.4.3 A specific multi–objective mixture–based IDEA 185
7.5 Experiments 185
7.5.1 Multi–objective optimization problems 186
7.5.2 Experiment setup 190
7.5.3 Results 193
7.5.4 Practitioner's summary 202
7.6 Discussion and future research 203

8 Discussion and conclusions **205**

A Maximum likelihood univariate conditional gpdfs **211**
A.1 Integer sample space 212
A.1.1 Frequency 212
A.2 Real–valued sample space 213
A.2.1 Histogram 213
A.2.2 Normal 213
A.2.3 Normal kernels and normal mixture 215

B Tabulated experimental results **217**
B.1 Numerical optimization 218
B.2 Permutation optimization 226
B.3 Multi–objective optimization 229

Bibliography **237**

Samenvatting **251**

Acknowledgements **257**

Curriculum Vitae **259**

Publications **261**

Index **265**

List of notations, symbols and pseudo–code conventions **269**

'Every new idea is an impossibility until it is born.'
Ron Brown

Introduction

In many fields of society people are called upon to solve complex optimization problems. Optimization is the central theme of this thesis. More specifically, this thesis is about optimization by induction, where an algorithm makes educated guesses about what makes a good solution to a problem under study, building upon solutions that were evaluated earlier in the execution of the algorithm. In this thesis, we consider the class of evolutionary algorithms (EAs) as a specific class of optimization algorithms. By implementing certain abstractions of the processes observed in nature, EAs are designed to perform induction. Different designs lead to different EAs and consequently to different behavior in induction and optimization. In this thesis, we combine EAs with tools from probability theory and study the applicability and performance of these probability–based EAs with respect to different types of optimization problem. In this introductory chapter, we briefly review the concepts of optimization and induction and give an overview of the contents of the subsequent chapters in this thesis.

The chapter is organized as follows. In Section 1.1 we review the main concepts involved in optimization. Subsequently, in Section 1.2, we discuss the concept of induction as a mechanism for solving optimization problems. In Section 1.3, we briefly indicate how we shall use tools from probability theory to arrive at a new type of EA. We further outline the scope of the research presented in this thesis and give a guide to subsequent chapters.

1.1 Optimization

The concept of optimization pervades our every–day lives. We want to optimize tasks and designs for many purposes. Consider, for instance, the task of shopping at a supermarket. We typically want to optimize the distance we have to travel through the supermarket to obtain all the items on our shopping list. Another optimization problem arises if we have only a limited amount of space for the groceries, or if there is a limit to the weight that we can carry. The optimization problem then is to buy items to make optimal use of our carrying capacities. Optimization can become more complex if we want to optimize more than one objective at the same time, especially when these objectives are conflicting. For example, on the one hand we may want to minimize the total weight of our groceries whereas on the other hand, we may want to maximize their nutricial value. In this optimization problem, there are combinations of groceries that we do not want to choose. These combinations have a larger weight and lower nutricial value than some other combinations. However, there may also be two combinations such that the first one is lighter to carry yet less nutricious, whereas the other is heavier but more nutricious. Without any preference for nutrition versus carrying weight, there is *no way* to distinguish between these two sets of groceries.

In many fields of society, people are called upon to solve optimization problems. Because of the importance of optimization, it has been a prominent research topic for several decades. A vast amount of knowledge exists about how to efficiently solve specific types of optimization problem. Also considerable insight has been gained on how difficult optimization problems can be. Given an optimization problem, it is a challenging task to design an algorithm that can find the optimal solution. It is even more challenging to find an algorithm that does so as fast as possible and to use it to solve problems with as many parameters as possible. In the remainder of this section, we will review the most important concepts in optimization.

1.1.1 Definitions and notation

We define five key concepts which are important to any optimization problem. These concepts are depicted schematically in Figure 1.1.

- **Parameter space**
  The *parameter space* (also called the *domain* or *search space*) is the set of all *solutions* to the optimization problem; we denote the parameter space by $\mathbb{P}$. Optimization problems are mostly specified using *variables* (also called *parameters*). The parameter space then consists of all possible combinations of values that can be assigned to these variables. Ideally, each combination of values in the parameter space is *feasible*. Unfortunately, it is not always possible to obtain a (mathematical) characterization of the optimization problem such that each possible combination of values is feasible. In such a case, *constraints* on the parameter space define which

combinations of values are feasible and which are not. For instance, if solutions are vectors of five binary values ($\mathbb{P} = \{0,1\}^5 = \mathbb{B}^5$), a constraint may be that no solution has more than three variables set to 1. The feasibility of a solution is determined by a function $\mathfrak{C} : \mathbb{P} \rightarrow \{\textit{feasible}, \textit{infeasible}\}$.

- **Objective space**
  The *objective space* is the set of all possible *optimization values* that can be associated with a solution; we denote the objective space by $\mathbb{O}$. An optimization value in general captures the values of m *objectives*. Many well–known optimization problems have only a single objective. However, there are many interesting problems that have multiple objectives. Since objectives are almost always functions that return a real *value* (e.g. $\mathbb{O} = \mathbb{R}^m$), it is common practice to use the terms objective *value* and optimization *value*.

- **Optimization function**
  The *optimization function* assigns an optimization value in $\mathbb{O}$ to each solution in $\mathbb{P}$; we denote the optimization function by $\mathfrak{F} : \mathbb{P} \rightarrow \mathbb{O}$. Since an optimization value consists of m objective values, an optimization function is composed of m *objective functions*, each of which contributes a single objective value to the optimization value.

- **Search space ordering**
  We assume that we can always decide whether or not the optimization values of two solutions A and B are identical, denoted by $\mathfrak{F}(A) = \mathfrak{F}(B)$. In addition, we assume that a (partial) ordering $\prec$ on the objective space is specified. With this ordering we can express that solution A is less preferred than solution B, denoted by $\mathfrak{F}(A) \prec \mathfrak{F}(B)$. Note that two solutions are equally preferred if $\mathfrak{F}(A) = \mathfrak{F}(B)$. Also note that two solutions may not be comparable, in which case they are also equally preferred, that is $\mathfrak{F}(A) \neq \mathfrak{F}(B)$, $\mathfrak{F}(A) \not\prec \mathfrak{F}(B)$ and $\mathfrak{F}(A) \not\succ \mathfrak{F}(B)$. This may happen for instance when there is more than one objective function. We denote this by $\mathfrak{F}(A) \sim \mathfrak{F}(B)$. In practice, we often want to minimize or maximize an optimization function that consists of a single objective function that returns a real value. In this case, minimization implies that $\prec$ equals $>$, whereas maximization implies that $\prec$ equals $<$.

- **Optimality**
  A solution A is *optimal* if A is feasible and there is no other feasible solution B such that $\mathfrak{F}(A) \prec \mathfrak{F}(B)$. Note that depending on $\mathfrak{F}$ and $\prec$, there may be more than one optimal solution.

As mentioned above, optimization problems are mostly specified using *problem variables* that can jointly be assigned a value from the parameter space $\mathbb{P}$. The number of problem variables is called the *dimensionality* d of the optimization problem, $d \geq 1$. For convenience, we introduce a vector $\boldsymbol{\mathcal{D}} = (0, 1, \ldots, d-1)$ of integers to indicate the problem variables ζ_i, $i \in \boldsymbol{\mathcal{D}}$. The

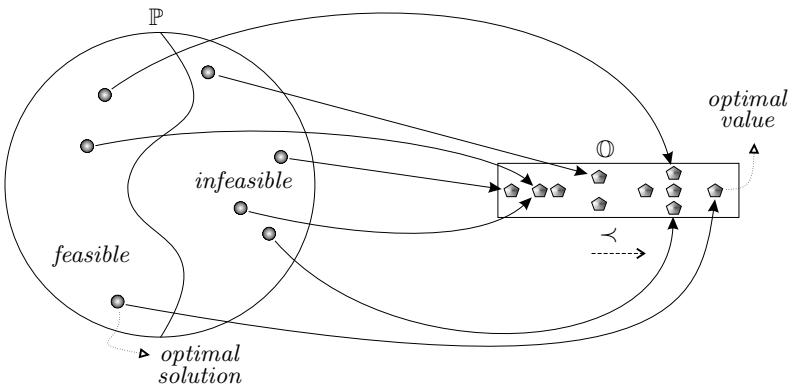


Figure 1.1: Graphical overview of an optimization problem. The arrows from $\mathbb{P}$ to $\mathbb{O}$ represent the optimization function $\mathfrak{F}$. The values in $\mathbb{O}$ are ordered according to increasing preference from left to right.

optimization function is often written as a function of the problem variables, that is, $\mathfrak{F}(\zeta_0, \zeta_1, \ldots, \zeta_{d-1}) = \mathfrak{F}(\zeta_{\boldsymbol{\mathcal{D}}})$. The domain of ζ_i is denoted by $\mathbb{D}_i$ and, if the parameter space is Cartesian, it is given by $\mathbb{P} = \times_{i=0}^{d-1} \mathbb{D}_i$.

1.1.2 Local, global, uni–modal and multi–modal optimization

Having defined the key concepts that constitute an optimization problem, we now turn to the output desired for such a problem, i.e. what it means to *solve* the problem. For most optimization problems, we are interested in just a single solution that is optimal according to the definition in Section 1.1.1. However, if the optimization problem has more than one objective, we are typically interested in a large variety of optimal solutions. More details on this are given in Chapter 7. The definition of optimality in Section 1.1.1 is usually called *global optimality*. The term *global* indicates that we consider the entire (feasible) parameter space. If a distance measure is associated with the parameter space $\mathbb{P}$, then the optimization problem can also have *locally optimal* solutions. A solution A is locally optimal if there is no other feasible solution B in the *neighborhood* of A such that $\mathfrak{F}(A) \prec \mathfrak{F}(B)$. Note that a globally optimal solution is therefore also locally optimal. The neighborhood of a solution A is a subset of $\mathbb{P}$ that only contains solutions that are close to A in terms of the associated distance measure. Often, the neighborhood of solution A is taken to be the set of solutions with the smallest possible nonzero distance to A. However, the definition of the neighborhood may differ for different optimization problems. For instance, when $\mathbb{P} = \mathbb{R}$, there is no solution closest to any other solution. In this case, the neighborhood is often defined to be larger.

Although the notion of (global) optimization as presented above is the most common, other types of optimization are also used in practice. The notion

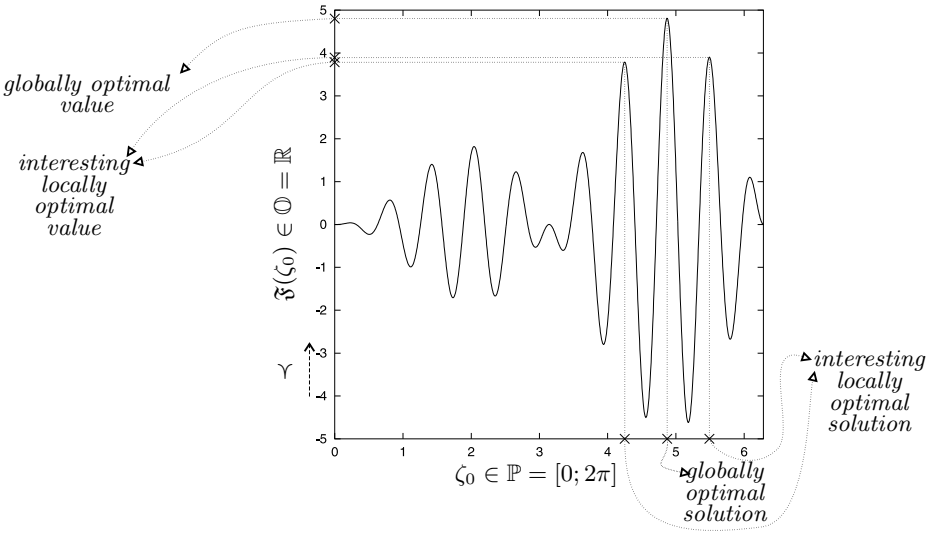


Figure 1.2: An illustration of globally optimal solutions and locally optimal solutions in the maximization of $\mathfrak{F}(\zeta_0) = \zeta_0 \sin(10\zeta_0)\sin(\zeta_0)$. There are 11 locally optimal solutions, including the globally optimal solution. Two of the locally optimal solutions are likely to be of interest in multi–modal optimization.

of optimization can be refined using the concepts of uni–modal optimization and multi–modal optimization. We will discuss these concepts assuming just a single objective. In uni–modal (global) optimization, we are interested in just a single *globally* optimal solution; any solution that is globally optimal will do. In *multi–modal optimization* the goal is to find a set of *globally* or *locally* optimal solutions. For example, we may want to find *all* locally optimal solutions. Usually we are interested in finding only globally optimal solutions, but we may also be interested in finding some additional, locally optimal solutions that have an optimization value that is close to the globally optimal value. The following example serves to illustrate this idea.

Example 1.1. Suppose that we want to maximize a particular real function of a single real variable ζ_0 in the domain $\mathbb{P} = [0; 2\pi]$; the function is $\mathfrak{F}(\zeta_0) = \zeta_0 \sin(10\zeta_0)\sin(\zeta_0)$ as depicted in Figure 1.2. The globally optimal solution and its corresponding globally optimal value can be clearly distinguished. Overall, this optimization problem has 11 locally optimal solutions, one of which is the globally optimal solution. In multi–modal optimization, the two peaks surrounding the globally optimal solution are likely to be of interest. In some cases, we might even be interested in all locally optimal solutions.

1.2 Induction

Any optimization problem has a structure. This structure is captured by the optimization function $\mathfrak{F}$ and the feasibility function $\mathfrak{C}$. Sometimes, features of this structure can be exploited to perform optimization more efficiently. For example, it may be possible to prove that certain parts of the feasible parameter space do not contain the optimal solution. Constraints can then be added to the problem specification that can subsequently be exploited by algorithms that systematically search the feasible parameter space to find the optimal solution. More often, however, we are not able to generate additional constraints or to give a better characterization of an optimization problem. Sometimes we might not even have an exact characterization of the optimization problem at all, for example when we are trying to optimize a problem that is modeled by a complex simulation. The parameters of the simulation define the search space, but their contribution to the optimization function is unclear. Often, therefore, there are many cases in which the optimization problem has exploitable structural features that aren't directly observable.

Optimization problems of which the structure is insufficiently known can be solved using the concept of *black–box optimization* (BBO). For BBO, we assume no prior knowledge on either $\mathfrak{F}$ or $\mathfrak{C}$. Knowledge of these functions can be gained, however, by computing the optimization value of a given solution and by verifying whether or not it is feasible. One way of solving such problems is a random or systematic search throughout the parameter space. If we assume, however, that the problem has some structural features that can be exploited, finding these features and subsequently using them in optimization is a more preferable way to traverse the search space. To find out the structural features, we must reason about the points in the parameter space for which their corresponding points in the objective space have been observed. This task is called *induction*. The following two examples illustrate the use of exploiting structural features and performing induction in optimization.

Example 1.2. In numerical optimization, we consider the parameter space $\mathbb{P} = \mathbb{R}^d$ for some $d \geq 1$, and the objective space $\mathbb{O} = \mathbb{R}$. If the optimization function $\mathfrak{F}$ is differentiable, the gradient of $\mathfrak{F}$ can be used to search for (local) optima more efficiently. If the definition of $\mathfrak{F}$ is not known or if $\mathfrak{F}$ can't be differentiated analytically, an approximation of the gradient at a particular point in the parameter space can be induced by probing $\mathfrak{F}$ in the neighborhood of that point.

Example 1.3. Suppose that $\mathbb{P} = \times_{i=0}^{d-1}\mathbb{B}$ and $\mathbb{O} = \mathbb{N}$. Further suppose that the optimization function is $\mathfrak{F}(\zeta_{\mathcal{D}}) = \sum_{i=0}^{d-1} \zeta_i$ and that the goal is to maximize the optimization value. This optimization problem is known as the *one–max* or *bitcounting* problem. Every problem variable ζ_i has a contribution to the optimization function that is *independent* of the other problem variables ζ_j, $j \neq i$. So, although the size of the parameter

space grows exponentially in d, the problem can be optimized very efficiently by assigning to each variable in turn a 0 and a 1, and observing which value for the problem variable results in the largest optimization value. We now assume that the optimization function is unknown and perform BBO. If the algorithm is able to induce the independence relationship of the problem variables during the optimization process, it can exploit this structural feature to perform efficient optimization as outlined above. The bitcounting problem is an example of an *additively decomposable* optimization problem. In an additively decomposable optimization problem, the optimization function $\mathfrak{F}$ is a sum of functions that pertain to mutually disjoint subsets of the problem variables. In general, if the size of these subsets is bounded, detection of the independence of the subsets can speed up optimization significantly.

The main focus of this thesis is on the use of induction in black–box optimization. More specifically, we focus on the type of induction performed by EAs and investigate to which extent we can improve the current induction capabilities of EAs by using techniques from probability theory.

1.3 Scope and overview

EAs are algorithms that mimic basic concepts in natural evolution to perform optimization by induction. The two main concepts used are genetic inheritance and survival of the fittest. In this thesis, we investigate a new type of EA that builds upon techniques from probability theory. We refer to this new type of algorithm as IDEA (*Iterated Density–Estimation Evolutionary Algorithm*). IDEA is a general framework for EAs that iteratively estimate probability distributions and draw new samples from the estimated distributions to perform iterated optimization. Probability theory offers a variety of effective tools for the modelling of dependencies between variables so as to provide accurate descriptions of dependencies in data. Such tools offer a means of identifying and modelling structural properties of the optimization problem during optimization and of exploiting these properties to arrive at more efficient optimization.

The goal of this thesis is to present the IDEA framework and to address the applicability of probability–distribution estimation techniques in EAs to solving different types of optimization problem. We restrict ourselves to the greedy estimation of multivariate and Bayesian factorized probability distributions and to the estimation of mixtures of factorized probability distributions by means of clustering. The optimization problems that we consider are within the fields of numerical function optimization, permutation optimization and multi–objective optimization. The remainder of this thesis is organized as follows.

Chapter 2
In this chapter we give an overview of concepts from probability theory that are relevant to the research described in this thesis. If the reader is already

familiar with a basic to firm understanding of probability theory, this chapter may not be required to understand the contents of this thesis. Probability distributions and probabilistic models are described in general, as are generalized probability density functions and factorizations. Properties of probability distributions such as likelihood and entropy are subsequently presented. The approaches used in this thesis to estimating factorizations and mixtures of factorizations is thereafter provided. A reflection on the use of the likelihood, the entropy, the Kullback–Leibler divergence and likelihood penalization metrics in probability distribution estimation and IDEAs was published before (Bosman and Thierens, 2000c).

Chapter 3
In this chapter we introduce evolutionary algorithms (EAs). We describe how natural evolution is used as a metaphor for constructing EAs and describe a general algorithmic framework for EAs. In addition, we outline the main streams in EA research. We then describe the classical genetic algorithm (GA) in more detail by instantiating the general EA framework. Subsequently, we discuss the results of three analyses from the literature to provide a basic background in GA dynamics and to stress a number of important limitations of the classical GA. These limitations indicate that the scale–up behavior of the classical GA may be improved by using more effective induction to determine the structure of the optimization problem during optimization. The definition of the general EA framework was published before (Bosman and Thierens, 1999b).

Chapter 4
In this chapter we show that the use of factorized probability distributions instead of recombination operators can lead to a scale–up behavior that is at least as good as that of the best possible two–parent crossover operator. Subsequently, we define the IDEA (*Iterated Density Estimation Evolutionary Algorithm*) framework that formalizes the use of probability distributions instead of recombination in EAs. The remainder of this chapter presents an overview of related algorithms reported in the literature. Algorithms of the IDEA type were first proposed for binary representations. The first algorithms used relatively simple factorized probability distributions. To ensure non–exponential scale–up behavior on additively decomposable optimization problems however, these simple factorizations do not suffice. An empirical validation of this observation is also presented. This empirical validation was published before (Bosman and Thierens, 1999a). Also, the IDEA framework was introduced before in a publication (Bosman and Thierens, 2001c).

Chapter 5
In this chapter, we apply the IDEA framework to numerical optimization. To this end, real–valued continuous probability distributions are used. Multivariate factorizations as well as Bayesian factorizations are estimated. Moreover, mixtures of factorized probability distributions are used. The generalized probability density functions (gpdfs) that are used to construct the factorizations,

are the single normal gpdf, the normal kernels gpdf, the normal mixture gpdf and the histogram gpdf. A combination with local search techniques based on the exploitation of local gradient information through the conjugate gradient algorithm is also studied in this chapter. This combination leads to new (hybrid) EAs that can be shown to indeed lead to good results on a variety of hard numerical optimization problems. Especially for larger problem dimensionalities, the IDEA–based approaches are capable of obtaining better results than other well–known EAs or classical purely gradient–based optimization techniques. Although the use of the probability distributions described in this chapter was published before (Bosman and Thierens, 2000a; Bosman and Thierens, 2000b; Bosman and Thierens, 2001a; Bosman and Thierens, 2001c), the experimental validation presented in this chapter is more extensive.

Chapter 6

In this chapter we focus on the estimation of probability distributions for permutations in evolutionary permutation optimization. In permutation optimization, the solutions to the optimization problem are permutations. Using a real–valued encoding of permutations, the IDEAs from Chapter 5 can be readily applied. However, since the real–valued parameter space has infinitely many solutions, a real–valued encoding is highly redundant. As a result of this redundancy, the dependencies between the problem variables are not efficiently modelled or processed. Therefore, we propose to estimate probability distributions for the permutations directly and use this information to construct new IDEAs. In this chapter, we show how multivariate factorized probability distributions can be estimated for permutations and how this information can be used in IDEA to construct efficient new EAs for permutation optimization. Experimental results indicate polynomial scale–up behavior on a difficult additively decomposable permutation optimization problem on which other classical permutation–based EAs are shown to scale up exponentially. This chapter combines three earlier publications (Bosman and Thierens, 2001b; Bosman and Thierens, 2001d; Bosman and Thierens, 2002b).

Chapter 7

In this chapter we show how efficient instances of the IDEA framework can be constructed for multi–objective optimization. Evolutionary algorithms perform very well in multi–objective optimization, since they are based on a population, which makes them natural candidates for finding multiple trade–off solutions of equal quality without having to repeat the search procedure. One of the IDEAs studied in this chapter is MIDEA (*Multi–objective Mixture–based* IDEA). In MIDEA, mixture probability distributions are used to promote diversity among solutions that are equally good in the multi–objective sense. A special selection operator is used to ensure that diversity is preserved. MIDEA instances are shown to be comparably efficient in multi–objective optimization as the current state–of–the–art multi–objective evolutionary algorithms. MIDEAs specifically excel in obtaining a diverse and extensive set of equally good multi–objective solutions. This observation is empirically supported using

a test suite of eight multi–objective optimization problems. A second observation that is supported by the experiments is that the selection operator in $\mathbb{MIDEA}$ is unique in the sense that it is the only selection operator in the current field of multi–objective EAs through which the balance between proximity and diversity can be changed. Although the $\mathbb{MIDEA}$ framework presented in this thesis does not differ from previous publications (Thierens and Bosman, 2001a; Thierens and Bosman, 2001b; Bosman and Thierens, 2002a; Bosman and Thierens, 2003), the experimental section of this chapter is more involved.

Chapter 8
We conclude this thesis in Chapter 8 by reflecting on the use of probability–distribution estimation in EAs and the capabilities of such EAs in solving optimization problems. We further discuss possible future directions for research.

Appendix A
In this appendix maximum–likelihood conditional gpdfs that are required for the estimation of Bayesian factorizations are derived.

Appendix B
In this appendix the results of the experiments in which IDEAs and other EAs were used to tackle the three main different types of optimization problem as presented in chapters 5, 6 and 7 are tabulated.

'The best ideas are common property.'
Lucius Seneca

Concepts from probability theory

In this chapter we describe the basic concepts and tools from probability theory that will be used throughout this thesis. The reader with a strong background in probability theory may want to skip this chapter, or use it merely for (notational) reference. The reader with a basic knowledge of probability–theoretic issues may want to skip the Sections 2.1 through 2.4, but is advised to read the remaining sections. Readers with only slight or virtually no knowledge of probability theory are encouraged to read the entire chapter.

The chapter is organized as follows. In Section 2.1, we review probability distributions and probabilistic models. Next, in Section 2.2, we discuss generalized probability density functions (gpdfs). Subsequently, in Section 2.3 we discuss probabilistic models. In Section 2.4, we describe the class of factorized probability distributions that are obtained by composition of gpdfs. In Section 2.5 we discuss a number of basic measures for probability distributions such as the Kullback–Leibler divergence between probability distributions and the entropy of a probability distribution. A general algorithmic framework for learning probabilistic models from data is presented in Section 2.6, along with example instances for two types of factorized probability distribution. Finally, in Section 2.7, we discuss clustering algorithms and the expectation maximization algorithm for estimating mixture probability distributions from data.

2.1 Basic concepts and properties

In this section we review some of the basic concepts and definitions from probability theory. For a more elaborate introduction to these concepts, we refer the reader to the literature.

2.1.1 Random variables and sample spaces

In probability theory, events are related to variables that can be assigned values from a fixed domain. In the field of probability theory, variables are commonly referred to as *random variables*. In this thesis, we assume to have l random variables $Z_0, Z_1, \ldots, Z_{l-1}$, $l \geq 1$, written as a vector $\boldsymbol{\mathcal{Z}} = (Z_0, Z_1, \ldots, Z_{l-1}) = Z_{\boldsymbol{\mathcal{L}}}$ with $\boldsymbol{\mathcal{L}} = (0, 1, \ldots, l-1)$. The domain of values that can be assigned to $\boldsymbol{\mathcal{Z}}$ is called the *sample space* Ω. An *event* A now is defined as $\boldsymbol{\mathcal{Z}}$ being assigned a value from $A \subseteq \Omega$. With each subvector $Z_{\boldsymbol{j}} \sqsubset \boldsymbol{\mathcal{Z}}$ of random variables, a separate sample space $\Omega^{\boldsymbol{j}}$ is identified. Unless explicitly stated otherwise, sample spaces are assumed to be Cartesian, that is, $\Omega^{\boldsymbol{j}} = \times_{i=0}^{|\boldsymbol{j}|-1} \Omega^{\boldsymbol{j}_i}$ for any $\boldsymbol{j} \sqsubseteq \boldsymbol{\mathcal{L}}$. In the remainder of this chapter we will consider two specific Cartesian sample spaces, which we refer to as the *integer* sample space and the *real–valued* sample space. If each random variable $Z_i \in \boldsymbol{\mathcal{Z}}$ can be assigned a value from a finite, discrete sample space Ω^i, that is, if we may assume, without loss of generality, that the sample space is equivalent to a finite subset of the natural numbers, $\Omega^i = \{0, 1, \ldots, \left|\Omega^i\right| - 1\}$, we say that it is an *integer* sample space and write X_i for the i–th random variable instead of Z_i. We have a *real–valued* sample space for Z_i if $\Omega^i = \mathbb{R}$; we then denote the i–th random variable by Y_i instead of Z_i.

2.1.2 Probability measures and probabilities

Probabilities are defined through *probability measures*. A probability measure Γ on a sample space Ω is a function $\Gamma\colon \mathcal{P}(\Omega) \rightarrow \mathbb{R}$, such that

1. $\Gamma(\Omega) = 1$
2. $\forall A \subset \Omega : \Gamma(A) \geq 0$
3. $\forall A^0 \subset \Omega, A^1 \subset \Omega : A^0 \cap A^1 = \emptyset \rightarrow \Gamma(A^0 \cup A^1) = \Gamma(A^0) + \Gamma(A^1)$

$\Gamma(A)$ is termed the probability of event A, which we also denote by $\Pr(\boldsymbol{\mathcal{Z}} \in A)$.

2.1.3 Probability distributions

A probability distribution $P(\boldsymbol{\mathcal{Z}})$ is a real–valued function $P(\boldsymbol{\mathcal{Z}}) : \Omega \rightarrow \mathbb{R}$. Since $P(\boldsymbol{\mathcal{Z}})$ is a function, we write $P(\boldsymbol{\mathcal{Z}})(\boldsymbol{z})$ to get the function value for $\boldsymbol{z} \in \Omega$. With each type of sample space, a specific probability measure is associated that is always defined in terms of a probability distribution. Therefore, a function $P(\boldsymbol{\mathcal{Z}}) : \Omega \rightarrow \mathbb{R}$ may be called a probability distribution over Ω if and only if the probability measure that is predefined for Ω meets the three constraints. The probability measure Γ^D for the integer sample space is defined as:

$$\Gamma^D(A) = \sum_{\boldsymbol{x} \in A} P(\boldsymbol{\mathcal{X}})(\boldsymbol{x}) \tag{2.1}$$

The probability measure Γ^R for the real–valued sample space is defined as:

$$\Gamma^R(A) = \int_{\boldsymbol{y} \in A} P(\boldsymbol{\mathcal{Y}})(\boldsymbol{y}) d\boldsymbol{y} \tag{2.2}$$

Obtaining probabilities from probability distributions

It should be noted that $P(\boldsymbol{\mathcal{Z}})(\boldsymbol{z})$ is *not* defined to be a *probability*. To obtain a probability from a probability distribution, the definition of the probability measure of the random variables involved needs to be used. For the integer sample space we find that $\Pr(\boldsymbol{\mathcal{X}} = \boldsymbol{x}) = \sum_{\boldsymbol{x} \in \{\boldsymbol{x}\}} P(\boldsymbol{\mathcal{X}})(\boldsymbol{x}) = P(\boldsymbol{\mathcal{X}})(\boldsymbol{x})$, so in the case of the integer sample space $P(\boldsymbol{\mathcal{X}})(\boldsymbol{x})$ indeed is a probability. For the real–valued sample space, however, we find by definition of the probability measure for the real–valued sample space that $\Pr(\boldsymbol{\mathcal{Y}} = \boldsymbol{y}) = \int_{\boldsymbol{y}}^{\boldsymbol{y}} P(\boldsymbol{\mathcal{Y}})(\boldsymbol{y}) d\boldsymbol{y} = 0$. It is important therefore to understand that for real–valued random variables, the probabilities are hidden in the integrals. So $\Pr(\boldsymbol{y}^0 < \boldsymbol{\mathcal{Y}} < \boldsymbol{y}^1) = \int_{\boldsymbol{y}^0}^{\boldsymbol{y}^1} P(\boldsymbol{\mathcal{Y}})(\boldsymbol{y}) d\boldsymbol{y}$ is the only valid way of obtaining probabilities for real–valued random variables.

Permutation invariant notation

Since $(Z_0, Z_1) = (z_0, z_1)$ is the same event as $(Z_1, Z_0) = (z_1, z_0)$, we define the probability distribution $P(Z_{\boldsymbol{j}})$ ($\boldsymbol{j} \sqsubseteq \boldsymbol{\mathcal{L}}$) to be invariant under any permutation of the random variables, given that the input is permuted in the same way:

$$\forall \boldsymbol{j} \sqsubseteq \boldsymbol{\mathcal{L}}, \boldsymbol{k} \in perm(\boldsymbol{j}) : P(Z_{\boldsymbol{j}})(z_{\boldsymbol{j}}) = P(Z_{\boldsymbol{k}})(z_{\boldsymbol{k}}) \tag{2.3}$$

2.1.4 Marginalization

An important concept used throughout this thesis pertains to the probability of an event in which only a selection of all random variables $Z_{\boldsymbol{j}} \sqsubset \boldsymbol{\mathcal{Z}}$ is assigned a value $\boldsymbol{z} \in \Omega^{\boldsymbol{j}}$. We introduce the notation $P(Z_{\boldsymbol{j}})$ to denote the function that is obtained from $P(\boldsymbol{\mathcal{Z}})$ such that $P(Z_{\boldsymbol{j}})(\boldsymbol{z}) = \Pr(Z_{\boldsymbol{j}} = \boldsymbol{z})$. Using the definition of the probability measure for the integer sample space, we then find that for any $\boldsymbol{j} \sqsubset \boldsymbol{\mathcal{L}}$ and for any $x_{\boldsymbol{j}} \in \Omega^{\boldsymbol{j}}$:

$$\begin{aligned} P(X_{\boldsymbol{j}})(x_{\boldsymbol{j}}) &= \Pr(X_{\boldsymbol{j}} = x_{\boldsymbol{j}}) = \Gamma\left(\{x'_{\boldsymbol{\mathcal{L}}} | x'_{\boldsymbol{j}} = x_{\boldsymbol{j}} \wedge x'_{\boldsymbol{\mathcal{L}}} \in \Omega\}\right) = \\ &\sum_{x''_{\boldsymbol{\mathcal{L}}} \in \{x'_{\boldsymbol{\mathcal{L}}} | x'_{\boldsymbol{j}} = x_{\boldsymbol{j}} \wedge x'_{\boldsymbol{\mathcal{L}}} \in \Omega\}} P(\boldsymbol{\mathcal{X}})(x''_{\boldsymbol{\mathcal{L}}}) = \sum_{x_{\boldsymbol{\mathcal{L}}-\boldsymbol{j}} \in \Omega^{\boldsymbol{\mathcal{L}}-\boldsymbol{j}}} P(\boldsymbol{\mathcal{X}})(x_{\boldsymbol{\mathcal{L}}}) \end{aligned} \tag{2.4}$$

So, $P(X_{\boldsymbol{j}})$ is the function that is obtained from $P(\boldsymbol{\mathcal{X}})$ by summing over all possible values for $\boldsymbol{\mathcal{X}} - X_{\boldsymbol{j}}$. Moreover, $P(X_{\boldsymbol{j}})$ is again a probability distribution, but now for random variables $X_{\boldsymbol{j}}$ instead of $\boldsymbol{\mathcal{X}}$.

In a similar fashion we find that for any $\boldsymbol{j} \sqsubset \boldsymbol{\mathcal{L}}$ and for any $y_{\boldsymbol{j}} \in \Omega^{\boldsymbol{j}}$:

$$P(Y_{\boldsymbol{j}})(y_{\boldsymbol{j}}) \quad = \quad \int_{y_{\boldsymbol{\mathcal{L}}-\boldsymbol{j}} \in \Omega^{\boldsymbol{\mathcal{L}}-\boldsymbol{j}}} P(\boldsymbol{\mathcal{Y}})(y_{\boldsymbol{\mathcal{L}}}) d_{y_{\boldsymbol{\mathcal{L}}-\boldsymbol{j}}} \tag{2.5}$$

We say that in equations 2.4 and 2.5 $P(Z_{\boldsymbol{j}})$ is obtained from $P(\boldsymbol{\mathcal{Z}})$ by *marginalization* over random variables $Z_{\boldsymbol{\mathcal{L}}-\boldsymbol{j}}$.

2.2 Generalized probability density functions

We define a *generalized probability density function* (gpdf) to be a probability distribution for which the functional form is fixed. It thus only has set of parameters that determines its actual behavior. For discrete random variables, a gpdf is also known as a *probability mass function* (pmf); for real–valued random variables, we speak of *probability density functions*. However, since for both types of random variable, a gpdf is a function of a sample space to the real values, gpdfs may be referred to using a common name and a common representation (DeGroot, 1970). We denote a gpdf with parameters $\boldsymbol{\theta}$ by $P_{\boldsymbol{\theta}}(\boldsymbol{\mathcal{Z}})$. If we are given a predefined way of estimating the parameters $\boldsymbol{\theta}$ of a gpdf, we omit $\boldsymbol{\theta}$ and write $P(\boldsymbol{\mathcal{Z}})$.

A well–known gpdf for discrete integer random variables is the *frequency probability mass function* $P^{\mathcal{F}}_{\boldsymbol{\theta}}(\boldsymbol{\mathcal{X}}) : \Omega \rightarrow [0;1]$, characterized by $|\Omega|$ parameters:

$$P^{\mathcal{F}}_{\boldsymbol{\theta}}(\boldsymbol{\mathcal{X}})(\boldsymbol{x}) = \boldsymbol{\theta}_q \quad \text{where } q = \sum_{i=0}^{l-1} x_i \prod_{k=i+1}^{l-1} \left|\Omega^i\right| \tag{2.6}$$

For binary random variables for example, we have that $\Omega = \times_{i=0}^{l-1} \mathbb{B}$ and a binary string $\boldsymbol{x}$ is mapped onto its standard integer counterpart to index the vector of parameters:

$$P^{\mathcal{B}}_{\boldsymbol{\theta}}(\boldsymbol{\mathcal{X}})(\boldsymbol{x}) = \boldsymbol{\theta}_q \quad \text{where } q = \sum_{i=0}^{l-1} x_i 2^{l-1-i} \tag{2.7}$$

Given a vector of l–dimensional samples, denoted by $\boldsymbol{\mathcal{S}}$, the parameters for the frequency function $P^{\mathcal{F}}_{\boldsymbol{\theta}}(\boldsymbol{\mathcal{X}})$ are usually estimated by computing the proportion of each possible combination of values for the random variables, which leads to:

$$\hat{P}^{\mathcal{F}}(\boldsymbol{\mathcal{X}})(\boldsymbol{x}) = \frac{1}{|\boldsymbol{\mathcal{S}}|} \sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \begin{cases} 1 & \text{if } \boldsymbol{\mathcal{S}}_i = \boldsymbol{x} \\ 0 & otherwise \end{cases} \tag{2.8}$$

Note that the function requires the estimation of $2^l - 1$ parameters, since all parameters but one need to be estimated and the last parameter can be computed using the fact that the sum over all values must be equal to 1.

A well–known gpdf for real–valued random variables is the normal probability density function $P^{\mathcal{N}}_{\boldsymbol{\theta}}(\boldsymbol{\mathcal{Y}})$. The normal probability density function is

characterized by two parameters. These parameters are a mean vector $\boldsymbol{\mu}$ and a symmetric covariance matrix $\boldsymbol{\Sigma}$:

$$P^{\mathcal{N}}_{(\boldsymbol{\mu},\boldsymbol{\Sigma})}(\boldsymbol{\mathcal{Y}})(\boldsymbol{y}) = \frac{(2\pi)^{-\frac{|\boldsymbol{y}|}{2}}}{(\det \boldsymbol{\Sigma})^{\frac{1}{2}}} e^{-\frac{1}{2}(\boldsymbol{y}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{y}-\boldsymbol{\mu})} \tag{2.9}$$

Given a sample vector $\boldsymbol{\mathcal{S}}$, the mean vector and the covariance matrix are usually estimated by computing the sample average and the sample covariance matrix respectively:

$$\hat{\boldsymbol{\mu}} = \frac{1}{|\boldsymbol{\mathcal{S}}|} \sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \boldsymbol{\mathcal{S}}_i \tag{2.10}$$

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{|\boldsymbol{\mathcal{S}}|} \sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} (\boldsymbol{\mathcal{S}}_i - \hat{\boldsymbol{\mu}})(\boldsymbol{\mathcal{S}}_i - \hat{\boldsymbol{\mu}})^T$$

Since the covariance matrix $\boldsymbol{\Sigma}$ is symmetric, it requires the estimation of $l + \frac{1}{2}(l^2 - l)$ parameters.

2.3 Probabilistic models

A *probabilistic model* is closely related to a probability distribution. Informally speaking, a probabilistic model $\mathcal{M}$ is a computational description of a probability distribution. More formally, a probabilistic model consists of a structure ς and a vector of parameters $\boldsymbol{\theta}$, so $\mathcal{M} = (\varsigma, \boldsymbol{\theta})$. We denote the probability distribution associated with a probabilistic model $\mathcal{M}$ by $P_{\mathcal{M}}(\boldsymbol{\mathcal{Z}})$. The structure ς of a probabilistic model determines which parameters need to be estimated to obtain an actual probability distribution.

The task of estimating a probability distribution from data, is equivalent to the *learning* of a probabilistic model from data. Finding the structure of a probabilistic model is commonly known as *model selection*, whereas estimating the parameters of a probabilistic model is commonly known as *model fitting*. In some cases, the structure is predefined and only the parameters need to be estimated. Model selection then is trivial. To use predefined values for the parameters, however, makes no sense in most practical cases. Yet, it often does make sense to predefine the way in which the parameters of the model are estimated. In this case, we write $\boldsymbol{\theta} \xleftarrow{fit} \varsigma$ and allow ourselves to write $P_{\varsigma}(\boldsymbol{\mathcal{Z}})$ instead of $P_{\mathcal{M}}(\boldsymbol{\mathcal{Z}})$. Model fitting is required during model selection each time a structure needs to be compared to another structure. Thus, model selection and model fitting always go hand in hand in the learning of probabilistic models.

2.4 Factorized probability distributions

In this section, we review a specific class of probability distributions, called *factorized* probability distributions, or *factorizations* for short. Factorizations are

important from a computational point of view, since by exploiting the fact that random variables may be independent of each other, they reduce the number of parameters to be estimated to characterize a distribution. In this section, we focus on two types of factorization that can be obtained by combining *generalized probability density functions* (gpdfs). In Section 2.4.1 we describe multivariate factorizations and in Section 2.4.2, we review Bayesian factorizations.

2.4.1 Multivariately factorized probability distributions

Definition
A multivariately factorized probability distribution (also called multivariate factorization, marginal factorization or marginal product factorization) is a product of gpdfs over mutually exclusive vectors of random variables such that each $Z_i \in \boldsymbol{\mathcal{Z}}$ appears in exactly one such vector. For Cartesian sample spaces, such a product is a probability distribution over $\boldsymbol{\mathcal{Z}}$ (Lauritzen, 1996).

Probabilistic model
In a multivariate factorization, each random variable occurs in a single gpdf. Each vector of indices of random variables of a gpdf is called a *node vector*, denoted by $\boldsymbol{\nu}_i \sqsubseteq \boldsymbol{\mathcal{L}}$, $\boldsymbol{\nu}_i \sqcap \boldsymbol{\nu}_j = ()$ for each $i \neq j$. We call the vector of all node vectors of the multivariate factorization the *node partition vector* and denote it by $\boldsymbol{\nu}$. The node partition vector indicates the structure of a probabilistic model that describes the multivariate factorization. The parameters $\boldsymbol{\theta}$ of the probabilistic model are given by the parameters $\boldsymbol{\theta}^{\boldsymbol{\nu}_i}$ required for each gpdf. A multivariate factorization can now be characterized as follows:

$$P_{(\boldsymbol{\nu},\boldsymbol{\theta})}(\boldsymbol{\mathcal{Z}}) = \prod_{i=0}^{|\boldsymbol{\nu}|-1} P_{\boldsymbol{\theta}^{\boldsymbol{\nu}_i}}(Z_{\boldsymbol{\nu}_i}) \tag{2.11}$$

such that $\forall(\boldsymbol{\nu}_i, \boldsymbol{\nu}_j) \in (\boldsymbol{\nu} \times \boldsymbol{\nu}) : \boldsymbol{\nu}_i \neq () \ \wedge \ \boldsymbol{\nu}_i \sqsubseteq \boldsymbol{\mathcal{L}} \ \wedge \ i \neq j \rightarrow \boldsymbol{\nu}_i \sqcap \boldsymbol{\nu}_j = ()$.

Factorization graph
With any factorization, we associate a *factorization graph*. For a multivariate factorization, this graph has no edges. It consists of just nodes, where each node is identified with a gpdf from the factorization. The label of a node is given by the indices of the random variables in the gpdf associated with that node. Two examples of multivariate factorization graphs are given in Figure 2.1.

Independencies
In a multivariately factorized probability distribution, a random variable Z_i that is contained in some gpdf does *not* depend on any of the random variables that are contained in other gpdfs. Therefore, the random variables in a node vector are *independent* of all other random variables. If all random variables are independent of each other, meaning that each node vector contains only a single random variable, giving $|\boldsymbol{\nu}| = l$, the associated probability distribution is

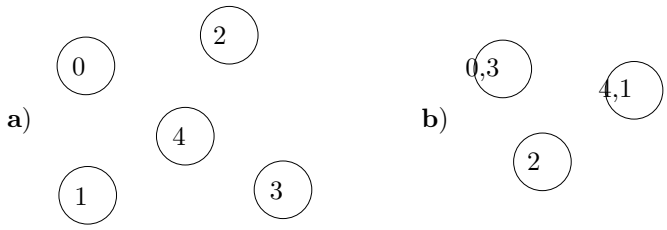


Figure 2.1: Two examples of factorization graphs for multivariately factorized probability distributions over 5 random variables. **a**) $P_{\boldsymbol{\nu}}(\boldsymbol{\mathcal{Z}}) = P(Z_0)P(Z_1)$ $P(Z_2)P(Z_3)P(Z_4)$, **b**) $P_{\boldsymbol{\nu}}(\boldsymbol{\mathcal{Z}}) = P(Z_0, Z_3)P(Z_2)P(Z_4, Z_1)$.

called the *univariate factorization*. Figure 2.1a shows the factorization graph of such a univariate factorization.

2.4.2 Bayesian factorized probability distributions

Definition

Let $\boldsymbol{\pi}_i \sqsubseteq \boldsymbol{\mathcal{L}}$, $i \in \boldsymbol{\mathcal{L}}$. A Bayesian factorization is a product $\prod_{i=0}^{l-1} P_{\boldsymbol{\theta}^i}(Z_i|Z_{\boldsymbol{\pi}_i})$ of l univariate conditional gpdfs. A univariate conditional gpdf $P_{\boldsymbol{\theta}^i}(Z_i|Z_{\boldsymbol{\pi}_i})$ is a function of $1 + |\boldsymbol{\pi}_i|$ random variables such that for each $z_{\boldsymbol{\pi}_i} \in \Omega^{\boldsymbol{\pi}_i}$ assigned to variables $Z_{\boldsymbol{\pi}_i}$, it is a gpdf over random variable Z_i. For the integer sample space, each gpdf $P^{\mathcal{F}}_{\boldsymbol{\theta}^i}(X_i|X_{\boldsymbol{\pi}_i})$ requires $(|\Omega^i| - 1)\prod_{k=0}^{|\boldsymbol{\pi}_i|-1} |\Omega^{(\boldsymbol{\pi}_i)_k}|$ parameters. For binary random variables for example, this number equals $2^{|\boldsymbol{\pi}_i|}$. In general, the conditional gpdf is defined as follows:

$$P(Z_{\boldsymbol{j}}|Z_{\boldsymbol{k}}) = \frac{P(Z_{\boldsymbol{j}\sqcup\boldsymbol{k}})}{P(Z_{\boldsymbol{k}})} \tag{2.12}$$

For $\prod_{i=0}^{l-1} P_{\boldsymbol{\theta}^i}(Z_i|Z_{\boldsymbol{\pi}_i})$ to be a probability distribution over the random variables $\boldsymbol{\mathcal{Z}}$, a permutation $\boldsymbol{\omega}$ of $\boldsymbol{\mathcal{L}}$ has to exist such that if we regard the variables in the order of the permutation, that is $Z_{\boldsymbol{\omega}_0}, Z_{\boldsymbol{\omega}_1}, \ldots Z_{\boldsymbol{\omega}_{l-1}}$, then upon regarding random variable $Z_{\boldsymbol{\omega}_i}$, variables $Z_{\boldsymbol{\pi}_{\boldsymbol{\omega}_i}}$ must already have been regarded. If this is not so, we can never draw a new sample from the product of univariate conditional gpdfs. With this restriction, such a product is again a probability distribution over $\boldsymbol{\mathcal{Z}}$ for Cartesian sample spaces (Lauritzen, 1996).

Probabilistic model

We call the vector of random variables indicated by $\boldsymbol{\pi}_i$ on which Z_i is conditioned, the vector of *parents* of Z_i. Moreover, we call $\boldsymbol{\pi}$ the *parent vector* that contains l vectors $\boldsymbol{\pi}_i$ that indicate the parents of each Z_i, $i \in \boldsymbol{\mathcal{L}}$. The parent vector indicates the structure of a probabilistic model that describes the Bayesian factorization. The parameters $\boldsymbol{\theta}$ of the probabilistic model are given by the parameters $\boldsymbol{\theta}^i$ required for each univariate conditional gpdf to be estimated. A Bayesian factorization can now be characterized as follows:

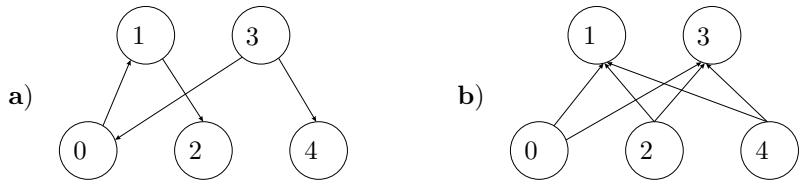


Figure 2.2: Two examples of factorization graphs for Bayesian factorized probability distributions over 5 random variables. The represented probability distributions and possible topological sorts are given by **a)** $P_{\boldsymbol{\pi}}(\boldsymbol{\mathcal{Z}}) = P(Z_0|Z_3) \cdot P(Z_1|Z_0)P(Z_2|Z_1)P(Z_3)P(Z_4|Z_3)$, $\boldsymbol{\omega} = (3,4,0,1,2)$, **b)** $P_{\boldsymbol{\pi}}(\boldsymbol{\mathcal{Z}}) = P(Z_0) \cdot P(Z_1|Z_0,Z_2,Z_4)P(Z_2)P(Z_3|Z_0,Z_2,Z_4)P(Z_4)$, $\boldsymbol{\omega} = (0,2,4,1,3)$.

$$P_{(\boldsymbol{\pi},\boldsymbol{\theta})}(\boldsymbol{\mathcal{Z}}) = \prod_{i=0}^{l-1} P_{\boldsymbol{\theta}^i}(Z_i|Z_{\boldsymbol{\pi}_i}) \tag{2.13}$$

such that $\exists \boldsymbol{\omega} \in perm(\boldsymbol{\mathcal{L}}) : \left(\forall i \in \boldsymbol{\mathcal{L}}, j \in \boldsymbol{\pi}_{\boldsymbol{\omega}_i} : j \in \bigcup_{k=0}^{i-1} \{\boldsymbol{\omega}_k\}\right)$.

Factorization graph

The factorization graph associated with a Bayesian factorization consists of a node for each random variable and an arc $i \to j$ if and only if Z_i is contained in the collection of variables that Z_j is conditioned on. This factorization graph is thus directed. From the definition of Bayesian factorization, it follows that in order for a directed graph to represent a Bayesian factorization, it must be acyclic. Therefore, a Bayesian factorization graph- allows a topological sorting of its nodes. Such a topological sort can be used to draw new samples from a Bayesian factorization. To ensure in a Bayesian factorization that we have values for the random variables that some random variable is conditioned on, it is sufficient to consider the factors in the order indicated by a topological sort $\boldsymbol{\omega}$. Two examples of Bayesian factorization graphs and possible topological sorts are given in the caption of Figure 2.2.

Independencies

The independencies described by a Bayesian factorization can be characterized in terms of conditional independence. A random variable Z_i is said to be *conditionally independent* of a collection of other random variables $Z_{\boldsymbol{j}}$, $\boldsymbol{j} \sqsubseteq \boldsymbol{\mathcal{L}} - (i)$ given a third collection of different random variables $Z_{\boldsymbol{k}}$, $\boldsymbol{k} \sqsubseteq \boldsymbol{\mathcal{L}} - ((i) \sqcup \boldsymbol{j})$, if $P(Z_i|Z_{\boldsymbol{j} \sqcup \boldsymbol{k}})$ is independent of the contribution by random variables $Z_{\boldsymbol{j}}$:

$$P(Z_i|Z_{\boldsymbol{j} \sqcup \boldsymbol{k}}) = P(Z_i|Z_{\boldsymbol{k}}) \tag{2.14}$$

Any random variable Z_j that can be reached from random variable Z_i in the factorization graph by following the arcs in the graph starting from Z_i, is called a *descendent* random variable of Z_i. The independence that is encoded in

a Bayesian factorization is that each random variable is conditionally independent of its non–descendent random variables given its parent random variables. This allows Bayesian factorizations to represent more fine–grained independency relations than do multivariate factorizations.

2.5 Likelihood, entropy and the Kullback–Leibler divergence

In this section we review three important basic measures that provide information about the relation between a probability distribution and a vector of samples. These measures are the likelihood, the entropy and the Kullback–Leibler divergence. Since in the remainder of this thesis, we will always use probability distributions through probabilistic models, we will introduce the measures using the notation for probabilistic models.

2.5.1 Likelihood

Given a vector $\boldsymbol{\mathcal{S}}$ of l–dimensional samples, the *likelihood* $\mathfrak{L}(P_{\boldsymbol{\mathcal{M}}}(\boldsymbol{\mathcal{Z}}), \boldsymbol{\mathcal{S}})$ of a probability distribution $P_{\boldsymbol{\mathcal{M}}}(\boldsymbol{\mathcal{Z}})$ over the random variables $\boldsymbol{\mathcal{Z}}$ is a measure of how likely this probability distribution is to give rise to the data in $\boldsymbol{\mathcal{S}}$. It is defined as the joint probability that all the samples were drawn independently from each other from the indicated probability distribution:

$$\mathfrak{L}\left(P_{\boldsymbol{\mathcal{M}}}(\boldsymbol{\mathcal{Z}}), \boldsymbol{\mathcal{S}}\right) = \prod_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} P_{\boldsymbol{\mathcal{M}}}(\boldsymbol{\mathcal{Z}})(\boldsymbol{\mathcal{S}}_i) \tag{2.15}$$

It is common practice to use the negative logarithm of the likelihood measure as it is often computationally more convenient. The resulting expression is called the *negative log–likelihood*:

$$-\ln\left(\mathfrak{L}\left(P_{\boldsymbol{\mathcal{M}}}(\boldsymbol{\mathcal{Z}}), \boldsymbol{\mathcal{S}}\right)\right) = -\sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \ln\left(P_{\boldsymbol{\mathcal{M}}}(\boldsymbol{\mathcal{Z}})(\boldsymbol{\mathcal{S}}_i)\right) \tag{2.16}$$

A probability distribution $P_{\boldsymbol{\mathcal{M}}}(\boldsymbol{\mathcal{Z}})$ over the random variables $\boldsymbol{\mathcal{Z}}$ is said to be a *maximum likelihood estimation* if and only if:

$$\neg\exists \boldsymbol{\theta}' : \mathfrak{L}\left(P_{(\varsigma,\boldsymbol{\theta}')}(\boldsymbol{\mathcal{Z}}), \boldsymbol{\mathcal{S}}\right) > \mathfrak{L}\left(P_{(\varsigma,\boldsymbol{\theta})}(\boldsymbol{\mathcal{Z}}), \boldsymbol{\mathcal{S}}\right) \tag{2.17}$$

The commonly used proportion estimation of the gpdf for the discrete integer sample space as well as the sample average and sample covariance matrix for the normal gpdf as given in Section 2.2, have been proven to result in *maximum* likelihood probability distributions (Anderson, 1958; Tatsuoka, 1971).

Each factor $P_{\boldsymbol{\theta}^{\boldsymbol{\nu}_i}}(Z_{\boldsymbol{\nu}_i})$ in a multivariate factorization $P_{\boldsymbol{\nu}}(\boldsymbol{\mathcal{Z}})$ as well as each factor $P_{\boldsymbol{\theta}^i}(Z_i|Z_{\boldsymbol{\pi}_i})$ in a Bayesian factorization $P_{\boldsymbol{\pi}}(\boldsymbol{\mathcal{Z}})$ has its own set of parameters to be estimated. If each gpdf $P_{\boldsymbol{\theta}^{\boldsymbol{\nu}_i}}(Z_{\boldsymbol{\nu}_i})$ is estimated to be of maximum

likelihood, then the multivariate factorization is also of maximum likelihood. Similarly, if each gpdf $P_{\boldsymbol{\theta}^i}(Z_i|Z_{\boldsymbol{\pi}_i})$ is estimated to be of maximum likelihood, then the Bayesian factorization is also of maximum likelihood. Furthermore, $P_{\boldsymbol{\theta}^i}(Z_i|Z_{\boldsymbol{\pi}_i})$ can be estimated to be of maximum likelihood by computing maximum likelihood estimations $\hat{P}(Z_{(i)\sqcup\boldsymbol{\pi}_i})$ and $\hat{P}(Z_{\boldsymbol{\pi}_i})$ and by subsequently computing $\hat{P}(Z_{(i)\sqcup\boldsymbol{\pi}_i})/\hat{P}(Z_{\boldsymbol{\pi}_i})$ in accordance with the definition of the conditional gpdf (Dawid and Lauritzen, 1993; Buntine, 1994; Heckerman and Geiger, 1995).

2.5.2 Entropy

The *entropy* of a probability distribution $P_{\boldsymbol{\mathcal{M}}}(\boldsymbol{\mathcal{Z}})$ over the random variables $\boldsymbol{\mathcal{Z}}$ is a measure of the amount of uncertainty inherent in this probability distribution. Entropy is high when all possibilities are almost equally probable, which corresponds to a situation of little information and thus high uncertainty. Entropy is low in the reverse situation. For the integer sample space, the entropy measure is defined as:

$$h\left(P_{\mathcal{M}}(\boldsymbol{\mathcal{X}})\right) = -\sum_{\boldsymbol{x}\in\Omega} P_{\mathcal{M}}(\boldsymbol{\mathcal{X}})(\boldsymbol{x})\ln\left(P_{\mathcal{M}}(\boldsymbol{\mathcal{X}})(\boldsymbol{x})\right) \tag{2.18}$$

For real–valued random variables, the entropy measure is also called *differential entropy*; it is defined as follows:

$$h\left(P_{\mathcal{M}}(\boldsymbol{\mathcal{Y}})\right) = -\int_{\boldsymbol{y}\in\Omega} P_{\mathcal{M}}(\boldsymbol{\mathcal{Y}})(\boldsymbol{y})\ln\left(P_{\mathcal{M}}(\boldsymbol{\mathcal{Y}})(\boldsymbol{y})\right) d\boldsymbol{y} \tag{2.19}$$

A probability distribution $P_{\mathcal{M}}(\boldsymbol{\mathcal{Z}})$ over the random variables $\boldsymbol{\mathcal{Z}}$ is said to have *minimal entropy* if and only if:

$$\neg\exists\boldsymbol{\theta}' : h\left(P_{(\varsigma,\boldsymbol{\theta}')}(\boldsymbol{\mathcal{Z}})\right) < h\left(P_{(\varsigma,\boldsymbol{\theta})}(\boldsymbol{\mathcal{Z}})\right) \tag{2.20}$$

It was shown by Kullback (1959) that a probability distribution is of maximum likelihood if and only if its *entropy* is minimal. In other words, minimal negative log–likelihood is equivalent to minimal entropy. In fact, it can be shown that if maximum likelihood gpdf estimations are used to construct a factorized probability distribution, then the relation between the entropy and the negative log–likelihood is $|\boldsymbol{\mathcal{S}}|h(P_{\mathcal{M}}(\boldsymbol{\mathcal{Z}})) = -\ln(\mathfrak{L}(P_{\mathcal{M}}(\boldsymbol{\mathcal{Z}}),\boldsymbol{\mathcal{S}}))$.

2.5.3 Kullback–Leibler divergence

The *Kullback–Leibler divergence* (KL divergence) is a measure that gives information about how different two probability distributions $P_{\mathcal{M}^0}(\boldsymbol{\mathcal{Z}})$ and $P_{\mathcal{M}^1}(\boldsymbol{\mathcal{Z}})$ are. As such, it can be considered to be a kind of a distance measure, though it is not a real distance measure because it is not symmetric. The KL divergence is also known as *relative entropy* (Kullback, 1959). For the integer sample space, the KL divergence equals:

$$d(P_{\mathcal{M}^0}(\boldsymbol{\mathcal{X}}) \,||\, P_{\mathcal{M}^1}(\boldsymbol{\mathcal{X}})) = \sum_{\boldsymbol{x}\in\Omega} P_{\mathcal{M}^0}(\boldsymbol{\mathcal{X}})(\boldsymbol{x}) \ln\left(\frac{P_{\mathcal{M}^0}(\boldsymbol{\mathcal{X}})(\boldsymbol{x})}{P_{\mathcal{M}^1}(\boldsymbol{\mathcal{X}})(\boldsymbol{x})}\right) \tag{2.21}$$

In the case of real–valued random variables, the KL divergence equals:

$$d(P_{\mathcal{M}^0}(\boldsymbol{\mathcal{Y}}) \,||\, P_{\mathcal{M}^1}(\boldsymbol{\mathcal{Y}})) = \int_{\boldsymbol{y}\in\Omega} P_{\mathcal{M}^0}(\boldsymbol{\mathcal{Y}})(\boldsymbol{y}) \ln\left(\frac{P_{\mathcal{M}^0}(\boldsymbol{\mathcal{Y}})(\boldsymbol{y})}{P_{\mathcal{M}^1}(\boldsymbol{\mathcal{Y}})(\boldsymbol{y})}\right) d\boldsymbol{y} \tag{2.22}$$

Using these definitions, it can be shown that the KL divergence for integer random variables between the multivariate gpdf for all random variables and a multivariate factorization $\boldsymbol{\nu}$ for which the gpdfs were estimated to be of maximum likelihood, equals:

$$d(P(\boldsymbol{\mathcal{X}}) \,||\, P_{\boldsymbol{\nu}}(\boldsymbol{\mathcal{X}})) = -h(P(\boldsymbol{\mathcal{X}})) + h(P_{\boldsymbol{\nu}}(\boldsymbol{\mathcal{X}})) \tag{2.23}$$

A similar property holds for real–valued random variables. Moreover, a similar property can be obtained for Bayesian factorizations for integer random variables as well as for real–valued random variables by using the definition of conditional entropy $h(P(X_i|X_{\boldsymbol{j}})) = h(P(X_{(i)\sqcup\boldsymbol{j}})) - h(P(X_{\boldsymbol{j}}))$ and conditional differential entropy $h(P(Y_i|Y_{\boldsymbol{j}})) = h(P(Y_{(i)\sqcup\boldsymbol{j}})) - h(P(Y_{\boldsymbol{j}}))$:

$$d(P(\boldsymbol{\mathcal{Y}}) \,||\, P_{\boldsymbol{\nu}}(\boldsymbol{\mathcal{Y}})) = -h(P(\boldsymbol{\mathcal{Y}})) + h(P_{\boldsymbol{\nu}}(\boldsymbol{\mathcal{Y}})) \tag{2.24}$$

$$d(P(\boldsymbol{\mathcal{X}}) \,||\, P_{\boldsymbol{\pi}}(\boldsymbol{\mathcal{X}})) = -h(P(\boldsymbol{\mathcal{X}})) + h(P_{\boldsymbol{\pi}}(\boldsymbol{\mathcal{X}})) \tag{2.25}$$

$$d(P(\boldsymbol{\mathcal{Y}}) \,||\, P_{\boldsymbol{\pi}}(\boldsymbol{\mathcal{Y}})) = -h(P(\boldsymbol{\mathcal{Y}})) + h(P_{\boldsymbol{\pi}}(\boldsymbol{\mathcal{Y}})) \tag{2.26}$$

2.6 Greedy model selection

To select a good probabilistic model structure is a difficult task. In this section, we outline a general greedy approach for model selection. First, we present a general algorithmic framework in Section 2.6.1. Subsequently, in Section 2.6.2, we describe two well known model selection techniques and indicate how they fit into the greedy algorithmic framework. Lastly, we show how we can use the greedy algorithm based on the extended likelihood principle to perform greedy incremental selection of multivariate factorizations (Section 2.6.3) and Bayesian factorizations (Section 2.6.4).

2.6.1 A general framework

To perform greedy model selection, we take an incremental approach. At any moment, we have a single current model $\mathcal{M}^0$. Furthermore, an operation is available that can be applied to a probabilistic model, resulting in a collection of candidate probabilistic models. The greedy aspect of our model selection framework lies in the fact that from this collection of candidate probabilistic models combined with the current model, we select the most promising one, according to certain criteria that are to be defined later. If, according to these criteria, the most promising candidate model is different from the current model, the current model is replaced with the most promising model and the process is repeated. Otherwise, the greedy model selection algorithm terminates and returns the current model as the result. Pseudo–code for this greedy model selection approach is given in Figure 2.3. Learning the structure of a probabilistic model is actually an optimization problem since we seek the best possible probabilistic model. Especially when building a probabilistic model is not the main goal, but a subtask in a larger dynamic system, a greedy approach to learning probabilistic models from data is very plausible.

```
GreedyModelSelection()
1  M^0 ← IntitialModel()
2  terminate ← false
3  while ¬terminate do
   3.1  C ← GetCandidateModels(M^0)
   3.2  M^1 ← GetBestModel(M^0, C)
   3.3  if M^1 ≠ M^0 then
            3.3.1  M^0 ← M^1
   3.4  else
            3.4.1  terminate ← true
4  return(M^0)
```

Figure 2.3: Pseudo–code for greedy model selection.

2.6.2 Choosing between models

To use the general greedy model selection algorithmic framework, we must be able to construct an initial model. Small changes are then usually made to this model to obtain the candidate models. We must then be able to distinguish between models and select the best one. In this section, we assume that a collection of candidate models is available and elaborate on how we can decide whether one model is better than another one. In sections 2.6.3 and 2.6.4 we shall show how such a set can be obtained if the probabilistic model represents a multivariate factorization or a Bayesian factorization.

There are many ways in which to choose between two given probabilistic models such as by means of the likelihood (Casella and Berger, 1990; Geiger,

1992), the extended likelihood (Sclove, 1994; Poland and Shachter, 1994), statistical hypothesis testing (Chernoff, 1954; Kendall and Stuart, 1967; Feller, 1968; Brandt, 1970; Trivedi, 1982; Edwards, 1995), resampling (Fung and Crawford, 1990; Efron and Tibshirani, 1991) or by Bayesian approaches (Tanner, 1992; Bernardo and Smith, 1994; Kass and Raftery, 1995). We refrain from giving an extensive overview of all different approaches known in the literature. Instead, we leave such information for the reader to find in the indicated specialized publications or literature overviews (Buntine, 1996). The model selection approaches that we do describe in this section, use the likelihood of the probability distribution or an extended version thereof. First, we indicate the consequences of using the likelihood directly to choose between probabilistic models. Then, we show how the likelihood principle can be extended to prevent overfitting. This extended likelihood approach is discussed in somewhat more detail as we use it throughout the remainder of this thesis.

Likelihood

In the likelihood approach, the model with largest likelihood is selected. There is a problem with this approach if we want to use it to select a factorization with our greedy algorithm however. As indicated in Section 2.5, the likelihood approach is equivalent to selecting the factorization with the smallest negative log–likelihood, which is equivalent with the smallest entropy if maximum likelihood gpdf estimations are used. Let $\mathcal{M}^0$ denote the probabilistic model of the current factorization and let $\mathcal{M}^1$ denote the probabilistic model of a candidate factorization. Using the results in Section 2.5 and letting d and h stand for the KL divergence and entropy for both the binary integer as well as the real–valued sample space momentarily, we have that:

$$d(P(\boldsymbol{\mathcal{Z}}) \,||\, P_{\mathcal{M}^0}(\boldsymbol{\mathcal{Z}})) - d(P(\boldsymbol{\mathcal{Z}}) \,||\, P_{\mathcal{M}^1}(\boldsymbol{\mathcal{Z}})) = h(P_{\mathcal{M}^0}(\boldsymbol{\mathcal{Z}})) - h(P_{\mathcal{M}^1}(\boldsymbol{\mathcal{Z}})) \quad (2.27)$$

Assuming maximum likelihood gpdf estimations, the KL divergence with respect to the full multivariate joint gpdf $P(\boldsymbol{\mathcal{Z}})$ increases monotonically as the complexity of the factorization increases. Thus, selecting a model based on the minimal entropy, the maximum likelihood or the minimal KL divergence to the full multivariate joint gpdf is equivalent and results in an approach that always prefers the more complex factorization, i.e. the one that yields the most parameters to be estimated:

$$\begin{aligned} &arg\ \max\{\mathfrak{L}(P_{\mathcal{M}^0}(\boldsymbol{\mathcal{Z}}), \boldsymbol{\mathcal{S}}), \mathfrak{L}(P_{\mathcal{M}^1}(\boldsymbol{\mathcal{Z}}), \boldsymbol{\mathcal{S}})\} = \\ &arg\ \min\{h(P_{\mathcal{M}^0}(\boldsymbol{\mathcal{Z}})), h(P_{\mathcal{M}^1}(\boldsymbol{\mathcal{Z}}))\} = \\ &arg\ \min\{d(P(\boldsymbol{\mathcal{Z}}) \,||\, P_{\mathcal{M}^0}(\boldsymbol{\mathcal{Z}})), d(P(\boldsymbol{\mathcal{Z}}) \,||\, P_{\mathcal{M}^1}(\boldsymbol{\mathcal{Z}}))\} = \\ &arg\ \max\{|\boldsymbol{\theta} \xleftarrow{fit} \mathcal{M}^0|, |\boldsymbol{\theta} \xleftarrow{fit} \mathcal{M}^1|\} \end{aligned} \quad (2.28)$$

Extended likelihood and minimum description length

The negative log–likelihood can be seen as an error that should be minimized. The more likely the fit, the larger the likelihood and thus the smaller the negative log–likelihood. However, even though a fit with a larger likelihood may indeed be better than a fit with a smaller likelihood, this does not mean that we are interested in it from either a *generalization* or a computational effectiveness point of view. If the complexity of one of the candidate factorizations is higher in that it has more parameters to be estimated, using the difference in negative log–likelihood in the greedy model selection algorithm will result in the maximum complexity factorization if maximum likelihood gpdf estimations are used. This is the result of *overfitting*. When using more complex models, more parameters can be tuned to fit the data perfectly. The most important point to note here is that the goodness of a probability distribution estimation is given by its ability to *generalize* the data. Since the collection of available samples is never the complete set of all possible samples, the probability distribution must be good at predicting new samples that are similar to the ones available. If the probability distribution is overfit, the likelihood on the available sample collection may be very high, but may actually be quite bad on another sample collection drawn from the same underlying true probability distribution. This has a direct relation to the concept of *Occam's razor*, which is a principle stated by William of Occam in the 14th century. Freely translated, his principle states that all things being equal, the simplest explanation tends to be better. To provide a means so as to avoid the introduction of unnecessary and unwanted complexity into the estimated probability distribution, a penalty term that increasingly penalizes more complex models can be introduced. The metric $\mathfrak{M}$ that should be minimized can then be formalized as follows:

$$\mathfrak{M}(P_{\mathcal{M}}(\boldsymbol{\mathcal{Z}}), \boldsymbol{\mathcal{S}}) = \text{Error}(P_{\mathcal{M}}(\boldsymbol{\mathcal{Z}}), \boldsymbol{\mathcal{S}}) + \text{Complexity}(P_{\mathcal{M}}(\boldsymbol{\mathcal{Z}})) \tag{2.29}$$

Choosing a model based on a metric of the form as specified in equation 2.29, is known as an *extended likelihood* or *complexity penalization* approach. These approaches have a quite similar foundation as do minimum information complexity approaches. Examples of minimum information complexity approaches are minimum description length (Rissanen, 1987), minimum message length (Wallace and Freeman, 1987) and minimum complexity (Barron and Cover, 1991). These approaches are also rooted in Occam's razor.

Regularization

The use of a complexity term is often termed *regularization*. Classically, the regularization equation has a *regularization parameter* λ as a factor times the complexity term. Setting $\lambda = 0$ then reduces the metric to an unpenalized one with respect to error minimization. Here, we have used a general complexity term that could be of such a form. However, we leave the contents of the complexity term completely unspecified, which gives no restrictions on the term.

A probabilistic derivation of extended likelihood metrics

Since the sample vector $\boldsymbol{\mathcal{S}}$ is given, from a probabilistic point of view we are interested in the probability of a probability distribution associated with a probabilistic model $\mathcal{M}$, given $\boldsymbol{\mathcal{S}}$. The model that results in the largest probability, is the most preferable. If we assume that we are given a predefined way to estimate the parameters for the probabilistic model given the structure of the model, that is $\boldsymbol{\theta} \overset{fit}{\leftarrow} \varsigma$, the probability that we are interested in, can be written as follows using Bayes' Rule:

$$\Pr(P_\varsigma(\boldsymbol{\mathcal{Z}})|\boldsymbol{\mathcal{S}}) = \frac{\Pr(P_\varsigma(\boldsymbol{\mathcal{Z}}))\Pr(\boldsymbol{\mathcal{S}}|P_\varsigma(\boldsymbol{\mathcal{Z}}))}{\Pr(\boldsymbol{\mathcal{S}})} \tag{2.30}$$

Because the sample vector $\boldsymbol{\mathcal{S}}$ is fixed and thus $\Pr(\boldsymbol{\mathcal{S}}) = 1$, we can discard $\Pr(\boldsymbol{\mathcal{S}})$ from equation 2.30. The probability of $\boldsymbol{\mathcal{S}}$ given $P_\varsigma(\boldsymbol{\mathcal{Z}})$, that is $\Pr(\boldsymbol{\mathcal{S}}|P_\varsigma(\boldsymbol{\mathcal{Z}}))$, can be seen as the likelihood as defined in equation 2.15, i.e. $\mathfrak{L}(P_\varsigma(\boldsymbol{\mathcal{Z}}), \boldsymbol{\mathcal{S}})$. To add a preference for simpler models following Occam's razor, we want to set $\Pr(P_\varsigma(\boldsymbol{\mathcal{Z}}))$ in such a way that this probability gets smaller if the complexity of $P_\varsigma(\boldsymbol{\mathcal{Z}})$ increases. The complexity of $P_\varsigma(\boldsymbol{\mathcal{Z}})$ is given by the number of parameters $\boldsymbol{\theta} \overset{fit}{\leftarrow} \varsigma$ that are required. Let $\mathfrak{S}$ be the space of all possible structures that we are interested in, such as for instance all possible Bayesian factorizations for all random variables $\boldsymbol{\mathcal{Z}}$. We can then write $\Pr(P_\varsigma(\boldsymbol{\mathcal{Z}}))$ as a function $\vartheta(\cdot)$, $\vartheta(\cdot) \geq 0$, of the parameter vector:

$$\Pr(P_\varsigma(\boldsymbol{\mathcal{Z}})) = \frac{\vartheta(\boldsymbol{\theta} \overset{fit}{\leftarrow} \varsigma)}{\sum_{\varsigma' \in \mathfrak{S}} \vartheta(\boldsymbol{\theta'} \overset{fit}{\leftarrow} \varsigma')} \tag{2.31}$$

To compare two models, the normalization factor $\sum_{\varsigma' \in \mathfrak{S}} \vartheta(\boldsymbol{\theta'} \overset{fit}{\leftarrow} \varsigma')$ in equation 2.31 may be discarded since this factor is the same for every model structure. As a result however, we then no longer have a probability distribution but a metric to be maximized that is a function of the probability distribution $P_\varsigma(\boldsymbol{\mathcal{Z}})$ and the sample collection $\boldsymbol{\mathcal{S}}$. Alternatively, we can minimize the negative logarithm of that metric and we obtain a metric that has a component that is dependent on the likelihood (error component) and a component that is dependent on the complexity of the probability distribution:

$$-\ln\left(\mathfrak{M}(P_\mathcal{M}(\boldsymbol{\mathcal{Z}}), \boldsymbol{\mathcal{S}})\right) = \underbrace{-\ln\left(\mathfrak{L}(P_\mathcal{M}(\boldsymbol{\mathcal{Z}})), \boldsymbol{\mathcal{S}}\right)}_{\text{Error}(P_\mathcal{M}(\boldsymbol{\mathcal{Z}}), \boldsymbol{\mathcal{S}})} - \underbrace{\ln\left(\vartheta(\boldsymbol{\theta} \overset{fit}{\leftarrow} \varsigma)\right)}_{\text{Complexity}(P_\mathcal{M}(\boldsymbol{\mathcal{Z}}))} \tag{2.32}$$

Depending on how we choose to weight the complexity of a probability distribution, we arrive at different metrics.

Likelihood as a (trivial) metric

If we take each probabilistic model to be equally likely, and thus actually perform no complexity penalization at all, we set $\vartheta(\cdot) = 1$ and simply end up with the likelihood, which is a trivial instance of equation 2.29:

$$\text{Neg. Log–Likelihood} = \underbrace{-\ln(\mathfrak{L}(P_{\mathcal{M}}(\boldsymbol{\mathcal{Z}})), \boldsymbol{\mathcal{S}})}_{\text{Error}(P_{\mathcal{M}}(\boldsymbol{\mathcal{Z}}), \boldsymbol{\mathcal{S}})} - \underbrace{0}_{\text{Complexity}(P_{\mathcal{M}}(\boldsymbol{\mathcal{Z}}))} \tag{2.33}$$

The Akaike Information Criterion

Alternatively, a simple way to add a bias towards less complex models is to let the probability of some probability distribution $\Pr(P_{\varsigma}(\boldsymbol{\mathcal{Z}}))$ decrease exponentially with its complexity by setting $\vartheta(\boldsymbol{\theta}) = e^{-|\boldsymbol{\theta}|}$. By doing so, we get a different instance of equation 2.29:

$$AIC(P_{\mathcal{M}}(\boldsymbol{\mathcal{Z}}), \boldsymbol{\mathcal{S}}) = \underbrace{-\ln(\mathfrak{L}(P_{\mathcal{M}}(\boldsymbol{\mathcal{Z}})), \boldsymbol{\mathcal{S}})}_{\text{Error}(P_{\mathcal{M}}(\boldsymbol{\mathcal{Z}}), \boldsymbol{\mathcal{S}})} + \underbrace{|\boldsymbol{\theta}|}_{\text{Complexity}(P_{\mathcal{M}}(\boldsymbol{\mathcal{Z}}))} \tag{2.34}$$

In the metric in equation 2.34, the negative log–likelihood is the error to minimize and the complexity term is the number of parameters in the probabilistic model. The resulting metric is known as the Akaike Information Criterion (AIC) (Akaike, 1973). If two probability distributions have the same likelihood, the one with the smaller number of parameters is favored by the AIC metric.

The Bayesian Information Criterion

At this point, we turn back to our note on regularization. Even though the AIC metric favors simpler models by increasingly penalizing more complex ones, the amount of penalization is fixed. By using a regularization parameter λ times the complexity term in the AIC metric, the regularization could be increased. Other than by increasing a regularization parameter, we can penalize the complexity differently. The derivation we have used so far is a very general one. To demonstrate its transparent use, we propose to set $\vartheta(\boldsymbol{\theta}) = e^{-\lambda \ln(|\boldsymbol{\mathcal{S}}|)|\boldsymbol{\theta}|}$. By doing so, we have introduced a parameter λ that has the same semantics as the regularization parameter. The resulting metric can be written as follows:

$$BIC(P_{\mathcal{M}}(\boldsymbol{\mathcal{Z}}), \boldsymbol{\mathcal{S}}) = \underbrace{-\ln(\mathfrak{L}(P_{\mathcal{M}}(\boldsymbol{\mathcal{Z}})), \boldsymbol{\mathcal{S}})}_{\text{Error}(P_{\mathcal{M}}(\boldsymbol{\mathcal{Z}}), \boldsymbol{\mathcal{S}})} + \underbrace{\lambda \ln(|\boldsymbol{\mathcal{S}}|)|\boldsymbol{\theta}|}_{\text{Complexity}(P_{\mathcal{M}}(\boldsymbol{\mathcal{Z}}))} \tag{2.35}$$

The metric in equation 2.35 with $\lambda = \frac{1}{2}$ is commonly known as the *Bayesian Information Criterion* (BIC) (Schwarz, 1978). The BIC metric has empirically been observed to give good results for the greedy estimation of factorized probability distributions (Mühlenbein and Mahnig, 1999; Bosman and Thierens, 2001a; Larrañaga and Lozano, 2001). Schwarz (1978) has indicated that the AIC metric cannot be asymptotically optimal under a few general assumptions that were used in the derivation of the BIC metric.

Some remarks

To choose between different probabilistic models, we will use the AIC and BIC penalization metrics in this thesis. The penalty in these metrics increases as the complexity of the model increases so as to prevent unnecessarily complex models to be selected. The difficulty with such an approach is choosing the *right*

amount of penalization, which is equivalent to the amount of *regularization*. Selecting the right amount of regularization is a central problem in statistical learning. In our application, the amount of regularization can be seen as a parameter that defines how much time the model selection algorithm is allowed to spend on model learning. When using factorizations for instance, a penalty term is something of a substitute for limiting the maximum order of dependency since the penalty term increases as the complexity of the factorization increases, which is directly related to the maximum order of dependency in the factorization. However, using a penalization metric in addition influences the decision of which operations on the factorization are beneficial. Seen in this way, a penalization metric is a practical substitute for using statistical hypothesis tests that can be used to determine whether some operation is significantly beneficial.

2.6.3 Greedy multivariate factorization selection

In this section we discuss the selection of multivariate factorizations using the greedy model selection framework from Section 2.6.1 and the extended likelihood metrics from Section 2.6.2. We sequentially discuss the initialization phase, the construction of candidate factorizations and the selection of a new factorization to replace the current one.

Initialization

Following the principle of Occam's razor in hoping to model the sample collection with the simplest, but still accurate, model possible, it makes sense to start out in the greedy model selection algorithm from Section 2.6.1 with the least complex model possible. In the case of a multivariate factorization, this is the univariate factorization, which is obtained by setting the model structure to $\boldsymbol{\nu} = ((0), (1), \ldots, (l-1))$.

Constructing candidate factorizations

For multivariate factorizations, we propose to use a single basic operation. The structures of the candidate models are obtained by splicing any two factors in the multivariately factorized probability distribution. In other words, if $\boldsymbol{\nu}^0$ is the structure of the current multivariate factorization, the set of candidate structures is given by:

$$\left\{(\boldsymbol{\nu}^0 - (\boldsymbol{\nu}_i^0, \boldsymbol{\nu}_j^0)) \sqcup (\boldsymbol{\nu}_i^0 \sqcup \boldsymbol{\nu}_j^0) \mid (i,j) \in \{0,1,\ldots,|\boldsymbol{\nu}^0|-1\}^2 \wedge j > i\right\} \tag{2.36}$$

Choosing the best candidate factorization

To find the best candidate model based on a complexity penalization metric to be minimized, we compare each candidate structure $\boldsymbol{\nu}^1$ that can be constructed using the splice operator, with the current structure $\boldsymbol{\nu}^0$ and observe

the difference in the penalization metric $\mathfrak{M}(P_{\boldsymbol{\nu}^0}, \boldsymbol{\mathcal{S}}) - \mathfrak{M}(P_{\boldsymbol{\nu}^1}, \boldsymbol{\mathcal{S}})$. The candidate structure that results in the largest difference, is selected. If no difference is positive, there is no improvement among the candidate structures. Since the penalization metrics in Section 2.6.2 are a sum of the negative log–likelihood and a complexity term, we can compute their contribution to the difference separately.

Negative log–likelihood term

Suppose that node vectors $\boldsymbol{\nu}_i^0$ and $\boldsymbol{\nu}_j^0$ were spliced to construct the candidate structure $\boldsymbol{\nu}^1$ from the current structure $\boldsymbol{\nu}^0$, so $\boldsymbol{\nu}^0 - (\boldsymbol{\nu}_i^0, \boldsymbol{\nu}_j^0) = \boldsymbol{\nu}^1 - (\boldsymbol{\nu}_i^0 \sqcup \boldsymbol{\nu}_j^0)$. Let $\boldsymbol{\mathcal{S}}^{\boldsymbol{j}} = (\boldsymbol{\mathcal{S}}_0^{\boldsymbol{j}}, \boldsymbol{\mathcal{S}}_1^{\boldsymbol{j}}, \ldots, \boldsymbol{\mathcal{S}}_{|\boldsymbol{\mathcal{S}}|-1}^{\boldsymbol{j}})$ be a vector containing a truncated version of each sample point in $\boldsymbol{\mathcal{S}}$, meaning that for each sample point only the dimensions indicated by $\boldsymbol{j}$ are in $\boldsymbol{\mathcal{S}}^{\boldsymbol{j}}$, that is, $(\boldsymbol{\mathcal{S}}^{\boldsymbol{j}})_i = \boldsymbol{\mathcal{S}}_i^{\boldsymbol{j}} = (\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{j}}$. Now, for the negative log–likelihood, we obtain the following for discrete integer random variables and real–valued random variables:

$$-\ln\left(\mathfrak{L}(P_{\boldsymbol{\nu}^0}(\boldsymbol{\mathcal{Z}}), \boldsymbol{\mathcal{S}})\right) - \left(-\ln\left(\mathfrak{L}(P_{\boldsymbol{\nu}^1}(\boldsymbol{\mathcal{Z}}), \boldsymbol{\mathcal{S}})\right)\right) = \tag{2.37}$$

$$\sum_{k=0}^{|\boldsymbol{\nu}^1|-1} \sum_{q=0}^{|\boldsymbol{\mathcal{S}}|-1} \ln\left(P(Z_{\boldsymbol{\nu}_k^1})((\boldsymbol{\mathcal{S}}_q)_{\boldsymbol{\nu}_k^1})\right) - \sum_{k=0}^{|\boldsymbol{\nu}^0|-1} \sum_{q=0}^{|\boldsymbol{\mathcal{S}}|-1} \ln\left(P(Z_{\boldsymbol{\nu}_k^0})((\boldsymbol{\mathcal{S}}_q)_{\boldsymbol{\nu}_k^0})\right) =$$

$$\sum_{q=0}^{|\boldsymbol{\mathcal{S}}|-1} \ln\left(P(Z_{\boldsymbol{\nu}_i^0 \sqcup \boldsymbol{\nu}_j^0})((\boldsymbol{\mathcal{S}}_q)_{\boldsymbol{\nu}_i^0 \sqcup \boldsymbol{\nu}_j^0})\right) -$$

$$\sum_{q=0}^{|\boldsymbol{\mathcal{S}}|-1} \ln\left(P(Z_{\boldsymbol{\nu}_i^0})((\boldsymbol{\mathcal{S}}_q)_{\boldsymbol{\nu}_i^0})\right) - \sum_{q=0}^{|\boldsymbol{\mathcal{S}}|-1} \ln\left(P(Z_{\boldsymbol{\nu}_j^0})((\boldsymbol{\mathcal{S}}_q)_{\boldsymbol{\nu}_j^0})\right) =$$

$$\ln\left(\mathfrak{L}(P(Z_{\boldsymbol{\nu}_i^0 \sqcup \boldsymbol{\nu}_j^0}), \boldsymbol{\mathcal{S}}^{\boldsymbol{\nu}_i^0 \sqcup \boldsymbol{\nu}_j^0})\right) - \ln\left(\mathfrak{L}(P(Z_{\boldsymbol{\nu}_i^0}), \boldsymbol{\mathcal{S}}^{\boldsymbol{\nu}_i^0})\right) - \ln\left(\mathfrak{L}(P(Z_{\boldsymbol{\nu}_j^0}), \boldsymbol{\mathcal{S}}^{\boldsymbol{\nu}_j^0})\right)$$

Complexity term

The parameters for a multivariate factorization are given by the parameters for each gpdf factor, $|\boldsymbol{\theta} \xleftarrow{fit} \boldsymbol{\nu}| = \sum_{i=0}^{|\boldsymbol{\nu}|-1} |\boldsymbol{\theta} \xleftarrow{fit} \boldsymbol{\nu}_i|$. In a similar manner as for the negative log–likelihood, the contributions of the factors in $\boldsymbol{\nu}^0$ that were not involved in the splice operation that formed $\boldsymbol{\nu}^1$, cancel out in the difference.

Penalization metric difference after a splice operation

As a result, we get the following final results for the decrease in the AIC and BIC metrics as a result of a splice operation:

$$\mathfrak{M}(P_{\boldsymbol{\nu}^0}(\boldsymbol{\mathcal{Z}}), \boldsymbol{\mathcal{S}}) - \mathfrak{M}(P_{\boldsymbol{\nu}^1}(\boldsymbol{\mathcal{Z}}), \boldsymbol{\mathcal{S}}) = \tag{2.38}$$

$$\ln\left(\mathfrak{L}(P(Z_{\boldsymbol{\nu}_i^0 \sqcup \boldsymbol{\nu}_j^0}), \boldsymbol{\mathcal{S}}^{\boldsymbol{\nu}_i^0 \sqcup \boldsymbol{\nu}_j^0})\right) - \ln\left(\mathfrak{L}(P(Z_{\boldsymbol{\nu}_i^0}), \boldsymbol{\mathcal{S}}^{\boldsymbol{\nu}_i^0})\right) - \ln\left(\mathfrak{L}(P(Z_{\boldsymbol{\nu}_j^0}), \boldsymbol{\mathcal{S}}^{\boldsymbol{\nu}_j^0})\right) -$$

$$\delta\left(|\boldsymbol{\theta} \xleftarrow{fit} (\boldsymbol{\nu}_i \sqcup \boldsymbol{\nu}_j)| + |\boldsymbol{\theta} \xleftarrow{fit} \boldsymbol{\nu}_i| + |\boldsymbol{\theta} \xleftarrow{fit} \boldsymbol{\nu}_j|\right)$$

where $\delta = 1$ for the AIC metric whereas $\delta = \lambda \ln(|\boldsymbol{\mathcal{S}}|)$ for the BIC metric. The effect of the local change caused by a splice operation can thus efficiently be computed by regarding only the node vectors that were involved in the splice operation. After actually performing a splice operation, we only need to compute the effects of any splice operation with the newly formed factor.

Characterizing the algorithmics in pseudo–code

Pseudo–code for learning multivariate factorizations through the iterated greedy splicing of node vectors, is given in Figure 2.4. This pseudo–code differs from the high–level pseudo–code for the general greedy model learning algorithm in that the candidate models do not need to be fully stored separately. Instead, since we know that the likelihood penalization metrics decompose over the factors in the multivariate factorizations, we can compute the parameters and the metrics for each factor separately and compute only the new parameters and metrics for the resulting new factor if a certain splice operation is performed. If a splice operation should be performed, we must first update the administration of what splice operations between which factors result in what metric improvements, since one factor will disappear after a splice operation. For a splice operation involving factors $\boldsymbol{\nu}_i$ and $\boldsymbol{\nu}_j$ $(i < j)$, both of these factors will disappear and the last factor will be moved to the j–th factor. The parameters for the new factor will have to be computed and the parameters and metric values for the splice operations involving the moved last factor will have to be moved in the administration. After this update has been performed, the actual splice operation can be applied to the node partition vector $\boldsymbol{\nu}$, after which the parameters and the metrics for the new factor can be computed.

2.6.4 Greedy Bayesian factorization selection

In this section we discuss the selection of Bayesian factorizations using the greedy model selection framework from Section 2.6.1 and the extended likelihood metrics from Section 2.6.2. We sequentially discuss the initialization phase, the construction of candidate factorizations and the selection of a new factorization to replace the current one.

Initialization

Similar to the case of multivariate factorizations, the least complex probabilistic model structure in the case of Bayesian factorizations is again the univariate factorization. We therefore construct the initial structure by setting $\boldsymbol{\pi}_i = (i)$.

Constructing candidate factorizations

To construct candidate models, we allow addition of a single arc to the factorization graph of the current Bayesian factorization graph such that no cycles are introduced.

```
LearnMultivariateFactorization()
1  ν ← new vector of (vector of integer) with size l
2  for i ← 0 to l − 1 do
   2.1  ν_i ← (i)
3  EstimateParametersAndComputeMetricsForCurrent-
     AndCandidateFactors()
4  while |ν| > 1 do
   4.1  (i, j) ← FindNodeVectorsForSplice(ν)
   4.2  if i < 0 then
          4.2.1  breakwhile
   4.3  UpdateAdministrationOfParameterEstimatesAnd-
          MetricsForCurrentCandidateFactorsAfterSplice(i, j, ν)
   4.4  for k ← 0 to |ν_j| do
          4.4.1  ν_i ← ν_i ⊔ ((ν_j)_k)
   4.5  ν_j ← ν_{|ν|−1}
   4.6  |ν| ← |ν| − 1
   4.7  EstimateParametersAndComputeMetricsForNew-
          CandidateFactorsAfterSplice(i, j, ν)
5  return(ν)
```

```
FindNodeVectorsForSplice( ν )
1  δ_max ← 0
2  i_max ← −1
3  j_max ← −1
4  for i ← 0 to |ν| − 1 do
   4.1  for j ← i + 1 to |ν| − 1 do
          4.1.1  δ ← ComputeMetricImprovementForSplice(i, j, ν)
          4.1.2  if δ > δ_max then
                   4.1.2.1  δ_max ← δ
                   4.1.2.2  i_max ← i
                   4.1.2.3  j_max ← j
5  return((i_max, j_max))
```

Figure 2.4: Pseudo–code for the greedy incremental learning of multivariate factorizations using splice operations. The splice operation that increases the metric the most, is actually performed. If no splice operation further improves the metric, the greedy incremental learning algorithm terminates.

Choosing the best candidate factorization

From the results obtained for multivariate factorizations, we expect that the effect of adding a single arc on the penalization metric can be computed locally and separately for the negative log–likelihood term and the penalization term. This is indeed known to be the case (Dawid and Lauritzen, 1993; Buntine, 1994; Heckerman and Geiger, 1995).

Negative log–likelihood term
Assume that we add an arc $i \rightarrow j$ to the factorization graph. Using the definition of negative log–likelihood and Bayesian factorizations, it becomes clear that all factors other than the one in which j is conditioned on parent variables, cancel out in the difference.

Complexity term
Furthermore, the number of parameters $|\boldsymbol{\theta} \xleftarrow{fit} \boldsymbol{\pi}|$ for a Bayesian factorization can be computed in a similar fashion as for multivariate factorizations. Writing $(i|\boldsymbol{\pi}_i)$ for a structure that indicates a conditional gpdf $P(Z_i|Z_{\boldsymbol{\pi}_i})$, we have that $|\boldsymbol{\theta} \xleftarrow{fit} \boldsymbol{\pi}| = \sum_{i=0}^{l-1} |\boldsymbol{\theta} \xleftarrow{fit} (i|\boldsymbol{\pi}_i)|$.

Penalization metric difference after a splice operation
Using this fact, we arrive at the following final result for the decrease in the AIC and BIC metrics as a result of the addition of a single arc to the Bayesian factorization graph, if we again write $\delta = 1$ for the AIC metric and $\delta = \lambda \ln(|\boldsymbol{\mathcal{S}}|)$ for the BIC metric:

$$\mathfrak{M}(P_{\boldsymbol{\pi}^0}(\boldsymbol{\mathcal{Z}}), \boldsymbol{\mathcal{S}}) - \mathfrak{M}(P_{\boldsymbol{\pi}^1}(\boldsymbol{\mathcal{Z}}), \boldsymbol{\mathcal{S}}) = \tag{2.39}$$

$$\ln(\mathfrak{L}(P(Z_j|Z_{\boldsymbol{\pi}_j^1}), \boldsymbol{\mathcal{S}}^{(j)\sqcup\boldsymbol{\pi}_j^1})) - \ln(\mathfrak{L}(P(Z_j|Z_{\boldsymbol{\pi}_j^0}), \boldsymbol{\mathcal{S}}^{(j)\sqcup\boldsymbol{\pi}_j^0})) -$$

$$\delta\left(|\boldsymbol{\theta} \xleftarrow{fit} (j|\boldsymbol{\pi}_j^1)| + |\boldsymbol{\theta} \xleftarrow{fit} (j|\boldsymbol{\pi}_j^0)|\right)$$

Again, the effect of a structure change as a result of the addition of a single arc to the Bayesian factorization graph, can be computed locally. Therefore, the effect of the addition of any arc $k \rightarrow q$ does not need to be recomputed when an arc $i \rightarrow j$ is added as long as $q \neq j$. The greedy model selection algorithm can thus be transformed to start out with the univariate factorization and compute for each arc the effect of its addition on the penalization metric. In each iteration, the arc that leads to the largest decrease in the penalization metric is then chosen. If an arc $i \rightarrow j$ is actually added, we recompute the effects caused by any arc $k \rightarrow j$ that hasn't been added yet and doesn't introduce a cycle into the Bayesian factorization graph. This makes the greedy search algorithm with arc additions more efficient since not all candidate models need to be reevaluated.

Characterizing the algorithmics in pseudo–code

Pseudo–code for the greedy incremental learning of Bayesian factorizations based on arc additions is given in Figure 2.5. Just as is the case for the multivariate factorization, we need only to focus on the change in the factors. However, the difference for Bayesian factorizations is that no factors disappear. Therefore, no administration update is required once an arc has been found to be added to the Bayesian factorization graph. To ensure that no cycles are introduced in the Bayesian factorization graph, an array of size $l \times l$ is maintained that indicates for each arc whether it may still be added to the graph without introducing cycles. This information is updated if an arc is actually added to the graph. To be able to do so, we need to keep track of the direct predecessors and successors of all nodes in the Bayesian factorization graph. If an arc $i \rightarrow j$ is added to the graph, the parameters and metrics for the factor in which Z_j is conditioned on its parents, need to be recomputed at the end of one iteration in the greedy incremental learning algorithm.

2.7 Mixture probability distributions

In this section, we discuss the class of *mixture* probability distributions. A mixture probability distribution is a weighted sum of $k > 1$ probability distributions. Each probability distribution in the mixture probability distribution is called a *mixture component*. Let $\boldsymbol{\mathcal{K}} = (0, 1, \ldots, k-1)$. The probabilistic model associated with a mixture probability distribution is a vector $\boldsymbol{\varsigma}$ of simpler probabilistic model structures and a vector $\boldsymbol{\theta}$ of vectors of parameters:

$$P_{(\boldsymbol{\varsigma},\boldsymbol{\theta})}(\boldsymbol{\mathcal{Z}}) = \sum_{i=0}^{k-1} \beta_i P_{(\boldsymbol{\varsigma}_i,\boldsymbol{\theta}_i)}(\boldsymbol{\mathcal{Z}}) \tag{2.40}$$

such that $\forall i \in \boldsymbol{\mathcal{K}} : \beta_i \geq 0$ and $\sum_{i=0}^{k-1} \beta_i = 1$. The β_i with which the mixture components are weighted in the sum are called *mixing coefficients*. They are also part of the model parameters, $|\boldsymbol{\theta}| = k + 1$, $\boldsymbol{\theta}_k = (\beta_0, \beta_1, \ldots, \beta_{k-1})$. Using mixture probability distributions, a larger class of relations between the random variables can be expressed than when using factorized probability distributions. If dependencies between random variables are different for various subvectors of the solutions for which the probability distribution must be estimated, each of these dependency relations can be accounted for by an individual mixture component. Capturing these different (simpler) dependency relations can be done by allowing each mixture component to be a factorization. By adding the k factorizations into the mixture probability distribution, the more complex dependency relation is modelled. Mixture probability distributions are expressionally powerful, yet computationally tractable probability distributions that can capture complicated non–linear dependencies.

We shall now describe two approaches to estimating mixture probability distributions from data. The first approach is by means of clustering, whereas the second approach is by means of the expectation maximization algorithm.

```
LearnBayesianFactorization()
1  allowed ← new array of boolean in 2 dimensions with size l × l
2  v^pred, v^succ ← 2 new arrays of (vector of integer) with size l
3  for i ← 0 to l − 1 do
   3.1  for j ← 0 to l − 1 do
        3.1.1  allowed[i, j] ← true
   3.2  allowed[i, i] ← false
   3.3  v^pred ← ()
   3.4  v^succ ← ()
4  EstimateParametersAndComputeMetricsForCurrent-
     AndCandidateFactors()
5  γ ← l^2 − l
6  while γ > 0 do
   6.1  (i, j) ← FindArcToAdd(allowed, v^pred, v^succ)
   6.2  if i < 0 then
        6.2.1  breakwhile
   6.3  γ ← γ − AddArcToAcyclicGraph(l, i, j, allowed, v^pred, v^succ)
   6.4  EstimateParametersAndComputeMetricsForNew-
          CandidateFactorsAfterArcAdd(i, j, v^pred)
7  return(v^pred)
```

```
FindArcToAdd( allowed[·, ·], v^pred[·], v^succ[·] )
1  δ_max ← 0
2  i_max ← −1
3  j_max ← −1
4  for i ← 0 to l − 1 do
   4.1  for j ← 0 to l − 1 do
        4.1.1  if allowed[i, j] then
                 4.1.1.1  δ ← ComputeMetricImprovementForArc-
                                Add(i, j, v^pred)
                 4.1.1.2  if δ > δ_max then
                              4.1.1.2.1  δ_max ← δ
                              4.1.1.2.1  i_max ← i
                              4.1.1.2.1  j_max ← j
5  return((i_max, j_max))
```

Figure 2.5: Pseudo–code for the greedy incremental learning of Bayesian factorizations using arc addition operations. The arc addition operation that increases the metric the most, is actually performed. If no arc addition operation further improves the metric, the greedy incremental learning algorithm terminates.

2.7.1 By means of clustering

Clusters are possibly overlapping subvectors of the original sample vector such that each sample point in the original sample vectors occur in at least one cluster. If we enforce that the subvectors are mutually disjoint, we speak of *partitions* rather than clusters. The use of clusters allows us to efficiently break up non–linear interactions. Thus, by estimating simpler probability distributions, such as factorizations, for each cluster separately, we can add these probability distributions into a mixture probability distribution to get an adequate representation of the complete sample vector. After clustering and the estimation of the simpler probability distributions for each cluster, we still have to choose the mixing coefficients β_i. This can be done in various ways. One of the common approaches if clusters are used is to set β_i to the proportion of the size of the i–th cluster with respect to the sum of the sizes of all clusters. Exact algorithms for partitioning exist (Hartigan, 1975), but the running times for these algorithms are of no practical use in our case. What we require, is a fast approximate assessment of clusters such that we can estimate a relatively simple probability distribution in each cluster in a good way. Computationally efficient clustering algorithms exist that provide useful results (Hartigan, 1975). In the following we first discuss distance metrics for clustering. Subsequently, we present two clustering algorithms. Afterwards, we discuss how to set the parameters for the clustering algorithms and present a few example results of the clustering algorithms on two–dimensional data.

Distances

One of the most fundamental aspects in clustering, is the distance function. The distance of a point to a cluster determines to which cluster the point will actually be assigned. Often, the *Euclidean* distance is used. However, for two variables with different scales, the Euclidean distance might not be the best choice. To treat each dimension as being equally important, the Euclidean distance must be used *after* the sample data has been rescaled in every dimension. Rescaling is however sensitive to outliers. For two l dimensional real–valued points $\boldsymbol{y}^i$ and $\boldsymbol{y}^j$ the Euclidean distance $d_E(\boldsymbol{y}^i, \boldsymbol{y}^j)$ between them is given by:

$$d_E(\boldsymbol{y}^i, \boldsymbol{y}^j) = \sqrt{(\boldsymbol{y}^i - \boldsymbol{y}^j)^T(\boldsymbol{y}^i - \boldsymbol{y}^j)} = \sqrt{\sum_{k=0}^{l-1}(\boldsymbol{y}_k^i - \boldsymbol{y}_k^j)^2} \tag{2.41}$$

A straightforward way to introduce scaling into the Euclidean distance, is obtained by dividing the distance in each dimension by the range of the samples in that dimension:

$$d_{ES}(\boldsymbol{y}^i, \boldsymbol{y}^j) = \sqrt{(\boldsymbol{y}^i - \boldsymbol{y}^j)^T \boldsymbol{R}^{-1}(\boldsymbol{y}^i - \boldsymbol{y}^j)} = \sqrt{\sum_{k=0}^{l-1}\frac{(\boldsymbol{y}_k^i - \boldsymbol{y}_k^j)^2}{r_k^2}} \tag{2.42}$$

$$\text{where } \boldsymbol{R} = \begin{bmatrix} r_0^2 & 0 & \cdots & 0 \\ 0 & r_1^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & r_{l-1}^2 \end{bmatrix} \text{ and } r_i = \max_j\{y_i^j\} - \min_j\{y_i^j\}$$

Given a cluster, we can use its centroid as a reference point. The distance between a point and the cluster is then the scaled Euclidean distance between the cluster centroid and the point. A drawback of this approach is that the distance to a cluster has ellipsoid shaped isolines that are aligned with the axes. Therefore, samples with linear interaction through correlation are likely to be broken up into multiple clusters. Alternatively, the *Mahalanobis* distance (Mahalanobis, 1930) can be used. The Mahalanobis distance can be seen as the result of fitting a normal pdf with a full covariance matrix. The clusters can therefore potentially be shaped better to the form of the data. Let $\boldsymbol{\mathcal{C}}$ be the cluster vector, $\boldsymbol{\mathcal{C}} = (\boldsymbol{c}^0, \boldsymbol{c}^1, \ldots, \boldsymbol{c}^{|\boldsymbol{\mathcal{C}}|-1})$, such that each $\boldsymbol{c}^i = (\boldsymbol{c}_0^i, \boldsymbol{c}_1^i, \ldots, \boldsymbol{c}_{|\boldsymbol{c}^i|-1}^i)$ is the vector of the indices of the samples in the sample vector $\boldsymbol{\mathcal{S}}$ that belong to the i–th cluster. Let $\overline{\boldsymbol{c}^i}$ be the centroid of the i–th cluster and let $\widetilde{\boldsymbol{S}^i}$ be the sample covariance matrix of the points in cluster i:

$$\overline{\boldsymbol{c}^i} = \frac{1}{|\boldsymbol{c}^i|} \sum_{j=0}^{|\boldsymbol{c}^i|-1} \boldsymbol{y}^{\boldsymbol{c}_j^i}, \qquad \widetilde{\boldsymbol{S}^i} = \frac{1}{|\boldsymbol{c}^i|} \sum_{j=0}^{|\boldsymbol{c}^i|-1} (\boldsymbol{y}^{\boldsymbol{c}_j^i} - \overline{\boldsymbol{c}^i})(\boldsymbol{y}^{\boldsymbol{c}_j^i} - \overline{\boldsymbol{c}^i})^T \tag{2.43}$$

The Mahalanobis distance $d_M(y^i, j)$ between a point $\boldsymbol{y}^i$ and the j–th cluster can now be written as:

$$d_M(\boldsymbol{y}^i, j) = \sqrt{(\boldsymbol{y}^i - \overline{\boldsymbol{c}^i})^T (\widetilde{\boldsymbol{S}^i})^{-1} (\boldsymbol{y}^i - \overline{\boldsymbol{c}^i})} \tag{2.44}$$

The disadvantage of the Mahalanobis distance is that the covariance matrices have to be determined, which becomes less reliable if the number of samples in a cluster becomes smaller. Furthermore, the memory and computation time requirements are $\mathcal{O}(l^2)$ instead of $\mathcal{O}(l)$.

Clustering algorithms

In this section, we shortly present two partitioning algorithms. To reduce additional computational requirements involved in estimating a mixture probability distribution, we want the partitioning algorithm to be fast and reliable. The first partitioning algorithm we discuss, which is the leader algorithm, is faster, whereas the second partitioning algorithm we discuss, which is the k–means algorithm, is more reliable.

The leader algorithm

The *leader* algorithm is one of the fastest partitioning algorithms. The use of it can thus be beneficial if the amount of overhead that is introduced by factorization mixture selection methods is desired to remain small. There is no need

to specify in advance how many partitions there should be. The first sample to make a new partition is appointed to be its leader. The leader algorithm goes over the sample vector exactly once. For each sample it encounters, it finds the first partition that has a leader being closer to the sample than a given threshold $\mathfrak{T}_d$. If no such partition can be found, a new partition is created containing only this single sample. To prevent the first partitions from becoming quite a lot larger than the later ones, we randomize the order in which the partitions are inspected. The asymptotic running time for finding the first partition with a leader closer than $\mathfrak{T}_d$ is the same as going over all partitions and finding the closest partition. Therefore, we prefer to find the closest partition.

One of the drawbacks of the (randomized) leader algorithm is that it is not invariant given the sequence of the input samples. Most partitioning algorithms do not have this property, but not as strongly as the leader algorithm. Therefore, to be sure that the ordering of the sample vector is not subject to large repeating sequences of samples, we propose to randomize the ordering of the samples as input to the leader algorithm as well.

To allow for more flexible partitions, we may want to use the Mahalanobis distance. This can however not be achieved by simply substituting the Mahalanobis distance for the Euclidean distance in the above algorithm because we need to know the sample covariance matrix of the samples that belong to a partition as well as their sample mean. To overcome this problem, we can determine the required parameters as the partitioning algorithm progresses through the samples. As a point is added to a partition, its sample covariance matrix and its sample mean are updated. However, the covariance matrix can only be computed and used in the Mahalanobis distance if we have at least two samples. Furthermore, based on only a few samples, the Mahalanobis distance may give unstable results, leading to unnatural partitions. Therefore, we propose that a partition must first grow to some minimum size $\mathfrak{T}_s$ before the Mahalanobis distance can be used for it.

The leader aspect in the Euclidean variant is given by the fact that the first sample to define a new partition is the designated leader of the partition, serving as a centroid for the Euclidean distance. In the variant that results when we use the Mahalanobis distance, the sample mean takes the role of the leader for some partition as soon as the size of that partition has reached $\mathfrak{T}_s$. The sample mean of a partition changes whenever a sample is added to that partition. The term leader is thus somewhat misplaced in the resulting algorithm.

The k–means algorithm

The *k–means* algorithm constructs exactly k partitions. Different from the leader algorithm, the k–means algorithm uses the partition centroids. First, k partitions are created at random. This can be done by partitioning the sample vector at random into k subvectors. The resulting centroids are however expected to lie close to each other. Therefore, the initial partitions are usually taken to consist of a single sample, which is chosen at random from the sample vector. Subsequently, the algorithm iterates until the means do not change to within a significance of ε anymore. In every iteration, each point is re–assigned

to the nearest partition based on the distance to the partition centroid. Ties between equally close partitions are broken randomly. Once all of the points have been assigned, the means of the partitions are recomputed.

An advantage over the leader algorithm is that the result of the k–means algorithm is less dependent on the input ordering. A disadvantage is that the algorithm takes N_I times as long as the leader algorithm where N_I is the number of iterations in the k–means partitioning algorithm. To avoid spending large computation efforts, a maximum of $\mathfrak{T}_i$ iterations is allowed.

Just as for the leader algorithm, we can alter the k–means clustering algorithm to make use of the Mahalanobis distance. If in the first iteration, a cluster has grown to some minimum size, the Mahalanobis distance may be used. Otherwise, the Euclidean distance is used. In subsequent iterations, the Mahalanobis distance is always used as the covariance values are then kept fixed in any single loop.

Setting parameters

An important question at this point is how to set the algorithmic parameters such that a desirable partitioning is obtained. Setting the parameters requires some experience. To automate this, we can apply the general approaches for model selection discussed in Section 2.6. For the k parameter in the k–means algorithm, we can increment k and compute the negative log–likelihood of the mixture probability distribution after estimating a factorized probability distribution in each cluster. If the resulting mixture probability distribution is significantly better than for a smaller value of k, this value is accepted and the search continues. A similar scheme can be used for the distance threshold $\mathfrak{T}_d$.

It should be noted however that using such methods requires a vast amount of computation time. Factorization selection for a single cluster can already take up a significant amount of computation time. Over k clusters, we thus require k times as much computation time. It therefore seems by far the most efficient choice to choose k or $\mathfrak{T}_d$ by experience in such a way that the expected number of samples in each cluster is large enough to reliably estimate a factorized probability distribution for it.

Examples

In this section, we present some examples of the leader and k–means partitioning algorithms that are described in the previous sections. We use sample vectors that can be displayed graphically to gain better insights into how the algorithms work. The results using both the scaled Euclidean as well as the Mahalanobis distance on 2 sample vectors are shown in Figure 2.6. The first sample vector is sampled from the uniform distribution. The second sample vector is sampled from uniform distributions over separated subregions.

The distance threshold for the scaled Euclidean leader algorithm was $\mathfrak{T}_d = 1.0$. For the Mahalanobis leader clustering algorithm, the distance threshold was $\mathfrak{T}_d = 2.5$ and the minimum cluster size threshold was $\mathfrak{T}_s = 10$. For both k–

means clustering algorithms, we used $k = 10, \varepsilon = 0.001, \mathfrak{T}_i = 10$. These settings were chosen such that a visually intuitive number of clusters was obtained.

The cluster boundaries of the results obtained with the k–means algorithm are strict in the sense that there is no overlap between the different clusters. The randomized approach to the leader algorithm proves to be effective in the sense that there are no extremely large clusters in any of its results and that the boundaries are not disturbingly overlapping. This means that the resulting densities that are estimated using either of the clustering results, will be quite similar. Especially if a good approximation is more important than the best possible density estimation, the results of the leader algorithm seem at least as promising as those of the k–means clustering algorithm. Since the running time of the leader algorithm is smaller than that of the k–means algorithm, the leader algorithm is preferable.

Comparing the results obtained with the Euclidean distance with the results obtained with the Mahalanobis distance, it is not directly clear which of these results is more desirable. One of the main drawbacks of using the Mahalanobis distance, is that small clusters are more likely to appear. Such clusters can for instance lie completely inside another cluster since the normal pdf that is based upon it, is sharply peaked. The k–means algorithm in this case leads to better formed clusters with less overlap. The true advantage of the Mahalanobis distance in that it can better describe linear relations, is not exploited in a useful way using the described clustering algorithms. As the Mahalanobis distance takes more computational effort, the Euclidean distance is preferable.

2.7.2 By means of expectation maximization

A different approach to estimating a mixture probability distribution from data is to compute a maximum likelihood estimation. On the one hand this seems a plausible approach since we do the same for a single normal pdf for instance. On the other hand, if we allow just as many mixture components as we have sample points, a maximum likelihood estimation will have a mixture component centered at each sample point with an infinite density at that point. From the latter point of view, a maximum likelihood estimation mixture probability distribution is far from desirable as it completely lacks any potential for generalization. Thus, as long as the number of mixture components is limited, a maximum likelihood estimation is desirable.

The Expectation Maximization algorithm

The *Expectation Maximization* (EM) algorithm (Dempster, Laird and Rubin, 1977) is a general iterative approach to computing a maximum likelihood estimate. The EM algorithm is required because a maximum likelihood estimation cannot always be analytically determined. In the EM algorithm, the difference in the negative log–likelihood of the estimated probability distribution between subsequent iterations is used to derive the parameters. The negative log–likelihood is thus seen as an error that we wish to minimize. Unfortunately,

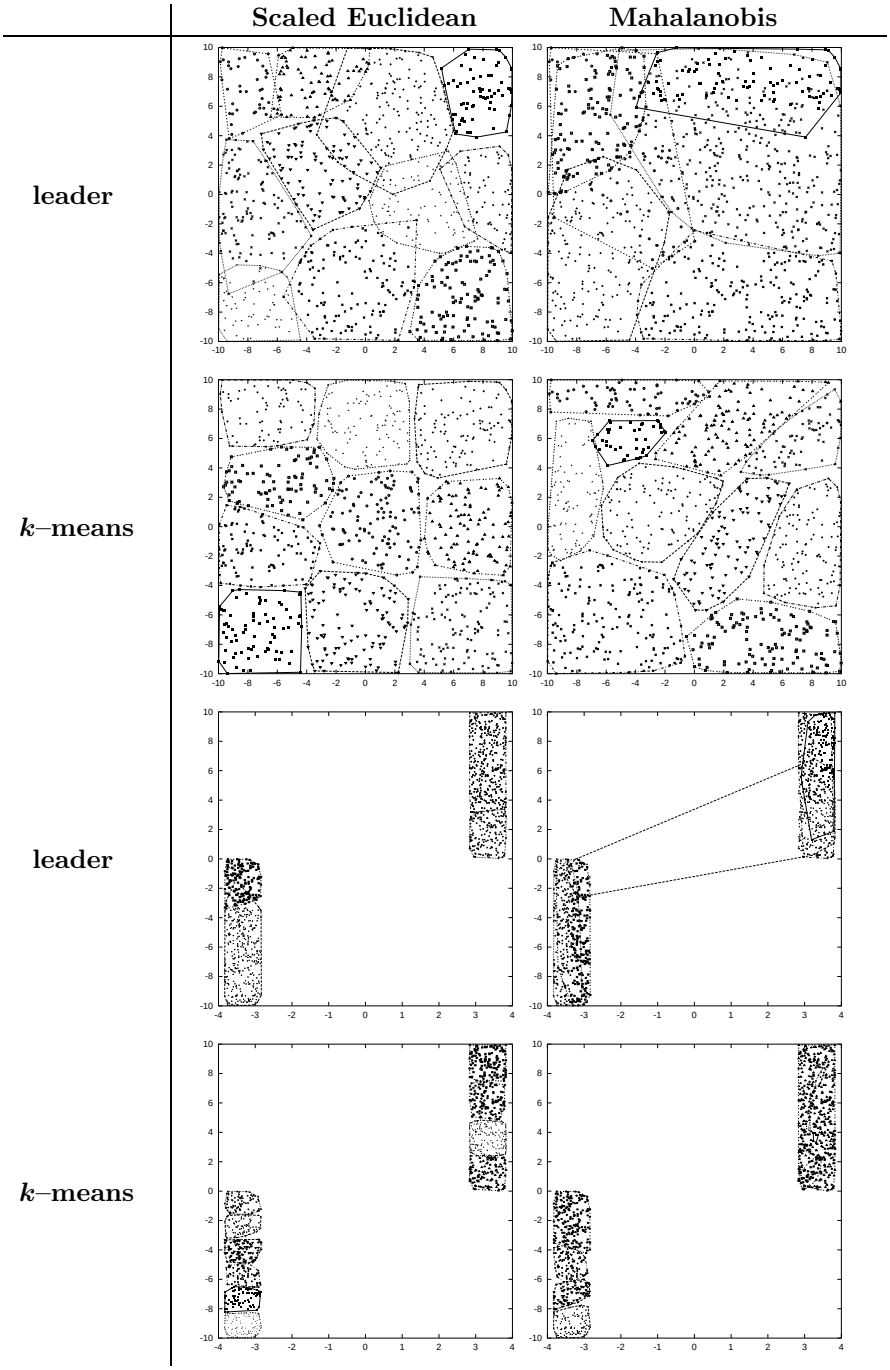


Figure 2.6: Clustering example results on two sample sets using the randomized leader clustering algorithm and the k–means clustering algorithm for both the scaled Euclidean distance and the Mahalanobis distance.

a straightforward application of the EM algorithm is highly likely to result in a non–maximum likelihood estimation. The reason for this is twofold. First, the minimization problem that the EM algorithm attempts to solve iteratively contains many local minima. Second, the EM algorithm is essentially a single–point gradient–descent type of algorithm, causing it to easily get stuck in a local minimum.

The Expectation Maximization algorithm for gpdfs

Mixture probability distributions can also be seen as gpdfs that can be used in a factorization. Moreover, these gpdfs can be estimated using the EM algorithm. If the univariate factorization is used, the number of parameters to be estimated remains small since the mixture probability distribution is estimated in a single dimension only. This makes the result that is obtained by applying the EM algorithm more reliable. Note that this *does* imply that if for instance the normal pdf is used in the mixture, linear correlations still cannot be modelled effectively using the univariate factorization.

Choosing the number of components

It should be noted that even though we get an approximation of the maximum likelihood estimation of a mixture probability distribution by using the EM algorithm, the number of components in the mixture probability distribution still has to be chosen. This choice is similar to the choice of the number of partitions when using a clustering approach to the estimation of a mixture probability distribution from data. Again, the value of k could be increased incrementally to observe the increase of the likelihood and the significance thereof, but as this should be done in addition to applying the EM algorithm, the amount of computational resources will increase rapidly. For this reason, k is often chosen by experience.

'Ideas move fast when their time comes.'
Carolyn Heilbrun

Evolutionary algorithms

In this chapter we focus on evolutionary algorithms (EAs). EAs mimic a few basic concepts in natural evolution to perform optimization. The two most important concepts are genetic inheritance and survival of the fittest. After presenting a general description of EAs and an outline of the main streams of EAs, we investigate the genetic algorithm (GA), a specific type of EA, in more detail. Since the introduction of the GA by Holland (1975), many variants of it have been introduced. The original GA by Holland (1975) has come to generally be known as the *simple GA*. It is this GA that we investigate in more detail.

The chapter is organized as follows. In Section 3.1 a general introduction to EAs is provided along with a general algorithmic framework for EAs. Subsequently, in Section 3.2 we give a detailed description of the simple GA by instantiating the general EA framework. We also discuss different types of analysis from which it becomes clear why and on what type of problem the simple GA performs well and on what type of problem it fails to perform useful induction. Based on these results, the simple GA can be enhanced to cope with its deficiencies by introducing more advanced tools to perform induction. Such enhanced GAs have been proposed by various researchers since the late 1980s. Some of these improved GAs are briefly discussed in Section 3.3. A special variant of these enhanced GAs, in which probability distributions are used as a tool for induction, is the topic of the remaining chapters of this thesis. The Chapter is concluded in Section 3.4 with a brief discussion on the use of induction in EAs in general.

3.1 Evolutionary algorithms

In *Evolutionary Computation* (EC), natural evolution is used as a metaphor for constructing optimization algorithms. These *Evolutionary Algorithms* (EAs) mimic abstractions of key features in natural evolution that are assumed to be responsible for the creation of the complex life forms known today. EAs are general–purpose non–linear optimization techniques. They are relatively easy to parallelize and to combine with existing optimization techniques and heuristics. Advantages such as these make EAs into attractive optimization algorithms.

In this section, we first describe how natural evolution can be seen as an optimization process in Section 3.1.1. In Section 3.1.2, we subsequently describe how abstractions of this process are used in EAs. Next, in Section 3.1.3 we present a general description of EAs by means of an algorithmic framework and outline the the main streams of algorithms in EC known today.

3.1.1 Natural evolution as an optimization process

Through natural evolution, species are able to adapt to their environment as generations pass. As a result of mating, new individuals are introduced that inherit the genetic material of their parents. This inherited material can differ from the original material as a result of mutations. The strongest individuals have the greatest chance of surviving and therefore of mating to generate offspring. This understanding of natural evolution was first published by Darwin (1859). He wrote about structures in nature that are copied over generations. He argued that the copied structures may vary partially from the original. Darwin furthermore stated that the structures are competing for a limited resource and that their relative reproductive success depends on their environment.

Natural evolution can be seen as an efficient and parallel optimization process based on natural selection and genetics. The goal is to generate individuals that are able to survive and thrive as good as possible in a certain environment. This is accomplished by allowing the best individuals to survive and mate, thereby increasing the chance of generating new individuals that are able to survive and thrive in their environment as well.

Each organism in nature is encoded by its DNA. DNA is basically a string or sequence of information that describes everything there physically is to know about the organism it encodes. The evolutionary process works using this sequence instead of the actual physical form. The encoding of the organism is called the *genotype*, whereas the physical form of the organism is called the *phenotype*. The phenotype determines how well the organism actually behaves in its environment, which we call the *fitness* of the organism. The inheritance of genetic material through mating is a result of combining the parent genotypes. The latter process is called *recombination*. Small additional changes in the offspring genotype are a result of the process of *mutation*. The survival of the fittest individuals in nature is called (natural) *selection*.

The optimization characteristic of natural evolution is witnessed by an increase of the (average) fitness of the individuals over the generations as a result

of selection, recombination and mutation. Although it has taken nature millions of years to evolve into the life forms that we know today, it is still an efficient process if we consider the vast complexity of these life forms and the variety in species. In natural selection and the recombination of genetic material implicitly lies the use of induction. Induction enables the determination of which parts of the genetic code result in organism characteristics that are beneficial in their environment. Those parts that do not result in beneficial characteristics, are gradually replaced with other genetic material.

3.1.2 Natural evolution as a metaphor for optimization

In our definition of optimization problems, the phenotype is a solution in $\mathbb{P}$. The objective space and the optimization function are given by the fitness of the organism. Optimization is led towards the individuals with the best fitness. What cannot be placed in our earlier definition of an optimization problem, is the genotype. The genotype is an encoding of the parameter space that is useful for a certain optimization process or algorithm. The algorithm can conveniently work with the encoding and translate this encoding into the phenotype when needed to compute the optimization function. In fact, the function that is actually being optimized, is given by $\mathfrak{G}(\textit{genotype}) = \mathfrak{F}(\mathfrak{D}(\textit{genotype}))$. We call $\mathfrak{G}$ the *fitness function*, and $\mathfrak{D}$ the *genotype decoding function*. With this definition, function $\mathfrak{G}$ can be seen as the actual optimization function. The space of all possible genotypes, called the *genotype space*, is denoted by $\mathbb{G}$. We could thus see the parameter space $\mathbb{P}$ as the set $\mathbb{G}$. This would eliminate the concept of decoding and the mapping between genotypes and phenotypes. However, decoding a genotype in $\mathbb{G}$ to get a solution in $\mathbb{P}$ can often be established in many ways. Moreover, there are many interesting details to the construction of such a mapping from genotypes to phenotypes (Kargupta, 2000). To reflect this fact, we respect the mapping between genotypes and phenotypes, but note that a thorough understanding of its details is outside the scope of this thesis.

In most cases, and at least in the cases treated in this thesis, a genotype is a vector of l symbols, $l \geq 1$. Using biological terms, each such symbol is called an *allele*. The location of the symbol in the genotype is called the *locus* of the allele. The i–th locus itself, which can be assigned different alleles, is called a *gene*. Each gene can be seen as a variable that can assume certain values. These values are represented by the alleles. When we disregard the domain type of the gene variables, we denote the gene variable at locus i by $\mathsf{Z}_i, i \in \boldsymbol{\mathcal{L}} = (0, 1, \ldots, l-1)$. We denote a genotype by $\mathbf{Z} = \mathsf{Z}_{\boldsymbol{\mathcal{L}}} = (\mathsf{Z}_0, \mathsf{Z}_1, \ldots, \mathsf{Z}_{l-1})$. The set of possible alleles at locus i is denoted by $\mathbb{A}_i$. The fitness function can now be written as a function of the gene variables, $\mathfrak{G}(\mathsf{Z}_0, \mathsf{Z}_1, \ldots, \mathsf{Z}_{l-1}) = \mathfrak{G}(\mathbf{Z}) = \mathfrak{F}(\mathfrak{D}(\mathbf{Z}))$. The genotype space is often given by $\mathbb{G} = \times_{i=0}^{l-1} \mathbb{A}_i$ (i.e. the Cartesian case). In this thesis, the genotype decoding function will often simply be the identity function, e.g. $\mathfrak{D}(\mathbf{Z}) = \mathbf{Z} = \zeta_{\boldsymbol{\mathcal{D}}}$. In this case, $\mathfrak{F} = \mathfrak{G}$ and either function can straightforwardly be written as a function of the gene variables Z_i. If $\mathfrak{D}$ is not the identity function (chapter 6), we will explicitly describe both $\mathfrak{F}$ and $\mathfrak{G}$ in terms of the problem variables ζ_i and the gene variables Z_i, respectively.

3.1.3 A general algorithmic framework for EAs

To give a general algorithmic framework for EAs, we focus on the algorithms instead of on the optimization problem. This clarifies what EAs are and how they can be composed. The framework that we present is capable of describing many variants of EAs and gives a good impression of the structure of EAs. After presenting the framework, we outline the main streams of EAs.

EA framework

At any moment in the execution of an EA, a vector $\boldsymbol{\mathcal{P}}$ of $n = |\boldsymbol{\mathcal{P}}| \geq 1$ genotypes (also called individuals or solutions) exists, called the *population*. At first, the population is initialized. This often means that n genotypes are generated at random, unless some prior information on the optimization problem is available. Sometimes, a fast local search algorithm is applied in the initialization phase. Subsequently, the individuals are evaluated so as to obtain their fitness values. If local search is used in the initialization procedure, the genotypes that have undergone local search, will already have been evaluated. Therefore, only the genotypes that haven't been evaluated yet must still be evaluated. The EA then evolves the population iteratively until a certain *termination criterion* is met. Each iteration in an EA is called a *generation*. The termination criterion is checked at the beginning of each generation. The termination criterion is often a maximum number of times that the fitness function may be evaluated. Another example of a frequently used termination criterion is the loss of diversity (different genotypes) in the population.

In each generation t, first a vector of selected solutions $\boldsymbol{\mathcal{S}}$ is made from the population. These selected solutions are commonly referred to as the *parents*. Selection can be done in various ways, but it mostly involves selecting solutions with better fitness values. Solutions in $\boldsymbol{\mathcal{P}}$ may be selected to appear in $\boldsymbol{\mathcal{S}}$ more than once. The parents are subsequently mated into groups. Some candidates may be selected to participate in multiple groups whereas other candidates may not be selected at all. Each of these groups then undergoes recombination to generate offspring. Recombination is applied with a certain probability p_r. If no recombination should be performed, each offspring is an exact copy of some parent. After recombination, the offspring undergo mutation with a certain probability p_m, which is usually kept small. The mutation probability is often interpreted as a probability of randomly changing a certain part of a genotype.

In the replacement phase that follows recombination and mutation, the offspring are placed into the population. This may be done by simply adding them to the population, but it is also possible that the offspring replace genotypes that are currently in the population or that they must compete with their parents to win a position in the population. Note that in the first case, there is no direct need to evaluate the offspring genotypes, whereas in the latter case evaluation during the replacement phase is imperative.

Finally, another selection phase determines which genotypes will survive the current generation. This selection phase sometimes only involves selecting the *entire* population after the replacement phase, in which case no additional

```
EvolutionaryAlgorithm( Initialize(), SelectParents(),
                       Mate(), Recombine(), Mutate(),
                       Replace(), SelectSurvivors(),
                       Terminate(), 𝔊(), p_r, p_m )
1  t ← 0
2  P ← Initialize()
3  for i ← 0 to n − 1 do
   3.1  if ¬evaluated[P_i] then
        3.1.1  fitness[P_i] ← 𝔊(P_i)
4  while ¬Terminate(P) do
   4.1  S ← SelectParents(P)
   4.2  mat ← Mate(S)
   4.3  off ← ()
   4.4  for each m ∈ mat do
        4.4.1  rec ← Recombine(p_r, m)
        4.4.2  mut ← ()
        4.4.3  for each g ∈ rec do
               4.4.3.1  mut ← mut ⊔ (Mutate(p_m, g))
        4.4.4  off ← off ⊔ mut
   4.5  rep ← Replace(off, mat, S, P)
   4.6  P ← SelectSurvivors(rep)
   4.7  for i ← 0 to n − 1 do
        4.7.1  if ¬evaluated[P_i] then
               4.7.1.1  fitness[P_i] ← 𝔊(P_i)
   4.8  t ← t + 1
```

Figure 3.1: Pseudo–code for the general evolutionary algorithm.

evaluations are required. Otherwise, the offspring will have to be evaluated first.

After replacement and survivor selection, the EA has come to the end of a generation, at which time the genotypes that haven't been evaluated yet, are evaluated. The entire process, with the exception of the initialization phase, is iteratively repeated. Upon termination, a genotype with the best fitness is usually reported as the final solution that the EA has found. The application of a specialized (local) optimization algorithm can also be taken into account. Such a specialized optimization algorithm is often able to quickly improve a solution. The added use of such an algorithm makes an EA *hybrid*. Since the use of local search techniques does not add to the clarity and understanding of EAs themselves, we do not explicitly mention it in our pseudo–code overview of EAs in Figure 3.1. To actually construct an EA, we must define how to encode a solution in a genotype vector and how to select parents, mate, recombine, mutate, replace, select survivors and evaluate fitness. The EA subsequently performs optimization by manipulating the genotypes.

EA main streams

Different genotypes and genetic operators have been proposed to arrive at different implementations of EAs. Over the years, four main streams of EAs have emerged and are commonly used today. Although the boundaries between these four main streams tend to be fuzzy at times in current EA research, the algorithms from which they originated clearly differ. One of these variants is *Evolution Strategies* (ES). The first variants of ES were proposed by Schwefel (1965). Improvements over these first variants were presented by Rechenberg (1973). Eight years later, Schwefel (1981) proposed the strategies that are today known as the basic evolution strategies (Bäck and Schwefel, 1993). ES are especially well suited for numerical optimization. Since we will use them in benchmark tests in Chapter 5, more detail on ES is presented there. The field of *Evolutionary Programming* (EP) was initiated by Fogel, Owens and Walsh (1966). EP is also especially well suited for numerical optimization. Although EP was developed separately from ES, the state of the art in EP has over the years developed many similarities with ES (Fogel, 1992). The *Genetic Algorithm* (GA) was first proposed and studied by Holland (1975). The GA genotype is a vector of l variables, each of which can take values from possibly different domains. In the classic form of the GA, all allele domains are identical and equal to $\mathbb{B}$, making a genotype a binary vector ($\mathbb{G} = \times_{i=0}^{l-1}\mathbb{B}$). The GA with this representation is the most frequently used EA (Goldberg, 1989). More details on the specific GA known as the simple GA are given in the next section. The last main stream we mention is the field of *Genetic Programming* (GP). Genetic programming is used to evolve tree or graph structures and is often applied to the construction of mathematical formulas and logical circuits. However, the field of GP is best known for the automatic construction of computer programs (Koza, 1992).

3.2 The simple genetic algorithm

First, the simple GA is presented in Section 3.2.1. Subsequently, Sections 3.2.2 through 3.2.4 are devoted to the dynamics and the competence of simple GA.

3.2.1 Algorithmic description

We define the simple GA by instantiating the general framework for EAs presented in Section 3.1.3. The general notion of an EA is already clear from this framework, so we restrict ourselves to giving exact definitions of the genetic operators and encodings to use in the framework to construct the simple GA.

Genotype

The genotype in the simple GA is a vector of l binary variables, $\mathbb{G} = \times_{i=0}^{l-1}\mathbb{B}$. We denote a binary gene variable by X_i instead of Z_i. The following example serves to provide some intuition as to how optimization problems can be encoded with binary gene variables.

Example 3.1. Suppose that the parameter space is a subset of all d–dimensional vectors of real values, $\mathbb{P} = [lb, ub]^d$. In that case, the number of binary gene variables l is often chosen as a multiple of d, i.e. $\frac{l}{d} \in \mathbb{N}$. This allows us to break up the binary genotype vector in equally–sized binary subvectors, each of which encodes a problem variable. To decode a binary genotype, each binary subvector is first interpreted using the standard binary encoding of natural numbers and subsequently scaled to the desired range of $[lb, ub]$:

$$\zeta_i = \left(\sum_{j=0}^{\frac{l}{d}-1} 2^{\frac{l}{d}-1-j} \mathsf{X}_{\frac{il}{d}+j} \right) \frac{(ub - lb)}{2^{\frac{l}{d}} - 1} + lb \tag{3.1}$$

Initialization

In most cases, no prior information on the promising regions of $\mathbb{P}$ is assumed. Therefore, initialization is mostly performed by generating n binary vectors randomly.

Parent selection

Proportionate selection

The selection of parents that are eligible for mating, was originally proposed by Holland (1975) to be performed using *proportionate selection*. In proportionate selection, a probability of selection is associated with each population member. This probability is given by the ratio of the fitness value of the population member and the total summed fitness of all individuals in the population. If we denote the associated selection probability of individual $\boldsymbol{\mathcal{P}}_i$ by $\mathit{proportion}[\boldsymbol{\mathcal{P}}_i]$, and recall that in the general EA framework we have stored the fitness of individual $\boldsymbol{\mathcal{P}}_i$ in a structure field that is denoted by $\mathit{fitness}[\boldsymbol{\mathcal{P}}_i]$, we get:

$$\mathit{proportion}[\boldsymbol{\mathcal{P}}_i] = \frac{\mathit{fitness}[\boldsymbol{\mathcal{P}}_i]}{\sum_{j=0}^{n-1} \mathit{fitness}[\boldsymbol{\mathcal{P}}_j]} \tag{3.2}$$

For this approach, we must always ensure that the goal is to maximize the fitness and that no fitness value can become negative. Otherwise, the respective problems are that selection will exert pressure on the wrong individuals and that the $\mathit{proportion}[\boldsymbol{\mathcal{P}}_i]$ values are no probabilities.

With the associated probabilities, the actual selection is made. In the simple GA, n individuals are usually selected. Because a population member may be selected more than once, the expected number of copies for population member $\boldsymbol{\mathcal{P}}_i$ is $|\boldsymbol{\mathcal{S}}| \cdot \mathit{proportion}[\boldsymbol{\mathcal{P}}_i]$. The implementation of proportionate selection can be achieved by first making a cumulative distribution over the proportionate probabilities of the population members.

$$\mathit{cumulative\_proportion}[\boldsymbol{\mathcal{P}}_i] = \sum_{j=0}^{i} \mathit{proportion}[\boldsymbol{\mathcal{P}}_j] \tag{3.3}$$

Subsequently, real values are drawn randomly from $[0, 1]$. For each randomly drawn value r, the individual $\boldsymbol{\mathcal{P}}_i$ for which $\mathit{cumulative\_proportion}[\boldsymbol{\mathcal{P}}_{i-1}] < r \leq \mathit{cumulative\_proportion}[\boldsymbol{\mathcal{P}}_i]$ holds, is selected. The exception being the individual at position 0, which is selected if and only if $\mathit{cumulative\_proportion}[\boldsymbol{\mathcal{P}}_0] \geq r$.

Next to the overhead of rescaling the fitness values in the case of minimization or when the fitness values can become negative, proportionate selection has some additional disadvantages which have rendered it obsolete in current EAs. The most important disadvantage is that individuals with a fitness far above average take over the population very quickly as they are selected many times. Another problem is that once an EA starts to converge and the fitness values of individuals become similar, there is no more *selection pressure* that ensures that the search continuously moves towards the best solutions. This prohibits efficient convergence to a final solution. For these reasons, tournament selection and truncation selection are used more often. Both these selection operators are able to preserve selection pressure since they do not perform selection based on absolute fitness values, but based on their relative ordering.

Tournament selection
Tournament selection selects individuals by first randomly picking s individuals from the population, $s \geq 1$. From these s individuals, the best individual is actually selected. We call s the *tournament size*, which is in most cases set to 2 or 4. Note that the selection pressure increases as s is increased.

Truncation selection
In *truncation selection*, $\lfloor \tau n \rfloor$ individuals are selected, $\tau \in [\frac{1}{n}, 1]$. We call τ the *truncation percentile*. The $\lfloor \tau n \rfloor$ individuals that are selected and placed into $\boldsymbol{\mathcal{S}}$ are simply the best individuals in the population $\boldsymbol{\mathcal{P}}$, i.e. $\forall \mathbf{Z} \in \boldsymbol{\mathcal{S}} : (\neg \exists \mathbf{Z}' \in (\boldsymbol{\mathcal{P}} - \boldsymbol{\mathcal{S}}) : \mathit{fitness}[\mathbf{Z}] \prec \mathit{fitness}[\mathbf{Z}'])$. The vector of selected solutions $\boldsymbol{\mathcal{S}}$ can be obtained efficiently by first sorting the individuals in $\boldsymbol{\mathcal{P}}$ on their fitness and by subsequently taking the top $\lfloor \tau n \rfloor$ individuals according to the ordering.

Mating

Since all classic recombination operators have two parent genotypes as input, the mating operation in the simple GA creates groups of two selected individuals. The classic recombination operators can return either a single offspring or two offspring. In the simple GA, two offspring are always returned. Since the offspring replace the current population in the simple GA, n new individuals have to be generated. This means that the mating operation should generate $\frac{n}{2}$ groups of 2 parents, which restricts n to be an even positive integer. The actual mating operation is performed by randomly picking 2 parents from the selected set until $\frac{n}{2}$ groups are generated. Note that this implies that certain selected genotypes may be picked as actual parents for recombination more than once whereas others are never picked. Furthermore, for a selection operator that selects less than n candidate individuals for recombination, mating *must* pick certain selected candidate individuals more than once for actual recombination.

Parents		Offspring
1 1 1 1 1 1 1 1 1 1 1 1 1	*one–point*	**1 1 1 1 1 1 1** 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0	*crossover*	0 0 0 0 0 0 0 0 **1 1 1 1 1**
1 1 1 1 1 1 1 1 1 1 1 1 1	*uniform*	0 **1 1** 0 0 **1** 0 **1** 0 0 **1** 0 **1**
0 0 0 0 0 0 0 0 0 0 0 0 0	*crossover*	**1** 0 0 **1 1** 0 **1** 0 **1 1** 0 **1** 0

Figure 3.2: Examples of one–point crossover and uniform crossover on the binary–vector genotype with $l = 13$.

Recombination

There are two classic recombination operators that are used in the simple GA, both of which take two parent genotypes and generate two offspring genotypes. Both operators are of the crossover type, which means that one offspring receives at locus i the allele at locus i from the one parent and the other offspring receives at locus i the allele at locus i from the other parent. Which alleles are copied into which offspring, is determined by the specific operator.

One–point crossover
In the *one–point crossover* operator, a crossover point $p \in \{0, 1, \ldots, l-1\}$ is selected at random. The first offspring receives the alleles at loci $\{0, 1, \ldots, p\}$ from the first parent. The remaining alleles at loci $\{p+1, p+2, \ldots, l-1\}$ are received from the second parent. The second offspring receives the alleles in the same way, but with the order of the parents reversed. An example of one–point crossover on the binary–vector genotype is given in Figure 3.2. Crossover is performed with a predefined recombination probability p_r. If no recombination should be performed, the offspring are be exact copies of the parents. This can be achieved by setting $p = l - 1$ in the one–point crossover operator.

Uniform crossover
In *uniform crossover*, it is determined for each locus i whether the first offspring will inherit the allele at locus i from the first parent or from the second parent. The second offspring will receive at locus i the allele at locus i from the other parent. The first parent is selected for transferring an allele to the first offspring with a probability called the *allele swapping probability* p_a. In the classic uniform crossover operator, this probability is set to $p_a = 0.5$. If no recombination should be performed, copies of the parents can be generated by temporarily setting $p_a = 0$. An example of uniform crossover is given in Figure 3.2.

Mutation

Mutation of a binary genotype in the simple GA changes every gene X_i to $1 - \mathsf{X}_i$ with the mutation probability p_m. This mutation operator is also known as *bit flipping mutation*.

Replacement

In the simple GA, the current population is completely replaced with the generated offspring. In our EA framework, this can be modelled by generating a replacement vector that contains only the offspring. This implies that the fitness of the best solution in the next generation might be worse than the fitness of the best solution in the current generation.

Survivor selection

The population at the beginning of the new generation in the simple GA consists of only the offspring of the previous generation. As a result, the survivor selection operation for the simple GA is similar to the replacement operation. Certainly, in any actual simple GA implementation, the replacement phase and the survivor selection phase would not be separately implemented.

3.2.2 Dynamics of the simple GA: schema analysis

In this section, we discuss an important basic analysis of the simple GA. Holland (1975) took the first steps to investigating the dynamics of the simple GA. His results provide an intuitive first explanation of how GAs work. Indirectly, these results also indicate why GAs work well on which types of problem. In the variant of the simple GA that was investigated by Holland, one–point crossover and proportionate selection are used. In the following, we briefly present the results by Holland.

Schemata

We begin by introducing the notion of *schemata* for binary genotypes. A *schema* is a vector of symbols with the same length l as the binary genotype. The alphabet for a schema is $\times_{i=0}^{l-1}\{0, 1, \star\}$. The $\star$ symbol is called a *wildcard*. A schema represents a subset of all possible binary genotypes of length l. It is also called a *similarity subset* since it defines a set of genotypes that are similar according to the schema. The binary genotypes that are represented by a schema are those that can be obtained by instantiating the $\star$ symbols in the schema with either a 0 or a 1. A binary genotype $\mathbf{X}$ is said to be *matched* by a schema $\boldsymbol{h}$ if and only if $\mathbf{X} \in \boldsymbol{h}$. For instance, we have that $11\star\star0 = \{11000, 11010, 11100, 11110\}$. The *order* $o(\boldsymbol{h})$ of a schema $\boldsymbol{h}$ equals the number of non–wildcard symbols. For instance, $o(11\star\star0) = o(\star01\star0\star\star) = 3$. Furthermore, the *defining length* $\delta(\boldsymbol{h})$ of a schema $\boldsymbol{h}$ is the distance between the positions of the outermost non–wildcard symbols in $\boldsymbol{h}$. This distance represents the number of points at which the schema can be cut in two so that some non–wildcard symbols are on the left of the cut and some are on the right of the cut. As an example, we have that $\delta(11\star\star0) = 4$ and $\delta(\star01\star0\star\star) = 3$.

Schema growth equation

The basic idea of the analysis by Holland is to observe the change in the average fitness of a schema in two subsequent generations. The observed result can then be extrapolated over many generations to predict the behavior of the average schema fitness. To do this, let $\boldsymbol{\mathcal{P}}^t$ denote the population at the beginning of generation t. The number of matches $\mu(\boldsymbol{h}, t)$ of schema $\boldsymbol{h}$ in population $\boldsymbol{\mathcal{P}}^t$ is:

$$\mu(\boldsymbol{h}, t) = \sum_{i=0}^{n-1} \begin{cases} 1 & \text{if } \boldsymbol{\mathcal{P}}_i^t \in \boldsymbol{h} \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

The population–based cumulative fitness $\varphi(\boldsymbol{h}, t)$ of a schema $\boldsymbol{h}$ at the beginning of generation t is the sum of the fitness values of all matched genotypes in $\boldsymbol{\mathcal{P}}^t$:

$$\varphi(\boldsymbol{h}, t) = \sum_{i=0}^{n-1} \begin{cases} \mathit{fitness}[\boldsymbol{\mathcal{P}}_i^t] & \text{if } \boldsymbol{\mathcal{P}}_i^t \in \boldsymbol{h} \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

The population–based average schema fitness $\overline{\varphi}(\boldsymbol{h}, t)$ of schema $\boldsymbol{h}$ at the beginning of generation t can now be computed as:

$$\overline{\varphi}(\boldsymbol{h}, t) = \frac{\varphi(\boldsymbol{h}, t)}{\mu(\boldsymbol{h}, t)} \tag{3.6}$$

Let $\bigstar$ be a schema of length l consisting only of $\star$ symbols, i.e. $\bigstar = \bigsqcup_{i=0}^{l-1}(\star)$. The cumulative population fitness at the beginning of generation t can now be written as $\varphi(\bigstar, t)$. Furthermore, the average population fitness can be written as $\overline{\varphi}(\bigstar, t)$. Also, note that $n = \mu(\bigstar, t)$. Since proportionate selection is used to select the parents, the probability $\mathit{proportion}[\boldsymbol{\mathcal{P}}_i^t]$ that the i–th genotype in population $\boldsymbol{\mathcal{P}}^t$ at the beginning of generation t is selected, is given by:

$$\mathit{proportion}[\boldsymbol{\mathcal{P}}_i^t] = \frac{\mathit{fitness}[\boldsymbol{\mathcal{P}}_i^t]}{\varphi(\bigstar, t)} \tag{3.7}$$

The probability $\psi(\boldsymbol{h}, t)$ that a genotype that is matched by $\boldsymbol{h}$ is selected at the beginning of generation t, is given by:

$$\psi(\boldsymbol{h}, t) = \sum_{i=0}^{n-1} \begin{cases} \mathit{proportion}[\boldsymbol{\mathcal{P}}_i^t] & \text{if } \boldsymbol{\mathcal{P}}_i^t \in \boldsymbol{h} \\ 0 & \text{otherwise} \end{cases} \quad = \quad \frac{\varphi(\boldsymbol{h}, t)}{\varphi(\bigstar, t)} \tag{3.8}$$

Without recombination and mutation, i.e. $p_r = 0$ and $p_m = 0$, the expected number of instances of schema $\boldsymbol{h}$ at the beginning of generation $t + 1$ equals n times the probability at selection at the beginning of generation t:

$$E[\mu(\boldsymbol{h}, t+1)] \;=\; n\psi(\boldsymbol{h}, t) \;=\; n\frac{\overline{\varphi}(\boldsymbol{h}, t)\mu(\boldsymbol{h}, t)}{\overline{\varphi}(\bigstar, t)\mu(\bigstar, t)} \;=\; \mu(\boldsymbol{h}, t)\frac{\overline{\varphi}(\boldsymbol{h}, t)}{\overline{\varphi}(\bigstar, t)} \tag{3.9}$$

This means that if the population–based average schema fitness $\overline{\varphi}(\boldsymbol{h}, t)$ remains larger than the average population fitness $\overline{\varphi}(\bigstar, t)$, then the number of matches of $\boldsymbol{h}$ in the population grows exponentially over subsequent generations.

Now, a schema may be disrupted by recombination and mutation since individuals that were first matched by a schema $\boldsymbol{h}$ become unmatched if a non–wildcard symbol changes. For the one–point crossover operator, an individual can only become unmatched if the crossover point falls between the two outer non–wildcard positions. If the non–wildcard positions in the schema contain the same values throughout the two parents that we are to apply one–point crossover to, no schema disruption can occur at all. So, using the defining length, we thus have that schema $\boldsymbol{h}$ survives recombination with probability *at least* $1 - p_r \frac{\delta(\boldsymbol{h})}{l-1}$. For bitwise mutation, a schema survives if none of the non–wildcard positions are mutated. Using the schema order, this means that schema $\boldsymbol{h}$ survives with probability $(1 - p_m)^{o(\boldsymbol{h})}$. Furthermore, additional schema matches may be introduced by the use of recombination and mutation on other schemata. These considerations lead to the final equation for the expected number of matches of schema $\boldsymbol{h}$ at the beginning of generation $t + 1$ in the simple GA:

$$E[\mu(\boldsymbol{h}, t+1)] \geq \mu(\boldsymbol{h}, t) \frac{\overline{\varphi}(\boldsymbol{h}, t)}{\overline{\varphi}(\bigstar, t)} \left(1 - p_r \frac{\delta(\boldsymbol{h})}{l-1}\right) \left((1 - p_m)^{o(\boldsymbol{h})}\right) \tag{3.10}$$

Equation 3.10 is also called the *schema growth equation*. This equation shows that if the part on the righthandside after $\mu(\boldsymbol{h}, t)$ remains greater than or equal to one in subsequent generations, there will be an exponential growth in the number of matches of schema $\boldsymbol{h}$ in the form of $E[\mu(\boldsymbol{h}, t)] = \mu(\boldsymbol{h}, 0)(1 + \varepsilon)^t$ for some $\varepsilon \geq 0$. However, the average fitness of the population $\overline{\varphi}(\bigstar, t)$ will finally catch up with the population–based average fitness of the schema $\overline{\varphi}(\boldsymbol{h}, t)$ since the population size n is finite. At such a time, most of the genotypes will already have converged to match schema $\boldsymbol{h}$.

Dynamics of the simple GA

The part of the schema growth equation concerning the survival probabilities as a result of recombination and mutation indicate that the exponential growth is more likely to occur when both $\delta(\boldsymbol{h})$ and $o(\boldsymbol{h})$ are small. Furthermore, a prerequisite is that the population–based average schema fitness of schema $\boldsymbol{h}$ remains larger than the average population fitness in subsequent generations. This leads to the expectation that the exponential growth of a schema is more likely to occur if the schema has an above average fitness, has a short defining length and is of low order. This is exactly what is stated in the *schema theorem* by Holland (1975):

> **Schema Theorem**
> Low–order, above–average schemata with a small defining length receive an exponentially increasing number of matches in subsequent generations in a simple genetic algorithm.

The theorem has led Holland (1975) to pose an hypothesis on how simple GAs work. This hypothesis is known as the *building block hypothesis*:

Building–Block Hypothesis
A simple genetic algorithm seeks near–optimal performance through the juxtaposition of low–order, high–performance schemata with a small defining length, called the building blocks.

Example 3.2. Suppose that we want to maximize the function $\mathfrak{F}(\zeta_0) = \zeta_0 + |\sin(32\zeta_0)|$ of a single real–valued variable $\zeta_0 \in [0, \pi]$. A graphical illustration of this function is shown in figure 3.3. We encode the real variable with a binary genotype of length $l = 16$ using the transformation described in equation 3.1. We use a simple GA with one–point crossover and proportionate selection and set $p_r = 1$ and $p_m = 0$. We set the population size to $n = 100$ and observe the properties of the schemata that are obtained by setting the first two symbols to 00, 01, 10 and 11 respectively, and by setting each of the remaining fourteen symbols to $\star$. These four schemata pertain to the genotypes that are mapped to the intervals $[0; \frac{1}{4}\pi)$, $(\frac{1}{4}\pi; \frac{1}{2}\pi)$, $(\frac{1}{2}\pi; \frac{3}{4}\pi)$ and $(\frac{3}{4}\pi; \pi]$ respectively. The last schema contains the fittest genotypes and is therefore guaranteed to have a higher average fitness than the entire population. Empirical validation of this observation can be seen in the graph of the population–based average schema fitness in Figure 3.3 at generation 0, the time at which the population contains only randomly generated genotypes. The number of matches of each schema in $\boldsymbol{\mathcal{P}}^0$ is expected to be around $\frac{1}{4}n = 25$ since each schema represents an equally large similarity subset. This can also be seen in Figure 3.3. The dynamics as indicated by the schema theorem can be seen in Figure 3.3 in the graph of the number of matches. The schema that describes the fittest part of the search space has an above–average fitness and an explosive increase in the number of matches of this schema results as generations pass. After sixteen generations, all population individuals are matched by the best schema and the average fitness of the population equals the population–based average fitness of the best schema.

The slow convergence behavior resulting from the use of proportionate selection can also be seen from this small experiment. The increase of the average fitness becomes smaller as generations pass, but still no termination is enforced because there are still a few different types of individual left. It takes unnecessarily long before proportionate selection has driven the population to only a single solution. The experiment was stopped after 40 generations, but at that time, the genotypes in the population were not yet identical.

From the schema theorem and the building–block hypothesis, it follows that the simple GA processes *building blocks*. Building blocks can be seen as (small) sets of positions in the binary genotype that together make an important contribution to the fitness of the genotype. It has also become apparent from the analysis that the building blocks that are processed by the simple GA with one–point crossover are those that have gene locations that are close together. If

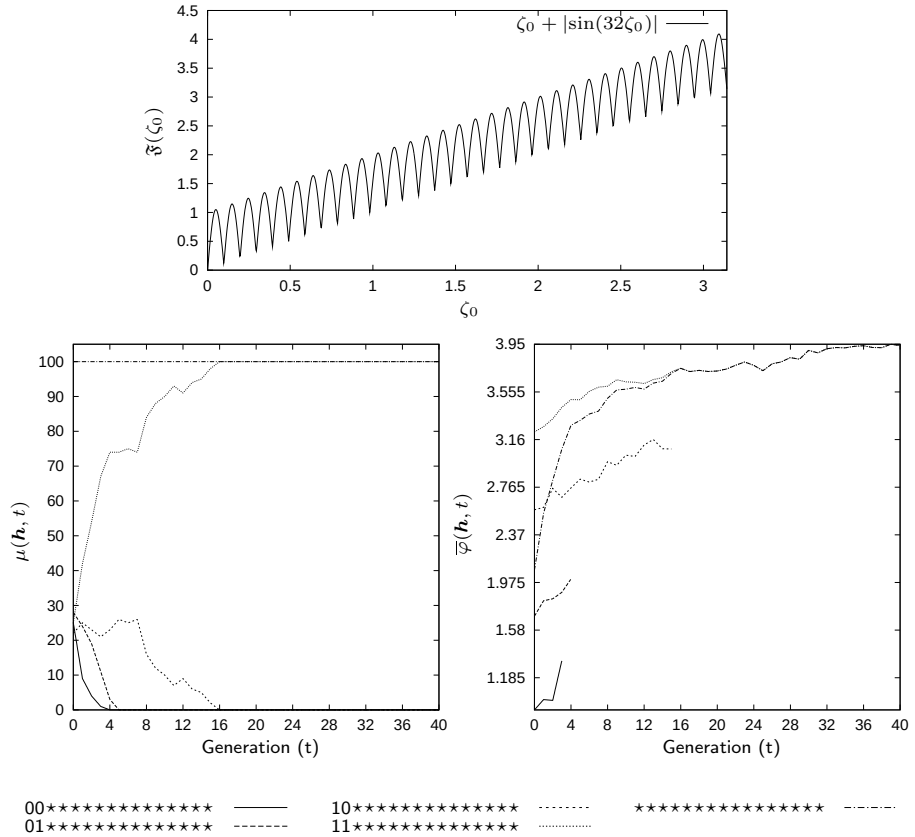


Figure 3.3: A single run of a simple GA ([$n = 100$]; proportionate selection; one–point crossover [$p_r = 1.0$]; no mutation; without elitism [*replace all solutions*]). The goal is to maximize $\mathfrak{F}(\zeta_0) = \zeta_0 + |\sin(32\zeta_0)|$ (top plot). Problem variable ζ_0 is encoded by sixteen binary variables. Five schemata indicate the working of the schema theorem.

such building blocks exist in the encoding of the optimization problem, we speak of *tight linkage* in the binary genotype. In general, the dependencies between the bits that together contribute to the fitness of a genotype is called *linkage*. Building blocks constitute one type of linkage.

3.2.3 Difficult problems for simple GAs

Building upon the schema theorem from the previous section, we now first elaborate on what is likely to be a difficult problem for a simple GA, but still qualifies as an optimization problem that we should be able to solve efficiently if we do some proper induction. In other words, these problems can be optimized very efficiently if the problem's structure is known beforehand so that we can adapt the recombination operator to it. According to the schema theorem, we at least require an optimization problem to contain indivisible building blocks of a length larger than 1 in order to not be able to guarantee efficient optimization by the simple GA. If a building block has length 1, it can't be disrupted by crossover and it would be processed efficiently according to the schema theorem.

Non–decomposable subfunctions

To design optimization problems that contain elementary and non–decomposable building blocks, we define subfunctions that pertain to a subvector of the complete genotype. We will make a combination of such subfunctions to compute the actual fitness of a complete genotype. The combination of bits that leads to the optimal value of the subfunction maps to a building block.

Needle–in–a–haystack subfunction
The simplest way to define a subfunction that ensures that the optimal combination of bits cannot be efficiently found by trying out smaller combinations of bits for subvectors of the given genes, is to assign a value of 0 to all combinations of bits, except for one combination, which is assigned a value of 1. Such a function is called a *needle–in–a–haystack* function. Let the optimal combination be given by only 1–symbols. The needle–in–a–haystack subfunction can be written as follows, given a vector of binary genes $\mathsf{X}_{\boldsymbol{j}}$, $\boldsymbol{j} \sqsubseteq \boldsymbol{\mathcal{L}}$:

$$g_{needle}(\mathsf{X}_{\boldsymbol{j}}) = \begin{cases} 1 & \text{if } \forall i \in \{0, 1, \ldots, |\boldsymbol{j}| - 1\} : \mathsf{X}_{\boldsymbol{j}_i} = 1 \\ 0 & \textit{otherwise} \end{cases} = \prod_{i=0}^{|\boldsymbol{j}|-1} \mathsf{X}_{\boldsymbol{j}_i} \quad (3.11)$$

Deceptive–trap subfunction
Another way to construct elementary building blocks is given by the *deceptive trap* function that was proposed by Deb and Goldberg (1994). Similar to the needle in a haystack subfunction, the combination of 1 symbols constitutes the optimal combination of bits for the deceptive trap subfunction. While induction on combinations other than the optimal combination, gives *no* additional information in the needle–in–a–haystack function, in the deceptive trap subfunction,

additional information can be gained by induction on other combinations of bits. However, this information is *deceptive* as it leads to the single *suboptimal* combination of bits, which is the combination of 0–symbols only. This suboptimal combination of bits is also called the deceptive *attractor*.

$$g_{deceptive}(\mathsf{X}_{\boldsymbol{j}}) = \begin{cases} 1 & \text{if } \forall i \in \{0,1,\ldots,|\boldsymbol{j}|-1\} : \mathsf{X}_{\boldsymbol{j}_i} = 1 \\ \frac{|\boldsymbol{j}|-1-\sum_{i=0}^{|\boldsymbol{j}|-1} \mathsf{x}_{\boldsymbol{j}_i}}{|\boldsymbol{j}|} & otherwise \end{cases} \tag{3.12}$$

The trap function has been designed to *deceive* a simple GA to find only suboptimal combinations of bits instead of the optimal combination. Let λ be the number of variables that the deceptive subfunction pertains to. The formal reason for the deception is that if we average the fitness over all combinations represented by a schema of order $k < \lambda$ with 0–symbols only, we get a lower average fitness than when we average the fitness of all combinations represented by a schema of order $k < \lambda$ with only 1–symbols. Only if we consider schemata of order λ do we find that a combination of 1–symbols only is more preferable (Deb and Goldberg, 1994). This implies that for optimizing the deceptive trap function, building blocks of length λ must have a good chance to survive recombination and mutation, otherwise they will not be processed. Without a good chance to survive, lower–order subschemata will have a larger probability to survive. Combined with the fact that such lower–order schemata containing 0–symbols only have a higher fitness on average, the schema theorem dictates that the GA will result in exponential growth of building blocks of a length smaller than λ containing 0–symbols. As a result, exponential growth of the schemata containing the optimal building block using one–point crossover in the simple GA is much more likely if the defining length of the schema is small.

Additively decomposable optimization problems of a bounded order

To ensure exponential growth of the schemata that contain the optimal building blocks when using one–point crossover, we thus require a tight linkage encoding of the genes over which the subfunctions are defined. However, there is no reason to assume such tight linkage in BBO. For simplicity, suppose that the fitness function consists of subfunctions that all concern binary vectors of length λ. Furthermore, let $\boldsymbol{\iota} = (\boldsymbol{\iota}_0, \boldsymbol{\iota}_1, \ldots, \boldsymbol{\iota}_{|\boldsymbol{\iota}|-1})$ be a vector of vectors describing the indices of the genes that the $|\boldsymbol{\iota}|$ subfunctions work on. We call $\boldsymbol{\iota}$ the *index cluster vector*. We call each component $\boldsymbol{\iota}_j$ of $\boldsymbol{\iota}$ an *index vector*. The index cluster vector satisfies the property that an index is contained in some index vector if and only if it is the index of a locus in the genotype, i.e. $i \in \boldsymbol{\mathcal{L}} \leftrightarrow \exists \boldsymbol{\iota}_j \in \boldsymbol{\iota} : i \in \boldsymbol{\iota}_j$. If a genotype $\mathbf{X}$ has an optimal combination of bits assigned to the genes indicated by an index vector $\boldsymbol{\iota}_j$, we say that the index vector $\boldsymbol{\iota}_j$ is optimal in $\mathbf{X}$.

The problems that we at least expect to be able to solve efficiently using induction are those in which no $\boldsymbol{\iota}_j$ is larger than a maximum order of interaction between genes κ^ι, i.e. $\forall \boldsymbol{\iota}_i \in \boldsymbol{\iota} : |\boldsymbol{\iota}_i| \leq \kappa^\iota$ and in which the index vectors themselves are mutually exclusive, i.e. $\forall (\boldsymbol{\iota}_i, \boldsymbol{\iota}_j) \in \boldsymbol{\iota} \times \boldsymbol{\iota} : i \neq j \rightarrow \boldsymbol{\iota}_i \sqcap \boldsymbol{\iota}_j = ()$. If

no constant bound κ^ι on the maximum size of an index vector exists, we can't optimize the problem efficiently because we require an exponentially growing number of samples, $\Theta(2^{\kappa^\iota})$, to find the optimal combination of bits for either of our two subfunction examples. However, assuming that κ^ι is constant and that the contributions of the subfunctions are completely independent from each other, just a constant number of samples is required to find the optimal combination of bits for an index vector. Ensuring that the contributions of the subfunctions are completely independent from each other can be achieved by summing the values of the subfunctions. By doing so, we get an *additively decomposable* fitness function. For example, we can define the additively decomposable needle–in–a–haystack and deceptive trap fitness functions given a genotype of length $l = |\boldsymbol{\iota}|\lambda$ as follows:

$$\mathfrak{G}_{additive\_needle}(\mathbf{X}) = \sum_{i=0}^{|\boldsymbol{\iota}|-1} g_{needle}(\mathsf{X}_{\boldsymbol{\iota}_i}) \tag{3.13}$$

$$\mathfrak{G}_{additive\_deceptive}(\mathbf{X}) = \sum_{i=0}^{|\boldsymbol{\iota}|-1} g_{deceptive}(\mathsf{X}_{\boldsymbol{\iota}_i}) \tag{3.14}$$

Regardless of the contents of the index vectors $\boldsymbol{\iota}_j$ we should be able to solve problems such as the ones in equations 3.13 and 3.14 efficiently by inducing that there are subfunctions that contribute to the fitness independently of each other. So, if $\boldsymbol{\iota}_j = (j\lambda, j\lambda+1, \ldots, j\lambda+\lambda-1)$, which ensures tight linkage, we expect one–point crossover to show good scale–up behavior with increasing $|\boldsymbol{\iota}|$ in terms of the minimally required population size, the required number of evaluations, and the actual running time of the simple GA. But in many optimization problems, we don't know what indices are contained in each $\boldsymbol{\iota}_j$. In the worst case, the index vectors may be encoded with as loose linkage as possible, so $\boldsymbol{\iota}_j = (j, j+|\boldsymbol{\iota}|, \ldots, j+(\lambda-1)|\boldsymbol{\iota}|)$. The defining length of each optimal building block is now very large and the schema theorem does not guarantee efficient growth of the building blocks using the one–point crossover operator. The question to answer now is *how* bad can the scale–up behavior of the simple GA get in such a case? This is the topic of the next section.

3.2.4 Competence of the simple GA: mixing analysis

In this section, we describe results by Thierens (1995) and results by Harik, Cantú-Paz, Goldberg and Miller (1999) that provide bounds on the minimally required population size n to solve optimization problems using different variants of the simple GA. The worst case scenario is obtained if a linkage unfriendly crossover operator is used. The best case scenario is obtained if a crossover operator is used that is extremely linkage friendly because it only has crossover points such that the important building blocks are not disrupted. The results serve to point out the disadvantages of the simple GA and the implications thereof. It is these disadvantages that we will remedy in this thesis using an approach based on learning probabilistic models.

Linkage unfriendly crossover operators

Thierens (1995) analyzed the scale–up behavior with increasing $|\boldsymbol{\iota}|$ of the simple GA with uniform crossover. Since we cannot assume tight linkage in optimization problems in general, any static crossover operator is very likely to disrupt building blocks. Therefore, the results apply to any simple GA that disregards linkage information, such as a simple GA with one–point crossover. Mutation is disregarded by Thierens since it is unlikely that mutation will generate a completely correct combination of bits for a subfunction. Furthermore, for deceptive building–block optimization functions, the deceptive attractor is likely to be quite different from the optimal combination of bits. Other combinations of bits are more likely to differ only a few bits from the deceptive attractor than from the optimal combination of bits since combinations that differ only a few bits from the deceptive attractor have a higher fitness contribution on average. The emergence of correct building blocks is therefore more likely to happen as a result of the mixing of genetic material due to crossover than due to mutation. Thierens (1995) showed that under the assumption that the selection pressure, the crossover probability, and the desired probability of failure of the simple GA are constant, the scale–up behavior of the simple GA with uniform crossover on additively decomposable deceptive optimization problems, is:

$$n \;=\; \Omega\left(2^{\lambda}\frac{2^{|\boldsymbol{\iota}|}}{|\boldsymbol{\iota}|^{2\frac{1}{2}}}\right) \;=\; \Omega\left(2^{\lambda}(2-\varepsilon)^{|\boldsymbol{\iota}|}\right) \text{ for some } 0<\varepsilon<1 \tag{3.15}$$

Thus, the requirement on the population size grows exponentially with the length of the index vectors. Moreover, the result shows that the requirement on the population size also grows exponentially with the *number* of index vectors. This behavior is not only to be expected when uniform crossover is used, but in general when a crossover operator is used that is likely to disrupt the building blocks. For instance, without tight linkage, one–point crossover has an even smaller chance of generating new optimal combinations for subfunctions than uniform crossover does.

Linkage friendly crossover operators

The schema theorem indicates that schemata that are tightly encoded and have an above average fitness, *will* be processed efficiently using one–point crossover since they will have an exponential number of matches over subsequent generations. However, we so far only know the scale–up behavior using linkage unfriendly crossover operators. The impression left by the schema theorem was quantified by Harik et al. (1999) in the scale–up behavior of a simple GA on additively decomposable problems under the assumption that recombination has only a very slight chance of disrupting optimal combination of bits for subfunctions. With respect to the simple GA with one–point crossover this situation corresponds to the *presence* of tight linkage. In their analysis, the use of gene–wise mutation was disregarded, similarly as in the analysis by Thierens (1995),

since such mutation is not likely to generate complete optimal building blocks. Harik et al. (1999) showed that under the assumption that the variance in fitness over the combinations for an index vector, the signal and the probability of failure of the simple GA are constant, the implications on the scale–up behavior of the simple GA on additively decomposable problems using a perfect crossover operator that respects the linkage, is:

$$n = \Theta(2^{\lambda}\sqrt{|\boldsymbol{\iota}|}) \tag{3.16}$$

In comparison with the result by Thierens (1995), the scale–up behavior with respect to the number of subfunctions shows a significant improvement if the recombination operator does not disrupt the optimal combinations of bits for the subfunctions. This is for instance the case for the one–point crossover operator in the presence of tight linkage. In this case, the required population size grows only as the square root of the number of index vectors, as is indicated by the result in equation 3.16. Without the ability of keeping the probability of disrupting optimal combinations of bits for the subfunctions low, such as is always the case when using uniform crossover or when using one–point crossover in the absence of tight linkage, the population size grows exponentially with the number of index vectors if the subfunctions are deceptive, as is indicated by the result in equation 3.15.

Experimental validation

An experimental validation of the analytical results by Thierens (1995) and by Harik et al. (1999) is given in figures 3.4 and 3.5. In Figure 3.4, the scale–up behavior of a simple GA is shown on the additively decomposable needle function $\mathfrak{G}_{additive\_needle}(\mathbf{X})$. Tournament selection was used with a tournament size of 4. The probability of recombination was set to $p_r = 0.5$ whereas the probability of mutation was set to $p_m = 0$. The minimal requirements averaged over 30 runs to find the optimum in all 30 runs are shown for uniform crossover, one–point crossover with tight linkage and loose linkage encodings, and perfect crossover. The perfect crossover operator exchanges information between parents in such a way that each offspring receives the information at all loci of the j–th index vector from one of the two parents. The donor parent is chosen anew at random for each index vector. In this way, no building blocks can ever be disrupted and the information is mixed perfectly. For longer index vectors, perfect crossover outperforms one–point crossover with a tight encoding of the index vectors, which in turn outperforms uniform crossover, which in turn outperforms one–point crossover with a loose encoding. In the last case, the optimal combinations of bits are almost certain to be disrupted as the defining lengths of the building blocks are quite large.

The figure shows that *all* variants of the simple GA scale up polynomially instead of exponentially on the additively decomposable needle function. The reason for this behavior is that the subfunctions are not deceptive. The premise of the analysis by Thierens that the mixing of optimal combinations of bits is

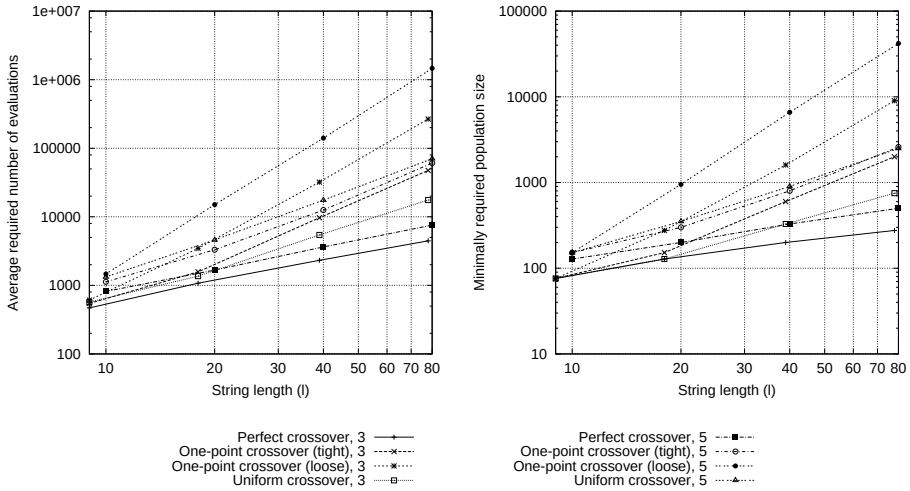


Figure 3.4: Scale–up behavior of simple GA variants on the additively decomposable needle function with subproblems of length 3 and 5. The results are averaged over 30 runs and indicate the minimal computational requirements to find the optimal solution in all 30 runs. The results are plotted on a log–log scale. Straight lines on this scale indicate polynomial scale–up behavior. (tournament selection [$s = 4$]; perfect crossover [$p_r = 0.5$], one–point crossover (tight linkage) [$p_r = 0.5$], one–point crossover (loose linkage) [$p_r = 0.5$], uniform crossover [$p_r = 0.5$]; no mutation; without elitism [*replace all solutions*]).

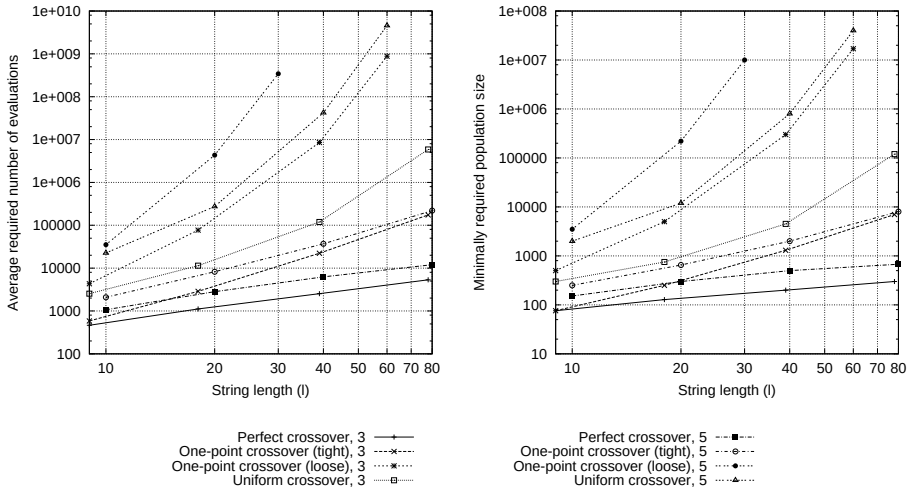


Figure 3.5: Scale–up behavior of the simple GA on the additively decomposable deceptive trap function. The same simple GA variants were used as for the results in Figure 3.4. The results are shown on a log–log scale.

the only significant way in which building blocks are introduced into offspring, is therefore not satisfied. Optimal combinations of bits are also likely to be created or to emerge through recombination since there is no deceptive attractor that causes good bits for any subfunction to disappear through selection. Without the deceptive attractor, any combination of bits other than the optimal one, is equally likely for any index vector. Therefore, recombination has a good chance at combining bits into a perfect sub–solution. Still, the scale–up behavior of the simple GA does become worse if the linkage information is disregarded. For the needle subproblem with index vector length 5, the minimally required population size for the perfect crossover operator scales approximately with $\mathcal{O}(l^{0.66})$ whereas the number of function evaluations scales approximately with $\mathcal{O}(l^{1.08})$. For one–point crossover with tight linkage, the scale–up behavior in terms of the minimally required population size and the number of function evaluations approximately are of $\mathcal{O}(l^{1.37})$ and of $\mathcal{O}(l^{1.92})$, respectively. For uniform crossover we get $\mathcal{O}(l^{1.35})$ and $\mathcal{O}(l^{1.91})$ respectively and for one–point crossover with loose linkage, we approximately obtain a scale–up behavior of $\mathcal{O}(l^{2.71})$ and $\mathcal{O}(l^{3.31})$ respectively. So, even though there is no exponential scale–up behavior, respecting the linkage of the problem does help to solve the optimization problem more efficiently.

In Figure 3.5, the scale–up behavior of the simple GA is shown for the additively decomposable deceptive trap function $\mathfrak{G}_{additive\_deceptive}(\mathbf{X})$ for different index vector lengths. The perfect crossover operator clearly scales up polynomially in terms of both the minimally required population size and the required number of evaluations. The regression slope is similar to that obtained for the additively decomposable needle function. The one–point crossover operator in the presence of tight linkage can also be observed to scale up polynomially, albeit worse than the perfect crossover operator. Both the uniform crossover operator and the one–point crossover operator with loose linkage scale up exponentially as the problem length increases because they are too disruptive.

3.3 Linkage–respecting genetic algorithms

The results presented in the previous section indicate the importance of processing the linkage information in an optimization problem by an EA. Polynomial scale–up behavior of the simple GA is only obtained if the optimization problem is encoded with tight linkage. But in most optimization problems, we do not know in advance whether tight linkage is present or not. To still be able to process linkage information in an EA, two main approaches can be taken. In this section, we briefly discuss some of the main existing contributions in this area. A full treatment of the theory and design behind the EAs introduced here, is out of the scope of this thesis. To this end, we refer the interested reader to the indicated publications.

The locus of an allele can be specified explicitly. A genotype then becomes a string of l pairs (*allele*, *locus*) $\in \mathbb{B} \times \{0, 1, \ldots, l-1\}$. The actual binary string is then obtained by placing each allele at its specified locus. This allows genes to

be placed close to each other such that a crossover operator that works on the genotype is less likely to disrupt the building blocks. This encoding was first proposed by Bagley (1967). Along with this encoding, the inversion operator was introduced. This mutation operator works directly on the (*allele*, *locus*) representation and reverses a part of this string to allow genes to be placed closer to each other in the string. However, the application of crossover to the fully specified (*allele*, *locus*) representation can lead to offspring genotypes in which loci are missing or in which they appear more than once. Crossover can only be applied to the (*allele*, *locus*) strings without problems if the parent strings have the same locus information at the each position in the (*allele*, *locus*) strings. This restricted application of recombination led to a GA in which crossover was hardly applied. Therefore, the added use of the inversion operator turned out not to be very effective using this recombination scheme (Frantz, 1972).

The same encoding is used in the messy GA (mGA) (Goldberg, Korb and Deb, 1989). In the messy GA, the (*allele*, *locus*) strings are not of a fixed length and may be overspecified or underspecified. Overspecification means that there are more genes with the same locus, whereas underspecification means that there are loci that do not occur in any gene. To resolve overspecification, the binary solution string is constructed by scanning the (*allele*, *locus*) strings from left to right and by placing an allele at a certain locus only if no allele has been placed at that locus as yet. To resolve underspecification, a *template* is used which is a binary string of length l. If a locus does not occur in the (*allele*, *locus*) string, the allele is copied from the template at the missing locus. The template is fixed during certain stages of the mGA and represents a locally optimal solution. As the mGA progresses, the template string is capable of representing local optimal solutions that can only be found by processing higher–order linkage information. Although the mGA is capable of solving additively decomposable problems in polynomial time, its scale–up behavior with respect to the number of required function evaluations, is $\mathcal{O}(l^{\kappa^\iota})$, where κ^ι is the maximum number of genes that are related in any subfunction.

The fast messy GA (fmGA) (Goldberg, Deb, Kargupta and Harik, 1993; Kargupta, 1995) uses the same encoding as is used in the mGA. One of the main problems with the mGA is that it requires a deterministically complete initialization of all strings of length k where k is the maximum order of interaction that can be processed in the mGA. This phase is replaced by a probabilistically complete initialization phase in the fmGA. Furthermore, enhanced operators are introduced in the fmGA that allows for filtering good building blocks more efficiently. As a result, the fmGA scales subquadratically with respect to the required number of function evaluations on additively decomposable problems.

The most recent approach that uses the special (*allele*, *locus*) string is known as the linkage learning GA (llGA) (Harik, 1997; Harik and Goldberg, 1997; Lobo, Deb, Goldberg, Harik and Wang, 1998). In this GA, each (*allele*, *locus*) string is seen as a circle. With each circle, a starting point is associated such that processing the circle in a clockwise fashion from the starting point, provides for obtaining a binary string in a similar fashion as is done in the mGA and fmGA. No circle can be underspecified, but overspecifications are allowed. The

llGA is similar to the simple GA in its generational dynamics, but has a specific recombination operator. In this operator, an offspring is constructed by first copying the second parent and by subsequently injecting a randomly chosen segment from the first parent into the second parent, after which genes with identical loci that occur more than twice, as encountered from the starting point, are deleted. To efficiently solve additively decomposable problems, Harik (1997) showed that additional redundant genes that do not encode parts of the binary string need to be incorporated in the circle. By doing so, the llGA was shown to scale up subquadratically on additively decomposable problems.

The results obtained from the gambler's–ruin model indicate that the scale–up behavior of the simple GA is very efficient if a crossover operator would be used that never disrupts building blocks yet mixes them perfectly. Trying to find such a crossover operator to use directly on the binary genotype is a second approach to improving the performance of the simple GA. Two efficient and competent GAs that take this approach are the gene expression messy genetic algorithm (gemGA) (Kargupta, 1996; Kargupta and Bandyopadhyay, 1998) and the building–block filtering GA (bbfGA) (van Kemenade, 1998). In both EAs, the effect on the fitness of changing bit values at certain loci is taken into account. Specifically, the genes that upon alteration result in the strongest decrease in fitness are expected to contribute to an important schema and are therefore regarded as being linked. This information is combined over multiple genotypes and is then used to construct linkage sets (gemGA) or crossover masks (bbfGA). This linkage information is respected when crossover is performed so as few building blocks as possible are disrupted. Although there are more details to these approaches, they are efficient. For the gemGA, results have been reported on additively decomposable fitness functions that indicate *linear* scale up behavior with respect to the required number of function evaluations.

3.4 Discussion

The linkage lessons learned from analyzing the simple GA can be used to design efficient GAs that are polynomially scalable on additively decomposable fitness functions. It should be noted though, that additively decomposable fitness functions are well suited to be solved using a crossover operator, since the combinations of bits for the subfunctions can be perfectly mixed to obtain a globally optimal solution. The boundaries of the index vectors are distinct and the non–linear interaction between the bits is bounded by a constant. The added value of a linkage–preserving crossover operator is important, but it still is quite a specific way to exploit a problem's structure. There may be other types of dependency that cannot be directly processed using crossover operators. If we for instance use the binary genotype to encode a solution in $\mathbb{R}^d$, the relations between the bits are unclear and may be quite complex, depending on the intrinsic structure of the real problem. If we use a vector of real values to directly encode a solution, we still want to have a way to exploit the problem's structure even though we cannot directly express the type of linkage between

real–valued variables in numerical optimization in a form that matches with the type of linkage information that can be processed by a crossover operator. To be able to express this broader concept of dependency in a model on which recombination is based in EAs and to exploit the structure of an optimization problem in general, we may require different tools. In the remainder of this thesis, we turn our attention to one such tool, which is based on probability theory.

'What you need is an idea.'
William Powell Lear

Iterated density–estimation evolutionary algorithms

The type of induction that can be performed by the simple GA is restricted, as was shown in Chapter 3. If one–point crossover is used, the simple GA can only process dependencies between genes effectively if the loci of these genes are close to each other. If uniform crossover is used, the simple GA cannot process such dependencies efficiently at all, regardless of the loci of the dependent genes. Without the ability to efficiently process dependencies, the minimally required population size and number of evaluations for a simple GA to solve certain additively decomposable optimization problems scale up exponentially. In this chapter, we study the use of probability distributions to learn and process dependencies between genes.

The chapter is organized as follows. We first show in Section 4.1 how a scale–up behavior that is at most as good as that of the simple GA with uniform crossover is achieved by an evolutionary algorithm (EA) that iteratively estimates a univariately factorized probability distribution and subsequently draws new solutions from this distribution. We then show that using a more complex factorization results in a scale–up behavior that is at least as good as that of the simple GA using perfect crossover. These results indicate that learning dependencies between genes is strongly related to estimating probability distributions. Through different classes of probability distributions, different types of dependency can be expressed, resulting in different types of induction. To investigate the impact of using different probability distributions in an EA on its performance, a new EA framework is defined in Section 4.2. The main research contributions in this thesis are based on this framework. In Section 4.3 we review the literature on related work by discussing different classes of probability distributions that have been used in EAs.

4.1 Prelude to the IDEA: from crossover to probability distributions

From the results presented in section 3.2.4 we know that the use of the uniform crossover (UX) operator in a GA may result in an exponential scale–up behavior, even if the fitness function for the optimization problem under consideration is additively decomposable and the subfunctions are no larger than a certain fixed maximum size. However, if a crossover operator can be defined for the problem that efficiently mixes partial solutions, the scale–up behavior can become much better. The scale–up behavior can in fact become sublinear for additively decomposable optimization problems.

As an alternative approach to using a crossover operator in a GA, we could generate a new offspring genotype by drawing it from a probability distribution that is estimated over the solutions that are selected from the current population. To this end, for example, a univariately factorized probability distribution, or univariate factorization for short, can be used. If such a factorization is used, the probability that a 1–symbol is generated for a certain locus i for the new offspring genotype, equals the proportion of 1–symbols at locus i in the vector of selected solutions. This approach is identical to using a variant of the uniform crossover operator in which the vector of parent genotypes equals the entire vector of selected solutions: since the donor parent is chosen randomly for each locus i, the probability that a 1–symbol is generated for that locus using this unusual version of uniform crossover equals exactly the proportion of 1–symbols at locus i in the vector of selected solutions. Similarly, an alternative to using perfect crossover for additively decomposable optimization problems is to use a probability distribution in which the probability of generating a new combination of bits for a certain subfunction equals the joint proportion of that combination of bits in the vector of selected solutions.

To investigate the properties of using probability distributions in a GA as outlined above, we could attempt an analysis of competence similar to the ones discussed in section 3.2.4. However, as the generation of a new offspring genotype with a crossover operator can also be described by a probability distribution, we propose to investigate the differences between the two approaches to predict the competence of a GA that uses a probability distribution.

4.1.1 Crossover probability distributions

In this section, we characterize the probability distributions that describe the use of the uniform crossover operator and of the perfect crossover operator, respectively. To do so, we introduce a binary stochastic random variable X_i for each gene variable X_i. Furthermore, we let $\boldsymbol{\mathcal{X}}$ denote the vector of all these random variables, $\boldsymbol{\mathcal{X}} = (X_0, X_1, \ldots, X_{l-1})$, $l \geq 1$, and let $P(\boldsymbol{\mathcal{X}})$ denote the true probability distribution from which a vector of selected solutions is drawn. A crossover operator now generates new solutions according to a probability distribution $\hat{P}(\boldsymbol{\mathcal{X}})$. Since recombination is a function of the selected solutions,

this probability distribution is derived from the vector of selected solutions and can thus be seen as an *estimation* of the true probability distribution $P(\boldsymbol{\mathcal{X}})$.

Uniform crossover probability distribution

To describe the use of the uniform crossover operator in terms of a probability distribution, we want to characterize the probability that a new offspring genotype $\boldsymbol{o}$ that is generated by the operator equals a given vector $\boldsymbol{x}$ of l bits. We call the probability distribution that returns this probability for any possible vector $\boldsymbol{x}$, the uniform crossover probability distribution, which we denote by $\hat{P}^{UX}(\boldsymbol{\mathcal{X}})$. Drawing samples from the probability distribution $\hat{P}^{UX}(\boldsymbol{\mathcal{X}})$ is now equivalent to performing uniform crossover. To characterize $\hat{P}^{UX}(\boldsymbol{\mathcal{X}})$, we first consider the case in which the uniform crossover operator is used to generate just a single offspring genotype $\boldsymbol{o}$ by deciding at random for each locus whether to inherit the allele from the first parent $\boldsymbol{p}^0$ or from the second parent $\boldsymbol{p}^1$. Without loss of generality, we assume that the first parent is the j–th solution from the selected solution vector $\boldsymbol{\mathcal{S}}$ and that the second parent is the k–th selected solution, so $\boldsymbol{p}^0 = \boldsymbol{\mathcal{S}}_j$ and $\boldsymbol{p}^1 = \boldsymbol{\mathcal{S}}_k$. The probability that the offspring $\boldsymbol{o}$ generated by uniform crossover from $\boldsymbol{p}^0$ and $\boldsymbol{p}^1$ equals the vector $\boldsymbol{x}$ now is:

$$\Pr(\boldsymbol{o} = \boldsymbol{x} \mid \boldsymbol{p}^0 = \boldsymbol{\mathcal{S}}_j, \boldsymbol{p}^1 = \boldsymbol{\mathcal{S}}_k) \; = \tag{4.1}$$
$$\prod_{i=0}^{l-1} \frac{1}{2}\boldsymbol{x}_i \left((\boldsymbol{\mathcal{S}}_j)_i + (\boldsymbol{\mathcal{S}}_k)_i \right) + (1 - \boldsymbol{x}_i) \left(1 - \frac{1}{2} \left((\boldsymbol{\mathcal{S}}_j)_i + (\boldsymbol{\mathcal{S}}_k)_i \right) \right)$$

To prove this property, we observe that the probabilities that $\boldsymbol{o}$ has a 1–symbol or a 0–symbol, respectively, at locus i, equal

$$\Pr(\boldsymbol{o}_i = 1 \mid \boldsymbol{p}^0 = \boldsymbol{\mathcal{S}}_j, \boldsymbol{p}^1 = \boldsymbol{\mathcal{S}}_k) \; = \; \frac{1}{2} \left((\boldsymbol{\mathcal{S}}_j)_i + (\boldsymbol{\mathcal{S}}_k)_i \right) \tag{4.2}$$
$$\Pr(\boldsymbol{o}_i = 0 \mid \boldsymbol{p}^0 = \boldsymbol{\mathcal{S}}_j, \boldsymbol{p}^1 = \boldsymbol{\mathcal{S}}_k) \; = \; 1 - \frac{1}{2} \left((\boldsymbol{\mathcal{S}}_j)_i + (\boldsymbol{\mathcal{S}}_k)_i \right)$$

Since $\boldsymbol{o}_i$ can only take on either a 0– or a 1–symbol, the equations can be combined to obtain a single expression for $\Pr(\boldsymbol{o}_i = \boldsymbol{x}_i \mid \boldsymbol{p}^0 = \boldsymbol{\mathcal{S}}_j, \boldsymbol{p}^1 = \boldsymbol{\mathcal{S}}_k)$. To do so, we note that we can isolate the term representing the case in which $\boldsymbol{x}_i = 1$ by multiplying it with $\boldsymbol{x}_i$ and the term representing the case in which $\boldsymbol{x}_i = 0$ by multiplying it with $1 - \boldsymbol{x}_i$:

$$\Pr(\boldsymbol{o}_i = \boldsymbol{x}_i \mid \boldsymbol{p}^0 = \boldsymbol{\mathcal{S}}_j, \boldsymbol{p}^1 = \boldsymbol{\mathcal{S}}_k) \; = \tag{4.3}$$
$$\frac{1}{2}\boldsymbol{x}_i \left((\boldsymbol{\mathcal{S}}_j)_i + (\boldsymbol{\mathcal{S}}_k)_i \right) + (1 - \boldsymbol{x}_i) \left(1 - \frac{1}{2} \left((\boldsymbol{\mathcal{S}}_j)_i + (\boldsymbol{\mathcal{S}}_k)_i \right) \right)$$

Since for each locus, the allele to be inherited by the offspring is selected independently from the other loci, the probability that the complete offspring genotype $\boldsymbol{o}$ equals the vector $\boldsymbol{x}$ is the product of the probabilities for the individual loci, which gives the result of equation 4.1.

Using equation 4.1 we can now characterize the uniform crossover probability distribution $\hat{P}^{UX}$. The distribution equals

$$\hat{P}^{UX}(\boldsymbol{\mathcal{X}})(\boldsymbol{x}) = \tag{4.4}$$
$$\sum_{j=0}^{|\boldsymbol{\mathcal{S}}|-1}\sum_{k=0}^{|\boldsymbol{\mathcal{S}}|-1}\frac{1}{|\boldsymbol{\mathcal{S}}|^2}\prod_{i=0}^{l-1}\frac{1}{2}\boldsymbol{x}_i\left((\boldsymbol{\mathcal{S}}_j)_i+(\boldsymbol{\mathcal{S}}_k)_i\right)+(1-\boldsymbol{x}_i)\left(1-\frac{1}{2}\left((\boldsymbol{\mathcal{S}}_j)_i+(\boldsymbol{\mathcal{S}}_k)_i\right)\right)$$

To prove this property, we begin by observing that, since we have allowed the mating phase in the simple GA in Chapter 3 to pick the same solution twice to serve as a parent for recombination, the probability of any pair of parents to be selected for uniform crossover equals

$$\Pr(\boldsymbol{p}^0=\boldsymbol{\mathcal{S}}_j,\boldsymbol{p}^1=\boldsymbol{\mathcal{S}}_k)=\frac{1}{|\boldsymbol{\mathcal{S}}|^2} \tag{4.5}$$

To obtain the uniform crossover probability distribution $\hat{P}^{UX}(\boldsymbol{\mathcal{X}})$, we must sum over all possible combinations of parents using the total law of probability.

$$\hat{P}^{UX}(\boldsymbol{\mathcal{X}})(\boldsymbol{x}) = \tag{4.6}$$
$$\sum_{j=0}^{|\boldsymbol{\mathcal{S}}|-1}\sum_{k=0}^{|\boldsymbol{\mathcal{S}}|-1}\Pr(\boldsymbol{o}=\boldsymbol{x}\mid\boldsymbol{p}^0=\boldsymbol{\mathcal{S}}_j,\boldsymbol{p}^1=\boldsymbol{\mathcal{S}}_k)\Pr(\boldsymbol{p}^0=\boldsymbol{\mathcal{S}}_j,\boldsymbol{p}^1=\boldsymbol{\mathcal{S}}_k)$$

Substituting equations 4.1 and 4.5 into equation 4.6 now gives the required result.

Perfect crossover probability distribution

In section 3.2.4 we described not just the uniform crossover operator, but also the perfect crossover operator for additively decomposable optimization problems. We recall that, with this crossover operator, the alleles at the loci defined for each index vector in the index cluster vector are copied simultaneously from a single parent to the offspring genotype. By doing so, the partial solution information is mixed in the most efficient manner. Since the donor parent is selected independently for each index vector, this perfect crossover operator is quite similar to uniform crossover in its behavior. Therefore, we can use a derivation similar to the one for the uniform crossover probability distribution to characterize the perfect crossover probability distribution $\hat{P}^{PX}(\boldsymbol{\mathcal{X}})$. For an additively decomposable optimization problem with index cluster vector $\boldsymbol{\iota}$, the perfect crossover probability distribution equals

$$\hat{P}^{PX}(\boldsymbol{\mathcal{X}})(\boldsymbol{x}) = \tag{4.7}$$
$$\sum_{j=0}^{|\boldsymbol{\mathcal{S}}|-1}\sum_{k=0}^{|\boldsymbol{\mathcal{S}}|-1}\frac{1}{|\boldsymbol{\mathcal{S}}|^2}\prod_{i=0}^{|\boldsymbol{\iota}|-1}\frac{1}{2}\left(\varphi((\boldsymbol{\mathcal{S}}_j)_{\boldsymbol{\iota}_i},\boldsymbol{x}_{\boldsymbol{\iota}_i})+\varphi((\boldsymbol{\mathcal{S}}_k)_{\boldsymbol{\iota}_i},\boldsymbol{x}_{\boldsymbol{\iota}_i})\right)$$
$$\text{where } \varphi(\boldsymbol{x},\boldsymbol{y})=\begin{cases}1 & \text{if } \boldsymbol{x}=\boldsymbol{y}\\ 0 & \text{otherwise}\end{cases}$$

4.1.2 Factorized probability distributions

Having characterized the uniform crossover probability distribution and the perfect crossover probability distribution, we define in this section the univariately factorized probability distribution and the perfectly factorized probability distribution. We note that upon using these distributions in a GA, they are estimated from the vector of selected solutions. We shall indicate the univariately factorized probability distribution by $P^{UF}(\boldsymbol{\mathcal{X}})$ and the perfectly factorized probability distribution by $P^{PF}(\boldsymbol{\mathcal{X}})$. If the univariately factorized probability distribution is used to generate an offspring genotype, the probability that the offspring receives a given symbol at a specific locus is independent of the probability of other symbols appearing at other loci. Therefore, the univariately factorized probability distribution is a product of the probability distributions for each random variable separately. More formally, for a vector $\boldsymbol{\mathcal{X}}$ of l random variables, the univariately factorized probability distribution, or univariate factorization for short, is defined by

$$P^{UF}(\boldsymbol{\mathcal{X}})(\boldsymbol{x}) = \prod_{i=0}^{l-1} P(X_i)(\boldsymbol{x}_i) \tag{4.8}$$

The factors $P(X_i)(\boldsymbol{x}_i)$ are generalized probability density functions (gpdfs) that will be estimated from the vector of selected solutions $\boldsymbol{\mathcal{S}}$ by means of the maximum likelihood principle:

$$\hat{P}(X_i)(\boldsymbol{x}_i) = \frac{1}{|\boldsymbol{\mathcal{S}}|} \sum_{j=0}^{|\boldsymbol{\mathcal{S}}|-1} \begin{cases} 1 & \text{if } (\boldsymbol{\mathcal{S}}_j)_i = \boldsymbol{x}_i \\ 0 & \text{otherwise} \end{cases} = \sum_{j|(\boldsymbol{\mathcal{S}}_j)_i = \boldsymbol{x}_i} \frac{1}{|\boldsymbol{\mathcal{S}}|} \tag{4.9}$$

The perfect factorization, in turn, is a product of probability distributions for the loci indicated by each index vector separately. More formally, for a vector $\boldsymbol{\mathcal{X}}$ of l random variables, the perfectly factorized probability distribution, or perfect factorization for short, for an additively decomposable optimization problem with index cluster vector $\boldsymbol{\iota}$ is defined by

$$P^{PF}(\boldsymbol{\mathcal{X}})(\boldsymbol{x}) = \prod_{i=0}^{|\boldsymbol{\iota}|-1} P(X_{\boldsymbol{\iota}_i})(\boldsymbol{x}_{\boldsymbol{\iota}_i}) \tag{4.10}$$

The gpdfs $P(X_{\boldsymbol{\iota}_i})(\boldsymbol{x}_{\boldsymbol{\iota}_i})$ are again estimated by means of the maximum likelihood principle:

$$\hat{P}(X_{\boldsymbol{\iota}_i})(\boldsymbol{x}_{\boldsymbol{\iota}_i}) = \frac{1}{|\boldsymbol{\mathcal{S}}|} \sum_{j=0}^{|\boldsymbol{\mathcal{S}}|-1} \begin{cases} 1 & \text{if } (\boldsymbol{\mathcal{S}}_j)_{\boldsymbol{\iota}_i} = \boldsymbol{x}_{\boldsymbol{\iota}_i} \\ 0 & \text{otherwise} \end{cases} = \sum_{j|(\boldsymbol{\mathcal{S}}_j)_{\boldsymbol{\iota}_i} = \boldsymbol{x}_{\boldsymbol{\iota}_i}} \frac{1}{|\boldsymbol{\mathcal{S}}|} \tag{4.11}$$

4.1.3 Comparing crossover with factorizations

In this section, we compare the crossover probability distributions derived in section 4.1.1 with the factorized probability distributions defined in section 4.1.2, under the assumption that the optimization problem under study is additively decomposable. To do so, we investigate the differences $\hat{P}^{UX}(\boldsymbol{\mathcal{X}}) - \hat{P}^{UF}(\boldsymbol{\mathcal{X}})$ and $\hat{P}^{PX}(\boldsymbol{\mathcal{X}}) - \hat{P}^{PF}(\boldsymbol{\mathcal{X}})$. An exact analysis of these differences is nearly impossible due to the complex form of the probability distributions for uniform crossover and perfect crossover. We therefore first discuss some example cases of the two differences. Subsequently, we present the results from a number of experiments to empirically support our observations. We now first turn our attention to uniform crossover and the univariate factorization. Based on the results found, the difference between perfect crossover and the perfect factorization is straightforward to predict.

Comparing uniform crossover with the univariate factorization

To investigate the difference $\hat{P}^{UX}(\boldsymbol{\mathcal{X}}) - \hat{P}^{UF}(\boldsymbol{\mathcal{X}})$, we begin by characterizing the uniform crossover probability distribution for a selection of all random variables $X_{\boldsymbol{j}} \sqsubseteq \boldsymbol{\mathcal{X}}$, that is obtained by marginalizing $\hat{P}^{UX}(\boldsymbol{\mathcal{X}})$. The probability distribution over $X_{\boldsymbol{j}}$ that results after applying uniform crossover equals

$$\hat{P}^{UX}(X_{\boldsymbol{j}})(\boldsymbol{x}) = \tag{4.12}$$

$$\sum_{q=0}^{|\boldsymbol{\mathcal{S}}|-1} \sum_{k=0}^{|\boldsymbol{\mathcal{S}}|-1} \frac{1}{|\boldsymbol{\mathcal{S}}|^2} \prod_{i=0}^{|\boldsymbol{j}|-1} \frac{1}{2} \boldsymbol{x}_i \left((\boldsymbol{\mathcal{S}}_q)_{\boldsymbol{j}_i} + (\boldsymbol{\mathcal{S}}_k)_{\boldsymbol{j}_i} \right) + (1 - \boldsymbol{x}_i) \left(1 - \frac{1}{2} \left((\boldsymbol{\mathcal{S}}_q)_{\boldsymbol{j}_i} + (\boldsymbol{\mathcal{S}}_k)_{\boldsymbol{j}_i} \right) \right)$$

Using this probability distribution, we can now investigate the difference between the uniform crossover probability distribution and the univariate factorization for the probability of a single variable X_i having a 1–symbol. The difference between the uniform crossover probability distribution and the univariate factorization for $X_i = 1$ equals

$$\hat{P}^{UX}(X_i)(1) - \hat{P}^{UF}(X_i)(1) = 0 \tag{4.13}$$

To prove this property, we observe that

$$\hat{P}^{UX}(X_i)(1) = \sum_{j=0}^{|\boldsymbol{\mathcal{S}}|-1} \sum_{k=0}^{|\boldsymbol{\mathcal{S}}|-1} \frac{1}{|\boldsymbol{\mathcal{S}}|^2} \frac{1}{2} \left((\boldsymbol{\mathcal{S}}_j)_i + (\boldsymbol{\mathcal{S}}_k)_i \right) = \tag{4.14}$$

$$\sum_{\substack{j|(\boldsymbol{\mathcal{S}}_j)_i = 0, \\ k|(\boldsymbol{\mathcal{S}}_k)_i = 0}} 0 + \sum_{\substack{j|(\boldsymbol{\mathcal{S}}_j)_i = 0, \\ k|(\boldsymbol{\mathcal{S}}_k)_i = 1}} \frac{1}{2|\boldsymbol{\mathcal{S}}|^2} + \sum_{\substack{j|(\boldsymbol{\mathcal{S}}_j)_i = 1, \\ k|(\boldsymbol{\mathcal{S}}_k)_i = 0}} \frac{1}{2|\boldsymbol{\mathcal{S}}|^2} + \sum_{\substack{j|(\boldsymbol{\mathcal{S}}_j)_i = 1, \\ k|(\boldsymbol{\mathcal{S}}_k)_i = 1}} \frac{1}{|\boldsymbol{\mathcal{S}}|^2}$$

If we now use equation 4.9, we find that:

$$\begin{aligned}\hat{P}^{UX}(X_i)(1) &= \hat{P}(X_i)(0)\hat{P}(X_i)(1) + \hat{P}(X_i)(1)^2 \\ &= (1-\hat{P}(X_i)(1))\hat{P}(X_i)(1) + \hat{P}(X_i)(1)^2 \\ &= \hat{P}(X_i)(1) \;=\; \hat{P}^{UF}(X_i)(1)\end{aligned} \tag{4.15}$$

From the property in equation 4.15 it may seem that the uniform crossover probability distribution actually equals a univariate factorization over $\boldsymbol{\mathcal{X}}$. However, the fact that $\hat{P}^{UF}(X_i)(1) = \hat{P}(X_i)(1)$ for each locus i does not necessarily mean that a given vector of offspring genotypes in which the proportion of 1–symbols at locus i equals $\hat{P}(X_i)(1)$, is more likely to have been generated using the univariate factorization than using another probability distribution that also has a probability $\hat{P}(X_i)(1)$ of generating a 1–symbol at that particular locus. The following example serves to illustrate this observation.

> **Example 4.1.** For a vector of selected solutions with 50 solutions that contain only 0–symbols and 50 solutions that contain only 1–symbols, we have $\hat{P}^{UX}(X_i)(1) = 0.5 = \hat{P}(X_i)(1)$. If we have a crossover operator that simply makes a copy of one of the parents, it is trivial to see that the probability of a 1–symbol being generated at locus i in an offspring is also 0.5. However, the probability that the vector of offspring solutions consists again of 50 solutions that contain only 0–symbols and 50 solutions that contain only 1–symbols, is extremely small if the univariate factorization is used, whereas using the copy crossover operator, this probability is quite large.

To further study the similarity of the uniform crossover probability distribution and the univariate factorization, therefore, it is important to investigate this difference for two or more random variables jointly. To this end, we analyze the probability of two random variables having a 1–symbol. The difference between the uniform crossover probability distribution and the univariate factorization for $X_{i_0} = 1$ and $X_{i_1} = 1$ equals

$$\hat{P}^{UX}(X_{i_0}, X_{i_1})(1,1) - \hat{P}^{UF}(X_{i_0}, X_{i_1})(1,1) = \tag{4.16}$$
$$\frac{1}{2}\left(\hat{P}(X_{i_0}, X_{i_1})(0,0)\hat{P}(X_{i_0}, X_{i_1})(1,1) - \hat{P}(X_{i_0}, X_{i_1})(0,1)\hat{P}(X_{i_0}, X_{i_1})(1,0)\right)$$

To prove this property, we observe that

$$\hat{P}^{UX}(X_{i_0}, X_{i_1})(1,1) = \tag{4.17}$$
$$\sum_{j=0}^{|\boldsymbol{\mathcal{S}}|-1}\sum_{k=0}^{|\boldsymbol{\mathcal{S}}|-1}\frac{1}{|\boldsymbol{\mathcal{S}}|^2}\frac{1}{4}\Big((\boldsymbol{\mathcal{S}}_j)_{i_0}+(\boldsymbol{\mathcal{S}}_k)_{i_0}\Big)\Big((\boldsymbol{\mathcal{S}}_j)_{i_1}+(\boldsymbol{\mathcal{S}}_k)_{i_1}\Big) =$$
$$\sum_{\substack{j|(\boldsymbol{\mathcal{S}}_j)_{(i_0,i_1)}=(0,0),\\ k|(\boldsymbol{\mathcal{S}}_k)_{(i_0,i_1)}=(1,1)}}\frac{1}{4|\boldsymbol{\mathcal{S}}|^2} + \sum_{\substack{j|(\boldsymbol{\mathcal{S}}_j)_{(i_0,i_1)}=(0,1),\\ k|(\boldsymbol{\mathcal{S}}_k)_{(i_0,i_1)}=(1,0)}}\frac{1}{4|\boldsymbol{\mathcal{S}}|^2} + \sum_{\substack{j|(\boldsymbol{\mathcal{S}}_j)_{(i_0,i_1)}=(0,1),\\ k|(\boldsymbol{\mathcal{S}}_k)_{(i_0,i_1)}=(1,1)}}\frac{1}{2|\boldsymbol{\mathcal{S}}|^2}+$$

$$\sum_{\substack{j|(\boldsymbol{\mathcal{S}}_j)_{(i_0,i_1)}=(1,0),\\ k|(\boldsymbol{\mathcal{S}}_k)_{(i_0,i_1)}=(0,1)}} \frac{1}{4|\boldsymbol{\mathcal{S}}|^2} + \sum_{\substack{j|(\boldsymbol{\mathcal{S}}_j)_{(i_0,i_1)}=(1,0),\\ k|(\boldsymbol{\mathcal{S}}_k)_{(i_0,i_1)}=(1,1)}} \frac{1}{2|\boldsymbol{\mathcal{S}}|^2} + \sum_{\substack{j|(\boldsymbol{\mathcal{S}}_j)_{(i_0,i_1)}=(1,1),\\ k|(\boldsymbol{\mathcal{S}}_k)_{(i_0,i_1)}=(0,0)}} \frac{1}{4|\boldsymbol{\mathcal{S}}|^2} +$$

$$\sum_{\substack{j|(\boldsymbol{\mathcal{S}}_j)_{(i_0,i_1)}=(1,1),\\ k|(\boldsymbol{\mathcal{S}}_k)_{(i_0,i_1)}=(0,1)}} \frac{1}{2|\boldsymbol{\mathcal{S}}|^2} + \sum_{\substack{j|(\boldsymbol{\mathcal{S}}_j)_{(i_0,i_1)}=(1,1),\\ k|(\boldsymbol{\mathcal{S}}_k)_{(i_0,i_1)}=(1,0)}} \frac{1}{2|\boldsymbol{\mathcal{S}}|^2} + \sum_{\substack{j|(\boldsymbol{\mathcal{S}}_j)_{(i_0,i_1)}=(1,1),\\ k|(\boldsymbol{\mathcal{S}}_k)_{(i_0,i_1)}=(1,1)}} \frac{1}{|\boldsymbol{\mathcal{S}}|^2} =$$

$$\frac{1}{2}\hat{P}(X_{i_0},X_{i_1})(0,0)\hat{P}(X_{i_0},X_{i_1})(1,1) + \frac{1}{2}\hat{P}(X_{i_0},X_{i_1})(0,1)\hat{P}(X_{i_0},X_{i_1})(1,0) +$$
$$\hat{P}(X_{i_0},X_{i_1})(0,1)\hat{P}(X_{i_0},X_{i_1})(1,1) + \hat{P}(X_{i_0},X_{i_1})(1,0)\hat{P}(X_{i_0},X_{i_1})(1,1) +$$
$$\hat{P}(X_{i_0},X_{i_1})(1,1)^2$$

The probability of finding a 1–symbol at locus i_0 as well as at locus i_1 in a single offspring genotype that has been generated with the univariate factorization now equals $\hat{P}(X_{i_0})(1)\hat{P}(X_{i_1})(1)$, which can be rewritten as follows:

$$\hat{P}^{UF}(X_{i_0},X_{i_1})(1,1) = \hat{P}(X_{i_0})(1)\hat{P}(X_{i_1})(1) = \tag{4.18}$$
$$\Big(\hat{P}(X_{i_0},X_{i_1})(1,0) + \hat{P}(X_{i_0},X_{i_1})(1,1)\Big)\Big(\hat{P}(X_{i_0},X_{i_1})(0,1) + \hat{P}(X_{i_0},X_{i_1})(1,1)\Big) =$$
$$\hat{P}(X_{i_0},X_{i_1})(1,0)\hat{P}(X_{i_0},X_{i_1})(0,1) + \hat{P}(X_{i_0},X_{i_1})(1,0)\hat{P}(X_{i_0},X_{i_1})(1,1) +$$
$$\hat{P}(X_{i_0},X_{i_1})(1,1)\hat{P}(X_{i_0},X_{i_1})(0,1) + \hat{P}(X_{i_0},X_{i_1})(1,1)^2$$

By canceling terms, the result now follows immediately. Since the difference $\hat{P}^{UX}(X_{i_0},X_{i_1})(1,1) - \hat{P}^{UF}(X_{i_0},X_{i_1})(1,1)$ is not equal to zero in general, we have that the use of the uniform crossover operator is *not* the same as the use of the univariate factorization for generating offspring. The following example serves to give some additional intuition as to why and how the two probability distributions differ.

> **Example 4.2.** We consider a vector of selected solutions of length $l = 2$ with $|\boldsymbol{\mathcal{S}}| = 10$. Suppose that two of the solutions in $\boldsymbol{\mathcal{S}}$ equal $(1,1)$ and that the remaining solutions equal $(0,0)$. Then, if we would use the univariate factorization to generate new offspring solutions, the probability that a genotype equal to $(1,1)$ would be generated equals $\frac{2}{10}\frac{2}{10} = \frac{1}{25}$. On the other hand, if we would use uniform crossover, this probability equals $\frac{1}{2}\frac{8}{10}\frac{2}{10} + (\frac{2}{10})^2 = \frac{3}{25}$. The probability that the $(1,1)$ genotype survives the recombination phase therefore is larger if the uniform crossover operator is used than if the univariate factorization is used to generate the offspring.

The difference in behavior originates from the fact that with uniform crossover information is exchanged between only two parents whereas with the univariate factorization the information is derived from all selected solutions simultaneously. Multivariate dependency information is processed to at least some extent

by the uniform crossover operator because the probability that two given symbols are placed at two specific loci in an offspring genotype depends on which parents were picked.

In Chapter 3, we have already seen that the simple GA with uniform crossover scales up exponentially in its population–size requirements for additively decomposable deceptive problems. From the previous observations we now have that using the univariate factorization for generating offspring instead of uniform crossover will scale up even worse for these problems. The univariate factorization, on the other hand, is expected to be more efficient than uniform crossover for problems where the lengths of the building blocks are equal to 1. For these problems, the probability of a particular symbol appearing in an offspring genotype at a given locus is truly independent of the probability of some symbol appearing at any other locus in the same genotype. While the univariate factorization captures this independence, uniform crossover will show a slight tendency to incorporate multivariate dependencies between loci as a result of the exchange of information between just two parents.

To support our observations, we have studied the difference between the uniform crossover probability distribution and the univariately factorized probability distribution experimentally. The results of our experiments are shown in Figures 4.2 and 4.3. In Figure 4.2, the scale–up behavior of a GA with uniform crossover and a recombination probability of $p_r = 0.5$ is shown, on the additively decomposable needle function $\mathfrak{G}_{additive\_needle}(\mathbf{X})$. Also, the behavior obtained with the univariate factorization is shown. To allow for faithful comparison, in the experiment with the univariate factorization, the probability of recombination is simulated by drawing an offspring from the univariate factorization with a probability of 0.5; if the offspring should not be sampled, a copy is made of a randomly picked genotype from the vector of selected solutions. The minimal requirements averaged over 30 runs to find the optimum in all 30 runs are shown. In Figure 4.3 the results for the same algorithms are shown for the additively decomposable deceptive trap function $\mathfrak{G}_{additive\_deceptive}(\mathbf{X})$. The results obtained on both optimization problems support our statement that the use of the univariate factorization results in worse scale–up behavior than does the use of the uniform crossover operator. The use of the univariate factorization therefore scales up *at least* exponentially on additively decomposable optimization problems if the subfunctions are deceptive.

Comparing perfect crossover with the perfect factorization

In the above, we have investigated the difference between the uniform crossover probability distribution and the univariately factorized probability distribution. We now focus on the difference between using the perfect crossover operator and the perfectly factorized probability distribution for generating offspring. We recall that the perfect crossover operator is quite similar to the uniform crossover operator. The only difference is that while the uniform crossover operator exchanges single alleles, the perfect crossover operator exchanges blocks of alleles. For the difference between the perfect crossover probability distribution and

the perfect factorization, therefore, results can be proven that are similar to the results for the difference between the uniform crossover probability distribution and the univariate factorization. It is again to be expected that the perfect factorization is capable of slightly more efficient mixing than the perfect crossover operator, since the perfect crossover operator will show a tendency to incorporate multivariate dependencies *between* the blocks of alleles while these blocks are completely independent in the optimization problem. The following example serves to illustrate this observation.

> **Example 4.3.** We consider a vector of selected solutions of length 10, in which the first five bits belong to a deceptive trap function as do the second five bits. Suppose that one of the selected genotypes equals $(1,1,1,1,1,0,0,0,0,0)$, that another genotype equals $(0,0,0,0,0,1,1,1,1,1)$, and that the remaining genotypes are composed of just 0–symbols. We observe that the probability of the optimal combination of bits for any of the subfunctions surviving the recombination phase is the same for both the perfect crossover operator and the perfectly factorized probability distribution. The probability of an offspring containing the first optimal block of 1–symbols, in the case where the perfect crossover operator is used, can be computed by considering the situations in which $(1,1,1,1,1,0,0,0,0,0)$ is either the first or the second parent: we obtain a probability of $\frac{1}{10}\frac{1}{10}\cdot 1 + \frac{1}{10}\frac{9}{10}\frac{1}{2} + \frac{9}{10}\frac{1}{10}\frac{1}{2} = \frac{1}{10}$. This probability equals the probability that an offspring contains the first optimal block of 1–symbols if the perfect factorization is used. The table in Figure 4.1 summarizes the probabilities of the different combinations of bits being constructed by using the perfect crossover operator and using the perfect factorization. We observe that the probability of juxtaposing the optimal combinations of bits for both subfunctions is twice as large for the perfect factorization than for the perfect crossover operator.

The observed difference in behavior indicates that the perfect factorization can mix the optimal building blocks slightly faster than the perfect crossover operator. To support our observations, we have studied the difference between the perfect crossover probability distribution and the perfect factorization experimentally. The results of our experiments are shown in Figures 4.2 and 4.3. The figures show the scale–up behavior of the perfect crossover operator and the use of the perfect factorization on the additively decomposable needle–in–a–haystack problem and on the additively decomposable deceptive problem. For the perfect operators, we set the probability of recombination at $p_r = 1.0$. The results obtained indicate that the use of the perfect factorization results in a GA that has a better scale–up behavior than does the use of the perfect crossover operator. Since we know from Chapter 3 that the use of the perfect crossover operator leads to a scale–up behavior that is sublinear on additively decomposable optimization problems if the subfunctions are deceptive, the use of the perfect factorization will consequently scale up at most sublinearly.

Probability of generating	Recombination Operator	
	Perfect Crossover	Perfect Factorization
1111111111	$\frac{1}{10}\frac{1}{10}\frac{1}{2}\frac{1}{2} + \frac{1}{10}\frac{1}{10}\frac{1}{2}\frac{1}{2} = 0.005$	$\frac{1}{10}\frac{1}{10} = 0.01$
1111100000	$\frac{1}{10}\frac{1}{10} + \frac{1}{10}\frac{1}{10}\frac{1}{2}\frac{1}{2} + \frac{1}{10}\frac{8}{10}\frac{1}{2} +$ $\frac{1}{10}\frac{1}{10}\frac{1}{2}\frac{1}{2} + \frac{8}{10}\frac{1}{10}\frac{1}{2} = 0.095$	$\frac{1}{10}\frac{9}{10} = 0.09$
0000011111	$\frac{1}{10}\frac{1}{10} + \frac{1}{10}\frac{1}{10}\frac{1}{2}\frac{1}{2} + \frac{1}{10}\frac{8}{10}\frac{1}{2} +$ $\frac{1}{10}\frac{1}{10}\frac{1}{2}\frac{1}{2} + \frac{8}{10}\frac{1}{10}\frac{1}{2} = 0.095$	$\frac{1}{10}\frac{9}{10} = 0.09$
0000000000	$\frac{1}{10}\frac{1}{10}\frac{1}{2}\frac{1}{2} + \frac{1}{10}\frac{8}{10}\frac{1}{2} + \frac{1}{10}\frac{1}{10}\frac{1}{2}\frac{1}{2} +$ $\frac{1}{10}\frac{8}{10}\frac{1}{2} + \frac{8}{10}\frac{1}{10}\frac{1}{2} + \frac{8}{10}\frac{1}{10}\frac{1}{2} +$ $\frac{8}{10}\frac{8}{10} = 0.805$	$\frac{9}{10}\frac{9}{10} = 0.81$

Figure 4.1: Probabilities of different combinations of bits being constructed by using the perfect crossover operator and using the perfect factorization for the additively decomposable deceptive trap function with subfunctions of length 5. For details on the vector of selected solutions, see Example 3.

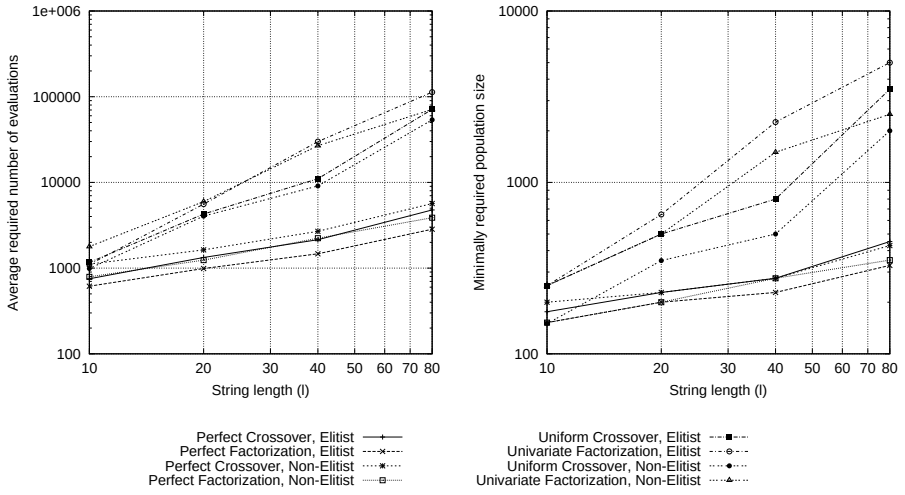


Figure 4.2: Scale–up behavior of various EAs on the additively decomposable needle function with subproblems of length 5. The results are averaged over 30 runs and indicate the minimal computational requirements to find the optimal solution in all 30 runs. The results are plotted on a log–log scale. Straight lines on this scale indicate polynomial scale–up behavior. (truncation selection [$\tau = 0.3$]; uniform crossover [$p_r = 0.5$], univariate factorization [$p_r = 0.5$], perfect crossover [$p_r = 1.0$], perfect factorization [$p_r = 1.0$]; no mutation; with elitism [*replace all non–selected solutions*], without elitism [*replace all solutions*]).

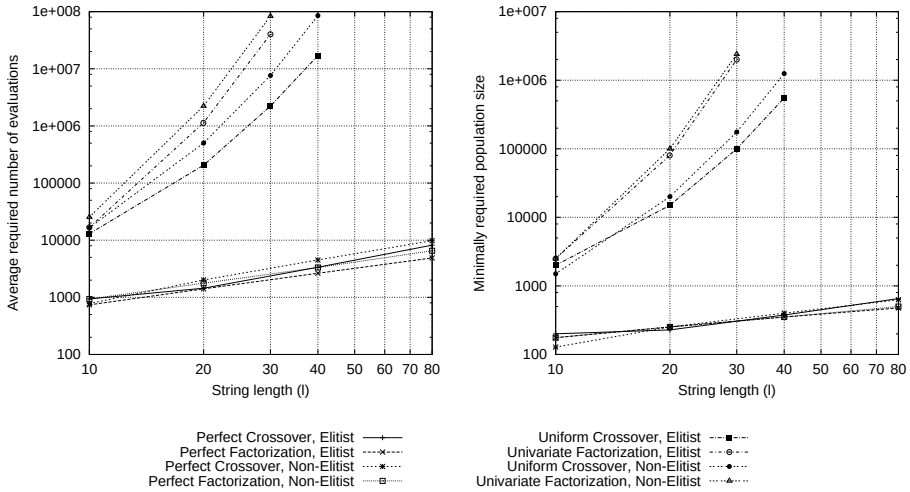


Figure 4.3: Scale–up behavior of various EAs on the additively decomposable deceptive trap function with subproblems of length 5. The same settings are used as for the results in Figure 4.2. The results are shown on a log–log scale.

4.1.4 From crossover linkage learning to factorization selection

In Chapter 3 we have argued that crossover linkage learning, that is, finding a good crossover operator, is required to prevent exponential scale–up behavior on additively decomposable optimization problems that can easily be solved in polynomial time if the decomposition is known. Probability distributions such as the perfect factorization for additively decomposable problems, are called *multivariately factorized probability distributions* or *multivariate factorizations* for short. The results from the previous section now indicate that the task of crossover linkage learning is closely related to the task of finding a good multivariate factorization. The latter task is commonly known as (multivariate) *factorization selection*. In fact, our results indicate that if a GA performs poorly at the task of factorization selection, it will have a bad scale–up behavior. If it does well at the task of factorization selection on the other hand, it may obtain a scale–up behavior that is better than can be achieved with a crossover operator.

Although the tasks of crossover linkage learning and factorization selection are closely related, they do not have the same search spaces. Only if we would restrict learning crossover linkage to finding crossover operators in which fixed sets of loci are transferred together from a single parent to the offspring, the search spaces of the two tasks would be equally large. The classical one–point crossover for instance does not have fixed sets of loci to be transferred together as these sets change for each new offspring genotype that is constructed. As a consequence, it cannot be described by a factorized probability distribution. On the other hand, factorization selection is only a specific case of the task commonly

known as *model selection*, which involves estimating probability distributions with different types of independency relation. In general, a probability distribution can represent many different types of independency relation, depending on the class it belongs to. In addition to multivariate factorizations, also more expressive probability distributions may be used. More complex probability distributions may be capable of modelling additional structural properties of optimization problems that can be exploited upon drawing new samples from a probability distribution that is estimated over the selected solutions.

From our observation that the use of probability distributions can indeed lead to efficient optimization by repeated estimation and sampling, we have a good motivation for replacing crossover linkage learning with learning and using probabilistic models. Surely, there are limitations to the approach of estimating probability distributions, since the tractability and scalability of estimating and sampling from probability distributions are bound to worsen as the complexity and expressiveness of the probability distribution used, increases. But there are classes of probability distributions that can be estimated from data efficiently just as there are many approaches to computing these estimates. The tractability of the resulting algorithms in terms of the running time as well as the number of required solutions to obtain good estimations for different classes of probability distributions, is the main topic of investigation in this thesis for different types of optimization problem.

4.2 The IDEA framework

In the previous section, we have argued that, instead of using a crossover operator, a probability distribution may be estimated over the vector of selected solutions from which new genotypes are drawn to perform induction. In this section, we present a general framework for evolutionary algorithms that are based upon this idea of using a probability distribution in the recombination phase. This framework is termed the IDEA framework of iterated density–estimation evolutionary algorithms. By instantiating some of its components, we arrive at the more specific framework of monotonic IDEAs that will be used to perform all experiments with throughout the remainder of the thesis.

4.2.1 Definition of IDEAs

To formalize the IDEA framework, we first introduce a random variable Z_i for each gene variable Z_i. We indicate the gene variables by Z_i now instead of X_i to distinguish them from the binary gene variables used in the previous sections. We write $\boldsymbol{\mathcal{Z}} = (Z_0, Z_1, \ldots, Z_{l-1})$, $l \geq 1$, for the vector of all the random variables for a problem under study. We further write $P^{\mathfrak{T}}(\boldsymbol{\mathcal{Z}})$ for a probability distribution over $\boldsymbol{\mathcal{Z}}$ that is uniform over the set of all genotypes $\mathbf{Z}$ with a fitness higher than a threshold $\mathfrak{T}$, that is $\mathfrak{G}(\mathbf{Z}) \succeq \mathfrak{T}$, and 0 otherwise. If we would know $P^{\mathfrak{T}^*}(\boldsymbol{\mathcal{Z}})$ for the optimal fitness value $\mathfrak{T}^*$, then drawing just a

single solution from this distribution would result in an optimal solution. Since we don't have access to $P^{\mathfrak{T}}(\boldsymbol{\mathcal{Z}})$ in general, we approximate this distribution.

If we have no prior information on a problem under study, we might as well generate a population $\boldsymbol{\mathcal{P}}$ of n random solutions initially, which represents the initial assumption of a uniform probability distribution over the search space. We then perform the following iterative inductive procedure. In each iteration t, we select a vector $\boldsymbol{\mathcal{S}}$ of solutions from the population; let $\mathfrak{T}$ be the fitness of the *worst* solution in $\boldsymbol{\mathcal{S}}$. Ideally, the vector $\boldsymbol{\mathcal{S}}$ of solutions is representative of the part of the search space that contains the solutions that have a fitness higher than $\mathfrak{T}$. At least, $\boldsymbol{\mathcal{S}}$ should be representative of the search space regions that we would like to further investigate. We now estimate a probability distribution over $\boldsymbol{\mathcal{S}}$ by learning a probabilistic model $\mathcal{M} = (\varsigma, \boldsymbol{\theta})$ from $\boldsymbol{\mathcal{S}}$. We recall that a probabilistic model $\mathcal{M}$ is a representation of a probability distribution that consists of a structure ς and an associated vector of parameters $\boldsymbol{\theta}$. Since we shall assume that the parameters for a given model structure are estimated in a fixed, predefined, way, we can uniquely characterize the model through its structure and denote the probability distribution that is represented by a specific model $\mathcal{M} = (\varsigma, \boldsymbol{\theta})$ by $P_\varsigma(\boldsymbol{\mathcal{Z}})$. By learning a probabilistic model $\mathcal{M}$ from $\boldsymbol{\mathcal{S}}$, we find a probability distribution $\hat{P}_\varsigma^{\mathfrak{T}}(\boldsymbol{\mathcal{Z}})$ that can be looked upon as an approximation of the true distribution $P^{\mathfrak{T}}(\boldsymbol{\mathcal{Z}})$. To further explore the part of the search space that we are interested in, we now use this distribution $\hat{P}_\varsigma^{\mathfrak{T}}(\boldsymbol{\mathcal{Z}})$ to draw new solutions from. To provide for iterated search, some of the new solutions replace some of the currently available solutions in $\boldsymbol{\mathcal{P}}$.

An *Iterated Density–Estimation Evolutionary Algorithm* (IDEA) now is an algorithm that maintains a collection of solutions $\boldsymbol{\mathcal{P}}$ and operates by iteratively repeating the following until a predefined termination criterion is met:

1. Select a collection of solutions $\boldsymbol{\mathcal{S}}$ from $\boldsymbol{\mathcal{P}}$
2. Learn a probabilistic model $\mathcal{M}$ from $\boldsymbol{\mathcal{S}}$
3. Draw a collection of new solutions $\boldsymbol{\mathcal{O}}$ from the estimated probability distribution $\hat{P}_\varsigma(\boldsymbol{\mathcal{Z}})$
4. Possibly mutate the new solutions in $\boldsymbol{\mathcal{O}}$
5. Replace some of the solutions in $\boldsymbol{\mathcal{P}}$ by solutions from $\boldsymbol{\mathcal{O}}$

Mühlenbein, Mahnig and Ochoa (1999) also proposed the use of iterated estimation of probability distributions within the framework of evolutionary algorithms, which they named *Estimation–of–Distribution Algorithms* (EDA). However, their EDA framework is slightly less general than the IDEA framework in the sense that the replacement phase is fixed to completely replace the current population with the new solutions. Algorithms that fit the IDEA description were introduced by different researchers using different acronyms, usually indicating a specific implementation for a specific type of random variable or probability distribution. These algorithms will be briefly reviewed in Section 4.3.

The evolutionary aspect of IDEAs is that a population of genotypes is used from which parent genotypes are selected to generate new offspring genotypes. Using these offspring along with the parents and the rest of the current population, a new population is constructed. By referring to the iterations in IDEAs as generations, the evolutionary aspect is even more evident. The main difference with most EAs is that the recombination operator is a function of all selected solutions rather than of just two solutions. Furthermore, the recombination operator generates all offspring at once, whereas in most EAs recombination is applied repeatedly. Within IDEAs, a probability p_r of applying recombination can be used by drawing a random real number from $[0; 1]$ for each new offspring genotype. If the number is smaller than p_r, then the offspring genotype is constructed by drawing a new sample from the estimated probability distribution. Otherwise, a genotype is picked at random from $\boldsymbol{\mathcal{S}}$ to be copied and serve as the new offspring genotype. To conclude, we would like to note that, although mutation can be applied as a separate operator in IDEAs, we do not use it in our research on the applicability of learning probabilistic models in EAs.

4.2.2 Monotonic IDEAs

An important issue in IDEAs is the discarding of solutions from the current generation in obtaining the solutions for the next generation. Since the probability distribution $\hat{P}_{\varsigma}^{\mathfrak{T}}(\boldsymbol{\mathcal{Z}})$ used to generate the new genotypes is merely an *approximation* of the true distribution $P^{\mathfrak{T}}(\boldsymbol{\mathcal{Z}})$ over all genotypes having a fitness higher than $\mathfrak{T}$, drawing new solutions from this distribution does not guarantee that these solutions will indeed have a fitness higher than $\mathfrak{T}$. And, even if this were so, the newly drawn solutions may not be better than the best solutions of the current generation. From this point of view, it seems a waste to discard the currently best–known solutions if they are not improved upon.

An EA that explicitly includes one or more of the better solutions from the population of the current generation in the population of the next generation, is said to be *elitist*. In EAs, the use of elitism has proven to be advantageous, in the sense of requiring fewer evaluations to find an optimal solution or obtaining better solutions in the same number of evaluations, if the recombination operator used is capable of effectively exploiting the structure of the problem (Thierens, 1995). However, if the optimization problem is very hard or if the recombination operator is not good enough at exploiting the structure of the problem, elitism will drive diversity out of the population very fast and enforce premature convergence of the EA to solutions of a lesser quality (de Jong, 1975).

Elitism can be used within the IDEA framework, yielding *monotonic* IDEAs. Let $\tau \in [\frac{1}{n}; 1)$. A monotonic IDEA performs the following specific operations within the general procedure outlined before:

1. Take the best $\lfloor \tau n \rfloor$ solutions from $\boldsymbol{\mathcal{P}}$ (truncation selection)
2. Draw a collection of $n - \lfloor \tau n \rfloor$ new solutions from the estimated probability distribution $\hat{P}_{\varsigma}(\boldsymbol{\mathcal{Z}})$
3. Use these solutions to replace the worst $n - \lfloor \tau n \rfloor$ solutions in $\boldsymbol{\mathcal{P}}$

IDEA(n, τ, LEARNMODEL(), SAMPLE(), TERMINATE()) (*monotonic instance*)	
Initialize the population	1 $\boldsymbol{\mathcal{P}} \leftarrow ()$ 2 ***for*** $i \leftarrow 0$ ***to*** $n-1$ ***do*** 2.1 $\boldsymbol{\mathcal{P}} \leftarrow \boldsymbol{\mathcal{P}} \sqcup (\text{RANDOMGENOTYPE}())$ 2.2 $\mathit{fitness}[\boldsymbol{\mathcal{P}}_i] \leftarrow \mathfrak{G}(\boldsymbol{\mathcal{P}}_i)$
Initialize the iteration counter	3 $t \leftarrow 0$
Iterate until termination	4 ***while*** $\neg$TERMINATE() ***do***
Select by truncation	4.1 $\boldsymbol{\mathcal{S}} \leftarrow \text{TRUNCATIONSELECTION}(\boldsymbol{\mathcal{P}}, \tau)$
Learn a probabilistic model	4.2 $\mathcal{M} \leftarrow \text{LEARNMODEL}(\boldsymbol{\mathcal{S}})$
Keep selected solutions (elitism)	4.3 $\boldsymbol{\mathcal{P}} \leftarrow \boldsymbol{\mathcal{S}}$
Draw additional offspring solutions from $\hat{P}_\varsigma(\boldsymbol{\mathcal{Z}})$ and evaluate them	4.4 ***for*** $i \leftarrow \lfloor \tau n \rfloor$ ***to*** $n-1$ ***do*** 4.4.1 $\boldsymbol{\mathcal{P}} \leftarrow \boldsymbol{\mathcal{P}} \sqcup (\text{SAMPLE}(\hat{P}_\varsigma(\boldsymbol{\mathcal{Z}})))$ 4.4.2 $\mathit{fitness}[\boldsymbol{\mathcal{P}}_i] \leftarrow \mathfrak{G}(\boldsymbol{\mathcal{P}}_i)$
Update the iteration counter	4.5 $t \leftarrow t+1$
Store required iterations t_{end}	5 $t_{\text{end}} \leftarrow t$

Figure 4.4: Pseudo–code for the monotonic IDEA.

As a consequence of using truncation selection, the probability distribution $\hat{P}_\varsigma^{\mathfrak{T}}(\boldsymbol{\mathcal{Z}})$ is an estimation over *all* selected solutions that *all* have a fitness higher than $\mathfrak{T}$. The replacement strategy in monotonic IDEAs thus results in a monotonically increasing series $\mathfrak{T}_0 \preceq \mathfrak{T}_1 \preceq \ldots \preceq \mathfrak{T}_{t_{\text{end}}}$. Assuming that the use of probability distributions for recombination leads to efficient induction and therefore to effective exploitation of the structure of an optimization problem, the use of elitism will be beneficial. In the remainder of this thesis, therefore, all our experiments with IDEAs will be performed using monotonic instances.

In figures 4.2 and 4.3, the effects of using elitism in monotonic IDEAs are illustrated on both the additively decomposable needle function and the additively decomposable deceptive function. Comparing the results obtained with elitism with those obtained without elitism, we can see that elitism indeed is beneficial in the case of perfect recombination operators. Elitism is also beneficial if the subfunctions are deceptive and therefore hard to construct optimally without a good recombination operator. We see that the scale–up behavior of the uniform crossover operator and of the univariate factorization, of which we know that they aren't capable of useful induction, can worsen if elitism is introduced: for the needle–in–a–haystack optimization problem, elitism then leads to premature convergence.

For clarity and reference, we present the monotonic IDEA in pseudo–code in Figure 4.4. The function RANDOMGENOTYPE returns a random genotype. Since we have not yet elaborated on genotypic encoding, we leave its implementation undefined for now.

> **Example 4.4.** Figure 4.5 illustrates how numerical optimization is performed by a monotonic IDEA using a mixture of normal probability distributions. The optimization problem under study is minimization of a two–dimensional version of the Rosenbrock function. In two dimensions,

the Rosenbrock function has two real–valued parameters and returns a single real value. In the example, two real–valued gene variables Y_0 and Y_1 are used to encode solutions. The fitness function to be minimized is $\mathfrak{G}(\mathsf{Y}_0, \mathsf{Y}_1) = 100(\mathsf{Y}_1 - \mathsf{Y}_0^2)^2 + (1 - \mathsf{Y}_0)^2$, $\mathsf{Y}_i \in [-5.12; 5.12]$; a surface plot for the function is shown in Figure 4.5.

For each real–valued gene variable Y_i, a real–valued random variable Y_i is introduced. In each generation, a mixture of normal probability distributions is estimated over $\boldsymbol{\mathcal{Y}}$ for the vector of selected solutions $\boldsymbol{\mathcal{S}}$. This mixture is estimated by first partitioning the vector of selected solutions by means of the leader partitioning algorithm as discussed in section 2.7.1. We recall that this algorithm computes partitions by going over the solution vector exactly once. For each subsequent solution, it finds the first partition that has a leader that is closer to this solution than a given threshold distance d. If no such partition can be found, a new partition is created containing this solution for its leader. A two–dimensional joint normal probability distribution is now estimated for each partition $\boldsymbol{c}^i \sqsubseteq (0, 1, \ldots, |\boldsymbol{\mathcal{S}}| - 1)$, using maximum likelihood estimates for the mean and covariance parameters. The distributions $\hat{P}_{\boldsymbol{\varsigma}_i}(\boldsymbol{\mathcal{Y}})$ estimated for the various partitions are combined into the final mixture using $\hat{P}_{\boldsymbol{\varsigma}}(\boldsymbol{\mathcal{Y}}) = \sum_{i=0}^{k-1} \beta_i \hat{P}_{\boldsymbol{\varsigma}_i}(\boldsymbol{\mathcal{Y}})$, where k is the number of partitions that were constructed by the leader algorithm. The mixing coefficients β_i are based on the relative size of the i–th partition compared to the total number of selected solutions, that is, $\beta_i = \frac{|\boldsymbol{c}^i|}{|\boldsymbol{\mathcal{S}}|}$.

In Figure 4.5, snapshots are presented from four different generations. Each snapshot presents a contour plot of the Rosenbrock function in the domain $[-5.12; 5.12] \times [-5.12; 5.12]$, along with points indicating the locations of the selected solutions. Below each Rosenbrock contour plot, a density contour plot is presented for the estimated probability distribution, together with the partitioning of $\boldsymbol{\mathcal{S}}$. The axes of these latter plots are scaled for clarity. After 16 generations, the Rosenbrock function was minimized within an additive precision of $5 \cdot 10^{-7}$.

4.3 A literature overview of IDEA instances and related algorithms

To obtain an instance of the IDEA framework, the type of probability distribution to be used must be specified as well as the way to estimate such a distribution from a given vector of solutions. In this section, we review related work that has been reported in the literature. This related work concerns both EAs that fit the IDEA framework and EAs that use a slightly different approach, yet also build upon probability distributions to generate new solutions. The types of probability distribution that we investigate in this thesis are factorizations and combinations of factorizations. Therefore, we will only review related work that is also based on these types of distribution. We furthermore point out

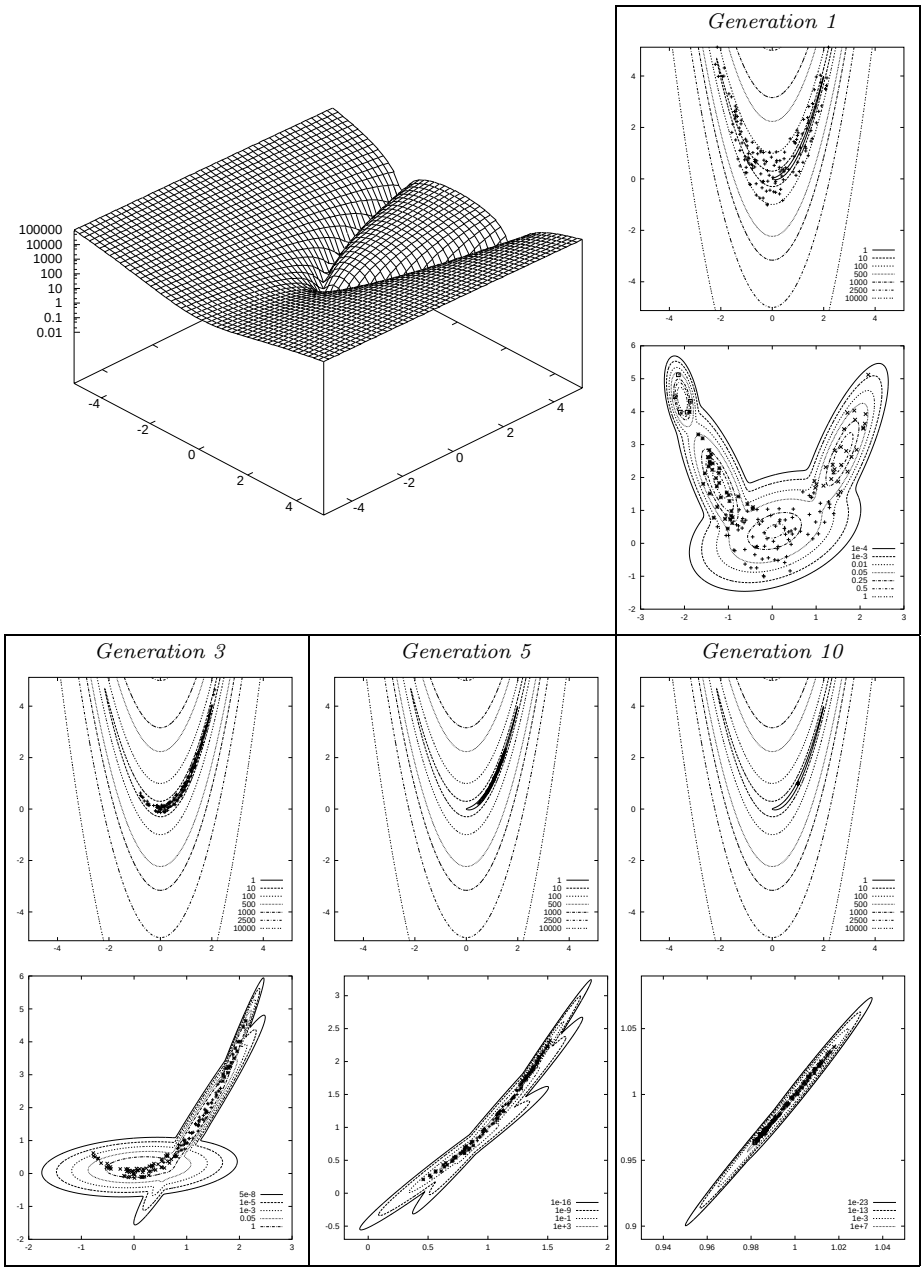


Figure 4.5: An example run of a monotonic IDEA minimizing the Rosenbrock function using a mixture of normal probability distributions. A Rosenbrock function surface plot is shown. Also, density contours are presented for the estimated probability distribution in different generations, along with Rosenbrock function contour plots showing the migration of the selected solutions towards the minimum.

that although approaches exist that use different representations of probability distributions (e.g. (Zhang and Shin, 2000; Shin, Cho, and Zhang, 2001)), the vast majority of the related literature concerns factorizations and combinations of factorizations. By reviewing the current state of the art of iterated density–estimation evolutionary algorithms and closely related algorithms, this section also serves to more clearly indicate the novelty of the research presented in the remainder of this thesis. In Section 4.3.1, we consider factorized probability distributions. In Section 4.3.2 we consider mixture probability distributions.

4.3.1 Factorized probability distributions

A *factorized probability distribution*, or *factorization* for short, is a probability distribution that can be written as a product of generalized probability density functions (gpdfs). Factorizations capture independencies between their random variables. Different types of factorization are capable of modelling different types of independency. Factorizations have the advantage that their maximum–likelihood estimates can be computed from the maximum–likelihood gpdf estimates for each factor. For discrete integer random variables and for real–valued random variables for which the normal gpdf is used, these maximum–likelihood gpdf estimates are well known (Anderson, 1958; Tatsuoka, 1971). Factorized probability distributions were the first to be used in evolutionary optimization. In the following, we first consider the univariate factorization. Subsequently, we discuss the use of multivariate factorizations and of Bayesian factorizations.

Univariate factorizations

A univariately factorized probability distribution equals a product of l one–dimensional gpdfs, one for each random variable. The distribution equals $\prod_{i=0}^{l-1} P_{\boldsymbol{\theta}^i}(Z_i)$, where $\boldsymbol{\theta}^i$ indicates the parameters for the gpdf $P_{\boldsymbol{\theta}^i}(Z_i)$ for the random variable Z_i. The univariate factorization does not model any dependencies between its variables since each gpdf is a function of just a single random variable.

Binary random variables

In the *Population–Based Incremental Learning* (PBIL) algorithm by Baluja and Caruana (1995), the genotype is a binary vector of length $l \geq 1$. A probability vector of the same length is maintained, where each value represents a factor of a univariate factorization. The i–th factor is the probability with which the i–th bit is set to 1 when constructing new solutions. The probability vector is updated each generation by selecting some of the best generated solutions and by updating each factor independently of the others. For each selected solution, updating the i–th factor amounts to multiplying the current factor i with $1 - \eta$ and adding the value at the i–th locus in the selected solution multiplied by η, where $\eta \in [0, 1]$ is a learning–rate parameter. The update rule of the probability vector serves to give PBIL a memory in which the impact of good solutions is not lost after a single generation. Similar to elitism, the use of the learning–rate parameter allows for slightly more efficient optimization of hard problems.

A probability vector is also at the core of the *compact Genetic Algorithm* (cGA) by Harik, Lobo and Goldberg (1998). The cGA, however, does not explicitly maintain a population of solutions. A population of size n is simulated by updating the probability vector after the generation of just two new solutions. The best of these new solutions is used to update the probability vector. If the best solution contains a 1–symbol at locus i, then the probability at the i–th position in the probability vector is increased by $\frac{1}{n}$ if it is currently smaller than 1; otherwise it is decreased by $\frac{1}{n}$ if it is currently larger than 0. The cGA can be looked upon as a memory–efficient implementation of an IDEA in which tournament selection is used in combination with the univariate factorization.

The *Univariate Marginal Distribution Algorithm* (UMDA) by Mühlenbein and Paaß (1998) is an instance of the EDA framework and is therefore also of the IDEA framework. A population of solutions is maintained and in each generation, a probability distribution is estimated from the selected solutions. The distribution used is the univariate factorization for binary random variables. From this factorization, n new solutions are drawn that completely replace the current population.

The algorithms discussed above all use a univariately factorized probability distribution over binary random variables. Since the genes of a solution are thus processed independently of each other, these algorithms are not capable of solving higher–order additively–decomposable non–linear optimization problems efficiently, as has been demonstrated in Section 4.1.

Real–valued random variables

The first application of evolutionary optimization driven by probabilistic model learning for real–valued random variables was an adaptation of the PBIL algorithm. The algorithm uses a normal gpdf for each of the l random variables (Rudlof and Köppen, 1996). To accommodate for these normal gpdfs, the probability vector from the original PBIL algorithm is replaced with a vector that specifies for each variable the mean and variance of the associated normal gpdf. The means are updated using a similar update rule as with the original binary PBIL. The variances are relatively large initially and are annealed to a small value using a geometrically decaying schedule. New solutions are generated by drawing values from the normal gpdfs for each variable separately.

The second application for real–valued variables was also an adaptation of the PBIL algorithm. In the algorithm by Servet, Trave-Massuyes and Stern (1997), a real–valued domain is explicitly stored for each variable. For each variable then, a histogram gpdf with two bins is maintained, where the first bin corresponds with the first half of the domain and the second bin corresponds with the second half. The probability vector from the original PBIL algorithm now specifies for each variable the probability with which a new value for that variable is generated in the second half of the domain currently stored for that variable. A domain is resized to contain exactly one of the two halves of that domain if the histogram starts to converge to that half of that domain.

A third adaptation of the original PBIL algorithm to real–valued random variables was introduced by Sebag and Ducoulombier (1998). Similar to the

approach by Rudlof and Köppen (1996), they proposed to use a normal gpdf for each variable. However, the variance is now updated using the same update rule as for the mean.

Gallagher, Fream and Downs (1999) were the first to propose for real–valued random variables, an algorithm in which a population of solutions is maintained and a probability distribution is estimated from a selection of solutions. Their algorithm is therefore more closely related to the IDEA framework than the PBIL adaptations mentioned before. In the algorithm by Gallagher et al. (1999), the adaptive mixture model by Priebe (1994) is used to estimate a normal mixture probability distribution for each variable separately. Another algorithm in which a population of solutions is maintained, was proposed by Tsutsui, Pelikan and Goldberg (2001). In their algorithm, the estimated probability distribution is a univariately factorized histogram distribution. Both the fixed–width as well as the fixed–height histogram variants were proposed.

The use of univariate factorizations for real–valued random variables was studied and compared against the use of more complex factorizations by various researchers (Bosman and Thierens, 2000a; Bosman and Thierens, 2000b; Larrañaga, Etxeberria, Lozano and Peña, 2000; Bosman and Thierens, 2001a); a more detailed description of the work by Bosman and Thierens is given in Chapter 5 of this thesis. In these studies, the use of univariately factorized normal probability distributions was shown to be inferior to the use of higher–order factorized normal probability distributions, for optimization problems containing linear interactions. This is to be expected since using higher–order factorizations allows for modelling dependencies that can be used to capture the interactions.

Multivariate factorizations

A multivariately factorized probability distribution (also called multivariate factorization, marginal factorization or marginal product factorization) is a product of multivariate joint gpdfs, with the gpdfs defined over mutually exclusive vectors of variables. Each random variable thus occurs in a single gpdf. The vector of variables of a gpdf is called a *node vector*, denoted by $\boldsymbol{\nu}_i$. We call the vector of all node vectors of a multivariate factorization the *node partition vector* and denote it by $\boldsymbol{\nu}$. The node partition vector in essence is the structure of a probabilistic model that describes the factorization. A multivariately factorized probability distribution is defined by $\prod_{i=0}^{|\boldsymbol{\nu}|-1} P_{\boldsymbol{\theta}^{\boldsymbol{\nu}_i}}(Z_{\boldsymbol{\nu}_i})$ where $\boldsymbol{\theta}^{\boldsymbol{\nu}_i}$ denotes the parameters for the multivariate joint gpdf $P_{\boldsymbol{\theta}^{\boldsymbol{\nu}_i}}(Z_{\boldsymbol{\nu}_i})$. The random variables indicated by a node vector are independent of all other random variables.

For efficient representation of factorized probability distributions, local structures were proposed by Friedman and Goldszmidt (1996). An example of a local structure is the *default table*. Recall that for discrete integer random variables, representing a multivariate joint probability distribution amounts to specifying a parameter estimate for each combination of values that can be assumed by the random variables associated with a single factor. In a default table, parameter estimates are stored for only a selection of all such combinations. For the combinations that are not explicitly stored in the table, a default value is assumed.

This default value is the average probability of all the non–present combinations, which results in a uniform probability distribution over the non–present combinations. Since often only few parameters will have to be estimated, which results in the penalization component of the model score to be lower, the main advantage of default tables is that more complex models containing larger multivariate factors may be dealt with when performing model selection.

Binary random variables

The use of multivariate factorizations in evolutionary optimization was first proposed for binary random variables by Harik (1999) (see also (Harik and Goldberg, 2000)). The *Extended Compact Genetic Algorithm* (ECGA) is an extension of the cGA that builds upon a univariately factorized probability distribution. In each generation, to estimate a multivariate factorization, the ECGA starts with a univariate factorization. The largest improvement in the minimum description length metric is then used to iteratively select and splice pairs of node vectors and increase the complexity of the multivariate factorization. The results obtained with the ECGA indicate a polynomial scale–up behavior of the minimally required population size and the required number of evaluations for additively–decomposable deceptive problems.

The use of multivariate factorizations has led to efficient scale–up behavior on additively–decomposable deceptive binary optimization problems. The main reason is that multivariate factorizations are well suited to capture the structure of an additively decomposable problem by placing the gene variables that contribute together to the fitness in a single node vector. The possible combinations of values for the variables per node vector are subsequently mixed efficiently as has been demonstrated in Section 4.1.

Real–valued random variables

The use of multivariate factorizations for real–valued random variables in evolutionary optimization has been limited to an application in permutation optimization in which permutation solutions are encoded with real values (Bosman and Thierens, 2001b). The results by Bosman and Thierens (2001b), which are discussed in more detail in chapter 6 of this thesis, indicate that the use of multivariate factorizations based on the normal gpdf only lead to polynomial scale–up behavior on additively–decomposable deceptive permutation optimization problems if a specialized crossover operator based on the estimated factorization is used to generate new solutions.

Permutation random variables

Multivariate factorizations for permutation random variables with frequency tables and with default tables have also been proposed for evolutionary optimization of permutation problems (Bosman and Thierens, 2001b; Bosman and Thierens, 2001d; Bosman and Thierens, 2002b); an in–depth description of the work by Bosman and Thierens is the topic of Chapter 6 of this thesis. Estimating a multivariately factorized probability distribution is slightly more involved for permutation random variables than for binary or real–valued ran-

dom variables. They can be estimated more efficiently, however, if in addition to the splicing of node vectors, node vectors are also allowed to exchange variables. Results obtained with both the AIC metric and the BIC metric indicate subquadratic scale–up behavior of the minimally required population size on additively–decomposable deceptive permutation problems.

Bayesian factorizations

A more general class of factorizations is the class of Bayesian factorized probability distributions or *Bayesian factorizations* for short. In essence, a Bayesian factorization equals a product of conditional gpdfs $P(Z_i|Z_{\boldsymbol{\pi}_i})$ for each random variable Z_i. We call the vector of random variables indicated by $\boldsymbol{\pi}_i$ on which Z_i is conditioned, the vector of *parents* of Z_i. The vector of all these vectors is called the *parent vector* of the Bayesian factorization, denoted by $\boldsymbol{\pi}$. The parent vector in essence is the structure of a probabilistic model that describes the Bayesian factorization. A Bayesian factorization is now defined by $\prod_{i=0}^{l-1} P_{\boldsymbol{\theta}^i}(Z_i|Z_{\boldsymbol{\pi}_i})$ where $\boldsymbol{\theta}^i$ indicates the parameters for the conditional gpdf $P_{\boldsymbol{\theta}^i}(Z_i|Z_{\boldsymbol{\pi}_i})$. By identifying a vertex with each variable Z_i and including an arc $Z_j \rightarrow Z_i$ if and only if Z_i is conditioned on Z_j ($j \in \boldsymbol{\pi}_i$), we get the Bayesian factorization *graph*. A Bayesian factorization is valid if and only if its Bayesian factorization graph is *acyclic*. Such representations of probability distributions are also called *graphical models* in literature (Lauritzen, 1996).

Local structures can also be used in Bayesian factorizations to improve the efficiency of representation. An example of a local structure for Bayesian factorizations is a *decision tree*. For each factor $P_{\boldsymbol{\theta}^i}(Z_i|Z_{\boldsymbol{\pi}_i})$ in the factorization, a tree is used to represent the parameter estimates. The tree has a unique root. Each internal node represents a variable $Z_{(\boldsymbol{\pi}_i)_j}$; it has as many children as there are values in the domain of $Z_{(\boldsymbol{\pi}_i)_j}$. Each leaf represents a parameter estimate, associated with all combinations of values for the parent variables $Z_{\boldsymbol{\pi}_i}$ that include the values along the path from the root to the leaf. An example decision tree for binary random variables is given in figure 4.6 on the left. A *decision graph* is an extension of a decision tree in which leaf nodes are allowed to have multiple parents. An example decision graph is given in figure 4.6 on the right.

Binary random variables

The first Bayesian factorizations used in evolutionary optimization were quite restricted. In the *Mutual Information Maximization Input Clustering* (MIMIC) algorithm by de Bonet, Isbell and Viola (1996), the Bayesian factorization used has a chain for its factorization graph. In a chain, each random variable has exactly one parent, with the exception of a single root variable, which has no parents; moreover, each random variable is the parent of exactly one other random variable, with the exception of a single leaf variable, which is a parent of no other random variable. The constraint that is thus placed on the Bayesian factorization is:

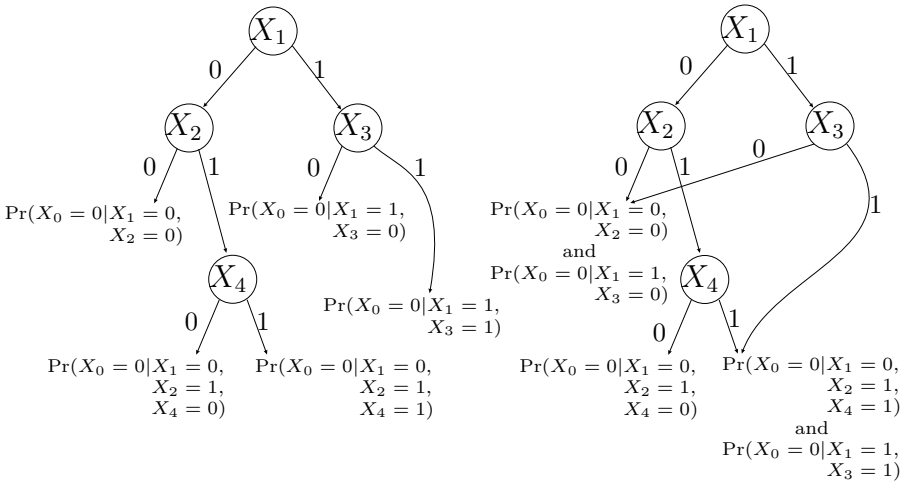


Figure 4.6: An example of a decision tree (left) and a decision graph (right) for random variable X_0. Both local structures represent $P_{\boldsymbol{\theta}^0}(X_0|X_1, X_2, X_3, X_4)$. The domain for each random variable is the binary domain $\mathbb{B}$.

$$\exists \boldsymbol{\omega} \in \mathit{perm}(\boldsymbol{\mathcal{L}}) : \boldsymbol{\pi}_{\boldsymbol{\omega}_{l-1}} = () \wedge \forall i \in \boldsymbol{\mathcal{L}} - (l-1) : \boldsymbol{\pi}_{\boldsymbol{\omega}_i} = (\boldsymbol{\omega}_{i+1}) \tag{4.19}$$

In MIMIC, the factorization is estimated from a vector of solutions by means of a greedy algorithm that minimizes the Kullback–Leibler divergence to the full multivariate joint probability distribution $P_{\boldsymbol{\theta}}(Z_{\boldsymbol{\mathcal{L}}})$. Minimizing this divergence amounts to minimizing the entropy of the estimated probability distribution. The greedy chain–learning algorithm used in MIMIC now first selects a variable with minimal univariate entropy. The next variable is selected by choosing one with minimal conditional entropy, *given* the previously selected variable. This process is repeated iteratively until all variables have been selected.

In the *Combining Optimizers with Mutual Information Trees* (COMIT) algorithm by Baluja and Davies (1997), the Bayesian factorization used has a tree for its factorization graph. In a tree, each variable but one is conditioned on exactly one other variable; many variables may have one and the same variable for their parent, however. The constraint that is thus placed on the Bayesian factorization is:

$$\exists \boldsymbol{\omega} \in \mathit{perm}(\boldsymbol{\mathcal{L}}) : \boldsymbol{\pi}_{\boldsymbol{\omega}_{l-1}} = () \wedge \forall i \in \boldsymbol{\mathcal{L}} - (l-1) : |\boldsymbol{\pi}_{\boldsymbol{\omega}_i}| = 1 \tag{4.20}$$

In the algorithm, the factorization is estimated by means of an algorithm by Chow and Liu (1968) that is guaranteed to yield the maximum–likelihood tree factorization. COMIT further uses a memory in a similar fashion as in PBIL to allow the impact of good solutions to span over multiple generations.

Pelikan and Mühlenbein (1999) proposed the *Bivariate Marginal Distribution Algorithm* (BMDA) in which the Bayesian factorization used has a *forest of trees*

for its factorization graph. In the forest, each variable has at most one parent. The constraint on the Bayesian factorization thus is:

$$\exists \boldsymbol{\omega} \in perm(\boldsymbol{\mathcal{L}}) : \boldsymbol{\pi}_{\boldsymbol{\omega}_{l-1}} = () \wedge \forall i \in \boldsymbol{\mathcal{L}} - (l-1) : |\boldsymbol{\pi}_{\boldsymbol{\omega}_i}| \leq 1 \tag{4.21}$$

In the BMDA, the factorization is constructed by means of χ^2 tests on the dependencies between pairs of variables.

All algorithms discussed so far use special–case Bayesian factorizations that share the property that in their factorization graph, each random variable may have just a single parent. Bayesian factorizations in general, are capable of modelling at least the same (in)dependency relations between their variables as multivariate factorizations, since each factor $P(Z_{\boldsymbol{\nu}_i})$ in a multivariate factorization can be written as:

$$P(Z_{\boldsymbol{\nu}_i}) = \prod_{j=0}^{|\boldsymbol{\nu}_i|-1} P\left(Z_{(\boldsymbol{\nu}_i)_j} \left| Z_{(\boldsymbol{\nu}_i)_{j+1}}, Z_{(\boldsymbol{\nu}_i)_{j+2}}, \ldots, Z_{(\boldsymbol{\nu}_i)_{|\boldsymbol{\nu}_i|-1}}\right.\right) \tag{4.22}$$

Using equation 4.22, therefore, a Bayesian factorization can be constructed that would result in the same good scale–up behavior on higher–order additively decomposable deceptive problems as when the corresponding multivariate factorization would be used. Bayesian factorizations, however, are also able to capture independencies that cannot be expressed by a multivariate factorization. An interesting question now is whether or not with a less complex Bayesian factorization as used in the algorithms reviewed so far, a good scale–up behavior can be attained on higher–order additively decomposable deceptive problems.

The results of an empirical study using Bayesian factorizations that have a tree for their factorization graph are presented in figures 4.7 and 4.8. Figure 4.7 reveals that the use of an optimal tree–structured Bayesian factorization in a monotonic IDEA results in a polynomial scale–up behavior on the additively–decomposable needle function. The scale–up behavior, however, is not as efficient as the one obtained with a perfect multivariate factorization in which the node vectors correspond with the index vectors. The difference between the two types of factorization becomes more prominent for the additively–decomposable deceptive function. Figure 4.8 reveals that the IDEA using the tree–structured Bayesian factorization in fact does not scale up polynomially. The reason for the relatively poor scale–up behavior is that the building blocks are not processed as a whole and as a result do not have a substantial chance to survive recombination. To achieve polynomial scale–up behavior, therefore, the restriction of allowing at most one parent for any variable in the factorization graph of a Bayesian factorization needs to be lifted.

Bayesian factorizations without any constraints on their factorization graphs were used, independently, in the *Bayesian Optimization Algorithm* (BOA) by Pelikan, Goldberg and Cantú-Paz (1999), in the *Learning Factorized Distribution Algorithm* (LFDA) by Mühlenbein and Mahnig (1999), and in the *Estimation of Bayesian Network Algorithm* (EBNA) by Etxeberria and Larrañaga (1999). In all three algorithms, the factorization is estimated by means of a greedy search algorithm that starts from a univariate factorization. The factorization graph of a univariate factorization has l nodes and no arcs. In each step,

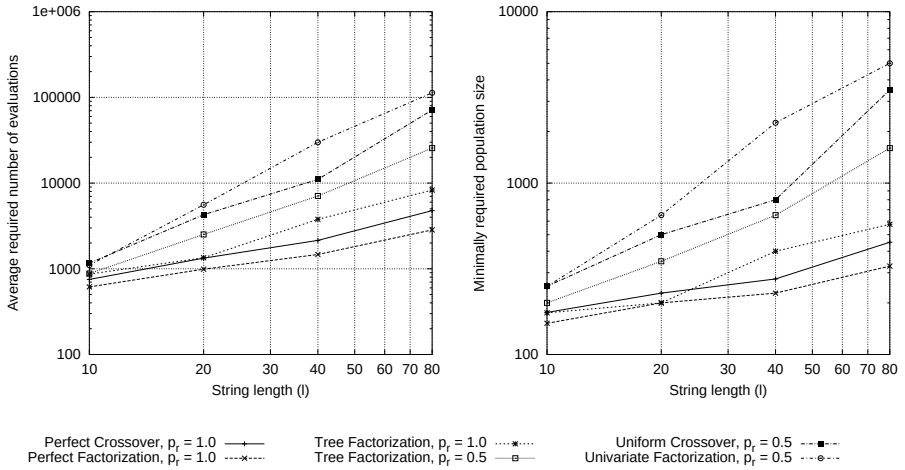


Figure 4.7: Scale–up behavior on the additively–decomposable needle function with subproblems of length 5. The results are averaged over 30 runs and indicate the minimal computational requirements to find the optimal solution in all 30 runs. The results are plotted on a log–log scale. (truncation selection [$\tau = 0.3$]; uniform crossover [$p_r = 0.5$], univariate factorization [$p_r = 0.5$], perfect factorization [$p_r = 1.0$], perfect crossover [$p_r = 1.0$], optimal dependency–tree [$p_r \in \{0.5, 1.0\}$]; no mutation; with elitism [*replace all non–selected solutions*]).

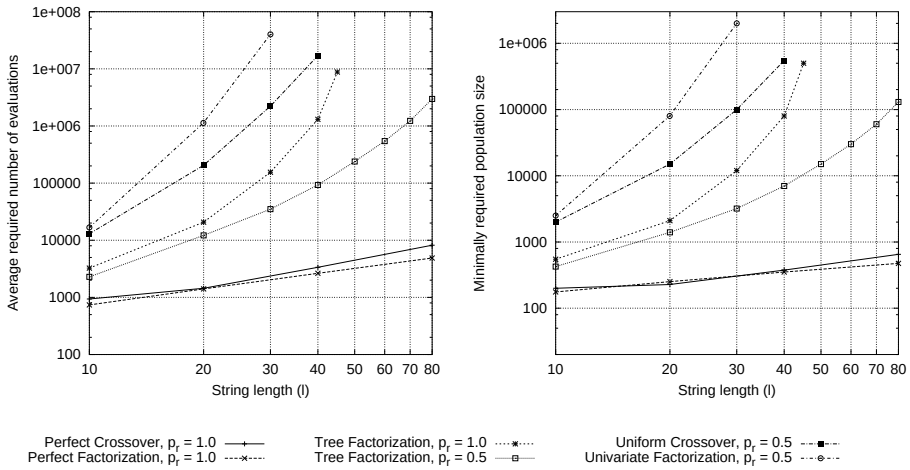


Figure 4.8: Scale–up behavior of IDEAs and GAs on the additively decomposable deceptive trap function with subproblems of length 5. The same IDEA and GA variants are used as for the results in Figure 4.7. The results are shown on a log–log scale.

the arc that improves the search metric the most, is added. In the first proposal of the BOA, the Bayesian–Dirichlet (Heckerman and Geiger, 1995) metric was used. Since the Bayesian–Dirichlet metric has properties similar to the entropy metric, in the sense that adding arcs will always improve the score of a factorization graph, an explicit limit on the number of parents per variable was used. In a later publication (Pelikan, Goldberg and Sastry, 2001), penalization metrics were proposed similar to the BIC metric. The BIC metric was also used in the LFDA and the EBNA. The use of an unrestricted Bayesian factorization combined with the BIC metric scales up subquadratically in the minimally required population size and required number of evaluations on additively decomposable optimization problems (Pelikan et al., 2001; Mühlenbein and Mahnig, 1999).

Unconstrained Bayesian factorizations represented by decision graphs were first used by Pelikan and Goldberg (2001), in the context of their BOA. The graphs were used in combination with niching. The term niching refers to ways of ensuring that only solutions that are similar will compete with each other. Niching was established in the BOA through restricted tournament selection. For each newly generated solution, a set of currently available solutions is picked randomly. The solution that is most similar, in terms of Hamming distance, to the new solution is replaced in the population if the fitness of the new solution is higher. The resulting variant of the BOA scales up subquadratically on very difficult *hierarchical* optimization problems. These problems are additively–decomposable and deceptive and have dependencies between combinations of bits for different variables which cannot be described efficiently using more straightforward representations than decision graphs because of the large number of variables that are dependent on each other.

To conclude, the polytree–structured Bayesian factorization has a polytree for its factorization graph, which is a directed graph having a tree for its underlying undirected graph. In a polytree–structured Bayesian factorization, the joint probability distribution of the parents of any node is univariately factorized:

$$\forall i \in \boldsymbol{\mathcal{L}} : P(X_{\boldsymbol{\pi}_i}) = \prod_{j=0}^{|\boldsymbol{\pi}_i|-1} P(X_{\pi_j^i}) \tag{4.23}$$

A polytree–structured factorization has a less complex factorization graph than an unrestricted Bayesian factorization may have. Yet, it is capable of capturing multivariate dependencies to at least some extent. As a result, polytree–structured factorizations can be learned from data more quickly than unrestricted Bayesian factorizations. Although experiments have been conducted with polytree–structured factorizations (Ochoa, Mühlenbein and Soto, 2000; Soto and Ochoa, 2000), it has not been shown as yet whether or not this class of Bayesian factorizations is capable of polynomial scale–up behavior on additively–decomposable deceptive optimization problems. The only algorithm using polytree–structured factorizations, which is called the *Polytree Approximation of Distribution Algorithm* (PADA)), however, has been shown to outperform algorithms using tree–structured factorizations (Ochoa et al., 2000).

Real–valued random variables

For real–valued random variables, Bayesian factorizations using normal gpdfs were proposed simultaneously by Bosman and Thierens (2000b) within the IDEA framework and by Larrañaga et al. (2000) in a variant of MIMIC that uses normal gpdfs, termed MIMIC_{C}, and in the *Estimation of Gaussian Network Algorithm* (EGNA). A detailed description of the work by Bosman and Thierens is given in Chapter 5. As a first approach, which is not covered in chapter 5 of this thesis, Bosman and Thierens (2000b) used an algorithm by Edmonds (1967) to find a Bayesian factorization of minimal entropy in which each variable has at most one parent. Also, the optimal dependency–tree algorithm used in COMIT and the greedy chain–learning algorithm used in MIMIC were used in the first publications on the IDEA framework (Bosman and Thierens, 2000a; Bosman and Thierens, 2000b). In a later publication (Bosman and Thierens, 2001a), the BIC metric was proposed in combination with a greedy factorization–learning algorithm. In the work by Larrañaga et al. (2000), finding a Bayesian factorization starts with a complete factorization graph. A likelihood–ratio test is then performed for each arc to determine whether or not that arc should be excluded from the graph. A greedy factorization–learning algorithm based on the BIC metric that starts from the univariate factorization was also used.

The notion of scale–up behavior is not well–defined for numerical optimization. If a fixed additive precision ε within which the problem should be optimized is used, some problems may become significantly easier or more difficult. Therefore, optimization performance is usually measured for different problems in different dimensionalities and not directly related to scale–up behavior. The results obtained with the IDEA framework with normal gpdfs and with EGNA indicate good optimization performance on problems with linear interactions and even on problems with many local optima, of both lower and higher dimensionality. However, the algorithms cannot efficiently solve optimization problems with non–linear interactions between their variables. The main reason is that the interactions that can be modelled using the normal gpdf are just linear.

4.3.2 Mixture probability distributions

In addition to factorized probability distributions, *mixture probability distributions* have been used in evolutionary optimization. A mixture probability distribution is a weighted sum of $k > 1$ probability distributions. Each probability distribution in the mixture probability distribution is called a *mixture component*. Let $\boldsymbol{\mathcal{K}} = (0, 1, \ldots, k-1)$. The probabilistic model associated with a mixture probability distribution is a vector $\boldsymbol{\varsigma}$ of simpler probabilistic model structures and a vector $\boldsymbol{\theta}$ of vectors of parameters:

$$P_{(\boldsymbol{\varsigma},\boldsymbol{\theta})}(\boldsymbol{\mathcal{Z}}) = \sum_{i=0}^{k-1} \beta_i P_{(\boldsymbol{\varsigma}_i,\boldsymbol{\theta}_i)}(\boldsymbol{\mathcal{Z}}) \tag{4.24}$$

where $\beta_i \geq 0$, $i \in \{0, 1, \ldots, k-1\}$, and $\sum_{i=0}^{k-1} \beta_i = 1$. The β_i with which the mixture components are weighted in the sum are called *mixing coefficients*.

They are part of the model parameters and constitute the $k+1$–st vector $\boldsymbol{\theta}_k = (\beta_0, \beta_1, \ldots, \beta_{k-1})$ within $\boldsymbol{\theta}$, $|\boldsymbol{\theta}| = k+1$. Using mixture probability distributions, a larger class of independency relations between the random variables can be expressed than when using factorized probability distributions. If different dependencies between random variables pertain to different subvectors of the solutions, each of these dependencies can be accounted for by a separate mixture component. Capturing these different (simpler) dependencies can be done by allowing each mixture component to be a factorization. By adding the k factorizations into the mixture probability distribution, the more complex dependency relation is modelled. By using mixture probability distributions, a powerful, yet computationally tractable type of probability distribution can be used within EAs, that provides for processing complicated non–linear interactions between a problem's variables.

Binary random variables

Multi–modal optimization can be performed by EAs using a mixture probability distribution as was demonstrated by Pelikan and Goldberg (2000). An adaptation of UMDA first performs clustering of the vector of selected solutions with the k–means clustering algorithm based on the Hamming distance. The number of clusters is user–defined. A univariate factorization is then estimated for each cluster. The i–th mixing coefficient is set proportional to the average fitness of the i–th cluster (assuming that we are maximizing a non–negative function). The resulting EA is shown to efficiently distribute the population over k local optima proportionally to the fitness of these local optima.

Meila and Jordan (1998) showed that learning a mixture of tree–structured factorizations using the EM algorithm leads to good probability–distribution estimations. This observation was exploited by Santana, Ochoa and Soto (2001) in an EA. Results obtained with this EA are shown to be better than results obtained with an EA using a single tree–structured factorization. However, the EA using a mixture of tree–structured factorizations does not show polynomial scale–up behavior on additively–decomposable deceptive problems. The main reason is that with the mixture of tree–structured factorizations no higher–order multivariate dependencies can be modelled.

Real–valued random variables

Mixture probability distributions for real–valued random variables were also used in evolutionary optimization. Results obtained by Bosman and Thierens (2001a) indicate that using mixture distributions can lead to more efficient optimization of real–valued problems with non–linear interactions. An in–depth discussion of these results can be found in Chapter 5 of this thesis. Similar to the approach by Pelikan and Goldberg (2000), clustering algorithms are used, after which a factorized normal probability distribution is estimated for each cluster. The number of clusters is again defined beforehand. An EM algorithm was also used to obtain a normal mixture probability distribution, but only results of using the univariate factorization for each mixture component were published. The number of mixture components in this case is also fixed beforehand. The

approach differs from the approach by Gallagher et al. (1999) reviewed above only in that a different method was used for estimating the normal mixture distribution. The results indicate however that problems with non–linear interactions cannot be solved efficiently using the univariate factorization for each mixture component.

Binary and real–valued random variables in multi–objective optimization
Mixture probability distributions have also been shown to be quite effective in evolutionary multi–objective optimization (Thierens and Bosman, 2001a; Thierens and Bosman, 2001b; Bosman and Thierens, 2002a). The work by Bosman and Thierens on multi–objective optimization is described in detail in Chapter 7 of this thesis. By clustering the vector of selected solutions based on the Euclidean distance in the objective space and by subsequently estimating a factorization for each cluster separately, the important tendency to explore and preserve diversity is introduced into a multi–objective EA. By combining the use of mixture probability distributions with a specialized diversity–preserving selection operator, good results are obtained with a proper spread of the solutions along the Pareto–front (Bosman and Thierens, 2002a).

'We like to test things... no matter how good an idea sounds, test it first.'
Henry Block

Numerical optimization

Most numerical optimization problems pertain to mathematical functions that are continuous and (partly) differentiable. Locally, the structure of these problems can be exploited by estimating the local gradient during optimization. The binary encoding of real–valued problem variables used in GAs introduce additional overhead in the representation. Using standard crossover operators, moreover, the GA is unaware that groups of binary variables represent a single real–valued problem variable and the GA is therefore less likely to efficiently exploit the structure of the problem. Real–valued ES are traditionally more successful in numerical optimization since they have a single real–valued random variable for each real–valued problem variable and build models for directions in which to move solutions to obtain better solutions, thus being able to exploit gradient features much more efficiently. From an IDEA point of view, it is therefore interesting to investigate whether the use of real–valued probability distributions can lead to efficient IDEAs for numerical optimization. This is exactly what we investigate in this chapter.

The chapter is organized as follows. In Section 5.1, we focus on the estimation of real–valued probability distributions. Both factorized probability distributions as well as mixtures of factorized probability distributions are discussed together with various gpdfs. In Section 5.2, we discuss the effect of integrating local optimization techniques based on gradient information in the IDEA framework by applying such techniques every iteration. We present results of an experimental study of a large variety of IDEA instances and discuss the performance of these IDEAs on a test suite of four well–known hard real–valued numerical optimization problems for three different dimensionalities. We compare the results obtained to a number of ESs and GAs as well as to a classical random restart conjugate gradient algorithm, and find that, on average, some real–valued IDEAs indeed tend to outperform the other tested algorithms.

5.1 Real–valued probability distributions

In order to apply IDEAs to real–valued numerical optimization, we must be able to estimate real–valued probability distributions from vectors of selected solutions and subsequently draw new solutions from them. In this section we present the probability distributions that we shall use in the IDEA framework in our experiments. In Section 5.1.1 we specify four real–valued generalized probability density functions and discuss their properties. These generalized probability density functions can be used to estimate real–valued factorized probability distributions. In Section 5.1.2, we discuss the estimation of real–valued mixture probability distributions by clustering.

5.1.1 Generalized probability density functions

We present the generalized probability density functions (gpdfs) that will be used in the sequel for the estimation of factorized probability distributions over real–valued random variables. For each gpdf, we give a definition. In Section 2.5 it was already noted that if the gpdfs in a factorization are estimated to be of maximum likelihood, then the factorization itself is also of maximum likelihood. Therefore, we define the gpdfs and their maximum likelihood estimations in terms of a selection of random variables $Z_{\boldsymbol{\nu}_i} \sqsubseteq \boldsymbol{\mathcal{Z}}$ so it is directly clear how these gpdfs can be used in a multivariate factorization. We shall do the same for the univariate conditional gpdfs required for Bayesian factorizations.

We further describe, for each gpdf, how it can be estimated for its use in the multivariate factorization as well as for its use in the Bayesian factorization. Subsequently, we discuss factorization selection for the specified gpdf and the indicated estimates. Next, we describe how new samples can be drawn from a factorization that was estimated using the gpdf. To conclude, we discuss some properties of the gpdf and visualize the estimated joint density for two random variables on two sample collections; the sample collections used for this purpose are shown in Figure 5.1.

Normal gpdf

Definition

A widely used generalized probability density function is the normal gpdf. It is a relatively simple, unimodal function. The normal gpdf $P^{\mathcal{N}}_{(\boldsymbol{\mu}_{\boldsymbol{\nu}_i}, \boldsymbol{\Sigma}^{\boldsymbol{\nu}_i})}$ for random variables $Y_{\boldsymbol{\nu}_i}$ is parameterized by a vector $\boldsymbol{\mu}_{\boldsymbol{\nu}_i}$ of means and a symmetric covariance matrix $\boldsymbol{\Sigma}^{\boldsymbol{\nu}_i}$ and is defined by

$$P^{\mathcal{N}}_{(\boldsymbol{\mu}_{\boldsymbol{\nu}_i}, \boldsymbol{\Sigma}^{\boldsymbol{\nu}_i})}(Y_{\boldsymbol{\nu}_i})(\boldsymbol{y}) = \frac{(2\pi)^{-\frac{|\boldsymbol{\nu}_i|}{2}}}{(\det \boldsymbol{\Sigma}^{\boldsymbol{\nu}_i})^{\frac{1}{2}}} e^{-\frac{1}{2}(\boldsymbol{y}-\boldsymbol{\mu}_{\boldsymbol{\nu}_i})^T (\boldsymbol{\Sigma}^{\boldsymbol{\nu}_i})^{-1} (\boldsymbol{y}-\boldsymbol{\mu}_{\boldsymbol{\nu}_i})} \tag{5.1}$$

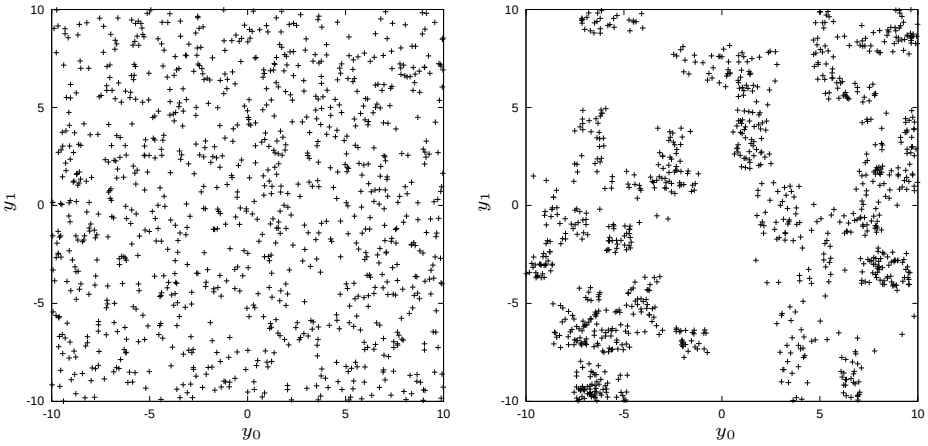


Figure 5.1: Two collections of samples. The collection on the left was obtained by drawing samples from a uniform distribution. The samples on the right were obtained by drawing samples from multiple clusters with uniform distributions.

Parameter estimation

A maximum likelihood estimation for the normal gpdf is obtained from a vector $\boldsymbol{\mathcal{S}}$ of samples if the parameters are estimated by the sample average and the sample covariance matrix (Anderson, 1958; Tatsuoka, 1971):

$$\hat{\boldsymbol{\mu}}_{\boldsymbol{\nu}_i} = \frac{1}{|\boldsymbol{\mathcal{S}}|} \sum_{j=0}^{|\boldsymbol{\mathcal{S}}|-1} (\boldsymbol{\mathcal{S}}_j)_{\boldsymbol{\nu}_i} \tag{5.2}$$

$$\hat{\boldsymbol{\Sigma}}^{\boldsymbol{\nu}_i} = \frac{1}{|\boldsymbol{\mathcal{S}}|} \sum_{j=0}^{|\boldsymbol{\mathcal{S}}|-1} ((\boldsymbol{\mathcal{S}}_j)_{\boldsymbol{\nu}_i} - \hat{\boldsymbol{\mu}}_{\boldsymbol{\nu}_i})((\boldsymbol{\mathcal{S}}_j)_{\boldsymbol{\nu}_i} - \hat{\boldsymbol{\mu}}_{\boldsymbol{\nu}_i})^T$$

To estimate a normal Bayesian factorization from a given vector of samples, we have to estimate the conditional gpdfs $P^{\mathcal{N}}(Y_i|Y_{\boldsymbol{\pi}_i})(y_{(i)\sqcup\boldsymbol{\pi}_i})$. Let $\boldsymbol{W^j}$ be the inverse of the symmetric covariance matrix, that is $\boldsymbol{W^j} = (\boldsymbol{\Sigma^j})^{-1}$. Matrix $\boldsymbol{W^j}$ is commonly called the *precision matrix*. In Appendix A, it is shown that for a maximum likelihood estimate of the required univariate conditional gpdf, maximum likelihood estimations as described in equation 5.2 need to be computed for the parameters $\boldsymbol{\mu}_{(i)\sqcup\boldsymbol{\pi}_i}$ and $\boldsymbol{\Sigma}^{(i)\sqcup\boldsymbol{\pi}_i}$. The maximum likelihood estimation for the required univariate conditional gpdf then is given by:

$$\hat{P}^{\mathcal{N}}(Y_i|Y_{\boldsymbol{\pi}_i})(y_{(i)\sqcup\boldsymbol{\pi}_i}) = \frac{1}{(\breve{\sigma}_i\sqrt{2\pi})} e^{\frac{-(y_i-\breve{\mu}_i)^2}{2\breve{\sigma}_i^2}} \tag{5.3}$$

$$\text{where } \begin{cases} \breve{\sigma}_i &= \frac{1}{\sqrt{\hat{\boldsymbol{W}}_{00}^{(i)\sqcup\boldsymbol{\pi}_i}}} \\ \breve{\mu}_i &= \frac{\hat{\boldsymbol{\mu}}_i \hat{\boldsymbol{W}}_{00}^{(i)\sqcup\boldsymbol{\pi}_i} - \sum_{j=0}^{|\boldsymbol{\pi}_i|-1} (y_{(\boldsymbol{\pi}_i)_j} - \hat{\boldsymbol{\mu}}_{(\boldsymbol{\pi}_i)_j}) \hat{\boldsymbol{W}}_{(j+1)0}^{(i)\sqcup\boldsymbol{\pi}_i}}{\hat{\boldsymbol{W}}_{00}^{(i)\sqcup\boldsymbol{\pi}_i}} \end{cases}$$

Factorization selection

To construct a factorization based on the normal gpdf for a given collection of samples, the greedy factorization selection algorithms described in Section 2.6 can be used. To evaluate the scoring metric in these greedy algorithms, the likelihood for a factor must be repeatedly computed. Using the relation between likelihood and entropy from Section 2.5, computing the negative log–likelihood $-\ln(\mathfrak{L}(\hat{P}^{\mathcal{N}}(Y_{\boldsymbol{j}}), \boldsymbol{\mathcal{S}}))$ in the case of maximum likelihood parameter estimates, is identical to computing $|\boldsymbol{\mathcal{S}}| h(\hat{P}^{\mathcal{N}}(Y_{\boldsymbol{j}}))$. The entropy $h(P^{\mathcal{N}}_{(\boldsymbol{\mu}_{\boldsymbol{j}}, \boldsymbol{\Sigma}^{\boldsymbol{j}})}(Y_{\boldsymbol{j}}))$ of the normal gpdf has been shown (Cover and Thomas, 1991) to be:

$$h\left(P^{\mathcal{N}}_{(\boldsymbol{\mu}_{\boldsymbol{j}}, \boldsymbol{\Sigma}^{\boldsymbol{j}})}(Y_{\boldsymbol{j}})\right) = \frac{1}{2}\left(|\boldsymbol{j}| + \ln\left((2\pi)^{|\boldsymbol{j}|}(\det \boldsymbol{\Sigma}^{\boldsymbol{j}})\right)\right) \tag{5.4}$$

Since the entropy of the normal gpdf can be evaluated faster than the likelihood of the normal gpdf, this gives a more convenient way to evaluating the required negative log–likelihood term in the scoring metric used in the greedy factorization selection algorithms.

Sampling

To draw new samples from a normal factorized probability distribution, we assume to have a method available for drawing a sample from a normal gpdf for a single random variable with a mean parameter of 0 and a variance parameter of 1 (see for instance (Knuth, 1981), Section 3.4.1, Algorithm C). It follows from the form of the normal gpdf that for a random variable Y that is distributed normally with a mean parameter of 0 and a variance parameter of 1 ($Y \sim \mathcal{N}(0,1)$), we have that if $Y' = a + bY$, then $Y' \sim \mathcal{N}(a, b^2)$. We can thus draw a sample from a normal gpdf for a single random variable with mean parameter μ and variance parameter σ^2 by drawing a single new value y that is normally distributed with a mean parameter of 0 and a variance parameter of 1 and compute $\mu + \sigma y$ to be the final result.

To draw a new sample from a multivariate factorization, we must be able to sample from a product of multivariate joint normal gpdfs that have been estimated with a maximum likelihood $\hat{P}^{\mathcal{N}}(Y_{\boldsymbol{\nu}_i})$. We can convert each multivariate joint factor to a Bayesian factorization using the definition of univariate conditional gpdfs from Section 2.4.2 and draw a sample from the so–obtained Bayesian factorization. The maximum–likelihood Bayesian factorization that we construct for the i–th multivariate joint factor is:

$$\hat{P}^{\mathcal{N}}(Y_{\boldsymbol{\nu}_i})(y_{\boldsymbol{\nu}_i}) = \tag{5.5}$$

$$\prod_{j=0}^{|\boldsymbol{\nu}_i|-1} \hat{P}^{\mathcal{N}}(Y_{(\boldsymbol{\nu}_i)_j} | Y_{((\boldsymbol{\nu}_i)_{j+1}, (\boldsymbol{\nu}_i)_{j+2}, \ldots, (\boldsymbol{\nu}_i)_{|\boldsymbol{\nu}_i|-1})})(y_{((\boldsymbol{\nu}_i)_j, (\boldsymbol{\nu}_i)_{j+1}, \ldots, (\boldsymbol{\nu}_i)_{|\boldsymbol{\nu}_i|-1})})$$

Since the required univariate conditional normal gpdf in equation 5.3 is actually a normal gpdf over a single random variable, we can draw a new l–dimensional sample from a Bayesian factorization based on normal gpdfs by repeatedly drawing samples from a one–dimensional normal gpdf. To do so, we first compute a topological sort $\boldsymbol{\omega}$ such that for each univariate conditional factor $\hat{P}^{\mathcal{N}}(Y_{\boldsymbol{\omega}_i}|Y_{\boldsymbol{\omega}_{\boldsymbol{\pi}_i}})$ in the Bayesian factorization we have that $\forall j \in \boldsymbol{\pi}_{\boldsymbol{\omega}_i} : j \in \bigcup_{k=0}^{i-1}\{\boldsymbol{\omega}_k\}$. Using this ordering we are able to compute $\breve{\mu}_{\boldsymbol{\omega}_i}$ and $\breve{\sigma}_{\boldsymbol{\omega}_i}$ for i increasing from 0 to $l-1$, since we have already drawn new samples for the variables that $Y_{\boldsymbol{\omega}_i}$ is conditioned on in the Bayesian factorization. We can then use the procedure for drawing a sample from a one–dimensional normal gpdf to obtain a value for the $\boldsymbol{\omega}_i$–th variable in the new solution. This procedure is known as sampling by *simulation*.

Properties

The number of parameters that need to be estimated for the multivariate joint normal gpdf over $|\boldsymbol{j}|$ random variables, equals $\frac{1}{2}|\boldsymbol{j}|^2 + \frac{3}{2}|\boldsymbol{j}|$. The number of parameters to be estimated therefore does not grow exponentially with the maximum degree of interaction in a factorization, but quadratically. As a result, estimating factorizations based on the normal gpdf is relatively fast and efficient. On the other hand, assuming that there is a bound on the maximum degree of interaction between the problem variables, the overhead for estimating the gpdf becomes a constant and algorithms for estimating factorizations for binary random variables are equally efficient as are algorithms for estimating factorizations based on the normal gpdf.

However, there is also an important limitation to the use of normal gpdfs that should be acknowledged. The density contours that result after estimating a normal factorized probability distribution, are ellipsoids. Depending on the dependencies modelled by the factorization, these ellipsoids can be aligned along any axis. If there is no dependency between a set of random variables, the projected density contours in those dimensions are aligned along the main axes. In either case, a normal gpdf is only capable of efficiently modelling *linear* relations in the sample collection.

Visualization of example estimations

For the sample collections in Figure 5.1 we have computed a maximum likelihood estimate of the multivariate joint normal gpdf. The results can be seen in Figure 5.2. Next to only being capable of expressing linear dependencies, the normal gpdf strongly generalizes the data. This aspect becomes clear from Figure 5.2 as clearly the clustered set is underfit.

Normal kernels gpdf

Definition

The normal kernels gpdf is parameterized by a vector of samples and a diagonal covariance matrix. The normal kernels gpdf is obtained by placing a multivariate normal gpdf over each point in the sample vector and by normalizing the sum of

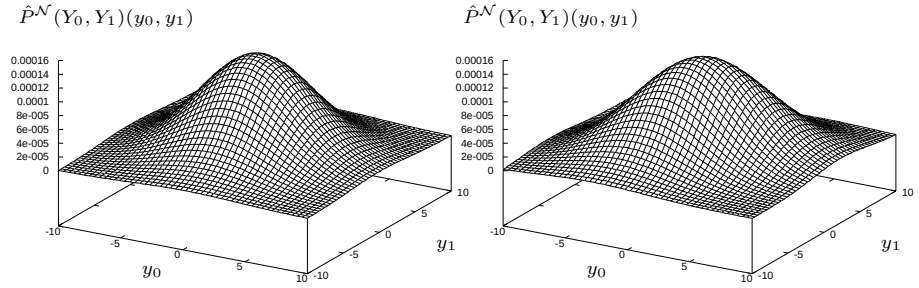


Figure 5.2: The result of fitting the sample collections from figure 5.1 with a multivariate joint normal gpdf using maximum likelihood estimates. The uniform sample collection is shown on the left and the clustered uniform sample collection is shown on the right.

the densities by dividing it by the number of samples. The symmetric covariance matrices of these normal kernels are all the same and are fixed to have non–zero entries *on* the diagonal and zero entries *off* the diagonal. The entries on the diagonal are the individual variances of the normal gpdf in each dimension. The normal kernels gpdf for random variables $Y_{\boldsymbol{\nu}_i}$ is parameterized by a vector of samples $\acute{\boldsymbol{\mathcal{S}}}$ and a diagonal covariance matrix $\boldsymbol{\Sigma}^{\boldsymbol{\nu}_i}$ and is defined by

$$P^{\mathcal{N}_K}_{(\acute{\boldsymbol{\mathcal{S}}}, \boldsymbol{\Sigma}^{\boldsymbol{\nu}_i})}(Y_{\boldsymbol{\nu}_i})(\boldsymbol{y}) = \frac{1}{|\boldsymbol{\mathcal{S}}|} \sum_{j=0}^{|\boldsymbol{\mathcal{S}}|-1} P^{\mathcal{N}}_{\left((\acute{\boldsymbol{\mathcal{S}}}_j)_{\boldsymbol{\nu}_i}, \boldsymbol{\Sigma}^{\boldsymbol{\nu}_i}\right)}(Y_{\boldsymbol{\nu}_i})(\boldsymbol{y}) \tag{5.6}$$

Parameter estimation

The parameter $\acute{\boldsymbol{\mathcal{S}}}$ for the normal kernels gpdf in an actual estimate over a sample collection $\boldsymbol{\mathcal{S}}$ is given by the actual sample collection $\boldsymbol{\mathcal{S}}$. The normal kernels gpdf is actually a special case of a normal mixture gpdf in which the number of mixture components equals $|\boldsymbol{\mathcal{S}}|$ and the centroids of the normal gpdf components coincide with the samples in the sample collection.

Deciding how to choose the fixed covariance matrix for each normal gpdf is hard. A maximum likelihood estimate is undesirable because in that estimate the variances are zero. The reason for this is that maximum likelihood is obtained by definition if the density at each point is maximum. In the case of a single normal gpdf, this can be obtained by setting the variance to 0, leading to a density of infinity at the mean of the normal gpdf.

Therefore, the variances in the covariance matrix that is used for each normal gpdf in the normal kernels gpdf should be set otherwise using a rule of thumb. One way of doing so, is to compute the range of the samples in $\boldsymbol{\mathcal{S}}$ in each dimension and to set the variance in the i–th dimension to a value based on the range such that it decreases as the number of samples increases, e.g. $\alpha \cdot$

$\text{range}_i/|\boldsymbol{\mathcal{S}}|$. The value of α allows to control the smoothness of the resulting density estimation. Let $\mathfrak{s}_i^2$ denote the variance to be used in the i–th dimension, the proposed parameter estimates can now be formalized as follows:

$$\acute{\boldsymbol{\mathcal{S}}} = \boldsymbol{\mathcal{S}}, \quad \boldsymbol{\Sigma}^{\boldsymbol{\nu}_i} = \begin{bmatrix} \mathfrak{s}^2_{(\boldsymbol{\nu}_i)_0} & 0 & \dots & 0 \\ 0 & \mathfrak{s}^2_{(\boldsymbol{\nu}_i)_1} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \mathfrak{s}^2_{(\boldsymbol{\nu}_i)_{|\boldsymbol{\nu}_i|-1}} \end{bmatrix} \tag{5.7}$$

$$\text{where} \quad \mathfrak{s}^2_{(\boldsymbol{\nu}_i)_j} = \frac{\alpha\,(\max\{\boldsymbol{y}_j \mid \boldsymbol{y} \in \boldsymbol{\mathcal{S}}\} - \min\{\boldsymbol{y}_j \mid \boldsymbol{y} \in \boldsymbol{\mathcal{S}}\})}{|\boldsymbol{\mathcal{S}}|}$$

In Appendix A, the univariate conditional form of the normal kernels gpdf that is required to construct Bayesian factorizations is derived (equation A.14). By using the same estimates as for the multivariate joint case, we also have a way of estimating the parameters for the factors in the Bayesian factorization.

Factorization selection

To use factorized probability distributions in combination with the normal kernels gpdf we note that we cannot use the greedy learning algorithms based on penalized likelihood maximization as presented in Section 2.6. The reason for this is that the number of parameters to be estimated does *not* increase as the complexity of the factorization increases. However, the likelihood of the estimated probability distribution *does* increase.

> **Example 5.1.** Consider the case of two random variables and two sample points $(0, 1)$ and $(1, 0)$. Regardless of the type of factorization, we must always compute the variance parameter in each dimension. Estimating a univariately factorized probability distribution for the two random variables leads to a joint probability density that has four equal normally distributed peaks with means $(0, 0)$, $(0, 1)$, $(1, 0)$ and $(1, 1)$. If we use a maximum complexity factorization, we get only normally distributed peaks with means $(0, 1)$ and $(1, 0)$, which by definition leads to a larger density at the sample points $(0, 1)$ and $(1, 0)$ and thus leads to a larger likelihood for the joint factorization.

As a consequence, the use of a greedy incremental factorization construction algorithm based on penalized likelihood maximization inavertably leads to the maximum complexity factorization. Since we only focus on penalized likelihood–based probabilistic model construction in this thesis, we will not use automated factorization selection in combination with the normal kernels gpdf. Instead, we will fix the factorization structure beforehand.

Sampling

To draw a new sample from a multivariate factorization based on the normal kernels gpdf, each multivariate joint factor can be regarded separately. For each

factor involving random variables $Y_{\boldsymbol{\nu}_i}$, one kernel is selected at random and the simulation approach for the single multivariate joint normal gpdf can be used to draw a new sample for variables $Y_{\boldsymbol{\nu}_i}$. However, since the covariance matrix is diagonal, the kernel can be written as a univariately factorized multivariate normal gpdf, from which a new sample can be drawn by drawing for each random variable $Y_{(\boldsymbol{\nu}_i)_j}$ a value from a normal gpdf that has a mean equal to the selected sample point in the $(\boldsymbol{\nu}_i)_j$–th dimension and a variance of $\mathfrak{s}_{(\boldsymbol{\nu}_i)_j}$.

Equation A.14 in Appendix A shows that each factor $P^{\mathcal{N}_K}(Y_i|Y_{\boldsymbol{\pi}_i})$ is a normal kernels gpdf for Y_i. Therefore, we can use a similar approach as for the single normal gpdf. Using a topological sort on the Bayesian factorization graph, the conditional factors can be regarded in the correct order. For each factor then, the mixture weights can be computed and a single component can be selected based on the weight proportion. The selected component is a normal gpdf for a single random variable from which we already know how to draw a new sample.

Properties
The main advantage of the normal kernels gpdf is that it has a natural tendency to fit the structure of the sample vector and is thereby capable of expressing complicated non–linear dependencies. A related disadvantage however, is that the quality of the density estimation heavily depends on the value of α. Intuitively, a larger α results in a smoother fit, but it is hard to predict beforehand what a good value for α would be. The normal kernels gpdf has a tendency to *overfit* a sample collection. This is directly related to the inability to use factorization selection based on penalized likelihood maximization. Without factorization selection, using the normal kernels approach is relatively fast. However, without good model selection and model fitting, the normal kernels gpdf is also hard to handle. One other possibility is to set the variances, or even covariances, for the normal kernels gpdf adaptively. If the adaptation is done for each normal kernel separately, the resulting approach is equivalent to the use of evolution strategies (Bäck and Schwefel, 1993). Concluding, the normal kernels gpdf certainly has interesting properties and potential to be used in IDEAs, but it is likely to be hard to handle.

Visualization of example estimations
The result of fitting a multivariate joint normal kernels gpdf over the sample collections in Figure 5.1, can be seen in Figure 5.3. The localized aspect of the normal kernels gpdf becomes clear immediately. Even though the samples were drawn from a uniform distribution, there are still peaks and valleys in the estimated distribution. On the other hand, the normal kernels estimate is much better for the clustered data than is the normal gpdf estimate in Figure 5.2.

Normal mixture gpdf

Definition
If we take w normal gpdfs instead of only a single one or as many as $|\boldsymbol{\mathcal{S}}|$, we have a trade–off between the cluster insensitive normal gpdf and the cluster

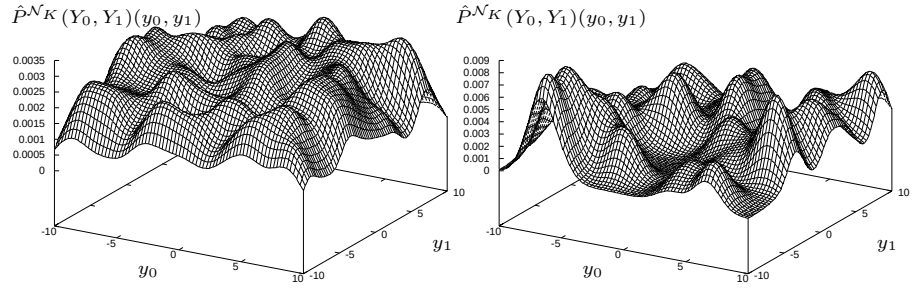


Figure 5.3: The result of fitting the sample collections from figure 5.1 with a multivariate joint normal kernels gpdf using variances $\mathfrak{s}_0^2 = \mathfrak{s}_1^2 = 1$. The uniform sample collection is shown on the left and the clustered uniform sample collection is shown on the right.

oversensitive normal kernels gpdf. For the normal mixture gpdf, we allow a full symmetric covariance matrix for each normal gpdf. The normal mixture gpdf for random variables $Y_{\boldsymbol{\nu}_i}$ is parameterized by a vector of w mixture coefficients $\boldsymbol{\beta}^{\boldsymbol{\nu}_i} = (\beta_0^{\boldsymbol{\nu}_i}, \beta_1^{\boldsymbol{\nu}_i}, \ldots, \beta_{w-1}^{\boldsymbol{\nu}_i})$ and a vector of w pairs consisting of a vector of $|\boldsymbol{\nu}_i|$ means and a symmetric covariance matrix of dimension $|\boldsymbol{\nu}_i| \times |\boldsymbol{\nu}_i|$ and equals

$$P^{\mathcal{N}_M}_{((\boldsymbol{\mu}^0_{\boldsymbol{\nu}_i}, \boldsymbol{\Sigma}^{0,\boldsymbol{\nu}_i}), \ldots, (\boldsymbol{\mu}^{w-1}_{\boldsymbol{\nu}_i}, \boldsymbol{\Sigma}^{w-1,\boldsymbol{\nu}_i}), \boldsymbol{\beta}^{\boldsymbol{\nu}_i})}(Y_{\boldsymbol{\nu}_i})(\boldsymbol{y}) = \sum_{j=0}^{w-1} \beta_i^{\boldsymbol{\nu}_i} P^{\mathcal{N}}_{(\boldsymbol{\mu}^j_{\boldsymbol{\nu}_i}, \boldsymbol{\Sigma}^{j,\boldsymbol{\nu}_i})}(Y_{\boldsymbol{\nu}_i})(\boldsymbol{y}) \quad (5.8)$$

Parameter estimation

A maximum likelihood estimation of the parameters for the multivariate joint gpdf in equation 5.8 cannot be obtained analytically. Therefore, as an alternative approach, the EM algorithm (Dempster et al., 1977) can be used. The EM algorithm is a general iterative approach to computing a maximum likelihood estimate. The derivation of the EM algorithm is rather involved. The EM algorithm was already discussed earlier in Section 2.7.2. The use of the EM algorithm for the estimation of the parameters of a normal mixture gpdf results in an algorithm that initializes each mixture coefficient, all means and all covariance matrices to certain values and then updates all of these values iteratively until the algorithm converges or until a maximum number of iterations has been reached. We refrain from a complete derivation of the required update equations. The interested reader may find more details elsewhere (Dempster et al., 1977; Bilmes, 1997). For a normal mixture gpdf (equation 5.8) that is to be estimated for random variables $Y_{\boldsymbol{\nu}_i}$, let $\{\hat{\beta}_q^{\boldsymbol{\nu}_i}\}^{\text{old}}$, $\{\hat{\mu}^q_{\boldsymbol{\nu}_i}\}^{\text{old}}$ and $\{\hat{\boldsymbol{\Sigma}}^{q,\boldsymbol{\nu}_i}\}^{\text{new}}$ be the parameters of the current model and let $\{\hat{\beta}_q^{\boldsymbol{\nu}_i}\}^{\text{new}}$, $\{\hat{\mu}^q_{\boldsymbol{\nu}_i}\}^{\text{new}}$ and $\{\hat{\boldsymbol{\Sigma}}^{q,\boldsymbol{\nu}_i}\}^{\text{new}}$ be the parameters of the new model. After the parameters for the mixture model

have been initialized, the EM algorithm iteratively updates the parameters until a certain convergence criterion is met. Common convergence criteria are a maximum number of iterations or all new parameters differing less than $\varepsilon > 0$ from their previous estimates. The required update equations are the following:

$$\{\hat{\beta}_q^{\boldsymbol{\nu}_i}\}^{\text{new}} = \frac{1}{|\boldsymbol{\mathcal{S}}|} \sum_{j=0}^{|\boldsymbol{\mathcal{S}}|-1} \frac{\{\hat{\beta}_q^{\boldsymbol{\nu}_i}\}^{\text{old}} P^{\mathcal{N}}_{\hat{\boldsymbol{\theta}}_q^{\text{old}}}(Y_{\boldsymbol{\nu}_i})((\boldsymbol{\mathcal{S}}_j)_{\boldsymbol{\nu}_i})}{P^{\mathcal{N}_M}_{\hat{\boldsymbol{\theta}}^{\text{old}}}(Y_{\boldsymbol{\nu}_i})((\boldsymbol{\mathcal{S}}_j)_{\boldsymbol{\nu}_i})} \tag{5.9}$$

$$\{\hat{\mu}_{\boldsymbol{\nu}_i}^q\}^{\text{new}} = \frac{\sum_{j=0}^{|\boldsymbol{\mathcal{S}}|-1} \frac{\{\hat{\beta}_q^{\boldsymbol{\nu}_i}\}^{\text{old}} P^{\mathcal{N}}_{\hat{\boldsymbol{\theta}}_q^{\text{old}}}(Y_{\boldsymbol{\nu}_i})((\boldsymbol{\mathcal{S}}_j)_{\boldsymbol{\nu}_i})(\boldsymbol{\mathcal{S}}_j)_{\boldsymbol{\nu}_i}}{P^{\mathcal{N}_M}_{\hat{\boldsymbol{\theta}}^{\text{old}}}(Y_{\boldsymbol{\nu}_i})((\boldsymbol{\mathcal{S}}_j)_{\boldsymbol{\nu}_i})}}{\sum_{j=0}^{|\boldsymbol{\mathcal{S}}|-1} \frac{\{\hat{\beta}_q^{\boldsymbol{\nu}_i}\}^{\text{old}} P^{\mathcal{N}}_{\hat{\boldsymbol{\theta}}_q^{\text{old}}}(Y_{\boldsymbol{\nu}_i})((\boldsymbol{\mathcal{S}}_j)_{\boldsymbol{\nu}_i})}{P^{\mathcal{N}_M}_{\hat{\boldsymbol{\theta}}^{\text{old}}}(Y_{\boldsymbol{\nu}_i})((\boldsymbol{\mathcal{S}}_j)_{\boldsymbol{\nu}_i})}} \tag{5.10}$$

$$\{\hat{\boldsymbol{\Sigma}}^{q,\boldsymbol{\nu}_i}\}^{\text{new}} = \frac{\sum_{j=0}^{|\boldsymbol{\mathcal{S}}|-1} \frac{\{\hat{\beta}_q^{\boldsymbol{\nu}_i}\}^{\text{old}} P^{\mathcal{N}}_{\hat{\boldsymbol{\theta}}_q^{\text{old}}}(Y_{\boldsymbol{\nu}_i})((\boldsymbol{\mathcal{S}}_j)_{\boldsymbol{\nu}_i})((\boldsymbol{\mathcal{S}}_j)_{\boldsymbol{\nu}_i} - \{\hat{\mu}_{\boldsymbol{\nu}_i}^q\}^{\text{new}})((\boldsymbol{\mathcal{S}}_j)_{\boldsymbol{\nu}_i} - \{\hat{\mu}_{\boldsymbol{\nu}_i}^q\}^{\text{new}})^T}{P^{\mathcal{N}_M}_{\hat{\boldsymbol{\theta}}^{\text{old}}}(Y_{\boldsymbol{\nu}_i})((\boldsymbol{\mathcal{S}}_j)_{\boldsymbol{\nu}_i})}}{\sum_{j=0}^{|\boldsymbol{\mathcal{S}}|-1} \frac{\{\hat{\beta}_q^{\boldsymbol{\nu}_i}\}^{\text{old}} P^{\mathcal{N}}_{\hat{\boldsymbol{\theta}}_q^{\text{old}}}(Y_{\boldsymbol{\nu}_i})((\boldsymbol{\mathcal{S}}_j)_{\boldsymbol{\nu}_i})}{P^{\mathcal{N}_M}_{\hat{\boldsymbol{\theta}}^{\text{old}}}(Y_{\boldsymbol{\nu}_i})((\boldsymbol{\mathcal{S}}_j)_{\boldsymbol{\nu}_i})}} \tag{5.11}$$

In Appendix A, the definition of the univariate conditional gpdf that is required to construct Bayesian factorizations, is derived. Similar to the normal kernels gpdf, the same parameter estimates can be used as for the multivariate joint case to automatically get a way of estimating the parameters for the conditional factors in the Bayesian factorization.

Factorization selection

Contrary to the case of the normal kernels gpdf, the greedy incremental maximum likelihood factorization selection algorithms from Section 2.6 based on maximum likelihood penalization metrics can be used to model dependencies between problem variables. However, to do so, we are required to use the EM algorithm many times to estimate the multivariate joint gpdfs that are required as a result of the factorization structure. Using the EM algorithm is unfortunately a time–consuming task, which makes automated factorization selection based on the greedy penalized maximum likelihood approach extremely costly. Therefore, similar to the case of the normal kernels gpdf, we will fix the factorization structure beforehand instead.

Sampling

Drawing new samples from marginal product factorizations or Bayesian factorizations in which the normal mixture gpdf is used, is quite similar to the case of the normal kernels gpdf. For the multivariate factorization, we must be able to draw a multivariate sample for each factor $Y_{\boldsymbol{\nu}_i}$. To do so, the j–th normal gpdf in the normal mixture estimate for random variables $Y_{\boldsymbol{\nu}_i}$ is selected with probability $\boldsymbol{\beta}_j^{\boldsymbol{\nu}_i}$. Subsequently, a multivariate sample can be drawn from the selected normal gpdf by simulating this normal gpdf.

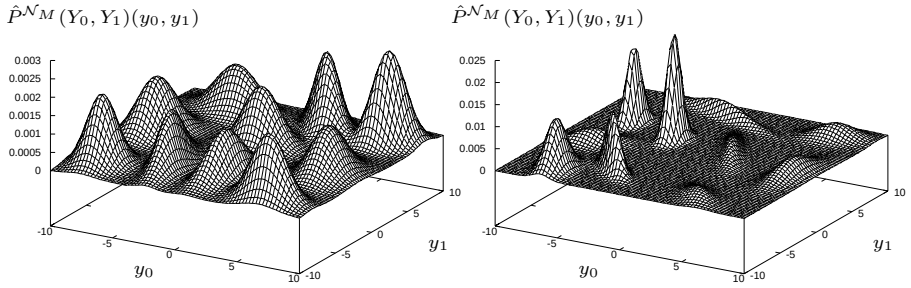


Figure 5.4: The result of fitting the sample collections from figure 5.1 with a multivariate joint normal mixture gpdf using the EM algorithm with 10 mixture components. The uniform sample collection is shown on the left and the clustered uniform sample collection is shown on the right.

To draw a sample from a Bayesian factorization based on normal mixture gpdfs, the same approach can be taken as explained for the normal kernels gpdf.

Properties

The main advantage of the normal mixture gpdf is that it provides a trade–off between the normal gpdf and the normal kernels gpdf. Furthermore, a maximum likelihood approach is available to estimate the normal mixture gpdf parameters. However, especially as the number of variables and the number of mixture components increases, the result of the EM algorithm becomes unreliable. Furthermore, the EM algorithm is a time–consuming approach. The main disadvantage of the normal mixture gpdf is that because of this unreliability and large required running time, we are not able to perform efficient induction to find dependencies that can be expressed by factorizations. Because we must fix the factorization structure beforehand, we are forced to use either a univariate factorization or a full joint factorization in which either no dependency or all possible dependencies are accounted for. In the latter case however, the resulting IDEA is highly likely to be inefficient due to the unreliable probability density estimates.

Visualization of example estimations

For the sample collections in Figure 5.1, we have used the EM algorithm to fit a normal mixture gpdf with $k = 10$ mixture components. The results can be seen in Figure 5.4. The trade–off between extreme generalization with only a single mixture component and extreme cluster sensitivity with $|\boldsymbol{\mathcal{S}}|$ mixture components can be seen from the density plots. For the uniform sample collection, the peaks are distributed quite well over the square. For the clustered sample collection, the mixture components are aligned quite well with the clusters.

Histogram gpdf

Definition

Perhaps one of the most straightforward ways to estimate a probability distribution over real–valued data is to use a histogram. The histogram gpdf can arbitrarily well represent the probability distribution of the data by choosing an arbitrarily small binwidth. However, the number of samples that is required to estimate the frequencies within each bin reliably increases significantly if the binwidth is decreased. Therefore, only a few bins are usually used. The histogram gpdf for random variables $Y_{\boldsymbol{\nu}_i}$ is parameterized by a vector of bin–boundary vectors $\boldsymbol{\beta}_{\boldsymbol{\nu}_i}$ and a bin–probability vector $\boldsymbol{\pi}$ and is defined by

$$P^{\mathcal{H}}_{(\boldsymbol{\beta}_{\boldsymbol{\nu}_i},\boldsymbol{\pi})}(Y_{\boldsymbol{\nu}_i})(\boldsymbol{y}) = \begin{cases} 0 & \text{if } \exists_j : \varphi(\boldsymbol{\beta}_{(\boldsymbol{\nu}_i)_j}, \boldsymbol{y}_j) = -1 \\ \eta \boldsymbol{\pi}_q & \text{otherwise} \end{cases} \tag{5.12}$$

$$\text{where } \begin{cases} \eta & = \frac{1}{\prod_{j=0}^{|\boldsymbol{\nu}_i|-1} ((\boldsymbol{\beta}_{(\boldsymbol{\nu}_i)_j})_{\varphi(\boldsymbol{\beta}_{(\boldsymbol{\nu}_i)_j}, \boldsymbol{y}_j)+1}) - ((\boldsymbol{\beta}_{(\boldsymbol{\nu}_i)_j})_{\varphi(\boldsymbol{\beta}_{(\boldsymbol{\nu}_i)_j}, \boldsymbol{y}_j)})} \\ q & = \sum_{j=0}^{|\boldsymbol{\nu}_i|-1} \varphi(\boldsymbol{\beta}_{(\boldsymbol{\nu}_i)_j}, \boldsymbol{y}_j) \prod_{k=j+1}^{l-1} \left|\boldsymbol{\beta}_{(\boldsymbol{\nu}_i)_k}\right| - 1 \\ \varphi(\boldsymbol{\beta}, y) & = \begin{cases} -1 & \text{if } \neg\exists_j : \boldsymbol{\beta}_j \leq y < \boldsymbol{\beta}_{j+1} \\ j & \text{otherwise (such that } \boldsymbol{\beta}_j \leq y < \boldsymbol{\beta}_{j+1}) \end{cases} \end{cases}$$

Although the formula in equation 5.12 looks quite complicated, it is relatively easy to interpret. The bin–boundary vectors $\boldsymbol{\beta}_{\boldsymbol{\nu}_i}$ indicate for each dimension the boundaries of the bins, which means that in dimension $\boldsymbol{\beta}_{(\boldsymbol{\nu}_i)_j}$ we have $|\boldsymbol{\beta}_{(\boldsymbol{\nu}_i)_i}|-1$ bins. Function $\varphi(\boldsymbol{\beta}, y)$ returns for any bin–boundary vector $\boldsymbol{\beta}$ and a query point y in a single dimension either the index of the bin in which the query point is contained or -1 if the query point is outside the range. The probability density is defined to be 0 whenever a query point is used that falls outside of the range, which means that there is at least one dimension for which the φ function returns -1. If the query point *does* fall into a bin, the probability for that bin as stored in the probability vector is returned in the same way as is done for a discrete integer variable in which a $|\boldsymbol{\nu}_i|$–dimensional discrete space is mapped onto a one–dimensional index. Since this probability is uniform over the complete area of the bin in which the query point is contained, we must multiply this probability by some normalization factor η to ensure that the histogram gpdf integrates to 1. This normalization constant is just one over the area of the bin.

Parameter estimation

The variant of the histogram gpdf that we will use, is known as the *fixed–width* histogram. In this variant of the histogram gpdf, the number of bins in each dimension is the same and the bin–boundaries in any single dimension are equidistant. We denote the number of bins in each dimension by r.

To use this histogram, the sample data can quite easily be mapped onto discrete integer data, after which the discrete integer frequency gpdf can be used with the extension that any integers outside of the set $\{0, 1, \ldots, r-1\}$ must be assigned a probability of 0.

We estimate the bin parameters of the fixed–width histogram by setting the width of the bins in a certain dimension relative to the range of the samples in that dimension. For each random variable Y_i a single discrete integer random variable X_i can be introduced with a domain of $\{0, 1, \ldots, r-1\}$. The real–valued samples can be mapped onto the discrete integer range by dividing the range of the samples in each dimension into r bins. Using the so–converted data a multivariate joint maximum likelihood estimate can be obtained for a vector $X_{\boldsymbol{j}}$ of discrete integer random variables by counting frequencies. Note that although formally we require the extension that any integer outside the range of $\{0, 1, \ldots, r-1\}$ is assigned the probability 0, we are never concerned with this case if we perform the estimation adaptive to the range of the samples and only draw new samples from the so estimated histogram gpdf. The required mapping from a vector of l real values $\boldsymbol{y}$ to a vector of l integers $\boldsymbol{x}$ is given by:

$$\boldsymbol{x}_i = \begin{cases} r-1 & \text{if } \boldsymbol{y}_i = \max\{\boldsymbol{y}_i \mid \boldsymbol{y} \in \boldsymbol{\mathcal{S}}\} \\ \left\lfloor \frac{\boldsymbol{y}_i - \min\{\boldsymbol{y}_i \mid \boldsymbol{y} \in \boldsymbol{\mathcal{S}}\}}{(\max\{\boldsymbol{y}_i \mid \boldsymbol{y} \in \boldsymbol{\mathcal{S}}\} - \min\{\boldsymbol{y}_i \mid \boldsymbol{y} \in \boldsymbol{\mathcal{S}}\})} r \right\rfloor & \text{otherwise} \end{cases} \tag{5.13}$$

After applying the transformation, maximum likelihood integer estimations can be used to estimate the bin probabilities.

Factorization selection

Depending on the bin width, more detailed or less detailed dependencies can be modelled using histogram gpdfs. However, to obtain a good estimate of the dependencies between the random variables, many bins may be required. Unfortunately, as the maximum order of interaction in a factorization increases, the number of parameters to be estimated grows extremely fast. Therefore, histograms are actually quite inefficient in expressing dependencies between problem variables. The number of parameters that are required to estimate the factorized probability distribution adequately, is extremely large. For example, if we use 10 bins in each dimension, placing only three variables together in a single node vector in a marginal product factorization already amounts to $10^3 = 1000$ bins, whereas in the binary case we could go as far as placing 10 variables together in a single node vector to get $2^{10} = 1024$ bins.

This phenomenon, which is known as the *curse of dimensionality*, is an important drawback of using histograms. In the use of the histogram gpdf in IDEAs, we will therefore fix the factorization structure to be the univariate one since it seems to be the only practical factorization structure to use.

Sampling

To draw new samples from a factorization based on the fixed–width histogram gpdf, we only need to realize that the probability in each histogram bin is uniform over all real values that are represented by the bin. The underlying discrete integer gpdf can thus be used to indicate in which bin a new sample should be generated, after which within the range of the bin, a point is chosen at random.

So for both the marginal product factorization as well as for the Bayesian factorization, after having drawn a new sample in the form of l–dimensional integer vector $\boldsymbol{x}$, a vector $\boldsymbol{y}$ of l real values can be constructed that was correctly drawn from the estimated histogram distribution as follows:

$$\begin{aligned} \boldsymbol{y}_i = \min\{\boldsymbol{y}_i \mid \boldsymbol{y} \in \boldsymbol{\mathcal{S}}\} + \\ (\boldsymbol{x}_i + \textsc{Random}([0,1])) \frac{(\max\{\boldsymbol{y}_i \mid \boldsymbol{y} \in \boldsymbol{\mathcal{S}}\} - \min\{\boldsymbol{y}_i \mid \boldsymbol{y} \in \boldsymbol{\mathcal{S}}\})}{r} \end{aligned} \tag{5.14}$$

Properties
An advantage of the histogram gpdf over the other real–valued gpdfs is that we can directly use well–known discrete integer gpdf estimates and factorization selection techniques. Furthermore, a more detailed estimate of the true underlying probability distribution can be obtained if more bins are used. However, as the number of bins increases, the efficiency of modelling dependencies with the histogram gpdf rapidly decreases. Furthermore, as the number of bins is increased, the danger of overfitting the given samples increases as well. If more bins are used, there will be more bins with a probability of 0 since there won't be any samples in them. As a result, drawing more samples from the estimated probability distribution will not produce any more samples in these areas, even though these areas might very well contain the global optimum. This is less likely to happen with gpdfs that have a better generalization property such as the normal gpdf.

Visualization of example estimations
For the sample collections in Figure 5.1 we have fitted a two–dimensional joint histogram gpdf with a maximum likelihood. The results can be seen in Figure 5.5. Similar to the normal kernels approach, the histogram gpdf has a strong localized aspect, especially with the quite large number of bins that we used in Figure 5.5. Although many parameters are required, the histogram gpdf is capable of modelling the clusters in the clustered sample collection quite well.

5.1.2 Mixture probability distributions by clustering

A mixture probability distribution is a weighted sum of probability distributions. Each of the probability distributions in the sum is a function of all random variables. To ensure that the weighted sum of probability distributions is again a probability distribution, the weights are all positive and sum up to 1. A mixture probability distribution for a vector of random variables $\boldsymbol{\mathcal{Z}}$ is parameterized by a vector of k probabilistic model structures $\boldsymbol{\varsigma}$ and a vector of k probabilistic model parameters $\boldsymbol{\theta}$ and is defined by

$$P_{(\boldsymbol{\varsigma},\boldsymbol{\theta})}(\boldsymbol{\mathcal{Z}}) = \sum_{i=0}^{k-1} \beta_i P_{(\boldsymbol{\varsigma}_i,\boldsymbol{\theta}_i)}(\boldsymbol{\mathcal{Z}}) \tag{5.15}$$

such that $\forall i \in \boldsymbol{\mathcal{K}} : \beta_i \geq 0$ and $\sum_{i=0}^{k-1} \beta_i = 1$.

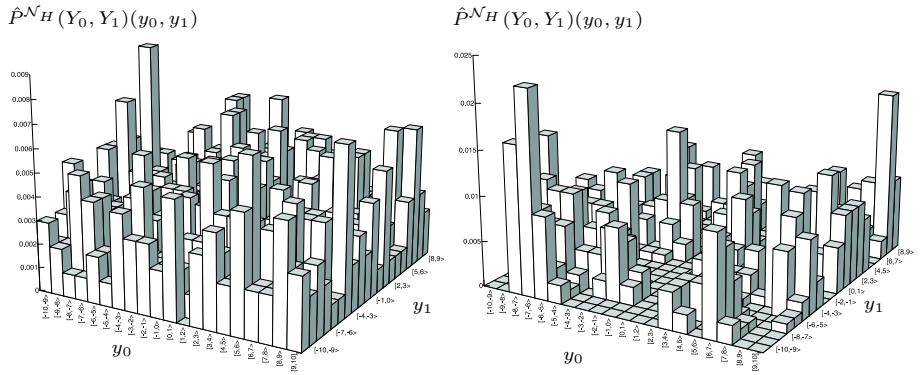


Figure 5.5: The result of fitting the sample collections from figure 5.1 with a multivariate joint normal histogram gpdf using maximum likelihood estimates for 20 bins in each dimension. The uniform sample collection is shown on the left and the clustered uniform sample collection is shown on the right.

Factorization mixture selection

Clustering the sample vector and subsequently estimating a factorized probability distribution in each cluster, is a fast way to construct mixture probability distributions through which dependencies in different parts of the sample vector can be expressed. Depending on the gpdf used in each cluster, any type of mixture probability distribution can be constructed. The drawback of this approach is that from a probabilistic viewpoint, the resulting probability distribution estimation is almost certain not to be a maximum likelihood estimate. However, the use of clustering does allow for the modelling of non–linear dependencies between the variables. Furthermore, the probability distribution estimate that results when using clustering, is better to handle than is the normal kernels gpdf. Also, we can still use the greedy incremental penalized maximum likelihood algorithm to estimate factorizations in each cluster and thereby model linear relations effectively in each cluster. The use of combinations of gpdfs allows us to efficiently break up non–linear interactions so that we can use combinations of simple factorizations to get an adequate representation of the sample vector, which is impossible with factorizations based on simpler gpdfs such as the normal gpdf. An example of this is depicted in Figure 5.6.

Mixture coefficient parameter estimation

Approaches to obtaining clusters are discussed in Section 2.7. Once the clusters have been constructed however, the mixture coefficients β_i still have to be determined. This can be done in different ways. One of the two most common ways, is to set β_i to the ratio of the size of the i–th cluster to the total size of all clusters. This approach enfavors regions in the sample vector that contain more evidence regarding the locations of local optima. Let $\boldsymbol{c^i}$ be a vector of

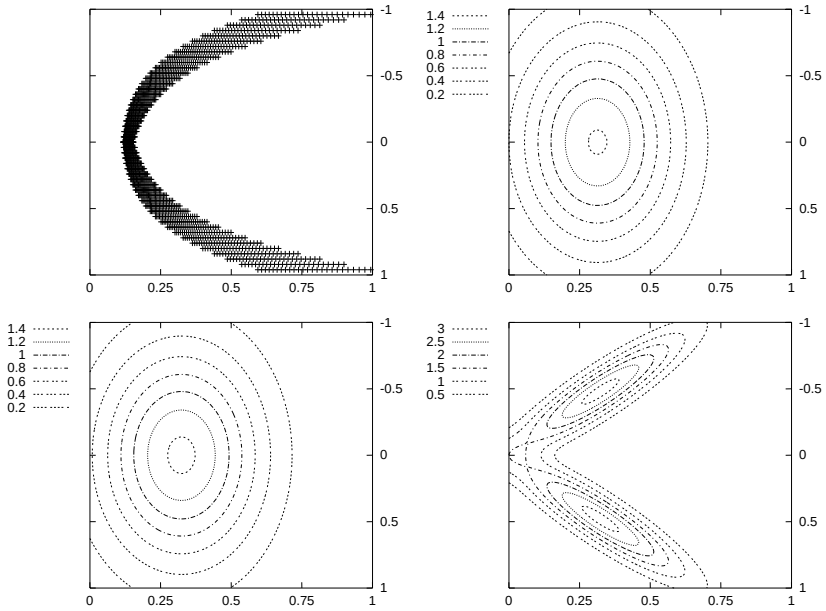


Figure 5.6: Density contours of maximum likelihood normal gpdf estimates on a sample vector that contains non–linear dependencies (top left). The density contours are shown for the product of two one–dimensional normal gpdfs (top right), a single multivariate joint two–dimensional normal gpdf (bottom left) and two multivariate joint two–dimensional normal gpdfs that have been fit after the sample vector was clustered (bottom right).

indices that indicates the samples in the sample vector $\boldsymbol{\mathcal{S}}$ that are assigned to the i–th cluster. The proportional cluster size approach to setting the mixture coefficients can now be written as follows:

$$\beta_i = \frac{|\boldsymbol{c^i}|}{\sum_{j=0}^{k-1} |\boldsymbol{c^j}|} \tag{5.16}$$

As selection will drive the optimization process toward the more promising regions of the search space, the clusters will be concentrated on different peaks in the optimization function. Such *niches* will however not remain stable using equation 5.16 if the peaks are not equally high in case of maximization, since selection will prefer the larger peak. In the case of equal peaks, the search may still converge to concentrate on a single peak because of finite statistical instability. Alternatively, the mixing coefficients can be determined as the average fitness of the niche divided by the sum of the average fitnesses over every niche. This approach enfavors regions in the sample vector that are on average more promising, regardless of the quantity of evidence available in the region. As a result, there is a natural tendency for the IDEA to divide the samples amongst different peaks. Formally, this approach can be written as follows:

$$\beta_i = \frac{\frac{1}{|\boldsymbol{c^i}|} \sum_{q=0}^{|\boldsymbol{c^i}|-1} \mathfrak{G}(\boldsymbol{\mathcal{S}}_{c_q^i})}{\sum_{j=0}^{k-1} \frac{1}{|\boldsymbol{c^j}|} \sum_{q=0}^{|\boldsymbol{c^j}|-1} \mathfrak{G}(\boldsymbol{\mathcal{S}}_{c_q^j})} \tag{5.17}$$

The formula in equation 5.17 doesn't work if the objective is to minimize and the values are below zero. To this end, rescaling is required.

The use of clustering algorithms to obtain niching in stochastic optimization, was first introduced in the field of genetic algorithms (Yin and Germay, 1993). Clustering algorithms and the mixing coefficients in equations 5.16 and 5.17 were later used in IDEAs to demonstrate the breaking of symmetry and the achievement of multi–modal optimization (Pelikan and Goldberg, 2000).

5.2 Exploiting local gradient information by hybridization: GLIDE

In the estimation of a probability distribution in IDEAs, no assumption is made on the source of the samples. Although this is a direct reflection of the fact that we focus on black box optimization, it does imply that for real–valued (partially) continuous functions that we seek to optimize with IDEAs, *gradient information* is disregarded. As a result, once the search has located an interesting region in the real–valued search space, efficiently finding the (local) optimum within that region is prohibited. To speed up convergence, it therefore appears to be a good idea to hybridize the IDEA to exploit local gradient information. One of the main reasons why ES have been quite successful in real–valued optimization, is their ability to evolve a probabilistic search direction, which enables the exploitation of gradient features.

To use the local gradient information, a straightforward manner is to perform *gradient descent* (in the case of minimization). This is an iterative approach that alters a point by moving it a short distance in the direction of the greatest rate of decrease in the optimization function (see for instance (Press, Teukolsky, Vetterling and Flannery, 1992)). By using *line minimization*, the distance that is moved in the direction of the steepest descent, takes the search to a point at which the gradient in that direction is 0. Subsequently, a new direction is taken until the search converges. However, following the direction of steepest descent in each step is in general not optimal. The reason for this is that if a descent is executed until the gradient in the current direction is 0, each subsequent search direction is orthogonal to the previous one. This can cause the search to oscillate around the optimal direction towards the optimum. The *conjugate gradient* algorithm (Hestenes and Stiefel, 1952; Press et al., 1992) overcomes this problem. In this algorithm, each subsequent search direction is *conjugate* with the previous one. This means that the new direction is chosen so that the component of the gradient in the previous direction remains zero along the new direction, resulting in more efficient local optimization. The conjugate gradient algorithm is a well–known tool for real–valued function optimization. By restarting the conjugate gradient algorithm from many different starting points in the search space, one of the best and most generally applicable non–evolutionary real–valued (continuous) optimization techniques is obtained (Press et al., 1992). This approach is called the *random restart conjugate gradient* ($\mathbb{R}$CG) algorithm.

By incorporating local gradient optimization algorithms such as the conjugate gradient algorithm in ID$\mathbb{E}$As, we obtain a specialized hybrid ID$\mathbb{E}$A. We call an algorithm that differs from an ID$\mathbb{E}$A only in that one or more local gradient optimization algorithms are used during a run, a *Gradient–Leveraged Iterated Density–Estimation Evolutionary Algorithm (GLIDE–EA)* or *GLIDE* for short. To use the conjugate gradient algorithm in GLIDE, we must be able to approximate the gradient. Since in BBO we assume that we do not know the analytical form of the optimization function, we cannot compute an exact solution to find the gradient at any point. To approximate the gradient at a single point, we therefore require l additional evaluations. For higher dimensional functions, using local gradient information therefore becomes much more expensive. In the specific GLIDE instance that we will experiment with, we apply the conjugate gradient algorithm to $\lfloor \tau_G n \rfloor$ randomly selected solutions at the end of each generation. We do not suggest that this hybridization approach is optimal, but it will point out whether a gradient search algorithm can aid in the optimization of (continuous) real–valued functions.

5.3 Experiments

In the previous section we have indicated different ways in which we can estimate real–valued probability distributions. In this section we will use these techniques to test different ID$\mathbb{E}$A and GLIDE instances on four well–known numerical optimization problems. We varied the dimensionality of these problems to get

Name	Definition	Range
Griewank	**Minimize** $\frac{1}{4000}\sum_{i=0}^{l-1}(\mathsf{Y}_i - 100)^2 - \prod_{i=0}^{l-1}\cos\left(\frac{\mathsf{Y}_i - 100}{\sqrt{i+1}}\right) + 1$	$\mathsf{Y}_i \in [-600, 600]$ $(0 \leq i < l)$
Michalewicz	**Minimize** $-\sum_{i=0}^{l-1}\sin(\mathsf{Y}_i)\sin^{20}\left(\frac{(i+1)\mathsf{Y}_i^2}{\pi}\right)$	$\mathsf{Y}_i \in [0, \pi]$ $(0 \leq i < l)$
Rosenbrock	**Minimize** $\sum_{i=0}^{l-2}100(\mathsf{Y}_{i+1} - \mathsf{Y}_i^2)^2 + (1 - \mathsf{Y}_i)^2$	$\mathsf{Y}_i \in [-5.12, 5.12]$ $(0 \leq i < l)$
Summation Cancellation	**Maximize** $100/(10^{-5} + \sum_{i=0}^{l-1}\lvert\gamma_i\rvert)$ **Where** $\gamma_0 = \mathsf{Y}_0, \gamma_i = \mathsf{Y}_i + \gamma_{i-1}$	$\mathsf{Y}_i \in [-3, 3]$ $(0 \leq i < l)$

Figure 5.7: Numerical optimization test problems.

a total of twelve problem instances to test the optimization algorithms on. To obtain an assessment of the performance and suitability of IDEAs and GLIDEs for numerical optimization, we compare the newly tested algorithms with other well–known evolutionary algorithms such as the simple GA and three variants of ES, but also a more traditional approach in the form of a random restart conjugate gradient algorithm. Since ES are commonly accepted to be among the state–of–the art in numerical BBO optimization, such a comparison will give a valuable indication of the potential of real–valued IDEAs and GLIDEs.

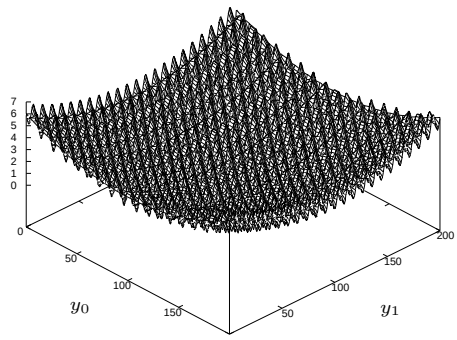
In Section 5.3.1 we describe our test problems and discuss their properties to indicate their specific difficulties. In Section 5.3.2 we describe our experiment setup and in Section 5.3.3 we present the obtained results. Finally, in Section 5.3.4 we give a short summary for the EA practitioner.

5.3.1 Numerical optimization problems

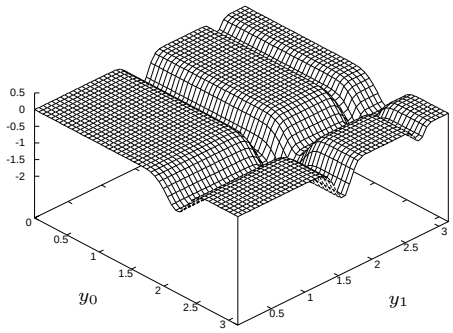
Our test suite consists of four problems that are defined for a general dimensionality of l. These four problems represent a variety of difficulties in numerical optimization. The definitions of the numerical optimization problems are given in Figure 5.7. For an intuitive impression of the characteristics of the problems in our test suite, two–dimensional surface plots are provided in Figure 5.8.

Griewank

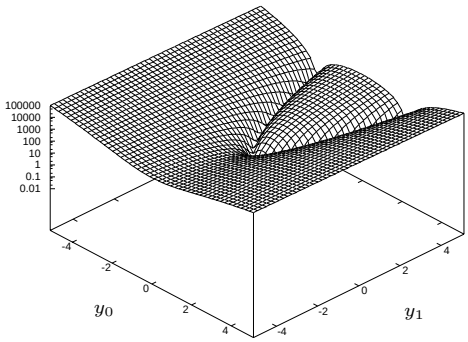
Griewank's function is a function with many local optima. Basically, it is a parabola superimposed with a sine function to obtain many local optima. As a result, if large steps are taken in Griewank's function, the so observed coarse–grained gradient information will quickly lead to a region close to the minimum of the parabola. However, if only small steps are taken, the many local optima will prevent efficient optimization of this function, even when a random restart strategy is used. Furthermore, for a fixed precision, Griewank's function becomes easier to optimize as l increases if large steps are taken. The minimum value for Griewank's function for any dimensionality is 0, which is obtained if all y_i are set to a value of 100.



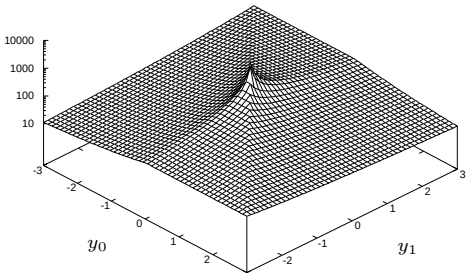
Griewank



Michalewicz



Rosenbrock



Summation cancellation

Figure 5.8: Two–dimensional surface plots for the four numerical optimization problems in the test suite. The ranges for Griewank's function were zoomed to get a better indication of the many local optima. Rosenbrock's function and the summation cancellation function are shown on a logarithmic scale for a better impression of their problem features. Note that the summit of the peak for summation cancellation is actually 10^7, but the drawing resolution prohibits accurate visualization thereof.

Michalewicz

Michalewicz's function is also a function with many local optima, albeit to a lesser degree than Griewank's function. An important difference is that Michalewicz's function has many long channels along which the minimum value throughout the channel is the same. The gradient information in such a channel therefore does not lead to the better local optima which are found at the intersections of the channels. Proper optimization of Michalewicz's function is therefore only possible if the channels of equal optimization value are explored or covered fully to find the intersections. The minimum value for Michalewicz's function depends on its dimensionality. A description of its solutions at which the minimum value is obtained for any dimensionality, has not been reported in the literature.

Rosenbrock

Rosenbrock's function is highly non–linear. It has a curved valley along which the quality of the solutions is much better than in its close neighborhood. Furthermore, this valley has a unique minimum of 0 itself for any dimensionality of Rosenbrock's function, which is obtained if all y_i are set to a value of 1. Rosenbrock's function has proven to be a real challenge for any numerical optimization algorithm. The gradient along the bottom of the non–linear valley is very slight. Any gradient approach is therefore doomed to follow the long road along the bottom of the valley, unless a starting point is provided in the vicinity of the optimum. Furthermore, since the valley is non–linear, simple gradient based approaches will oscillate from one side of the valley to the other, which does not result in efficient gradient exploitation. For an IDEA, capturing the valley in a probabilistic model is difficult, even if all of the points within the valley are known. The reason for this is that the valley is non–linear in the coding space. Therefore, it is to be expected that in order to be able to get any reasonable results, we are required to use mixture probability distributions.

Summation cancellation

The summation cancellation was proposed by Baluja and Caruana (1995). This optimization problem has multivariate linear interactions between the problem variables. This should allow algorithms that are capable of modelling linear dependencies to outperform algorithms that are not capable of doing so. Furthermore, the degree of multivariate interaction is as large as possible since each γ_i in the problem definition is defined in terms of all y_j with $j < i$. Finally, the optimum is located at a very sharp peak, which implies that the optimization algorithm needs to have a large precision and needs to be able to prevent premature convergence in order to reach the global optimum.

5.3.2 Experiment setup

Optimization problem dimensionalities

We varied the dimensionality for each problem in the test suite described in section 5.3.1 to get an indication of the applicability of each algorithm as the number of problem variables increases. To be precise, we have set the dimensionality for the test problems to $l \in \{5, 25, 100\}$.

General algorithmic setup

We ran tests for all algorithms to find the best results within a maximum of 10^6 evaluations and a maximum population size of 10^5. If all of the selected solutions differed by less than $5 \cdot 10^{-7}$, termination was enforced. Since we allow for a maximum of function evaluations, there is a value for n at which an algorithm will perform best. For too large population sizes, the search will move towards a random search and for too small population sizes, there is not enough information to perform adequate model learning and induction. We therefore ran tests so as to find the population size at which the best optimization performance is obtained. All results were averaged over 10 independent runs.

Algorithms

We have tested a variety of optimization algorithms. In the following, we shall briefly describe the different settings that we have used for these algorithms.

Simple GA
We have applied the simple GA by encoding each real–valued variable with 30 bits. Note that this number of bits is relatively small compared to the standard of 64 bits used to encode real–valued variables in programming languages today. However, the binary genotypic dimensionality is equal to $30l$, which already amounts to 3000 bits for a 100–dimensional numerical optimization problem instance. Finding problem structure among this many problem variables can be extremely time–consuming. Therefore, we chose a compromise precision encoding of 30 bits. We have instantiated the simple GA with one–point crossover as well as uniform crossover. We furthermore used truncation selection with a truncation percentile of $\tau = 0.3$ and used elitism such that the selected solutions are always copied into the population of the next generation. The recombination operators are used to fill the remainder of the population with new offspring genotypes. We set the probability at recombination to $p_r = 1$. The probability of mutation was set to $p_m = 0$.

ES
We have also tested three ES variants. ES are especially well suited for numerical optimization. The genotype in ES is a vector of real values. In ES, a normal probability distribution with zero mean is maintained for each genotype.

To mutate a genotype, a sample is drawn from the probability distribution associated with the genotype and added to the genotype. Thus, the probability distribution effectively encodes a probabilistic search direction. Moreover, these probabilistic search directions are evolved together with the solutions, because the covariances of the normal distribution are also stored in the genotype and are subject to specific recombination and mutation operations. This allows ES to induce search directions adaptively during optimization and to exploit local (gradient) features of the search space. This is especially beneficial if the function to be optimized is differentiable (Bäck and Schwefel, 1993).

The three ES variants that we have used for testing differ in the way mutation is performed. In the first variant, the l–dimensional normal distribution that represents the mutation probability distribution, is represented by only a single variance value such that the covariance matrix of the normal distribution is diagonal and the variance in each direction is identical. In the second ES variant, l variances are used to represent a diagonal covariance matrix with possibly differing values on the diagonal. In the third and final ES variant, a full symmetric covariance matrix is stored for each solution. The probability of recombination and mutation were set to $p_r = p_m = 1$. Recombination was performed by applying intermediate crossover to the solution genes and discrete crossover to the covariance matrix genes. Finally, a (μ, λ) selection strategy based on truncation selection was used with a ratio of $\mu = \frac{1}{7}\lambda$. This means that each generation, the number of offspring that is generated is 7 times the size of the population. From the $7n$ offspring, the n best genotypes are then selected by truncation selection. The $\frac{1}{7}$ ratio is a commonly used rule of thumb for ES (Bäck and Schwefel, 1993).

IDEA

We used the rule of thumb by Mühlenbein and Mahnig (1999) for FDA and set the selection threshold τ to 0.3. We further used both real–valued as well as binary encoded IDEAs.

- **Real–valued**. We used IDEA instances based on the normal gpdf by learning Bayesian factorizations using the BIC metric. We also learned normal mixtures of Bayesian factorizations by applying clustering. For the task of clustering, we used the leader algorithm. The distance threshold in the leader algorithm was set to a value that led to 4 clusters on average. The mixture coefficients β_i were set to the proportional cluster sizes. For reasons explained in earlier sections in this chapter, we applied the normal kernels gpdf, the histogram gpdf and the normal mixture gpdf without factorization selection. For the normal kernels gpdf, we used both the univariate factorization as well as the full joint factorization. For both factorizations, we experimented with variance factors of $\alpha \in \{1, 10\}$. For the histogram gpdf, we only used the univariate factorization since the full joint factorization would lead to an intractable number of parameters. We furthermore used 5 bins in each dimension for the histogram gpdf. Finally, we used the EM algorithm to estimate both univariately factorized normal

mixture probability distributions as well as full joint factorized normal mixture probability distributions.

- **Binary encoded**. For the binary encoding with 30 bits per real–valued problem variable, we instantiated the IDEA framework with the optimal dependency tree algorithm by Chow and Liu (1968) to learn tree factorizations. Since the number of binary random variables increases dramatically with the problem dimensionality, no higher–order factorization learning algorithms were applied.

ℝCG
Next to pure EAs, we also experimented with a classical gradient–based approach. More precisely, we have used the conjugate gradient algorithm (Hestenes and Stiefel, 1952; Press et al., 1992). Since we have no prior information on promising regions in the search space at which to start the conjugate gradient algorithm, the conjugate gradient algorithm is repeatedly started at a random starting point. The minimum value that is reached, is recorded in combination with the overall number of evaluations that have been required so far.

GLIDE
Lastly, we also tested the GLIDE instance that we proposed in Section 5.2. In GLIDE, we allowed the conjugate gradient algorithm to run for 10 iterations each time it was called. We set $\tau_G \in \{0.05, 0.25\}$. We computed the gradient information by using $\Delta y_i = 10^{-13}$. Furthermore, we have used the Polak–Ribiere variant of the conjugate gradient algorithm (Press et al., 1992). We have combined the hybrid application of the conjugate gradient algorithm with the learning of Bayesian factorized normal gpdfs and the learning of mixtures of Bayesian factorized normal gpdfs by means of clustering. Furthermore, we also used the univariate factorization and the full joint factorization based on the normal gpdf. The generalization property of the normal gpdf that leads to a relatively *global* type of search is expected to form a good combination with the use of the *local* efficient conjugate gradient optimization algorithm.

5.3.3 Results

For each optimization problem and each dimensionality, we computed for each algorithm the average and standard deviation of the best fitness and required number of evaluations upon termination over 10 runs. The averages for the population sizes that led to the best results, are tabulated in Appendix B Section B.1 (Figures B.1 up to and including B.6). For each IDEA and each GLIDE instance the superscript indicates which gpdf was used, whereas the subscript indicates the probabilistic model structure that was used. For the simple GA instances, the superscript indicates the type of crossover that was used. For the ES variants, the superscript indicates how many covariance values were used for adaptation.

To determine the true relative performance ordering of the tested algorithms with respect to optimization performance, we performed Aspin–Welch–Satterthwaite (AWS) statistical hypothesis T–tests at a significance level of $\alpha = 0.05$. The AWS T–test is a statistical hypothesis test for the equality of means in which the equality of variances is not assumed. For each problem and each dimensionality, we statistically verified whether the average fitness obtained differs significantly. We assigned a value of 1 if an algorithm scored significantly better and -1 if an algorithm scored significantly worse. If no statistical significance with respect to the difference in average fitness could be determined, the average number of evaluations was used instead. We summed the so obtained matrices over all problems and dimensionalities to obtain the statistically significant improvement matrices that are shown in Appendix B Section B.1 (Figures B.7 and B.8). For each dimensionality separately, we also computed for each algorithm the sum of the statistically significant improvement tests over all other algorithms to obtain an indicator of relative performance of the tested algorithms. These results are visualized for each dimensionality using histograms in Figure 5.9. In this figure, the average over all dimensionalities is also shown, which thus equals for each algorithm the sum of the statistically significant improvement tests over all other algorithms divided by three.

We will now investigate these results closer and draw some conclusions regarding the applicability of IDEAs and GLIDE–EAs to numerical optimization. We also highlight the most important summarized conclusions.

IDEA Instances

Small dimensionality

From the summary results in figure 5.9 the IDEAs that result in the best significant improvement sum with respect to all other EAs are the variants in which the normal gpdf is used in combination with either Bayesian factorization selection and clustering or the fixed joint factorization with clustering and the variant in which the histogram gpdf is used. The added use of clustering is clearly beneficial if the dimensionality of the optimization problem is small. The use of the EM algorithm to fit a normal mixture gpdf does however not lead to effective IDEAs. Only if the univariate factorization is used, the results are better than when a single normal gpdf is used. The problem of fine–tuning the α parameter for the normal kernels gpdf becomes clear from the overall results since although Figures B.1 and B.2 sometimes indicate good results for IDEAs based on the normal kernels gpdf, the same approach fails on other problems. Thus, without the possibility of adaptively controlling the α parameter or the shape of each individual kernel, this gpdf does not lead to efficient IDEAs.

It is also interesting to note that the use of tree factorization selection on the binary encoding of the real–valued variables in an IDEA leads to better results than if either uniform crossover or one–point crossover in the simple GA is used. Thus, there are important multivariate dependencies between the binary variables that can be exploited. However, because of the large number of binary random variables, this approach requires quite a lot of time.

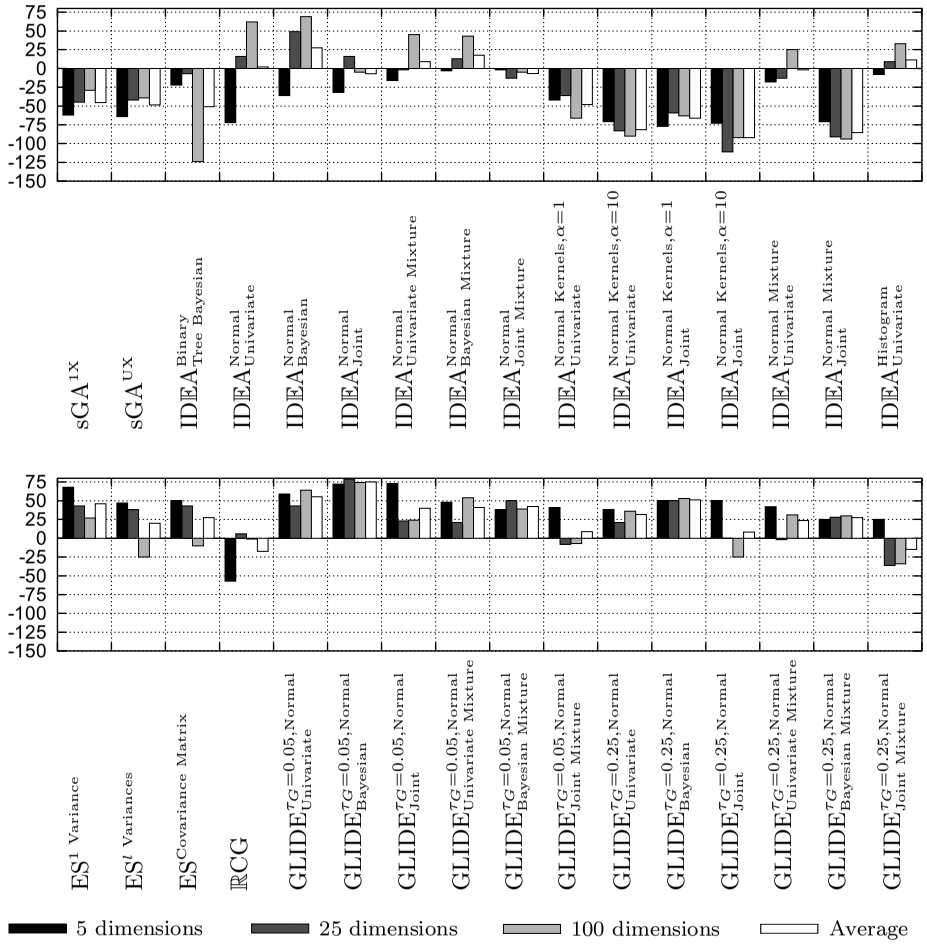


Figure 5.9: A summary of the results of the statistical hypothesis tests performed for each pair of algorithms. For each algorithm, the sum over all problems of the outcome of the statistical hypothesis tests is shown for each dimensionality separately. Furthermore, the average of these values is also shown, which serves as a global indicator of the performance of an algorithm relative to the other tested algorithms.

Medium dimensionality
As the dimensionality increases, the results start to become different. Clearly, the added use of clustering with a small number of clusters does not contribute as much as is the case if the dimensionality is smaller. The main reason for this is that in most cases, the maximum number of evaluations was reached. Before the added use of clustering can truly come into effect, the maximum number of evaluations has already passed. Moreover, it is to be expected that a larger number of clusters is required to effectively model dependencies for larger dimensional search spaces. However, this also implies that the population size has to increase in order to ensure an adequate number of solutions in each cluster. Therefore, clustering for larger dimensional spaces in an IDEA can cause us to lose more than we gain. Of the other real–valued gpdfs, again the histogram gpdf is the best alternative. However, detecting dependencies with Bayesian factorizations in combination with a normal gpdf is clearly the best choice.

It is again interesting to note that the use of the tree factorization leads to much better results than those obtained by the simple GA variants. Again this shows in general that an attempt to induce and exploit problem structure from previously evaluated solutions can be beneficial in optimization.

Large dimensionality
For numerical optimization problems with a large dimensionality, IDEAs using the normal gpdf are clearly the best. Again, the added use of clustering does not help to improve the results. What also becomes clear is that the use of the univariate factorization becomes more preferable as the dimensionality increases. This can be seen from the fact that, with the exception of the normal kernels gpdf, each IDEA in which the univariate factorization is used, obtains good results, regardless of the choice of gpdf. The main reason for this is that the maximum number of evaluations is reached quite quickly if the dimensionality is large, but the added value of using more sophisticated models may only become significantly visible after a larger number of evaluations such that the attractive regions of the optimization problem are located more precisely. Furthermore, to within our fixed precision, some optimization problems only become easier to solve as the number of dimensions is increased. This is for instance the case for Griewank's function. Therefore, simpler models are more likely to solve the problem within this fixed precision efficiently, giving them an added preference in the results of our benchmark.

The use of the tree factorization for the binary encoding seems to score terribly bad in Figure 5.9. The reason for this is that the time that is required to use this model on 3000 binary variables far exceeds any reasonable time limit for a benchmark such as this one. A single run that uses all 10^6 evaluations takes approximately 10^6 seconds on a 1GHz Intel Pentium–III processor.

Overall
We can summarize the most important observations regarding the optimization performance of IDEAs on real–valued, continuous and mostly smooth optimiza-

tion problems with possible multi–modality and non–linear interactions between problem variables, as follows:

- If the dimensionality is small (approximately $l \leq 25$), the normal gpdf combined with clustering and the Bayesian factorization selection in each cluster should be used.
- If the dimensionality is medium (approximately $25 \leq l \leq 100$), the normal gpdf combined with Bayesian factorization selection should be used.
- If the dimensionality is large (approximately $l \geq 100$), the normal gpdf should be used combined with Bayesian factorization selection or with the univariate factorization. Although using the Bayesian factorization selection is expected to give slightly better results, the univariate factorization will take considerably less time.

GLIDE Instances

If we only regard GLIDE instances, the results in Figure 5.9 indicate that using Bayesian factorization selection based on the normal gpdf leads to the best results, regardless of the dimensionality and the percentage of conjugate gradient local search. Furthermore, applying conjugate gradient local search to 5% of all selected solutions leads to better results than using a percentage of 25%. Thus, using only a small percentage of local search in GLIDE is preferable, regardless of the dimensionality or the gpdf and the probabilistic model structure.

Interestingly, the capability of Bayesian factorizations to model simple linear dependencies between variables is enough to exploit the global structure of the optimization problem if the gradient information of the optimization problem is exploited by a gradient optimization procedure. Moreover, it also truly adds something to the optimization process to model dependencies since the resulting GLIDE–EA outperforms the use of a univariate factorization.

Based on these observations and the strikingly unanimous results, we can directly summarize the most important observations regarding GLIDE. For the best expected performance Bayesian factorization selection should be combined with a gradient local search percentage of $\tau_G \approx 0.05$ in GLIDE. Note that this result only applies to the relative comparison of individual GLIDE variants. The true added value of exploiting gradient information by using the conjugate gradient algorithm as a local search procedure will be discussed next.

IDEA and GLIDE versus other EAs and RCG

Small dimensionality

The most remarkable thing to note from the results in Figure 5.9 is that, with the exception of the random restart conjugate gradient algorithm, all algorithms that explicitly (GLIDE–EAs) or implicitly (ES variants) exploit gradient information, have a better overall performance on the low–dimensional problems than the EAs that do not have a means to perform gradient exploitation. Real–valued continuous IDEAs are capable of detecting and modelling dependencies

between the problem variables and to give a global indication of the structure of the optimization landscape. Depending on the gpdf used and whether or not we are estimating mixture probability distributions, the type of dependencies that can be modelled range from linear to strongly non–linear. By drawing more samples from the estimated probability distributions, the modelled regions are investigated and explored more closely. For numerical optimization, this approach meets with two important troublesome issues.

1. By drawing samples from a probability distribution that was estimated over a set of samples, we obtain a set of samples that is relatively close to the set of samples that the probability distribution was estimated over. Therefore, if the search has already focused on a certain part of a region surrounding a local optimum, an IDEA approach is not likely to find the local optimum since it will converge onto a range near the one that the algorithm is currently already focusing on. The existence of this problem is exemplified by the fact that on Rosenbrock's function, solutions inside the narrow valley that contains the optimum can be found using even only a single normal gpdf. These solutions are likely to be focused on one or more subregions of the valley, upon which the search converges. An IDEA approach is not capable of following the gradient along the valley to the optimum, since points are not likely to be sampled outside the regions in which the current solutions reside. For this reason, the ES approach is often better suited for numerical optimization since it is essentially an adaptive normal kernels approach in which the goal is not to estimate the probability of the current available samples but to estimate the probability of the promising search directions, allowing the ES to adaptively exploit the local gradient structure of the optimization problem.

2. The second troublesome issue is closely related to the first one. Even if a region of interest that contains a local optimum is adequately sampled, an IDEA approach will not efficiently be able to locate the local optimum since the gradient information is ignored. In the extreme case in which a region of interest contains only a single optimum, using gradient information to locate this optimum is the most efficient approach since this information points directly towards the optimum.

Still, the use of real–valued continuous IDEA approaches for numerical optimization is a good way to globally indicate the structure and the presence of promising regions of the search space. We already saw that the use of Bayesian factorizations can indeed overall lead to the best expected performance of an IDEA. For this reason, allowing to learn more advanced probability distributions leads to a more efficient description of the presence of local optima. In this way, learning probability distributions can be beneficial for numerical optimization. However, as a result of not using gradient information at all, IDEAs are not the best approach if the optimization problem is (partly) continuous, which are the problems we focus on in this thesis. This is especially true for lower dimensional problems as is exemplified by the results in Figure 5.9.

The unique capability of continuous IDEAs to identify and model in a more global fashion the structure of the optimization landscape, makes them ideal candidates for providing good starting points from which to use local methods to exploit the gradient information and find local minima. The hybridization of the IDEA framework with a classical conjugate gradient approach in the GLIDE evolutionary algorithm does indeed lead to quite efficient results. These results are sometimes evenly matched by the ES. The main reason for this is clearly the fact that the ES has a gradient–oriented mutation operator. Still, for the type of optimization problem that we have used in our benchmark, the GLIDE approach is preferable.

A last thing to note for the low–dimensional case is that the classical random restart conjugate gradient approach is clearly not as efficient as the GLIDE approach or the ES approach. The added use of a population instead of only a single solution thus allows for more efficient exploitation of problem structure. It is interesting to note that although the IDEA instances and the $\mathbb{R}$CG algorithm are inferior to the ES, the combination of the global and local search found in these two separately inferior algorithms leads to a superior algorithm.

Medium dimensionality

If the dimensionality of the optimization problem is increased, using gradient information becomes more problematic. The reason for this is that if we want to explicitly estimate local gradients, we have to perform $\mathcal{O}(l)$ additional computations. If we want to extract gradient information during the optimization process implicitly such as the way in which this is done in the ES, it must therefore certainly also take more effort to extract gradient information, especially if a full covariance matrix is used. Indeed, we can observe from figure 5.9 that the significant improvement of the ES variants over all other algorithms decreases. Note that, similar to the IDEA variants, the case in which only a single variance variable is used for each dimension in the ES keeps working relatively well. This can be explained in a similar manner as was done for the univariate factorization for the IDEA instances.

The success of the algorithms that do not use gradient information clearly improves with the increase in dimensionality. This is especially true for the IDEA based on the normal gpdf and Bayesian factorization selection. Moreover, for some problems it is easier to find the optimum to within the demanded precision. Furthermore, for problems such as the Rosenbrock function, the explicit use of gradient information will still significantly help optimization as the dimensionality of the problem is increased. This explains why the $\mathbb{R}$CG algorithm indicates a significant improvement over the results obtained for the small dimensionality case. For these reasons, the performance of the GLIDE–EA does not decrease as the dimensionality is increased to a medium size.

Large dimensionality

As the dimensionality is increased further, the decay of the success of ES due to the more costly exploitation of gradients can again be observed. At this point, the ES variants are at most as successful as the best IDEA instance. From

the summarized results in figure 5.9 it even seems that the GLIDE–EA based on the normal gpdf, Bayesian factorization selection and $\tau_G = 0.05$ is still the best algorithm. However, if we look at the pairwise statistical improvement matrix in Figures B.7 and B.8, we see that over all dimensionalities, this GLIDE variant is slightly inferior to the IDEA based on the normal gpdf and Bayesian factorization selection. The main reason for this is that with the exception of Rosenbrock's function, the IDEA approach outperforms the GLIDE approach for the largest dimensionality, whereas the GLIDE approach outperforms the IDEA approach for the smallest dimensionality. The reason for this is that the GLIDE approach explicitly has to compute gradients which becomes more expensive as the dimensionality is increased. Moreover, in our current GLIDE hybridization scheme, the conjugate gradient local search is *always* applied to a percentage of the solutions. Even if at times the IDEA part of the GLIDE approach is more efficient and the local search does not significantly add much to the optimization process, local search is still applied. If this could be circumvented, the GLIDE approach would be superior to all tested algorithms.

5.3.4 Practitioner's summary

We can summarize the most important observations regarding the optimization performance of the tested EAs on real–valued, continuous and mostly smooth optimization problems with possible multi–modality and non–linear interactions between problem variables, as follows:

- If the dimensionality is small (approximately $l \leq 25$), the GLIDE–EA in which the normal gpdf is used in combination with Bayesian factorization selection and the application of the conjugate gradient algorithm to 5% of all selection solutions should be used.
- If the dimensionality is medium (approximately $25 \leq l \leq 100$), either the IDEA in which the normal gpdf is used combined with Bayesian factorization selection or the GLIDE–EA in which the normal gpdf is used in combination with Bayesian factorization selection and the application of the conjugate gradient algorithm to 5% of all selection solutions should be used.
- If the dimensionality is large (approximately $l \geq 100$), either the IDEA in which the normal gpdf is used combined with Bayesian factorization selection or the IDEA in which the univariate factorization is used with the normal gpdf should be used. Although using the Bayesian factorization selection is expected to give slightly better results, the univariate factorization will take considerably less time.

The GLIDE–EA performs the best on average. To implement this algorithm, the IDEA framework needs to be instantiated with the greedy learning of Bayesian factorizations. The details of this can be found in Section 2.6. Moreover, the gpdf that should be used is the normal gpdf. The details of this gpdf

regarding its estimation from data and the drawing of new samples are given in Section 5.1.1. Finally, the random restart conjugate gradient algorithm needs to be implemented to make the IDEA implementation hybrid and obtain the GLIDE–EA (Section 5.2). The details required for implementing the random restart conjugate gradient algorithm are best described by Press et al. (1992).

The best alternative to the GLIDE–EA and the best non–hybrid EA for numerical optimization is still the ES. The details required for implementing the most common ES variants are best described by Bäck and Schwefel (1993).

5.4 Discussion and future research

The combination of the IDEA with explicit gradient information exploitation in the GLIDE framework has been shown to result in efficient algorithms for numerical optimization. A combination of IDEA with the gradient–oriented mutation operator in ES would most likely also yield a powerful mix of global and local search for numerical optimization. A first attempt to do so was recently done by Pelikan, Goldberg and Tsutsui (2002). Although the benchmark used was completely different and not as complete as the one presented in this chapter, the results were encouraging. One of the issues that needs to be researched further in this context, is how much local search should be applied at any point during the optimization process. One way of doing so, is to set the amount of local search adaptively based on the average rate of success of using local search versus global search. Another issue is that although IDEAs can indeed globally indicate the structure of the optimization problem and as such are a good candidate for hybridization with local search methods such as the conjugate gradient algorithm, the efficiency of the other EAs might also be increased by using the same hybridization scheme. For completeness, the algorithms so obtained should also be tested to determine whether GLIDE–EAs are truly one of the most efficient new tools for numerical optimization.

We have introduced clustering as a means of allowing the modelling of non–linear interactions between problem variables. Our experimental results indicate that mixture probability distributions allow for the construction of more efficient IDEAs for numerical optimization, especially if the dimensionality of the optimization problem is small. However, we have only experimented with a predefined number of clusters. It would be better to be able to compute the number of clusters that are required adaptively so as to meet the actual modelling needs for a given set of samples. One way of doing so is to use a penalization metric and compute the metric difference if the number of clusters is increased. However, this leads to very large requirements on the computation time since a factorization has to be estimated in each cluster anew. Therefore, alternative methods are required. Another important question is whether and how the added effort of clustering is affected by the increase of the dimensionality if the number of clusters is not fixed beforehand. A study in which the scaling behavior of the required number of clusters is investigated, would be interesting.

'To stay ahead, you must have your next idea waiting in the wings.'
Rosabeth Moss Kanter

Permutation optimization

In this chapter, we focus on the use of the IDEA paradigm for solving permutation optimization problems. On the practical side, important real–life problems such as scheduling and the traveling salesman problem (TSP) are permutation optimization problems. On the theoretical side, the search space in these problems grows *factorially* as the number of problem variables increases. Furthermore, the search space consists of permutations, which is fundamentally different from the binary or real spaces that most EAs have been designed for. As a result, only a few very recent attempts at induction of problem structure features during optimization are known in permutation optimization with EAs.

The chapter is organized as follows. In Section 6.1, we describe two ways of encoding permutations in a genotype and discuss a few related permutation optimization EAs that provide a basis for comparison in the light of black box optimization. Subsequently, in Section 6.2, we focus on problem difficulty in permutation optimization and introduce some problems for which traditional permutation EAs, that are not capable of competently detecting and exploiting problem structure features, display exponential scale–up behavior. In Section 6.3 we show how multivariate factorizations can be estimated for permutation random variables. Using such factorizations, new IDEAs can be constructed for permutation optimization. In Section 6.4, we discuss how the exploitation of problem structure in such IDEAs may be improved by introducing crossover operators that are guided by probabilistic model learning. Finally, we empirically investigate the optimization performance of a variety of IDEAs on two difficult permutation optimization problems in Section 6.5 and observe that for the problems on which traditional EAs scale–up exponentially, the scale–up behavior for almost all tested IDEAs is polynomial.

6.1 Evolutionary algorithms and genotypes for permutations

In most EAs, the domain of the genotype is a Cartesian product space, that is, $\mathbb{G} = \times_{i=0}^{l-1}\mathbb{A}_i$. Examples are the simple GA and the binary IDEAs in chapters 3 and 4, for which we have that $\mathbb{G} = \times_{i=0}^{l-1}\mathbb{B}$. A permutation of length l is a vector of l unique objects. For simplicity, we assume that each of these objects is an integer and that the permutations are *compact*. We say that a permutation of length l is *compact* if it is a permutation of $(0, 1, \ldots, l-1)$. There are a large number of interesting and real–life practical problems for which the solutions are naturally represented by permutations. Examples of such problems are scheduling and routing problems. Unfortunately, the space of permutations can not be represented by a Cartesian–product space without adding constraints.

In this section we discuss two genotypes that encode permutations. For both genotypes we mention a few EAs that do not use problem specific information and thus target black box permutation optimization problems. With the exception of a single EA, none of these EAs explicitly attempts to induce problem structure features during optimization and are thus similar to simple binary GAs in which uniform crossover or one–point crossover is used. Since we know from Chapter 4 that the simple GA scales up exponentially on a class of problems for which we typically want a GA to scale up polynomially, we want to investigate whether existing permutation EAs have similar limitations. For this reason, we mention a few of the most common permutation EAs.

6.1.1 Integer permutation genotype

The integer permutation genotype was the first encoding of permutations to be used in EAs. This encoding is still the most frequently used genotype because the encoding is simply a permutation itself.

The genotype

The space of all integer permutation genotypes is defined by

$$\mathbb{G} = perm(\boldsymbol{\mathcal{L}}) = \tag{6.1}$$

$$\left\{ \boldsymbol{g} \;\middle|\; \boldsymbol{g} \in \times_{i=0}^{l-1}\{0, 1, \ldots, l-1\} \;\wedge\; \forall (i,j) \in \boldsymbol{\mathcal{L}} \times \boldsymbol{\mathcal{L}} : j \neq i \rightarrow \boldsymbol{g}_i \neq \boldsymbol{g}_j \right\}$$

The additional constraint on the Cartesian product in the above definition enforces that no integer can appear twice in the genotype. The decoding function $\mathfrak{D} : \mathbb{G} \rightarrow \mathbb{P}$ that is associated with mapping an integer permutation genotype $\boldsymbol{g} \in \mathbb{G}$ to an integer permutation solution $\boldsymbol{p} \in \mathbb{P}$ is the identity function, that is, $\mathfrak{D}(\boldsymbol{g}) = \boldsymbol{g}$ $(= \boldsymbol{p})$.

Related EAs

Because an additional constraint is required on the Cartesian–product space in the integer permutation genotype, the application of classical crossover operators, such as one–point crossover, is useless. Using classical crossover operators genotypes can be constructed that do not represent a permutation. For instance, if the parent genotypes are $(0, 1, 2, 3)$ and $(3, 2, 1, 0)$, an application of one–point crossover with a crossover point in the middle may result in two offspring genotypes $(0, 1, 1, 0)$ and $(2, 3, 3, 2)$, both of which are not permutations. To ensure feasibility of the offspring genotypes, specialized recombination and mutation operators have been designed that ensure that the offspring genotypes are feasible permutations. Some of these operators have been designed for specific permutation optimization problems using domain knowledge. However, we are interested in the case in which we are unaware of such additional information. The four most commonly known permutation recombination operators are the partially mapped crossover (PMX) operator by Goldberg and Lingle, Jr. (1985), the cycle crossover (CX) operator by Oliver, Smith and Holland (1987), the order crossover (OX) operator by Davis (1985) and the edge map recombination (ER) operator by Whitley, Starkweather and Fuquay (1989). Although this last recombination operator does not use any specific problem knowledge, it was designed with the TSP in mind because it specifically focuses on processing information regarding successors and predecessors of integers with respect to the ordering in the permutation.

6.1.2 Random keys genotype

An alternative approach to designing EAs for permutation optimization, is to use a different encoding of permutations such that well known crossover operators can straightforwardly be applied. The most successful example of such an encoding is the *random keys* encoding, which was proposed by Bean (1994).

The genotype

The random keys encoding of a permutation of length l is a string of l real values. The space of all random keys genotypes is defined by

$$\mathbb{G} = \times_{i=0}^{l-1} \mathbb{R} \tag{6.2}$$

The decoding function $\mathfrak{D} : \mathbb{G} \to \mathbb{P}$ that is associated with mapping a random keys genotype $\boldsymbol{g} \in \mathbb{G}$ to an integer permutation $\boldsymbol{p} \in \mathbb{P}$ in the parameter space is such that integer i occurs before integer j in $\boldsymbol{p}$ if and only if $\boldsymbol{g}_i < \boldsymbol{g}_j$. In practice, each real value in the random keys genotype is usually defined to be in $[0, 1]$ instead of in the whole space of real values. Since there are no additional constraints on the random keys genotype, the main advantage of the random keys genotype is that by crossing over the real values, no genotype can be constructed that doesn't represent a permutation.

In the remainder of this chapter, we will denote a random keys genotype by $\boldsymbol{r}$ instead of $\boldsymbol{g}$. A gene at locus i in a random keys genotype $\boldsymbol{r}$ is called a random key $\boldsymbol{r}_i$. From the definition of the associated decoding function it follows that the value of a random key $\boldsymbol{r}_i$ determines the *position* of the integer i in the decoded permutation $\boldsymbol{p}$. To actually decode a random keys genotype, the random keys can be sorted in ascending order. We denote this by $\boldsymbol{p} = \sigma(\boldsymbol{r})$, where σ denotes the ascending sorting function such that $\boldsymbol{r}_{\boldsymbol{p}_0} < \boldsymbol{r}_{\boldsymbol{p}_1} < \ldots < \boldsymbol{r}_{\boldsymbol{p}_{|\boldsymbol{r}|-1}}$.

> **Example 6.1.** Consider the random keys genotype $\boldsymbol{r} = (0.61, 0.51, 0.62, 0.31)$. The integer permutation that is associated with $\boldsymbol{r}$ is $\sigma(\boldsymbol{r}) = (3, 1, 0, 2)$. Random key $\boldsymbol{r}_2$, which is assigned the value 0.62, determines that the integer 2 in the integer permutation is placed at the end of the permutation because 0.62 is the largest value in $\boldsymbol{r}$.

Note that for any permutation $\boldsymbol{p}$, the sorting operation $\sigma(\boldsymbol{p})$ tells for each i the position $\sigma(\boldsymbol{p})_i$ at which to find the integer of rank i in $\boldsymbol{p}$. Thus, if the integer at position j in $\boldsymbol{p}$ has rank i, then $\boldsymbol{p}_{\sigma(\boldsymbol{p})_i} = \boldsymbol{p}_j$. Now suppose that $\boldsymbol{p}$ is a compact permutation. Then each integer in $\boldsymbol{p}$ also occurs in $\sigma(\boldsymbol{p})$ because $\sigma(\boldsymbol{p})$ is also a compact permutation. Moreover, the rank of the integer at position j in $\boldsymbol{p}$ is just that integer itself, that is $\boldsymbol{p}_j$, which gives $\boldsymbol{p}_{\sigma(\boldsymbol{p})_{\boldsymbol{p}_j}} = \boldsymbol{p}_j$ and thus $\sigma(\boldsymbol{p})_{\boldsymbol{p}_j} = j$. This means that integer j can be found at position $\boldsymbol{p}_j$ in $\sigma(\boldsymbol{p})$. If we now repeat the meaning of σ, we find that $\sigma(\sigma(\boldsymbol{p}))$ tells for each i the position $\sigma(\sigma(\boldsymbol{p}))_i$ at which to find the integer of rank i in $\sigma(\boldsymbol{p})$. As noted earlier, the rank of i in any compact permutation is just i and thus $\sigma(\sigma(\boldsymbol{p}))$ tells for each i the position $\sigma(\sigma(\boldsymbol{p}))_i$ at which to find i in $\sigma(\boldsymbol{p})$. But we also know that the position at which to find i in $\sigma(\boldsymbol{p})$ is just $\boldsymbol{p}_i$ and thus we obtain the interesting property that for any compact permutation $\boldsymbol{p}$ we have $\sigma(\sigma(\boldsymbol{p})) = \boldsymbol{p}$.

Related EAs

If the simple GA is combined with the random keys encoding, a simple *Random Keys Genetic Algorithm* (RKGA) is obtained, which is the EA that was proposed by Bean (1994) for permutation optimization. The results by Bean (1994) indicate that the use of random keys in combination with one–point crossover gives good results on a variety of permutation problems. However, the resulting EA is less well suited for the TSP. This result is an important drawback of using classical crossover operators in combination with the random keys genotype. We will discuss the cause of this important remark in section 6.6.

One notable exception to all other previously mentioned general permutation optimization EAs, is the *Ordering Messy Genetic Algorithm* (OmeGA) by Knjazew and Goldberg (2000). This algorithm is essentially a fast messy GA that has been adapted to work with random keys. As such, it is the first EA for permutation spaces that attempts to induce structural features of the optimization problem and exploit them in a linkage–friendly fashion. The OmeGA was shown to have a good performance on problems in which the relative ordering of the integers in the permutation is important (Knjazew and Goldberg, 2000).

The relative ordering of integers i and j integer j corresponds to the question of whether i occurs before or after j in a permutation, regardless of how many integers are in between them.

6.2 Hard relative ordering problems for simple permutation EAs

Problem difficulty and deception of permutation optimization problems with respect to EAs was first investigated by Kargupta, Deb and Goldberg (1992). Briefly summarized, permutation EAs may be deceived in optimization in a similar fashion as is the case for binary GAs. In this thesis, we focus on the case in which the *relative* ordering of the gene loci is important and investigate *relative* deception. To this end, we first introduce the notion of relative order schemata. Subsequently, we discuss relative deceptive permutation optimization functions on the basis of which we define two permutation optimization problems based that we will use for experimentation.

6.2.1 Relative order schemata

One of the driving forces behind crossover–based GAs is the expectation that the recombination operator will copy sets of related genes from a single parent to an offspring genotype to preserve important partial solution information. As was indicated for binary representations in Chapter 3, depending on the actual implementation of the recombination operator, a varying spectrum of schemata may be processed efficiently in the GA. For the permutation representation, we find quite similar properties. Most recombination operators that are not problem–specific tend to preserve the some of the relative ordering of the integers in the permutation and derive integers for the remainder of the positions in the permutation in such a way that the resulting genotype is a permutation. Using random keys and one–point crossover for instance, the relative ordering of the loci in both parents on either side of the crossover point is transferred to the offspring since the random key values are themselves copied. Using the uniform crossover operator on the other hand has a strong tendency to preserve little relative ordering information similar to the fact that the uniform crossover operator for binary variables does not process multivariate dependencies efficiently. Moreover, if integers in the permutation, or equivalently, random key values at specific gene locations that are dependent on each other such that their relative ordering is of importance, are not inherited from a single parent in recombination, schemata that represent their relative ordering are likely to be disrupted. This is for instance the case if the dependent genes are spread throughout the solution and one–point crossover is used or if uniform crossover is used in general. If this spreading is combined with a deceptive fitness component for the related genes, exponential scale–up behavior can be expected for simple RKGAs, because the same behavior occurs in the case of binary variables as was indicated in Section 3.2.4.

To analyze this expected behavior more precisely, we introduce the notion of relative order schema (Goldberg and Lingle, Jr., 1985). A relative order schema $\boldsymbol{h}$ of length k and order $o(\boldsymbol{h})$, is a vector of length $|\boldsymbol{h}|$ such that the subvector $(\boldsymbol{h}_0, \boldsymbol{h}_1, \ldots, \boldsymbol{h}_{o(\boldsymbol{h})-1})$ is a permutation of $o(\boldsymbol{h})$ unique integers from $\{0, 1, \ldots, k-1\}$. Moreover, the remainder of $\boldsymbol{h}$ is filled with wildcard symbols !. A permutation $\boldsymbol{p}$ of length k is said to match a relative order schema $\boldsymbol{h}$, denoted $\boldsymbol{p} \in \boldsymbol{h}$, if and only if the relative ordering of the integers in $\boldsymbol{h}$ is the same as the relative ordering of the same integers in $\boldsymbol{p}$. The !–symbols in a relative order schema $\boldsymbol{h}$ indicate that these symbols may be replaced using exactly those integers in $\{0, 1, \ldots, k-1\}$ that have not yet been used in $\boldsymbol{h}$. Moreover, contrary to what the notation suggests, these numbers may be placed *anywhere* among the non–wildcard symbols, since only the *relative* ordering of the non–wildcard symbols is of importance.

Example 6.2. For $k = 5$, the relative order schema $h = (0, 1, 4, !, !)$ represents all of the following $\binom{5}{2} = 20$ permutations:

(0,1,4,2,3)	(0,1,2,4,3)	(0,1,2,3,4)	(0,2,1,4,3)	(0,2,1,3,4)
(0,2,3,1,4)	(2,0,1,4,3)	(2,0,1,3,4)	(2,0,3,1,4)	(2,3,0,1,4)
(0,1,4,3,2)	(0,1,3,4,2)	(0,1,3,2,4)	(0,3,1,4,2)	(0,3,1,2,4)
(0,3,2,1,4)	(3,0,1,4,2)	(3,0,1,2,4)	(3,0,2,1,4)	(3,2,0,1,4)

6.2.2 Relative–ordering deceptive problems

Similar to the case of binary variables, relative order *deceptive* permutation optimization problems of order k can be constructed by ensuring that for each relative order schema of order $1 < o(\boldsymbol{h}) < k$, the average relative order schema fitness is better for a relative order schema that contains a substring of the suboptimal permutation than for a relative order schema concerning the same integers such that they form a substring of the optimal permutation. In addition, the average relative order schema fitness for schemata of order $o(\boldsymbol{h}) = k$ must be the best for the optimal permutation.

Example 6.3. For $k = 5$, let the optimal permutation be $(0, 1, \ldots, 4)$ and the suboptimal permutation be $(4, 3, \ldots, 0)$. Let $\boldsymbol{j}_0, \boldsymbol{j}_1, \ldots, \boldsymbol{j}_{o(\boldsymbol{h})-1}$ be $o(\boldsymbol{h})$ unique integers from $(0, 1, 2, 3, 4)$. Without loss of generality, we can order these integers such that $\boldsymbol{j}_i < \boldsymbol{j}_{i+1}$ for all $i \in \{0, 1, \ldots, o(\boldsymbol{h}) - 2\}$. The average relative order schema fitness for a relative order schema of order $1 < o(\boldsymbol{h}) < 5$ containing integers $\boldsymbol{j}_0, \boldsymbol{j}_1, \ldots, \boldsymbol{j}_{o(\boldsymbol{h})-1}$ should be better for schema $(\boldsymbol{j}_{o(\boldsymbol{h})-1}, \boldsymbol{j}_{o(\boldsymbol{h})-2}, \ldots, \boldsymbol{j}_0, !, !, \ldots, !)$, which contains a substring of the suboptimal permutation, than for schema $(\boldsymbol{j}_0, \boldsymbol{j}_1, \boldsymbol{j}_{o(\boldsymbol{h})-1}, \ldots, , !, !, \ldots, !)$, which contains a substring of the optimal permutation. Let $\eta = 1/\binom{5}{3}$. We then get for $o(\boldsymbol{h}) = 2$, for instance:

- $\eta \sum_{\boldsymbol{p} \in (0,1,!,!,!)} \mathfrak{F}(\zeta_{(0,1,2,3,4)})(\boldsymbol{p}) < \eta \sum_{\boldsymbol{p} \in (1,0,!,!,!)} \mathfrak{F}(\zeta_{(0,1,2,3,4)})(\boldsymbol{p})$
- $\eta \sum_{\boldsymbol{p} \in (0,2,!,!,!)} \mathfrak{F}(\zeta_{(0,1,2,3,4)})(\boldsymbol{p}) < \eta \sum_{\boldsymbol{p} \in (2,0,!,!,!)} \mathfrak{F}(\zeta_{(0,1,2,3,4)})(\boldsymbol{p})$
- $\eta \sum_{\boldsymbol{p} \in (0,3,!,!,!)} \mathfrak{F}(\zeta_{(0,1,2,3,4)})(\boldsymbol{p}) < \eta \sum_{\boldsymbol{p} \in (3,0,!,!,!)} \mathfrak{F}(\zeta_{(0,1,2,3,4)})(\boldsymbol{p})$
- $\eta \sum_{\boldsymbol{p} \in (0,4,!,!,!)} \mathfrak{F}(\zeta_{(0,1,2,3,4)})(\boldsymbol{p}) < \eta \sum_{\boldsymbol{p} \in (4,0,!,!,!)} \mathfrak{F}(\zeta_{(0,1,2,3,4)})(\boldsymbol{p})$
- $\eta \sum_{\boldsymbol{p} \in (1,2,!,!,!)} \mathfrak{F}(\zeta_{(0,1,2,3,4)})(\boldsymbol{p}) < \eta \sum_{\boldsymbol{p} \in (2,1,!,!,!)} \mathfrak{F}(\zeta_{(0,1,2,3,4)})(\boldsymbol{p})$
- $\eta \sum_{\boldsymbol{p} \in (1,3,!,!,!)} \mathfrak{F}(\zeta_{(0,1,2,3,4)})(\boldsymbol{p}) < \eta \sum_{\boldsymbol{p} \in (3,1,!,!,!)} \mathfrak{F}(\zeta_{(0,1,2,3,4)})(\boldsymbol{p})$
- $\vdots \qquad < \qquad \vdots$

At order $o(\boldsymbol{h}) = 5$, the best relative order schema should be $(0, 1, \ldots, 4)$:

- $\mathfrak{F}(\zeta_{(0,1,2,3,4)})((0,1,2,3,4)) > \mathfrak{F}(\zeta_{(0,1,2,3,4)})((4,3,2,1,0))$

For such problems, the GA must disrupt the relative ordering of the integers in the relative order schema *as little as possible* during recombination and mutation in order not to be deceived by the lower order relative order schemata that lead to the suboptimal permutation $(4, 3, \ldots, 0)$.

Such a relative–ordering deceptive permutation problem was first constructed by Kargupta et al. (1992) for permutations of length 4. Knjazew (2000) proposed a permutation optimization problem for a variable length k and showed this problem to be deceptive for relative order schemata of lengths $k \in \{4, 5, 6\}$. The optimum integer permutation for the relative–ordering deceptive optimization of length k problem by Knjazew (2000) is $(0, 1, \ldots, k-1)$. To further specify the optimization problem, a distance measure for permutations is used. This distance from any permutation $\boldsymbol{p}$ to the optimum equals $k - |\textsc{lis}(\boldsymbol{p})|$, where $\textsc{lis}(\boldsymbol{p})$ is a *longest increasing subsequence* in $\boldsymbol{p}$. For example, if $\boldsymbol{p} = (1, 2, 0, 4, 3)$, then $\textsc{lis}(\boldsymbol{p}) \in \{(1, 2, 4), (1, 2, 3)\}$. Furthermore, $k - |\textsc{lis}(\boldsymbol{p})| = 5 - 3 = 2$. Note that the reverse permutation $(k-1, k-2, \ldots, 0)$ is the only permutation with a distance of $k - 1$. The relative–ordering deceptive optimization problem for $|\boldsymbol{j}|$ problem variables $\zeta_{\boldsymbol{j}_0}, \zeta_{\boldsymbol{j}_0}, \ldots, \zeta_{\boldsymbol{j}_{|\boldsymbol{j}|-1}}$ by Knjazew (2000) can now be written as follows:

$$\mathfrak{F}_{relative\_deceptive}(\zeta_{\boldsymbol{j}}) = \begin{cases} 1 & \text{if } |\textsc{lis}(\zeta_{\boldsymbol{j}})| = |\boldsymbol{j}| \\ 1 - \frac{|\textsc{lis}(\zeta_{\boldsymbol{j}})|}{|\boldsymbol{j}|} & \text{if } |\textsc{lis}(\zeta_{\boldsymbol{j}})| < |\boldsymbol{j}| \end{cases} \tag{6.3}$$

6.2.3 Decomposable relative–ordering deceptive test problems

Based on the relative–ordering deceptive permutation optimization problem, Knjazew (2000) defined several difficult decomposable relative–ordering deceptive permutation optimization problems. We will test our algorithms on two

of these problems. Both problems are a sum of the relative–ordering deceptive function $\mathfrak{F}_{relative\_deceptive}$, but only one of these two problems is additively decomposable. The sums can be defined using the index clusters that were introduced in chapter 3. The only difference for relative–ordering permutation optimization problems is that the index clusters are interpreted differently for integer permutations. Whereas for the binary encoding an index vector indicates the gene loci for which a subfunction is defined, for integer permutations an index vector indicates the integers for which their relative ordering is used in a subfunction. This means that in the i–th function in the sum it is the relative ordering in a given permutation $\boldsymbol{p}$ of the integers defined by $\boldsymbol{\iota}_i$ that determines the contribution of the i–th function in the sum. To obtain the subpermutation $\boldsymbol{p}^{\boldsymbol{\iota}_i}$ of $\boldsymbol{p}$ that contains only the integers in $\boldsymbol{\iota}_i$ such that the relative ordering in $\boldsymbol{p}$ of the integers in $\boldsymbol{\iota}_i$ is the same as the relative ordering in $\boldsymbol{p}^{\boldsymbol{\iota}_i}$ of the integers in $\boldsymbol{\iota}_i$, we can simply delete all other integers from $\boldsymbol{p}$ because $\boldsymbol{p}$ is a vector, which guarantees the preservation of ordering of non–deleted components. This deletion operation can be written as $\boldsymbol{p}^{\boldsymbol{\iota}_i} = \boldsymbol{p} - (\boldsymbol{\mathcal{L}} - \boldsymbol{\iota}_i)$. The general form of the two decomposable relative–ordering deceptive test problems that we will use can now be written as follows:

$$\mathfrak{F}(\zeta_{\boldsymbol{\mathcal{L}}}) = \sum_{j=0}^{|\boldsymbol{\iota}|-1} \mathfrak{F}_{relative\_deceptive}(\zeta_{\boldsymbol{\mathcal{L}}} - (\boldsymbol{\mathcal{L}} - \boldsymbol{\iota}_j)) \tag{6.4}$$

If the random keys genotype is used, the fitness function can be written directly as a function of the random keys. The relative ordering of the integers $\boldsymbol{\iota}_i$ in a permutation $\boldsymbol{p} = \sigma(\boldsymbol{r})$ is completely determined by the random keys $\boldsymbol{r}_{\boldsymbol{\iota}_i}$. As a result, we can compute the permutation $\boldsymbol{p}^{\boldsymbol{\iota}_i}$ that describes the required relative ordering by sorting only the random keys $\boldsymbol{r}_{\boldsymbol{\iota}_i}$, that is, $\boldsymbol{p} - (\boldsymbol{\mathcal{L}} - \boldsymbol{\iota}_i) = \sigma(\boldsymbol{r}_{\boldsymbol{\iota}_i})$. Since we shall indeed use the random keys genotype in our new IDEAs for permutation optimization, we state for completeness the general form of the fitness function as a function of the random keys gene variables $\mathsf{R}_i, i \in \boldsymbol{\mathcal{L}}$:

$$\mathfrak{G}(\mathsf{R}_{\boldsymbol{\mathcal{L}}}) = \sum_{j=0}^{|\boldsymbol{\iota}|-1} \mathfrak{F}_{relative\_deceptive}(\sigma(\mathsf{R}_{\boldsymbol{\iota}_j})) \tag{6.5}$$

In the two decomposable problems that we will use, each index cluster has length $\kappa^{\boldsymbol{\iota}}$. In the first problem, the index clusters are *mutually disjoint*, making the problem additively decomposable. To avoid the possibility that static crossover operators are biased in optimization because the random key genes contributing to each subfunction are closely located, the locations for each $\boldsymbol{\iota}_j^{loose}$ are chosen *loosely*, meaning as well spread as possible.

$$\boldsymbol{\iota}_j^{loose} = (j, j + |\boldsymbol{\iota}|, j + 2|\boldsymbol{\iota}|, \ldots, j + (\kappa^{\boldsymbol{\iota}} - 1)|\boldsymbol{\iota}|) \tag{6.6}$$

In the second permutation problem, the j–th index cluster *shares* its first position with index cluster $j-1$ and its last position with index cluster $j+1$. This overlapping property makes the problem *significantly* more difficult as there are

no clear index cluster boundaries and the decomposability is harder to detect. We will only used non–static crossover operators on the second permutation problem. Therefore we may, for simplicity of encoding the problem, encode the overlapping index cluster vector $\boldsymbol{\iota}^{overlap}$ tightly so that a random keys genotypes $\boldsymbol{r}$ represents an optimal solution if and only if $\sigma(\boldsymbol{r}) = (0, 1, \ldots, l-1)$.

$$\boldsymbol{\iota}_j^{overlap} = (j(\kappa^{\boldsymbol{\iota}} - 1), j(\kappa^{\boldsymbol{\iota}} - 1) + 1, j(\kappa^{\boldsymbol{\iota}} - 1) + 2, \ldots, j(\kappa^{\boldsymbol{\iota}} - 1) + \kappa^{\boldsymbol{\iota}} - 1) \quad (6.7)$$

6.2.4 Scale–up behavior of simple permutation EAs

Classical non–linkage friendly permutation recombination operators were shown to have great difficulty in optimizing an additively decomposable permutation optimization problem based on relative–ordering deceptive permutation subfunctions (Kargupta et al., 1992; Knjazew and Goldberg, 2000). Similar to the case of binary deceptive optimization problems and simple binary GAs, the main reason for this is that the (relative–ordering) linkage information is disrupted. The actual scale–up behavior of the discussed general permutation recombination operators, except the OmeGA, is shown on the loosely encoded additively–decomposable relative deceptive permutation problem of order $\kappa^{\iota} = 4$ in Figure 6.1. The use of any of these recombination operators scales up exponentially with respect to the minimally required population size and the required number of evaluations as the problem length increases. The reason for this is that these recombination operators are too disruptive with respect to the relative–ordering linkage information. The reason why even one–point crossover is not capable of polynomial scale up behavior is the presence of non–tight linkage in the random keys encoding. The edge map recombination operator is capable of efficiently processing neighboring information, but since only the relative ordering information is important, neighboring information is extremely uninformative.

The only permutation EA that was ever shown to not scale–up exponentially on the additively decomposable relative–ordering deceptive problem is the OmeGA. The OmeGA scales up subquadratically in the minimum required number of evaluations and population size (Knjazew and Goldberg, 2000). Since the OmeGA however is a variant of the fmGA, it is better suited for processing non–overlapping building blocks and less well suited for overlapping building blocks. Moreover, OmeGA does not use probabilistic models. Therefore it is interesting to investigate whether a similar or perhaps even bigger improvement over classical non–linkage learning permutation EAs can be obtained with an IDEA approach, which we turn to next.

6.3 Factorizations for random keys random variables

To design IDEAs for permutation optimization problems, we must estimate probability distributions in the space of permutations. We can do so directly

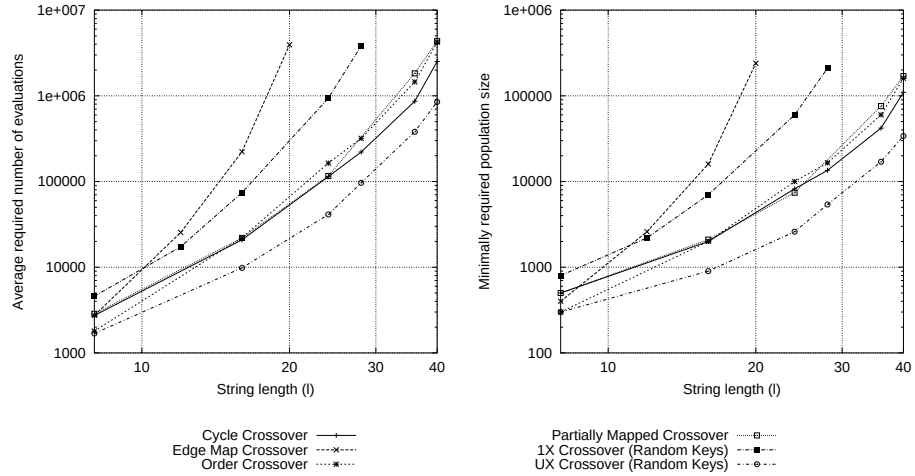


Figure 6.1: Scale–up behavior on the additively decomposable deceptive permutation problem with subproblems of length 4. The results are averaged over 30 runs and indicate the minimal computational requirements to find the optimal solution in all 30 runs. The results are plotted on a log–log scale. Straight lines on this scale indicate polynomial scale–up behavior. (truncation selection [$\tau = 0.3$]; recombination [$p_r = 0.5$, *crossover* $\in$ {*cycle*, *edge map*, *order*, *partially mapped*, *one–point (random keys)*, *uniform (random keys)*}]; no mutation; with elitism [*replace all non–selected solutions*]).

by introducing random variables that represent integer permutations. An alternative is to use the random keys representation. The advantage of using of random keys is that we can interpret them as real values or as an encoding of permutations. The latter interpretation leads to a similar situation as when random variables are introduced that represent integer permutations, but it turns out that random keys can conveniently be used to draw new samples from probability distributions for permutation random variables. In either way of interpretation, we introduce a random keys random variable R_i for each $i \in \boldsymbol{\mathcal{L}}$.

In the remainder of this section we focus on defining and estimating factorizations for permutations using the random keys representation. First, we show how this can be done by interpreting the random keys as real values. Second, we investigate the use of permutation random variables.

6.3.1 Interpreting random keys as real values

If the random keys are interpreted as real values, real–valued probability distributions such as the ones presented in Chapter 5 can directly be applied to obtain IDEAs for permutation optimization. This approach has been taken by Bosman and Thierens (2001b) as well as by Robles, de Miguel and Larrañaga (2001). The probability distributions used in these studies over $\boldsymbol{\mathcal{R}} = (R_0, R_1, \ldots, R_{l-1})$ are multivariate factorizations and Bayesian factorizations in combination with normal gpdfs:

$$\hat{P}^{\mathcal{N}}_{\boldsymbol{\nu}}(\boldsymbol{\mathcal{R}}) = \prod_{i=0}^{l-1} \hat{P}^{\mathcal{N}}(R_{\boldsymbol{\nu}_i}), \qquad \hat{P}^{\mathcal{N}}_{\boldsymbol{\pi}}(\boldsymbol{\mathcal{R}}) = \prod_{i=0}^{l-1} \hat{P}^{\mathcal{N}}(R_i | R_{\boldsymbol{\pi}_i}) \tag{6.8}$$

Maximum likelihood estimates for the normal gpdfs involved in the factorizations can be used to estimate the factorizations from the random keys in the same way as was proposed in Chapter 5 for numerical optimization.

A problem that will arise when using real–valued probability distributions is that the real–valued space is largely redundant with respect to the permutation space that is being encoded, since all points in a single one of the $l!$ convex partitions of the $[0,1]^l$ hypercube represent the same permutation. Normal gpdfs may therefore have great difficulty in modelling competitions between the different partitions of the $[0,1]^l$ hypercube that represent different permutations. Therefore, we expect that it is more likely that relative–ordering dependencies can be modelled efficiently if the probability distribution is estimated in the space of permutations directly.

6.3.2 Interpreting random keys as permutations

In this section, we elaborate on the estimation of probability distributions in the space of permutations. To do so, we will first give a definition of integer permutation random variables and show how we can define probability distributions for these random variables. Subsequently, we describe the semantics of these integer permutation random variables with respect to the random keys random

variables. We then describe a gpdf for the integer permutation random variables and give a characterization of multivariate factorizations for integer permutation random variables. Finally, we show how the parameters of a multivariate factorization for integer permutation random variables can be represented and can be estimated greedily from data.

Random variables and probability distributions for permutations

We introduce l permutation random variables O_i, $i \in \boldsymbol{\mathcal{L}}$. A vector of permutation random variables $O_{\boldsymbol{j}}$ can be assigned $|\boldsymbol{j}|!$ different values, since there are $|\boldsymbol{j}|!$ ways to permute $O_{\boldsymbol{j}}$ and obtain the only valid ways of denoting the probability of an event related to permutations.

> **Example 6.4.** If $|\boldsymbol{j}| = 3$, there are $3! = 6$ ways to query the probability distribution:
>
> $$\Pr(O_{\boldsymbol{j}_0} < O_{\boldsymbol{j}_1} < O_{\boldsymbol{j}_2}), \quad \Pr(O_{\boldsymbol{j}_0} < O_{\boldsymbol{j}_2} < O_{\boldsymbol{j}_1}), \quad \Pr(O_{\boldsymbol{j}_1} < O_{\boldsymbol{j}_0} < O_{\boldsymbol{j}_2}),$$
>
> $$\Pr(O_{\boldsymbol{j}_1} < O_{\boldsymbol{j}_2} < O_{\boldsymbol{j}_0}), \quad \Pr(O_{\boldsymbol{j}_2} < O_{\boldsymbol{j}_0} < O_{\boldsymbol{j}_1}), \quad \Pr(O_{\boldsymbol{j}_2} < O_{\boldsymbol{j}_1} < O_{\boldsymbol{j}_0})$$

The domain of a collection of permutation random variables $O_{\boldsymbol{j}}$ is thus the set of all compact permutations of length $|\boldsymbol{j}|$, that is $O_{\boldsymbol{j}} \in \Omega^{P,\boldsymbol{j}} = perm((0, 1, \ldots |\boldsymbol{j}|-1))$. To formalize this in terms of values that can be assigned to all l permutation random variables $\boldsymbol{\mathcal{O}}$, we can thus use exactly these permutations. Since we constrained the values to be compact permutations, exactly $|\boldsymbol{j}|!$ values are allowed for random variables $O_{\boldsymbol{j}}$. If we now write $P(\boldsymbol{\mathcal{O}})(\boldsymbol{o})$, this means we are querying the probability distribution for permutation random variables $\boldsymbol{\mathcal{O}}$ for the event in which the relative value–ordering of the permutation random variables is indicated by $\boldsymbol{o}$. In other words, if and only if $\boldsymbol{o}_i < \boldsymbol{o}_j$ for some $(i, j) \in \boldsymbol{\mathcal{L}} \times \boldsymbol{\mathcal{L}}, i \neq j$, then $O_i < O_j$ holds in the query. This means that we can sort the compact permutation using the ascending sorting function defined in Section 6.1.2 and order the random variables according to the ascending order to get the full ordering of the random variables that corresponds to the event that we are interested in:

$$\begin{aligned} P(\boldsymbol{\mathcal{O}})(\boldsymbol{o}) &= \Pr(\boldsymbol{\mathcal{O}} = \boldsymbol{o}) \\ &= \\ &\Pr(\boldsymbol{\mathcal{O}}_{\sigma(\boldsymbol{o})_0} < \boldsymbol{\mathcal{O}}_{\sigma(\boldsymbol{o})_1} < \ldots < \boldsymbol{\mathcal{O}}_{\sigma(\boldsymbol{o})_{|\boldsymbol{j}|-1}}) \end{aligned} \tag{6.9}$$

We now recall that the definitions regarding probability theory in Chapter 2 are only valid for Cartesian–product sample spaces. The integer permutation sample space, however, is not a Cartesian product. Fortunately, the probability theoretic definitions from Chapter 2 can be extended to the non–Cartesian–product integer permutation sample space quite intuitively. For instance, as is to be expected for any discrete sample space, the sum over all probabilities for the different permutations that can be assigned to permutation random

variables $\boldsymbol{\mathcal{O}}$ must be equal to 1. In other words, the probability measure Γ^P for the integer permutation sample space is defined as follows:

$$\Gamma^P(A) = \sum_{\boldsymbol{o} \in A} P(\boldsymbol{\mathcal{O}})(\boldsymbol{o}) \tag{6.10}$$

The marginalization property also requires a different formulation for permutation random variables. The reason for this is that it is possible that a variable O_i ($i \in \boldsymbol{j} \sqsubset \boldsymbol{\mathcal{L}}$) can be assigned different values in $\Omega^{P,\boldsymbol{j}}$ than in the complete sample space for all permutation random variables Ω^P. For instance, $\Pr(O_0 < O_1 < O_2) = P(O_0, O_1, O_2)(0, 1, 2)$ is a valid way to query for a probability, but so is $\Pr(O_2 < O_3 < O_1 < O_0) = P(O_0, O_1, O_2, O_3)(3, 2, 0, 1)$. Whereas in the first case, O_0 is *not* allowed to be assigned the value of 3, in the second case it *is*. To formalize the marginalization property we therefore need a surjective function that, given a vector that is constructed by taking the values at indices $\boldsymbol{k} \sqsubset \boldsymbol{j}$ from a vector in $\Omega^{P,\boldsymbol{j}}$, returns a vector in $\Omega^{P,\boldsymbol{k}}$. This amounts to copying the required $|\boldsymbol{k}|$ integers from the permutation of length $|\boldsymbol{j}|$ and transforming them to a permutation of $(0, 1, \ldots, |\boldsymbol{k}| - 1)$ such that the relative ordering of the permutation random variables remains the same. To this end, the sorting function σ that we introduced earlier in Section 6.1.2 can be used. Note that for any permutation $\boldsymbol{p}$ and for any $\boldsymbol{p_j} \sqsubset \boldsymbol{p}$, $\sigma(\sigma(\boldsymbol{p_j}))$ is a permutation of $(0, 1, \ldots, |\boldsymbol{j}| - 1)$ that has the property that $\sigma(\sigma(\boldsymbol{p_j}))_i < \sigma(\sigma(\boldsymbol{p_j}))_{i+1}$ if and only if $(\boldsymbol{p_j})_i < (\boldsymbol{p_j})_{i+1}$.

Example 6.5. Consider 6 random variables $O_0, O_1, \ldots, O_5$. We want to obtain the probability distribution $P(O_2, O_3, O_4, O_5)$ by marginalization of $P(O_0, O_1, O_2, O_3, O_4, O_5)$ over random variables O_0 and O_1. Then for each permutation $o_{(2,3,4,5)}$ of length 4 in $\Omega^{P,(2,3,4,5)}$ we must sum over all permutations $o'_{(0,1,2,3,4,5)}$ of length 6 in $\Omega^{P,(0,1,2,3,4,5)}$ for which the indicated ordering of random variables O_2, O_3, O_4 and O_5 is the same as the ordering indicated by $o_{(2,3,4,5)}$. To check whether this is the case, we must find a compact permutation of $(0, 1, 2, 3)$ that indicates the same ordering of random variables O_2, O_3, O_4 and O_5 as these random variables have when ordering all random variables $O_0, O_1, \ldots, O_5$ according to $o'_{(0,1,2,3,4,5)}$. We can then compare this compact alternative with $o_{(2,3,4,5)}$ and add the probability $P(O_0, O_1, O_2, O_3, O_4, O_5)(o'_{(0,1,2,3,4,5)})$ to the sum if and only if the permutations are equal.

We illustrate the finding of the compact alternative permutation using an example instance. Let $o'_{(0,1,2,3,4,5)} = (0, 5, 3, 4, 2, 1)$. The ordering of random variables $O_0, O_1, \ldots, O_5$ indicated by this permutation is $O_0 < O_5 < O_4 < O_2 < O_3 < O_1$. It is important to note that we cannot simply take the subvector $o'_{(2,3,4,5)}$ of $o'_{(0,1,2,3,4,5)}$ to get the ordering of random variables O_2, O_3, O_4 and O_5 only, because this subpermutation may no longer be compact. According to the ordering indicated by

$o'_{(0,1,2,3,4,5)}$, vector $o_{(2,3,4,5)}$ must represent that $O_5 < O_4 < O_2 < O_3$, since $o'_5 < o'_4 < o'_2 < o'_3$. Using the definition of of permutation random variables, this means that the correct compact alternative permutation for random variables O_2, O_3, O_4 and O_5 is $(2, 3, 1, 0)$. To derive this permutation from $o'_{(0,1,2,3,4,5)}$, we first take subvector $o'_{(2,3,4,5)} = (3, 4, 2, 1)$, since it validly indicates the relative ordering of the random variables, albeit not in a compact way. If we now sort this permutation, we get a compact permutation, i.e. $\sigma(o'_{(2,3,4,5)}) = (3, 2, 0, 1)$. Although this permutation is compact, it no longer represents the same relative ordering of the random variables. However, it does indicate how such an ordering can be constructed, since for any permutation $\boldsymbol{p}$, the sorting operation $\sigma(\boldsymbol{p})$ tells for each rank i in the sorted ordering, the position $\sigma(\boldsymbol{p})_i$ at which to find the object of rank i in $\boldsymbol{p}$. Thus, if we place the integers $(0, 1, \ldots, |\boldsymbol{p}|-1)$ in a permutation $\boldsymbol{p}'$ in such a way that $\sigma(\boldsymbol{p}') = \sigma(\boldsymbol{p})$, $\boldsymbol{p}'$ is a compact alternative to $\boldsymbol{p}$. Since $\boldsymbol{p}'$ is compact, we have $\sigma(\sigma(\boldsymbol{p}')) = \boldsymbol{p}'$ (see Section 6.1.2), and thus we can construct $\boldsymbol{p}'$ by performing a double sort on $\boldsymbol{p}$, since $\sigma(\boldsymbol{p}') = \sigma(\boldsymbol{p}) \Leftrightarrow \sigma(\sigma(\boldsymbol{p}')) = \sigma(\sigma(\boldsymbol{p})) \Leftrightarrow \boldsymbol{p}' = \sigma(\sigma(\boldsymbol{p}))$. Thus, the required compact alternative permutation for the example is $\sigma(\sigma(o'_{(2,3,4,5)})) = (2, 3, 1, 0)$.

Using the required surjective function $\sigma(\sigma(\boldsymbol{o}))$, we can now formalize the marginalization property for permutation random variables for any $\boldsymbol{k} \sqsubset \boldsymbol{j} \sqsubseteq \boldsymbol{\mathcal{L}}$ and for any $o_{\boldsymbol{k}} \in \Omega^{P,\boldsymbol{k}}$ and for any $o_{\boldsymbol{j}} \in \Omega^{P,\boldsymbol{j}}$:

$$P(O_{\boldsymbol{k}})(o_{\boldsymbol{k}}) = \sum_{o'_{\boldsymbol{j}} \in \Omega^{\boldsymbol{j}}} \begin{cases} P(O_{\boldsymbol{j}})(o'_{\boldsymbol{j}}) & \text{if } \sigma(\sigma(o'_{\boldsymbol{k}})) = o_{\boldsymbol{k}} \\ 0 & otherwise \end{cases} \tag{6.11}$$

Interpretation of permutation random variables

We will first discuss how we will interpret permutation random variables. Second, we show how this interpretation of permutation random variables allows us to continue to describe probability distributions over permutation random variables instead of random–keys random variables because there is a clear and easy–to–use correspondence between the probability distribution for permutation random variables and the real–valued probability distribution for random keys random variables that we are ultimately interested in.

Semantics

For clarity and good understanding, it is very important to note that we can assign two different semantics to a permutation random variable O_i. A straightforward interpretation for O_i is that this permutation random variable represents the integer at position i in a solution in $\mathbb{P}$. We call this interpretation *relative–value–at–position semantics*. In this case, we cannot directly identify permutation random variable O_i with random keys random variable R_i since by definition of the random keys genotype the probability $\Pr(O_i < O_j)$ in this interpretation corresponds to the probability that the i–th largest random key

value in a random keys genotype is found at a locus smaller than the locus of the j–th largest random key value in the same random keys genotype.

The discrepancy between the random keys encoding and the relative–value–at–position interpretation of the permutation random variables is not convenient. We therefore choose a different approach. By decoding a random keys substring $\boldsymbol{r_j}$, an integer permutation is obtained that indicates the relative ordering of the integers $\boldsymbol{j}$ in the complete decoded random keys genotype $\sigma(\boldsymbol{r})$. We let permutation random variable O_i represent the *position* of integer i in the integer permutation solution in $\mathbb{P}$. We call this interpretation *relative–position–of–value semantics*. With this interpretation, $\Pr(O_i < O_j)$ denotes the probability that integer i *precedes* integer j in a solution. Note that this interpretation for a permutation random variable O_i is valid since the positions of all integers $i \in \{0, 1, \ldots, l-1\}$ themselves form again a permutation. It should furthermore be noted that with these semantics, a permutation random variable O_i can directly be identified with a random keys random variable R_i since by definition of the random keys genotype we have that $\Pr(O_i < O_j) = \Pr(R_i < R_j)$.

Obtaining random keys distributions from permutation distributions

Because of the correspondence between random keys and the permutation random variables with the relative–position–of–value semantics, we may continue to describe probability distributions over permutation random variables $O_{\boldsymbol{j}}$ instead of over the random keys random variables $R_{\boldsymbol{j}}$ in the remainder of this chapter. Any probability distribution estimated for random variables $O_{\boldsymbol{j}}$ can be translated into a probability distribution for random keys random variables $R_{\boldsymbol{j}}$ using this correspondence. The probability distribution over the real space $[0,1]^{\boldsymbol{j}}$ that corresponds to some probability distribution for random variables $O_{\boldsymbol{j}}$ consists of $|\boldsymbol{j}|!$ convex hypervolumes, one associated with each ordering $\boldsymbol{o} \in \mathit{perm}((0, 1, \ldots, |\boldsymbol{j}| - 1))$. Each of these hypervolumes is equally large because for any two points $\boldsymbol{r}^0_{\boldsymbol{j}}$ and $\boldsymbol{r}^1_{\boldsymbol{j}}$ in this hypervolume, the ordering of their real values is the same, that is $\sigma(\boldsymbol{r}^0_{\boldsymbol{j}}) = \sigma(\boldsymbol{r}^1_{\boldsymbol{j}})$ The probability distribution is uniform in each hypervolume and the integral over all points in a hypervolume corresponding to ordering $\boldsymbol{o}$ is just the probability $\Pr(O_{\boldsymbol{j}}) = \boldsymbol{o}$. In other words, the probability distribution for the real–valued space can be seen as a form of a histogram in which each bin represents all points of equal ordering. Drawing a new sample from the random keys probability distribution for random variables $R_{\boldsymbol{j}}$ associated with some integer permutation probability distribution can be done by first probabilistically determining an ordering $o_{\boldsymbol{j}}$ for random variables $O_{\boldsymbol{j}}$ according to their estimated probability distribution and subsequently drawing $|\boldsymbol{j}|$ random real–values $r_{\boldsymbol{j}}$ in $[0,1]^{|\boldsymbol{j}|}$ that are placed in the unique order such that random variables $R_{\boldsymbol{j}}$ represent the same permutation as do random variables $O_{\boldsymbol{j}}$ according to the relative–position–of–value semantics, that is, $\sigma(\sigma(r_{\boldsymbol{j}})) = o_{\boldsymbol{j}}$.

A gpdf for random keys based on the permutation gpdf

We will define a gpdf for permutation random variables and show how to estimate it with a maximum likelihood from a given vector of random keys encod-

ings. According to the proposed interpretation of permutation random variables, the type of dependency that will be modelled will concern the relative ordering of the integers in the permutations. In Section 2.5 it was already noted that if the gpdfs in a factorization are estimated to be of maximum likelihood, then the factorization itself is also of maximum likelihood. Therefore, we define the gpdf and its maximum likelihood estimation in terms of a selection of random variables $Z_{\boldsymbol{\nu}_i} \sqsubseteq \boldsymbol{\mathcal{Z}}$ so it is directly clear how these gpdfs can be used in a multivariate factorization.

Definition
The permutation gpdf for random variables $O_{\boldsymbol{\nu}_i}$ is simply a discrete gpdf that has $|\boldsymbol{\nu}_i|!$ parameters. Each parameter equals the probability of a certain ordering $o_{\boldsymbol{\nu}_i} \in perm(0, 1, \ldots, |\boldsymbol{\nu}_i|-1)$ of the random variables $O_{\boldsymbol{\nu}_i}$, that is, $P_{\boldsymbol{\theta}}(O_{\boldsymbol{\nu}_i})(o_{\boldsymbol{\nu}_i})$.

Parameter estimation
It is relatively easy to estimate the parameters for the permutation gpdf with a maximum likelihood. Since our ID$\mathbb{E}$A will be working with random keys genotypes, let $\boldsymbol{\mathcal{S}}$ be a vector of random keys genotypes of length l for which our parameter estimates must be made. Because we have a discrete space with a finite number of parameters, a straightforward way to estimate the probability $P(O_{\boldsymbol{\nu}_i})(o_{\boldsymbol{\nu}_i})$ is to compute the average frequency of finding random keys in the data at the positions indicated by $\boldsymbol{\nu}_i$ such that they encode the same ordering of random variables $O_{\boldsymbol{\nu}_i}$ as does $o_{\boldsymbol{\nu}_i}$. Such proportion estimates for discrete gpdfs are known to result in a maximum likelihood estimate (Anderson, 1958; Tatsuoka, 1971). If we now realize that $\sigma(\sigma(\boldsymbol{o})) = \boldsymbol{o}$ if $\boldsymbol{o}$ is a compact permutation, we obtain the following characterization of the maximum likelihood estimation of the permutation gpdf:

$$
\begin{aligned}
&\hat{P}(O_{\boldsymbol{\nu}_i})(o_{\boldsymbol{\nu}_i}) \qquad (6.12)\\
&= \\
&\hat{\Pr}((O_{\boldsymbol{\nu}_i})_{\sigma(o_{\boldsymbol{\nu}_i})_0} < (O_{\boldsymbol{\nu}_i})_{\sigma(o_{\boldsymbol{\nu}_i})_1} < \ldots < (O_{\boldsymbol{\nu}_i})_{\sigma(o_{\boldsymbol{\nu}_i})_{|\boldsymbol{\nu}_i|-1}}) \\
&= \\
&\frac{1}{|\boldsymbol{\mathcal{S}}|} \sum_{j=0}^{|\boldsymbol{\mathcal{S}}|-1} \begin{cases} 1 & \text{if } ((\boldsymbol{\mathcal{S}}_j)_{\boldsymbol{\nu}_i})_{\sigma(o_{\boldsymbol{\nu}_i})_0} < ((\boldsymbol{\mathcal{S}}_j)_{\boldsymbol{\nu}_i})_{\sigma(o_{\boldsymbol{\nu}_i})_1} < \ldots < ((\boldsymbol{\mathcal{S}}_j)_{\boldsymbol{\nu}_i})_{\sigma(o_{\boldsymbol{\nu}_i})_{|\boldsymbol{\nu}_i|-1}} \\ 0 & \textit{otherwise} \end{cases} \\
&= \\
&\frac{1}{|\boldsymbol{\mathcal{S}}|} \sum_{j=0}^{|\boldsymbol{\mathcal{S}}|-1} \begin{cases} 1 & \text{if } \sigma((\boldsymbol{\mathcal{S}}_j)_{\boldsymbol{\nu}_i}) = \sigma(o_{\boldsymbol{\nu}_i}) \\ 0 & \textit{otherwise} \end{cases} \\
&= \\
&\frac{1}{|\boldsymbol{\mathcal{S}}|} \sum_{j=0}^{|\boldsymbol{\mathcal{S}}|-1} \begin{cases} 1 & \text{if } \sigma(\sigma((\boldsymbol{\mathcal{S}}_j)_{\boldsymbol{\nu}_i})) = o_{\boldsymbol{\nu}_i} \\ 0 & \textit{otherwise} \end{cases}
\end{aligned}
$$

Sampling
To generate a random keys sequence by sampling an estimated gpdf for random variables $O_{\boldsymbol{\nu}_i}$, the same approach can be used as discussed above for obtaining random keys probability distributions from permutation probability distributions. Obtaining an integer permutation $o_{\boldsymbol{\nu}_i}$ from the permutation gpdf for random variables $O_{\boldsymbol{\nu}_i}$ is straightforward since $\Omega^{P,\boldsymbol{\nu}_i}$ is finite and discrete.

Multivariate factorizations for permutation random variables

Defining the multivariate factorization for integer permutation random variables is not as straightforward as is the case for integer random variables. To get some intuition as to why this is so, we first present an extensive example.

Example 6.6. Consider the case in which $l = 6$ and we want to characterize the multivariate factorization based on $\boldsymbol{\nu} = ((0,1),(2,3,4,5))$. To this end, we first observe an example query for obtaining the probability of a permutation $\boldsymbol{o} = (0,5,3,4,2,1)$. In other words, we want to know how to compute $P_{((0,1),(2,3,4,5))}(O_0,O_1,O_2,O_3,O_4,O_5)(0,5,3,4,2,1)$. Note that this means that we are interested in the probability of the event that $O_0 < O_5 < O_4 < O_2 < O_3 < O_1$.

According to the definition of independence in multivariate factorizations, we must be able to compute the required probability by using the permutation gpdfs $P(O_0,O_1)$ and $P(O_2,O_3,O_4,O_5)$. In the definition of the marginalization property for permutation random variables, we already saw that to obtain the correct compact query permutations for $P(O_0,O_1)$ and $P(O_2,O_3,O_4,O_5)$ that indicate the same ordering of the random variables as does $\boldsymbol{o}$ for all 6 random variables, we must perform a double sort on subvectors $(\boldsymbol{o}_0,\boldsymbol{o}_1)$ and $(\boldsymbol{o}_2,\boldsymbol{o}_3,\boldsymbol{o}_4,\boldsymbol{o}_5)$. The required compact alternative permutations are thus given by $\sigma(\sigma(\boldsymbol{o}_{(0,1)})) = (0,1)$ and $\sigma(\sigma(\boldsymbol{o}_{(2,3,4,5)})) = (2,3,1,0)$.

The frequency tables of the 2 individual multivariate joint permutation gpdfs in our example are of size $2! = 2$ and $4! = 24$. Assume that all parameters are uniform, i.e. both parameters for $P(O_0,O_1)$ are $\frac{1}{2}$ and all 24 parameters for $P(O_2,O_3,O_4,O_5)$ are $\frac{1}{24}$. We find that unlike in cases such as the discrete integer case and the case in which the normal gpdf is used for real–valued random variables, we have that

$$P_{((0,1),(2,3,4,5))}(O_0,O_1,O_2,O_3,O_4,O_5)(\boldsymbol{o}) \neq \tag{6.13}$$
$$P(O_0,O_1)(\sigma(\sigma(\boldsymbol{o}_{(0,1)})))P(O_2,O_3,O_4,O_5)(\sigma(\sigma(\boldsymbol{o}_{(2,3,4,5)})))$$

The multivariate factorization does not equal the product of the individual gpdfs for the mutually exclusive factors because the righthandside in equation 6.13 is not a probability distribution over the 6 random variables combined. The reason for this is that the number of possible permutations of length 6 is $6! = 720$. Therefore, the summation of the wrongly factorized probabilities over all of these 720 permutations equals

$720 \cdot \frac{1}{2} \cdot \frac{1}{24} = 15 \neq 1$. In the case of binary random variables, the number of possible combinations would be $2^6 = 64$ and the individual frequency tables would have been of size $2^2 = 4$ and $2^4 = 16$. If each individual probability would then have been $\frac{1}{4}$ and $\frac{1}{16}$ respectively for example, the summation over all possible combinations would be $64 \cdot \frac{1}{4} \cdot \frac{1}{16} = 1$.

Multivariate factorizations for permutations are different because if we know that $O_0 < O_1$ and $O_2 < O_3 < O_4 < O_5$, then there are a multiple of events regarding all six permutation random variables in which both equations hold instead of only a single permutation. Two examples of such events are $O_0 < O_1 < O_2 < O_3 < O_4 < O_5$ and $O_0 < O_2 < O_3 < O_4 < O_5 < O_1$. There are 15 of such "indistinguishable" events since the total number of possible permutations is $6! = 720$ and the total number of groups of permutations that can be made with the 2 shorter permutations is $2! \cdot 4! = 48$. This means that the correct factorization of the probability distribution is given by:

$$P_{((0,1),(2,3,4,5))}(O_0, O_1, O_2, O_3, O_4, O_5)(\boldsymbol{o}) = \tag{6.14}$$
$$\frac{2!4!}{6!} P(O_0, O_1)(\sigma(\sigma(\boldsymbol{o}_{(0,1)}))) P(O_2, O_3, O_4, O_5)(\sigma(\sigma(\boldsymbol{o}_{(2,3,4,5)})))$$

The reason for the observed difference between the multivariate factorization as defined in Chapter 2 and the result in our example is that the definition of multivariate factorizations in Chapter 2 is only valid for Cartesian–product sample spaces. The general definition of multivariate factorizations is slightly different:

$$P_{\boldsymbol{\nu}}(\boldsymbol{\mathcal{Z}})(\boldsymbol{z}) = \frac{1}{|\Omega|} \prod_{i=0}^{|\boldsymbol{\nu}|-1} |\Omega^{\boldsymbol{\nu}_i}| P_{\boldsymbol{\theta}^{\boldsymbol{\nu}_i}}(Z_{\boldsymbol{\nu}_i})(\psi(\boldsymbol{z}_{\boldsymbol{\nu}_i})) \tag{6.15}$$
$$=$$
$$\frac{\prod_{i=0}^{|\boldsymbol{\nu}|-1} |\Omega^{\boldsymbol{\nu}_i}|}{|\Omega|} \prod_{i=0}^{|\boldsymbol{\nu}|-1} P_{\boldsymbol{\theta}^{\boldsymbol{\nu}_i}}(Z_{\boldsymbol{\nu}_i})(\psi(\boldsymbol{z}_{\boldsymbol{\nu}_i}))$$

In equation 6.15, $\psi(\boldsymbol{z}_{\boldsymbol{\nu}_i})$ is a function that reformats a subvector of a value for all random variables into a value for a selection of all random variables. If the random variables are in a Cartesian–product sample space, the definition of a multivariate factorization simplifies significantly. For binary random variables for instance, $|\Omega^{\boldsymbol{j}}| = 2^{|\boldsymbol{j}|}$ and thus $\frac{\prod_{i=0}^{|\boldsymbol{\nu}|-1} |\Omega^{\boldsymbol{\nu}_i}|}{|\Omega|} = 1$. For real–valued random variables, $|\Omega^{\boldsymbol{j}}| = |\mathbb{R}|^{|\boldsymbol{j}|}$, and we again find that the definition of a multivariate factorization simplifies to $\prod_{i=0}^{|\boldsymbol{\nu}|-1} P_{\boldsymbol{\theta}^{\boldsymbol{\nu}_i}}(Z_{\boldsymbol{\nu}_i})(\psi(\boldsymbol{z}_{\boldsymbol{\nu}_i}))$. For permutation random variables however, such a simplification does not occur and the factor $\frac{\prod_{i=0}^{|\boldsymbol{\nu}|-1} |\Omega^{\boldsymbol{\nu}_i}|}{|\Omega|}$ serves as a normalization of the product of the gpdfs such that the resulting function over all random variables is a probability distribution. Function $\psi(\boldsymbol{z}_{\boldsymbol{\nu}_i})$ is simply the identity function for Cartesian–product sample spaces. For non–Cartesian–product spaces, such as the space of permutations,

the number of values is smaller for a subset of all random variables than for all random variables. To make sure that a subvector of a value for a larger set of random variables is a value for the subset of all random variables, some type of transformation must be made. We have already seen from the marginalization property for permutation random variables, that for permutation random variables we have that $\psi(\boldsymbol{z}_{\boldsymbol{\nu}_i}) = \sigma(\sigma(\boldsymbol{z}_{\boldsymbol{\nu}_i}))$.

The number of values that can be assigned to random variables $O_{\boldsymbol{\nu}_i}$ of a single multivariate joint factor is $|\Omega^{\boldsymbol{\nu}_i}| = |\boldsymbol{\nu}_i|!$. Since the individual factors in a multivariate factorization are taken to be totally *independent* of each other, the total number of values that the product of the gpdfs can uniquely assign a probability to, is $\prod_{i=0}^{|\boldsymbol{\nu}|-1} |\boldsymbol{\nu}_i|!$. On the other hand, the number of values that can be assigned to all random variables $\boldsymbol{\mathcal{O}}$ equals the number of permutations of length l, that is, $|\Omega| = l!$. Therefore, to construct a probability distribution over all possible permutations of length l, the product of multivariate gpdfs $\prod_{i=0}^{l-1} P(O_{\boldsymbol{\nu}_i})(\sigma(\sigma(\boldsymbol{o}_{\boldsymbol{\nu}_i})))$ must be normalized by multiplication with $\frac{\prod_{i=0}^{|\boldsymbol{\nu}|-1} |\boldsymbol{\nu}_i|!}{l!}$. The multivariate factorization for l permutation random variables $\boldsymbol{\mathcal{O}}$ can now be defined as follows:

$$P_{\boldsymbol{\nu}}(\boldsymbol{\mathcal{O}})(\boldsymbol{o}) = \frac{\prod_{i=0}^{|\boldsymbol{\nu}|-1} |\boldsymbol{\nu}_i|!}{l!} \prod_{i=0}^{l-1} P(O_{\boldsymbol{\nu}_i})(\sigma(\sigma(\boldsymbol{o}_{\boldsymbol{\nu}_i}))) \tag{6.16}$$

Parameter representation

The parameters that need to be estimated for the multivariate factorization for permutation random variables need to be stored somehow. One way of doing so is to store them in a frequency table, just as is common practice in the use of binary random variables. An alternative approach is given by default tables. In the following, we discuss the realization of both approaches.

A direct representation through frequency tables

From the definition of the multivariate factorization, and the maximum likelihood permutation gpdf estimate it follows that we have to be able to count the frequencies for a selection of integer permutation random variables $O_{\boldsymbol{\nu}_j}$. The overhead that we at least have for this task according to equation 6.12, is in the decoding of all selected random keys genotypes at the positions indicated by the j–th factor $\boldsymbol{\nu}_j$. This can be done in $\mathcal{O}(|\boldsymbol{\mathcal{S}}||\boldsymbol{\nu}_j|\log(|\boldsymbol{\nu}_j|))$ time by sorting subsequences of all selected random keys genotypes. To construct a frequency table for permutations of length $|\boldsymbol{\nu}_j|$, we require a frequency table of minimum size $|\boldsymbol{\nu}_j|! - 1$. In order to generate such a frequency table, we can take two approaches, which we discuss next in turn.

One approach is to construct a bijective mapping between permutations and integers and use this mapping to index an array of size $|\boldsymbol{\nu}_j|! - 1$. The reason why this mapping must be bijective is that once we have computed the frequency tables, we must also be able to sample from them. To do so, we must know for each index what permutation it is associated with.

To explain how the bijective mapping can be constructed, we start with the inverse case that maps integers to integer permutations. If we have a number n that represents a permutation $\boldsymbol{p}$, we know that the number lies in the range $\{0, 1, \ldots |\boldsymbol{p}|! - 1\}$. Therefore, $\lfloor \frac{n}{(|\boldsymbol{p}|-1)!} \rfloor$ lies in the range $\{0, 1, \ldots, |\boldsymbol{p}| - 1\}$. Furthermore, n rem $(|\boldsymbol{p}| - 1)!$ lies in the range $\{0, 1, \ldots, (|\boldsymbol{p}| - 1)! - 1\}$. We can generate the corresponding permutation in $|\boldsymbol{p}|$ steps by first creating an initial temporary permutation. Then, in each step i, we take the final permutation element at position i to be the element at position $\lfloor \frac{n}{(|\boldsymbol{p}|-i)!} \rfloor$ in the temporary permutation. To ensure that no elements from the temporary permutation are used twice, the $(|\boldsymbol{p}| - i)$–th element in the temporary permutation replaces the used element at position i. For the j–th factor in the multivariate factorization, the resulting algorithm runs in $\mathcal{O}(|\boldsymbol{\nu}_j|)$ time and is given in Figure 6.2. Also in this figure, the function results are given for $k = 4$.

To map integer permutations of length $|\boldsymbol{\nu}_j|$ to integers, we have to know in each iteration at which position the i–th permutation element was located in the temporary permutation array. To this end, we make a second temporary array in which this information is stored. Before the first temporary array is altered in the same way as is done in the inverse case, we update the location information in the second temporary array. The resulting algorithm runs in $\mathcal{O}(|\boldsymbol{\nu}_j| \log(|\boldsymbol{\nu}_j|))$ time and is given in Figure 6.2.

One problem with this approach is that when the factor size $|\boldsymbol{\nu}_j|$ becomes larger, the maximum integer that must be representable in the bijective mapping, grows extremely fast and will no longer be efficiently representable. It is therefore to be expected when using frequency tables that the maximum factor size will be limited in any practical use of multivariate factorized probability distributions, either forced or by penalization of the likelihood while learning the probabilistic model.

Using a bijective mapping between permutations and integers leads to a required running time of $\mathcal{O}(|\boldsymbol{\nu}_j|! + |\boldsymbol{\mathcal{S}}||\boldsymbol{\nu}_j| \log(|\boldsymbol{\nu}_j|))$, since each selected permutation of length $|\boldsymbol{\nu}_j|$ can be mapped onto an integer in $|\boldsymbol{\nu}_j| \log(|\boldsymbol{\nu}_j|)$ time.

An alternative approach to using the bijective mapping is to make a table of size $|\boldsymbol{\nu}_j|! - 1$ that can be indexed using a permutation. To estimate the frequencies as efficiently as possible in this case, the available permutations must be sorted, which can be done in $\mathcal{O}(|\boldsymbol{\nu}_j||\boldsymbol{\mathcal{S}}| \log(|\boldsymbol{\mathcal{S}}|))$ time. By scanning the sorted list of permutations simultaneously with the full frequency table of size $|\boldsymbol{\nu}_j|!$, the required average frequencies can then be counted in $\mathcal{O}(|\boldsymbol{\nu}_j|! + |\boldsymbol{\nu}_j||\boldsymbol{\mathcal{S}}|)$ time. The total amount of required time using permutation indexed tables is thus given by $\mathcal{O}(|\boldsymbol{\nu}_j|! + |\boldsymbol{\nu}_j||\boldsymbol{\mathcal{S}}|(1 + \log(|\boldsymbol{\mathcal{S}}|)))$.

Introducing local structures through default tables

The *default table* (Friedman and Goldszmidt, 1996) offers an alternative to the frequency table. In a default table, the probabilities are explicitly specified for a subset of all available values. For the absent values, a *default value* is used, which is the average probability of all absent values. Examples of default tables for permutations are given in Figure 6.3. One straightforward way to use default

```
IntegerToIntegerPermutation( n, length )
1  o ← new array of integer with size length
2  p ← new array of integer with size length
3  for i ← 0 to length − 1 do
   3.1  o[i] ← i
4  fac ← (length − 1)!
5  for i ← 0 to length − 1 do
   5.1  pos ← n / fac
   5.2  n ← n − pos · fac
   5.3  p[i] ← o[pos]
   5.4  o[pos] ← o[length − 1 − i]
   5.5  if i < length − 1 then
        5.5.1  fac ← fac / (length − i − 1)
6  Return(p)
```

```
IntegerPermutationToInteger( p, length )
1  o^0 ← new array of integer with size length
2  o^1 ← new array of integer with size length
3  for i ← 0 to length − 1 do
   3.1  o^0[i] ← i
   3.2  o^1[i] ← i
4  n ← 0
5  fac ← (length − 1)!
6  for i ← 0 to length − 1 do
   6.1  pos ← o^1[p[i]]
   6.2  n ← n + pos · fac
   6.3  o^1[o^0[length − 1 − i]] ← pos
   6.4  o^0[pos] ← o^0[length − 1 − i]
   6.5  if i < length − 1 then
        6.5.1  fac ← fac / (length − i − 1)
7  Return(n)
```

Perm.	$\mathbb{N}$
0 3 2 1	0
0 3 1 2	1
0 1 3 2	2
0 1 2 3	3
0 2 3 1	4
0 2 1 3	5
1 0 2 3	6
1 0 3 2	7

Perm.	$\mathbb{N}$
1 3 0 2	8
1 3 2 0	9
1 2 0 3	10
1 2 3 0	11
2 0 3 1	12
2 0 1 3	13
2 1 0 3	14
2 1 3 0	15

Perm.	$\mathbb{N}$
2 3 0 1	16
2 3 1 0	17
3 0 2 1	18
3 0 1 2	19
3 1 0 2	20
3 1 2 0	21
3 2 0 1	22
3 2 1 0	23

Figure 6.2: Pseudo–code for a bijective mapping between integers and integer permutations. On the left, the pseudo–code is given for converting integers to integer permutations (top) and for converting integer permutations to integers (bottom). On the right, an example of the bijective mapping is given for integer permutations of length 4.

$\boldsymbol{\mathcal{S}}$					
0	1	2	3	4	5
3	5	4	2	1	0
3	4	2	0	1	2
3	5	4	2	1	0
0	1	2	3	4	5
0	1	2	3	4	5
3	4	5	1	2	0

Default table						
0	1	2	3	4	5	$\frac{3}{7}$
default						$\frac{4}{5033}$

Default table						
0	1	2	3	4	5	$\frac{3}{7}$
3	4	2	0	1	2	$\frac{1}{7}$
3	4	5	1	2	0	$\frac{1}{7}$
3	5	4	2	1	0	$\frac{2}{7}$
default						0

Figure 6.3: Two examples of default tables. On the left, a sample vector is shown that contains 7 permutations. The center default table has only an entry for the single most frequently appearing permutation. On the right, a default table is shown in which the same information is stored as would be contained in a frequency table, only more efficient due to the use of a default entry.

tables, is to only specify the average frequency for each value that occurs in the sample vector. By doing so, no factor in a multivariate factorization can give rise to more parameters than $|\boldsymbol{\mathcal{S}}|$. Although the information in the so–constructed default table is the same as that contained in a frequency table, the number of parameters that need to be estimated equals the number of different samples in $\boldsymbol{\mathcal{S}}$. For this reason, if the number of different samples is significantly less than the total number of different possible samples, a default table containing the same information as a frequency table can be estimated more efficiently than can a frequency table. For permutations for instance, the list of selected permutations is first sorted in $\mathcal{O}(|\boldsymbol{\nu}_i||\boldsymbol{\mathcal{S}}|\log(|\boldsymbol{\mathcal{S}}|))$ time, after which the frequencies are counted in $\mathcal{O}(|\boldsymbol{\nu}_i||\boldsymbol{\mathcal{S}}|)$ time, completing the construction of the default table. The total running time for constructing a default table is thus $\mathcal{O}(|\boldsymbol{\nu}_i||\boldsymbol{\mathcal{S}}|(1+\log(|\boldsymbol{\mathcal{S}}|)))$, whereas the construction of a frequency table requires an additional $\mathcal{O}(|\boldsymbol{\nu}_j|!)$. Note that we cannot map permutations to integers for faster sorting because the integers would become too large to efficiently represent with increasing $|\boldsymbol{\nu}_j|$, which is likely to happen using default tables.

Default tables containing the same information as frequency tables may not always be estimated faster than a frequency table since it may still be required that $|\boldsymbol{\mathcal{S}}| = \mathcal{O}(\kappa^{\boldsymbol{\iota}}!)$ when there are subproblems with a maximum length of $\kappa^{\boldsymbol{\iota}}$ that need to be exhaustively sampled. However, when we must combine lower order solutions to get solutions of a higher order, the default tables can give us a much more efficient representation of the few good solutions to the subproblems.

This latter issue is an important benefit of using *local structures* in probability distributions. A local structure allows for a more explicit representation of dependencies between the *values* for random variables instead of dependencies between the random variables *themselves*. As a result, less parameters need to be estimated. Probabilistic models that are capable of expressing more complex dependencies now become eligible for selection when using a penalization metric, whereas otherwise non–local structure models expressing similar dependencies

would never have been regarded because of the large number of (redundant) parameters they impose (Friedman and Goldszmidt, 1996).

Factorization selection

To find a good multivariate factorization for permutation random variables, we intend to use the greedy penalized likelihood maximization algorithm as described in Chapter 2. However, the definition of the multivariate factorization is different for permutation random variables than for binary random variables or real–valued random variables. Therefore, we investigate the implication of this difference for the use of the greedy penalized likelihood maximization algorithm and propose remedies for the additional problems that are encountered.

Greedy multivariate factorization selection by splicing factors

Briefly recapitulated, the greedy multivariate factorization selection algorithm starts from the univariate factorization, $\boldsymbol{\nu} = ((0), (1), \ldots, (l-1))$. In each iteration, the factorization may be altered by the splicing (joining) of two factors $\boldsymbol{\nu}_{s_0}$ and $\boldsymbol{\nu}_{s_1}$. The splice operation that decreases a certain metric, is actually performed. This process is repeated until there are no splice operations left that further improve the metric. The metric that is used, is either the AIC metric or the BIC metric, both of which are a penalized negative log–likelihood metric.

In the greedy algorithm, we search for the largest value of the penalized negative log–likelihood of the current factorization $\boldsymbol{\nu^0}$ *minus* the penalized negative log–likelihood of the candidate factorization $\boldsymbol{\nu^1}$. Since the penalization is additive to this difference, it is already indicated in Chapter 2 that for the AIC and BIC metrics the penalization for this difference may be determined separately. We recall that this difference equals $\delta\,(|\boldsymbol{\theta} \xleftarrow{fit} (\boldsymbol{\nu}_{s_0} \sqcup \boldsymbol{\nu}_{s_1})| + |\boldsymbol{\theta} \xleftarrow{fit} \boldsymbol{\nu}_{s_0}| + |\boldsymbol{\theta} \xleftarrow{fit} \boldsymbol{\nu}_{s_1}|)$ where $\delta = 1$ for the AIC metric and $\delta = \lambda \ln(|\boldsymbol{\mathcal{S}}|)$ for the BIC metric. Using the definition of the multivariate factorization for permutation random variables from equation 6.16, and by realizing that $\sigma(\sigma(\sigma(\sigma(\boldsymbol{\mathcal{S}}_i))_{\boldsymbol{j}})) = \sigma(\sigma((\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{j}}))$, the negative log–likelihood of the multivariately factorized probability distribution for permutation random variables can be written as:

$$
\begin{aligned}
-\ln(\mathfrak{L}(\boldsymbol{\mathcal{S}}|\hat{P}_{\boldsymbol{\nu}}(\boldsymbol{\mathcal{O}}))) &= -\sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \ln\left(\hat{P}_{\boldsymbol{\nu}}(\boldsymbol{\mathcal{O}})(\sigma(\sigma(\boldsymbol{\mathcal{S}}_i)))\right) \qquad (6.17)\\
&= \\
-\sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \ln\left(\frac{\prod_{j=0}^{|\boldsymbol{\nu}|-1} |\boldsymbol{\nu}_j|!}{l!} \prod_{j=0}^{|\boldsymbol{\nu}|-1} \hat{P}(O_{\boldsymbol{\nu}_j})(\sigma(\sigma((\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{\nu}_j})))\right)&\\
&= \\
|\boldsymbol{\mathcal{S}}|\ln(l!) - \sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1}\sum_{j=0}^{|\boldsymbol{\nu}|-1} \ln\left(|\boldsymbol{\nu}_j|!\hat{P}(O_{\boldsymbol{\nu}_j})(\sigma(\sigma((\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{\nu}_j})))\right)&
\end{aligned}
$$

We can now determine the difference in negative log–likelihood between two factorizations that differ only by means of a single splice of two factors. Let

$\boldsymbol{\nu^0}$ and $\boldsymbol{\nu^1}$ be node partition vectors for multivariate factorizations such that $\boldsymbol{\nu^1}$ contains all node vectors in $\boldsymbol{\nu^0}$ except two node vectors ${\boldsymbol{\nu^0}}_{s_0}$ and ${\boldsymbol{\nu^0}}_{s_1}$. Furthermore, the only additional node vector that is contained in $\boldsymbol{\nu^1}$ is the node vector ${\boldsymbol{\nu^0}}_{s_0} \sqcup {\boldsymbol{\nu^0}}_{s_1}$. The negative log–likelihood for the multivariate factorization based on $\boldsymbol{\nu^0}$ minus the negative log–likelihood for the multivariate factorization based on $\boldsymbol{\nu^1}$ can be found by canceling terms to be:

$$\sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \ln\left(\hat{P}_{\boldsymbol{\nu}^1}(\boldsymbol{\mathcal{O}})(\sigma(\sigma(\boldsymbol{\mathcal{S}}_i)))\right) - \sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \ln\left(\hat{P}_{\boldsymbol{\nu}^0}(\boldsymbol{\mathcal{O}})(\sigma(\sigma(\boldsymbol{\mathcal{S}}_i)))\right) \tag{6.18}$$

$$=$$

$$\sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \Big[\ln\left((|\boldsymbol{\nu}_{s_0} \sqcup \boldsymbol{\nu}_{s_1}|)!\hat{P}(O_{\boldsymbol{\nu}_{s_0}\sqcup\boldsymbol{\nu}_{s_1}})(\sigma(\sigma((\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{\nu}_{s_0}\sqcup\boldsymbol{\nu}_{s_1}})))\right)$$
$$-\ln\left(|\boldsymbol{\nu}_{s_0}|!\hat{P}(O_{\boldsymbol{\nu}_{s_0}})(\sigma(\sigma((\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{\nu}_{s_0}})))\right) - \ln\left(|\boldsymbol{\nu}_{s_1}|!\hat{P}(O_{\boldsymbol{\nu}_{s_1}})(\sigma(\sigma((\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{\nu}_{s_1}})))\right)\Big]$$

$$=$$

$$|\boldsymbol{\mathcal{S}}| \ln\left(\frac{(|\boldsymbol{\nu}_{s_0} \sqcup \boldsymbol{\nu}_{s_1}|)!}{|\boldsymbol{\nu}_{s_0}|!|\boldsymbol{\nu}_{s_1}|!}\right) + \sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \Big[\ln\left(\hat{P}(O_{\boldsymbol{\nu}_{s_0}\sqcup\boldsymbol{\nu}_{s_1}})(\sigma(\sigma((\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{\nu}_{s_0}\sqcup\boldsymbol{\nu}_{s_1}})))\right)$$
$$-\ln\left(\hat{P}(O_{\boldsymbol{\nu}_{s_0}})(\sigma(\sigma((\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{\nu}_{s_0}})))\right)$$
$$-\ln\left(\hat{P}(O_{\boldsymbol{\nu}_{s_1}})(\sigma(\sigma((\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{\nu}_{s_1}})))\right)\Big]$$

Swap and transfer operations to solve problems with the splice operation

The difference in negative log–likelihood after a splice operation in equation 6.18 indicates that there is a problem if *only* the splice operator is used in the greedy search algorithm when trying to solve additively decomposable optimization problems. The following example serves to makes this intuitively clear.

> **Example 6.7.** Consider an additively decomposable permutation optimization problem. Moreover, consider a run of an IDEA in which two index vectors $\boldsymbol{\iota}_0$ and $\boldsymbol{\iota}_1$ of length 5 have already converged optimally. In other words, every solution in the population now contains the optimal permutation for index vectors $\boldsymbol{\iota}_0$ and $\boldsymbol{\iota}_1$. Furthermore, without loss of generality, assume that the optimal permutations are such that $O_{(\boldsymbol{\iota}_0)_0} < O_{(\boldsymbol{\iota}_0)_1} < \ldots < O_{(\boldsymbol{\iota}_0)_4}$, $i \in \{0, 1\}$. Towards successful convergence of these two index vectors it is quite likely that $\boldsymbol{r}_{(\boldsymbol{\iota}_0)_0} < \boldsymbol{r}_{(\boldsymbol{\iota}_1)_4}$ holds for each random–keys encoded solution in $\boldsymbol{\mathcal{S}}$. If this is so, the greedy splice algorithm for multivariate factorization selection is in its first stages just as likely to choose to splice the singleton node vectors that contain $((\boldsymbol{\iota}_0)_0)$ and $((\boldsymbol{\iota}_0)_4)$, which we want, as it is to splice the singleton node vectors that contain $((\boldsymbol{\iota}_0)_0)$ and $((\boldsymbol{\iota}_1)_4)$, which we do *not* want. When the splicing algorithm has mixed up the index vectors in this way, drawing new random key sequences from the

estimated multivariate factorization is *very* unlikely to generate new correct permutations. This is a problem that does not occur for binary variables, because once all 10 bits have for instance fully converged to 1, it doesn't matter whether we use a full joint factorization $\boldsymbol{\nu} = (((\boldsymbol{\iota}_0)_0, (\boldsymbol{\iota}_0)_1, \ldots, (\boldsymbol{\iota}_0)_4, (\boldsymbol{\iota}_1)_0, (\boldsymbol{\iota}_1)_1, \ldots, (\boldsymbol{\iota}_1)_4))$ or a univariate factorization $\boldsymbol{\nu} = (((\boldsymbol{\iota}_0)_0), ((\boldsymbol{\iota}_0)_1), (\ldots), ((\boldsymbol{\iota}_0)_4), ((\boldsymbol{\iota}_1)_0), ((\boldsymbol{\iota}_1)_1), (\ldots), ((\boldsymbol{\iota}_1)_4))$. Upon sampling, both factorizations will return a binary genotype that has a 1–symbol at all positions of index vectors $\boldsymbol{\iota}_0$ and $\boldsymbol{\iota}_1$.

This problem occurs because *decision errors* are made at a lower dependency level. These lower order errors cannot be avoided and only become visible at a higher dependency level. When regarding dependencies of level 5 for instance, it *does* become clear that the index vectors must be separated. The reason for this is that the individual random key sequences for the index vectors have converged to a single permutation, but other combinations of length 5 lead to random key sequences that represent different permutations throughout $\boldsymbol{\mathcal{S}}$. By definition, the likelihood of the correct factorization is therefore larger.

To overcome this problem, either the size of the sample vector has to increase significantly to reduce the probability of $r_{\iota_0^0} < r_{\iota_4^1}$ or we require a way to correct for lower order decision errors. The problem can be avoided by the greedy incremental algorithm if we allow the splicing of more than 2 vectors at once. However, by allowing the splicing of k vectors, we get a running time complexity of $\mathcal{O}(l^{k+1})$ for the greedy search algorithm. Since this significantly influences the scale–up behavior of the algorithm, we propose to extend the greedy search algorithm by allowing a second operator. This operator allows to correct for lower order decision errors that were made at an earlier stage. This is enforced by allowing two subvectors of the node vector to exchange an element. We call this the *swap* operator. The swap operation that decreases the negative log–likelihood the most is performed first. Since the complexity of the factorization does not increase, no penalization is required for this operation. To ensure that splice operations are only performed when lower order decision errors are no longer visible at the current stage of the greedy algorithm, a swap operation is always preferred over a splice operation.

However, there are situations in which even a swap operation cannot undo a low order decision error, for instance if all but a single index in index vector $\boldsymbol{\iota}_i$ end up in a single node vector and all indices in index vector $\boldsymbol{\iota}_j$, $j \neq i$ plus the missing index from index vector $\boldsymbol{\iota}_i$ end up in a single node vector, then a swap operation at this time can not resolve the problem anymore since for the incomplete node vector to obtain the missing index from the other node vector by a swap operation, it must turn over another index, which will then become the missing index. To this end, we propose to allow a third operator in the greedy learning algorithm, which we call the *transfer* operator. With this operator, we allow a factor $\boldsymbol{\nu}_i$ to transfer a single index to another factor $\boldsymbol{\nu}_j$, which resolves the aforementioned problem.

Additional problems with the use of default tables
Another problem arises when we consider the case in which we use default tables to store the probabilities for each factor. If the length of the default table for factor $\boldsymbol{\nu}_i$ is close to $|\boldsymbol{\mathcal{S}}|$, there is no telling whether this length is representative of the number of *true* permutations or whether this is due to the maximum length of $|\boldsymbol{\mathcal{S}}|$ that any default table can have. This reduces the reliability of the BIC metric, because if factor $\boldsymbol{\nu}_i$ becomes even larger, it *cannot* have a much larger default table length because of the limiting size of the selected sample vector. However, because the number of possible permutations increases factorially, the likelihood of the probability distribution will increase significantly as can be observed in equation 6.18. Since $|\boldsymbol{\theta} \xleftarrow{fit} \boldsymbol{\nu}_i|$ now equals the default table length instead of $|\boldsymbol{\nu}_i|!$, there is hardly any complexity penalization.

To remedy this problem, we require a cutoff value $\xi \in [0, 1]$ that defines the maximum default table length to be $\xi|\boldsymbol{\mathcal{S}}|$. No operation is allowed to create a factor that has a default table longer than $\xi|\boldsymbol{\mathcal{S}}|$. The rationale behind ξ is that if the default table length becomes larger than $\xi|\boldsymbol{\mathcal{S}}|$, we decide that the reason for this length is uncertain. Without this restriction, there would be an *early* drift towards large factors when $|\boldsymbol{\nu}_i|!$ starts to get larger than $\boldsymbol{\mathcal{S}}$.

6.4 More efficient probabilistic processing of random keys: $\mathbb{ICE}$

If we use normal gpdfs to estimate a real–valued probability distribution based on random keys genotypes, we are likely to have difficulties with representing the underlying permutation–based dependencies because of the redundancy of the random keys encoding. Furthermore, the normal gpdf has some limitations with respect to non–linearity and multimodality. To overcome these problems, the random keys sequences should be interpreted in the permutation space. In Section 6.3 we already introduced a way to estimate probability distributions over the permutations that are encoded by the random keys directly. Another way to cope with the redundancy of the random keys encoding, is to *mix* the random keys sequences using crossover. Only combinations of the initial strings are thereby generated, just as is done in GAs, which inherently reduces the influence of the redundancy of the random keys encoding.

In this section we propose a framework in which crossover is performed in such a way that the crossover operator respects the dependency information that is obtained by the selection of a probabilistic model as well as possible. This approach is thus actually a way to perform crossover linkage learning for permutation spaces. Based on the type of probabilistic model that is used, it is to be expected that more or less efficiently scalable permutation EAs can be constructed that are at least able to efficiently solve additively decomposable relative–ordering deceptive permutation optimization problems.

6.4.1 The ℐℂ𝔼 framework

The way in which crossover is done, determines the rate of success for a GA. It should be noted that *if* the combinations of random keys for the dependent positions in a relative–ordering permutation optimization problem are exchanged between parents as a whole, the information is mixed in the most efficient mixing manner, which resolves the redundancy problem. To find the information regarding the dependent random keys positions, we rely on the probabilistic model learning part in the ID𝔼A framework to find a structure that contains this information. For instance, if we have a multivariate factorization, we are given groups of random keys that should each be processed in a multivariate joint gpdf. Therefore, these random keys should be processed together as a block, reflecting our belief that such blocks are a good approximation of the true building blocks in the problem. By respecting the boundaries of these blocks in crossover, a linkage friendly crossover operator is obtained.

In general, we can attempt to construct a crossover operator based on a probabilistic model that was learned from the selected solutions in such a way that the dependencies between the random variables are respected as well as possible. If these dependencies reflect the structure of the problem well, we are also more likely to mix good subsolutions, which we know to be an important prerequisite for the success of a GA.

Although this way of constructing crossover operators is applicable to both integer genotypes for other types of optimization problems as well as to random keys genotypes for permutation optimization problems, there is an important additional transformation that can be performed in the case of random keys genotypes. Before copying a block to the offspring, we propose to allow each block to be transformed using a function $\varrho(\cdot)$. This function *rescales* the random keys to a randomly selected subinterval of $[0, 1]$ with probability p_ϱ. We propose to divide the $[0, 1]$ interval into equal–sized subintervals from which one subinterval may be chosen randomly. The number of subintervals is a parameter to the operator. For instance, if we scale $(0.1, 0.2, 0.3)$ to $[0.9, 0.95]$, we get $(0.9, 0.925, 0.95)$. Note that this doesn't change the permutation that is encoded. We call this transformation function *random rescaling*. The optimization problem can require a relative ordering of the building blocks. Without random rescaling, we have to rely on the random key combinations that are generated initially. Rescaling the blocks can increase the probability that they will be combined properly.

By crossing over chromosome elements, or genes, instead of using sampling from the estimated probability distribution, EAs can be constructed that differ slightly from ID𝔼As. We call an algorithm that differs from an ID𝔼A only in that the sampling from the estimated probability distribution in the recombination phase is replaced with the crossing over of blocks of genes such that the block information is derived from the probabilistic model that was learned, an ID𝔼A *Induced Chromosome Elements Exchanger* (ℐℂ𝔼).

In the instances of ℐℂ𝔼 that we shall work with, crossover is performed by randomly selecting two parents from the selected samples and crossing over

```
ℐℂℰ
(Two–parent crossover instance for random keys genotypes)
(Procedure to generate a single new solution)
1  par_0 ← RANDOM({0, 1, ..., |S| − 1})
2  par_1 ← RANDOM({0, 1, ..., |S| − 1} − {par_0})
3  B ← CROSSOVERBLOCKS(ς)
4  r ← new vector of real with size l
5  for i ← 0 to |B| − 1 do
     5.1  par ← RANDOM({par_0, par_1})
     5.2  for j ← 0 to |B_i| − 1 do
            5.2.1  r_(B_i)_j ← ϱ((S_par)_(B_i)_j)
6  RETURN(r)
```

Figure 6.4: Pseudo–code for the phase in $\mathbb{ICE}$ in which new solutions are generated by using block–crossover. The blocks are determined based on the information learned and stored in the probabilistic model structure ς.

blocks in the solutions. Which blocks we actually perform crossover with, is determined by the type of probabilistic model. Since the resulting EA uses crossover, it can validly be argued that we have designed a GA. The specialty of this GA is that it attempts to learn linkage information and use this linkage information in a linkage preserving crossover operator. Pseudo–code for the phase in which new solutions are generated in $\mathbb{ICE}$ using block–crossover for our two–parent crossover instance combined with the random keys genotype is given in Figure 6.4.

6.4.2 Specific $\mathbb{ICE}$ instances based on factorizations

To obtain actual instances of the $\mathbb{ICE}$ framework, we are required to indicate how the crossover blocks $\boldsymbol{\mathcal{B}}$ can be constructed from a probabilistic model structure ς. We focus on block–construction mechanisms for factorizations since factorizations are capable of directly representing multivariate dependencies. Although multivariate factorizations are the most straightforward for deriving the crossover blocks, Bayesian factorizations allow for more precise probabilistic modelling since any multivariate factorization can be expressed as a Bayesian factorization, but not vice versa. Therefore, it is interesting to base crossover operators on both types of factorizations.

Crossover blocks based on multivariate factorizations

For multivariate factorizations, the resulting crossover operator is a true multivariate joint dependency block mixing operator. By crossing over the random keys based on the node partition vector $\boldsymbol{\nu}$, we attempt to directly exchange and mix the important blocks of information. This results in an approach similar to the ECGA for binary variables by Harik (1999). Pseudo–code for the resulting crossover operator is given in Figure 6.5.

```
CrossoverBlocks(ν)
1 Return(ν)
```

Figure 6.5: Pseudo–code for deriving crossover blocks from a multivariate factorization structure. The blocks to perform crossover with are given by the node vectors that indicate the belief in multivariate joint dependencies within blocks of random variables.

```
CrossoverBlocks(π)
1 pos_0 ← Random({0, 1, ..., l})
2 pos_1 ← Random({0, 1, ..., l})
3 if pos_1 < pos_0 then
   3.1 pos_0 ↔ pos_1
4 ω ← TopologicalSort(π)
5 Return(((ω_0, ω_1, ..., ω_(pos_0−1)),
          (ω_pos_0, ω_(pos_0+1), ..., ω_(pos_1−1)),
          (ω_pos_1, ω_(pos_1+1), ..., ω_(l−1))))
```

Figure 6.6: Pseudo–code for deriving crossover blocks from a chain Bayesian factorization structure. The blocks to perform crossover with are given by an application of two–point crossover to the ordering of the variables in the chain. This ordering can be obtained by performing a topological sort on the parent structure of the Bayesian factorization. The resulting $\mathbb{ICE}$ algorithm is essentially a position biased two–point crossover GA.

Crossover blocks based on Bayesian factorizations

To use Bayesian factorizations, we cannot use the unrestricted Bayesian factorization since there is no straightforward way in which we can order the variables to derive a linkage friendly crossover operator. To still be able to use Bayesian factorizations, we propose to learn a chain of dependencies in which random keys that are important together, are placed close to each other. Subsequently, we can apply for instance two point crossover such that the linkage information in the chain is respected. This approach was recognized earlier (Bosman and Thierens, 1999a) to be interesting for processing linkage. To find a chain, the greedy entropy algorithm in MIMIC (de Bonet et al., 1996) can be used. The entropy, which equals the average negative log–likelihood for normal gpdfs (Bosman and Thierens, 2000c), can be computed using the normal gpdf as we have done for all real–valued $\mathbb{IDEA}$s. Pseudo–code for determining the blocks to be exchanged based on the chain–structured factorization, is given in Figure 6.6.

6.5 Experiments

In this section, we test different $\mathbb{IDEA}$ and $\mathbb{ICE}$ instances on the two additive optimization problems as defined in Section 6.2. We varied the dimensionality to

investigate the scale–up behavior on relative–ordering additively decomposable permutation optimization problems. A large variety of IDEA and ICE instances have been tested to get a good impression of what operators are important in learning and using probabilistic models in permutation optimization. In Section 6.5.1 we shortly describe our test problems. In Section 6.5.2 we describe our experiment setup and in Section 6.5.3 we present the obtained results. Finally, in Section 6.5.4 we give a short summary for the EA practitioner.

6.5.1 Permutation optimization problems

Our test suite consists of the two problems as defined in Section 6.2. Both problems are both a sum of multiple repetitions of the deceptive permutation subfunction by Knjazew (2000). If the subfunctions do not overlap, the optimization problem is additively decomposable.

Non–overlapping subfunctions
For the problem variant that is based on loosely encoded non–overlapping index vectors $\boldsymbol{\iota}_j^{loose}$, we have used index vector lengths of $\kappa^\iota \in \{4, 5\}$.

Partly–overlapping subfunctions
For the additive problem variant based on tightly encoded overlapping index vectors $\boldsymbol{\iota}_j^{overlap}$, we have only used an index vector length of $\kappa^\iota = 4$ since the overlapping property of the problem makes optimization much harder.

6.5.2 Experiment setup

Optimization problem dimensionalities

We have used different problem dimensionalities for the non–overlapping and the overlapping variants of the sum of deceptive subfunctions.

Non–overlapping subfunctions
For the problem variant that is based on loosely encoded non–overlapping index vectors $\boldsymbol{\iota}_j^{loose}$, we have used different problem lengths of $l \in [15, 76]$ such that $l \bmod \kappa^\iota = 0$. We have used these different problem lengths to investigate the scale–up behavior of different algorithms on the additively decomposable variant of the sum–of–deceptive–subfunctions optimization problem.

Partly–overlapping subfunctions
For the overlapping problem, we have fixed the problem length to $l \in \{10, 19\}$. This corresponds to index cluster vector lengths of $|\boldsymbol{\iota}| \in \{3, 6\}$, which equals the number of subfunctions in the partially–overlapping variant of the sum–of–deceptive–subfunctions optimization problem.

General algorithmic setup

We ran tests for all algorithms to find the minimum required population size and associated average required number of evaluations to solve the optimization problems. We consider an optimization problem to be solved if the optimal solution is found in all of the 30 independent runs that we carried out. Furthermore, we did not allow the population size to become larger than 10^5.

Algorithms

We tested both IDEAs as well as ICE variants using different gpdfs. In the following, we shall briefly describe the different settings that we have used.

IDEA

We used the rule of thumb by Mühlenbein and Mahnig (1999) for FDA and set the selection threshold τ to 0.3. Furthermore, we set the probability at recombination to $p_r = 1$ for all IDEA instances. We tested IDEA instances based on the normal gpdf for the random keys encoding by learning multivariate factorizations with both the AIC and the BIC metric. We applied the use of the permutation gpdf by learning multivariate factorizations for them, using both the AIC and BIC metric in combination with frequency tables. Furthermore, next to using only the splice operation in the greedy factorization selection algorithm, we also investigated the added use of the swap operation to greedily construct a multivariate factorization.

ICE

For each ICE instance, we used the same general settings of $\tau = 0.3$ and $p_r = 1$ that we used for each IDEA instance.

We tested ICE instances based on the normal gpdf for the random keys encoding by learning multivariate factorizations with both the AIC and the BIC metric. Although the swap operation was proposed and motivated in combination with the permutation gpdf, we investigate its added use in combination with the normal gpdf as well.

We also learned constrained Bayesian factorizations based on the normal gpdf by using the greedy chain learning algorithm as used in MIMIC (de Bonet et al., 1996) that was described earlier in Chapter 4. The chain–normal Bayesian factorization was used in combination with the position–based two–point crossover operator in ICE that was described in Section 6.4.

We learned multivariate factorizations using the permutation gpdf in the same way as we did for the IDEA instances that are based on the permutation gpdf. In addition to the swap operation, we also investigated the added use of the transfer operation to greedily construct a multivariate factorization. Next to using the frequency tables representation based on the bijective integer mapping, we also used the default tables representation and empirically determined a good value for the cutoff parameter ξ as is described in Section 6.5.3.

We also learned constrained Bayesian factorizations based on the permuta-

tion gpdf by using the same greedy chain learning algorithm as we used for the normal gpdf and used the same position biased two–point crossover operator to obtain an $\mathbb{ICE}$ instance.

6.5.3 Results

We have tested many different combinations of gpdfs and greedy factorization selection operations for the IDEA instances as well as for the $\mathbb{ICE}$ instances. In the following, we will present and describe the obtained results on each optimization problem in turn. First, we present the results obtained for the additively decomposable problem. Subsequently, we present the results obtained for the problem in which the subfunctions partly overlap.

Non–overlapping subfunctions of length 4

Normal gpdf

- **IDEA based on multivariate factorizations**
  Figure 6.7 shows the average number of evaluations and the minimal population size that are required for various IDEAs and $\mathbb{ICE}$ algorithms to terminate such that the algorithm has converged to the optimal solution in all of the 30 independent runs. The results in Figure 6.7 pertain only to EAs based on the normal gpdf. None of the pure IDEAs in Figure 6.7 scale up polynomially. It is clear that using the univariate factorization scales up significantly worse than when problem structure is attempted to be exploited by using multivariate factorizations. However, even using the perfect factorization leads to an IDEA that is not capable of polynomial scale–up behavior. Therefore, it is of no use to attempt to exploit additively decomposable relative–ordering deceptive problem structure by learning factorizations for the normal gpdf. Indeed, using either the AIC metric or the BIC metric to learn a multivariate factorization also leads to an IDEA that scales up exponentially. The bad scale–up behavior of real–valued IDEAs using the random keys representation is a result of the redundancy problems as described earlier in Section 6.3 and the limitations of the normal gpdf. For these reasons, we will not investigate the use of IDEAs based on the normal gpdf and the random keys representation for permutation optimization any further.

- **$\mathbb{ICE}$ based on Bayesian chain factorizations**
  Figure 6.7 also shows results for $\mathbb{ICE}$ variants based on the normal gpdf in combination with Bayesian chain factorizations. Unfortunately, using this approach does not lead to polynomial scale–up behavior unless the chain structure is fixed beforehand. In that case, the two–point crossover operator is likely to effectively mix the building blocks. Since in a practical application we do not know this optimal structure beforehand, we must attempt to learn it. However, it is hard to find the right second order linkage information if we are not allowed to induce higher order linkage information as well. The experimental results verify this statement because the

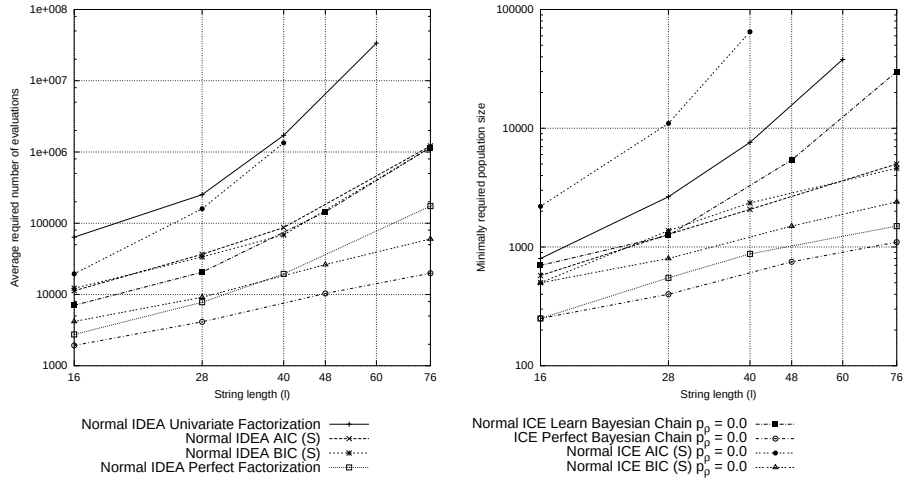


Figure 6.7: Scale–up behavior on the additively decomposable relative–ordering deceptive problem with subfunctions of length 4. The results are averaged over 30 runs and indicate the minimal computational requirements to find the optimal solution in all 30 runs. The results are plotted on a log–log scale. (truncation selection $[\tau = 0.3]$; $\{\mathrm{IDEA}, \mathrm{ICE}\}$ $[p_r = 1.0, p_\varrho = 0.0$, *gpdf* $=$ *normal*, *factorization* $\in$ {*(univariate)*, *(multivariate, AIC)*, *(multivariate, BIC)*, *(perfect)*}, *operators* $=$ *splice*]; no mutation; with elitism [*replace all non–selected solutions*]).

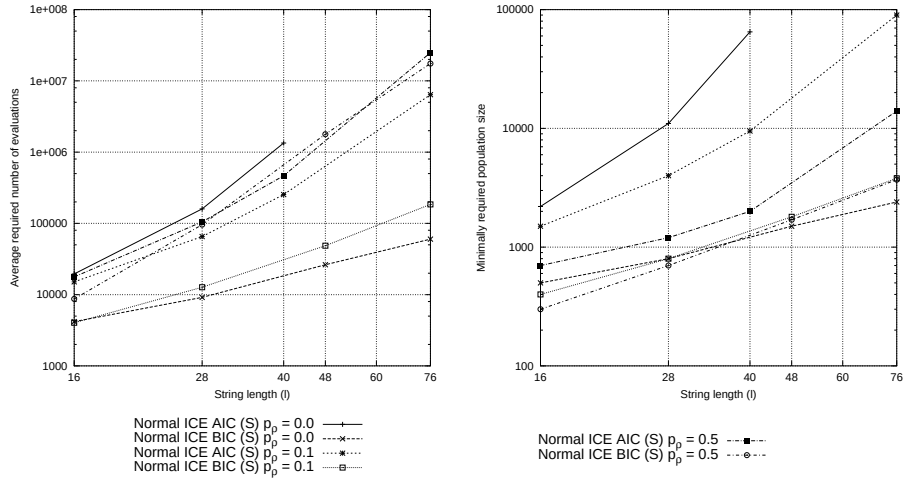


Figure 6.8: Scale–up behavior on the additively decomposable relative–ordering deceptive problem with subfunctions of length 4. The results are averaged over 30 runs and indicate the minimal computational requirements to find the optimal solution in all 30 runs. The results are plotted on a log–log scale. (truncation selection $[\tau = 0.3]$; ICE $[p_r = 1.0, p_\varrho \in \{0.0, 0.1, 0.5\}$, *gpdf* $=$ *normal*, *factorization* $\in$ {*(multivariate, AIC)*, *(multivariate, BIC)*}, *operators* $=$ *splice*]; no mutation; with elitism [*replace all non–selected solutions*]).

learning approach for Bayesian chain factorizations clearly scales up a lot worse and not polynomially. We therefore conclude in a similar fashion as has been done for binary spaces (Pelikan and Mühlenbein, 1999; Bosman and Thierens, 1999a) that finding and exploiting lower order linkage information is less efficient than finding and exploiting higher order linkage information for problems such as the one at hand. For these reasons, we will not investigate the use of $\mathbb{ICE}$ variants based on Bayesian factorizations for the normal gpdf any further. If we still want to use Bayesian factorizations and achieve polynomial scale–up behavior, we must be able to effectively learn Bayesian factorizations without additional restrictions on the factorization graph and use these in an ID$\mathbb{E}$A. From the results of the perfect multivariate factorization, it should be clear that to this end the Bayesian factorization should not be combined with the normal gpdf.

- **$\mathbb{ICE}$ based on multivariate factorizations**
  If the BIC metric is combined with the learning of multivariate factorizations based on the normal gpdf, the use of $\mathbb{ICE}$ gives significantly better results. The experiments indicate that the scale–up behavior of this type of $\mathbb{ICE}$ algorithm is polynomial. The penalization of the AIC metric combined with the normal gpdf is too small. The use of the AIC metric causes the factors in the multivariate factorization to become too large and to sometimes even form a single factor containing all random variables, which does not lead to efficient exploitation of the decomposable problem structure.

- **Influence of random rescaling**
  Although random rescaling can help if the optimization problem has subfunctions that partially overlap, we are normally not aware of whether or not partially overlapping subfunctions exist. Therefore, we would like to always apply random rescaling at some rate to make sure that if partial overlapping subfunctions exist, our EA can tackle this difficulty efficiently. It is therefore of great interest to see the implication of random rescaling on solving the additively decomposable optimization problems. If the structure of additively decomposable problems is correctly found, introducing random rescaling should not matter. However, if the index cluster boundaries are not completely found, random rescaling may introduce additional disruptiveness to the crossover operation. This can indeed be seen to be the case in Figure 6.8 as the scale–up coefficients for the results obtained with the BIC metric worsen as p_ϱ increases. However, the scale- up behavior does stay polynomial. This indicates that the basis of problem structure induction in this variant of $\mathbb{ICE}$ is competent for solving relative–ordering additively decomposable optimization problems. It should be noted that the use of the AIC metric does not lead to polynomial scale–up for any probability of random rescaling. However, the performance does improve if this probability is set larger than 0. The reason for this is that the entropy of the selected set of solutions

increases, which reduces the size of the multivariate factors that are created. However, the use of random rescaling does not help in finding the perfect factorization since it adds additional disruptiveness. Therefore the scale–up behavior is not reduced to polynomial scale–up. Furthermore, if the probability of random rescaling is increased further, the increase in disruptiveness only causes the scale–up behavior to become worse since it does not additionally help in finding the perfect factorization.

- **Overall**
  Concluding, the only way in which additive decomposability of relative–ordering deceptive problems is effectively exploited using IDEAs and ICE algorithms based on the normal gpdf, is to use the BIC metric in combination with multivariate factorizations in ICE.

Permutation gpdf

Before we investigate the scale–up behavior of the different proposed EAs based on the permutation gpdf, we must first decide which value of ξ to use for the variants in which we use default tables to store the parameters of the permutation multivariate factorization. In figure 6.9, the performance for ICE with permutation default tables is plotted for different values for ξ on the additively decomposable problem with $\kappa^{\iota} = 5$ and $l = 25$. The results show a wide range of values that lead to a similar performance. The optimal value of $\xi = 0.1$ is most likely biased towards the exploitation of the subfunction size. Since $\xi = 0.3$ is therefore probably a more generally applicable value, we will use this setting in the remainder of our experiments whenever we use default tables. Figures 6.10 and 6.11 show the scale–up behavior for various IDEAs and ICE algorithms that use the permutation gpdf.

- **IDEA based on multivariate factorizations**
  The experiments indicate that all pure IDEAs scale up polynomially. However, pure IDEAs are clearly outperformed by their corresponding ICE algorithms in which the sampling phase is replaced by the block crossover operator as can be seen in Figure 6.11. One of the most important reasons for this is that the crossover operator is less disruptive. This is especially true if we have an additively decomposable optimization problem. Assume for an index vector $\boldsymbol{\iota}_i$ of length 4 that the factorization has spliced no further than $((\iota_0^i, \iota_1^i), (\iota_2^i, \iota_3^i))$. If only one of the parents has a good solution for the index vector, the probability that it is crossed over to the offspring is $\frac{1}{4}$. But even if the block has *fully* converged in $\boldsymbol{\mathcal{S}}$, the probability that the optimal block will be constructed by sampling the factorization given above, is only $\frac{2!2!}{4!} = \frac{1}{6}$. A similar argument holds when lower order decision errors have found their way into the final factorization. Because of this inherent more efficient capability of ICE to exploit relative–ordering additive decomposability if we are not given the perfect factorization, we will not investigate the use of pure IDEAs using multivariate factorizations based on the permutation gpdf any further.

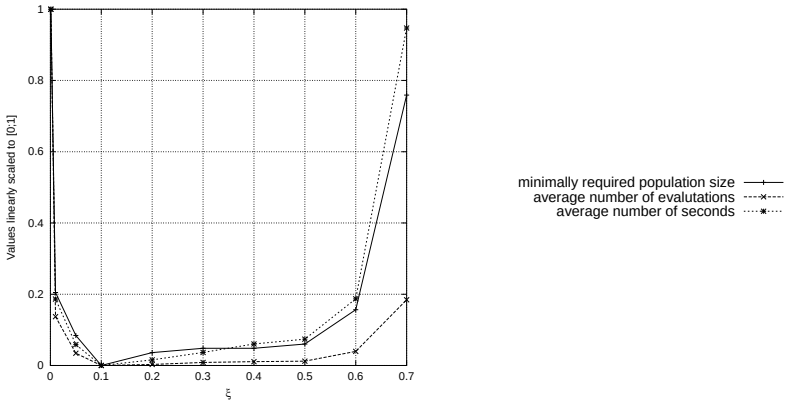


Figure 6.9: The performance of $\mathbb{ICE}$ using default tables to represent the parameters of the permutation multivariate factorization on the additively decomposable relative–ordering deceptive problem with 5 subfunctions of length 5 as a function of ξ. The results are averaged over 30 runs and indicate the minimal computational requirements to find the optimal solution in all 30 runs. Since we are not interested in the absolute values, we scaled the results to $[0; 1]$.

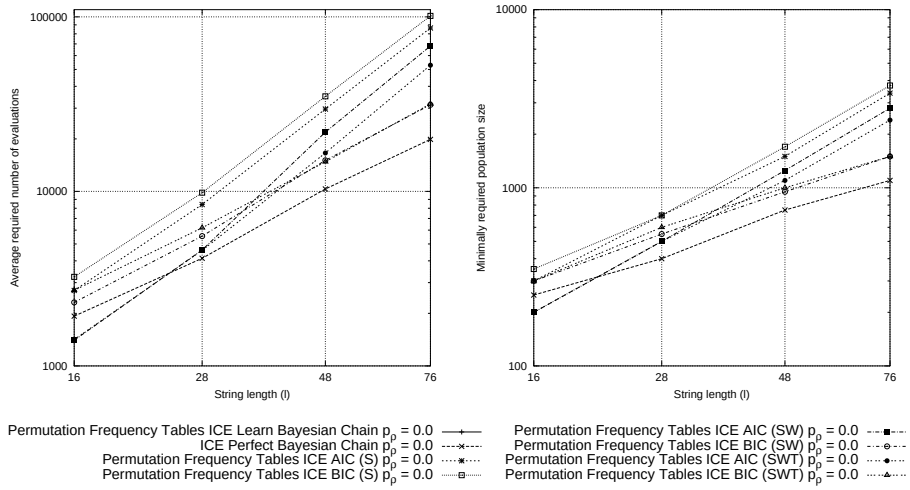


Figure 6.10: Scale–up behavior on the additively decomposable relative–ordering deceptive problem with subfunctions of length 4. The results are averaged over 30 runs and indicate the minimal computational requirements to find the optimal solution in all 30 runs. The results are plotted on a log–log scale. (truncation selection $[\tau = 0.3]$; $\mathbb{ICE}$ $[p_r = 1.0, p_\varrho = 0.0$, *gpdf* = *permutation (frequency tables)*, *factorization* $\in \{$*(multivariate, AIC)*, *(multivariate, BIC)*$\}$, *operators* $\in \{$*splice*, (*splice*,*swap*), (*splice*, *swap*,*transfer*)$\}$]; no mutation; with elitism [*replace all non–selected solutions*]).

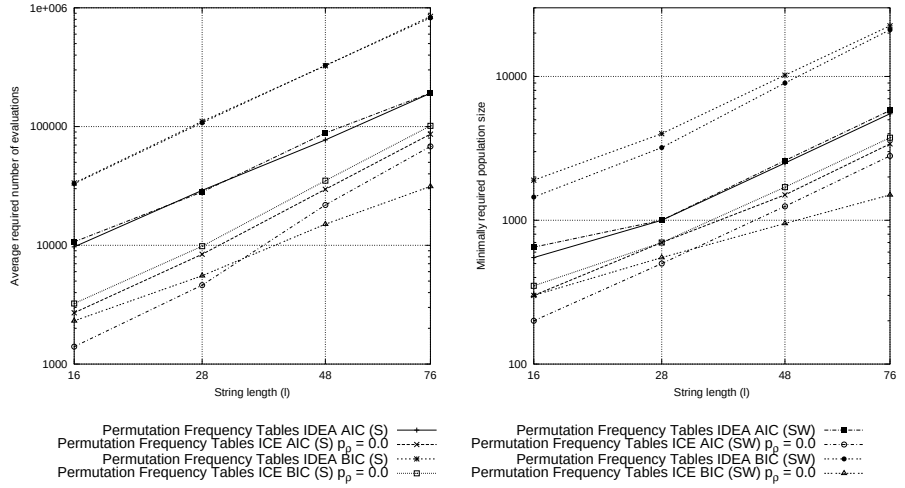


Figure 6.11: Scale–up behavior on the additively decomposable relative–ordering deceptive problem with subfunctions of length 4. The results are averaged over 30 runs and indicate the minimal computational requirements to find the optimal solution in all 30 runs. The results are plotted on a log–log scale. (truncation selection [$\tau = 0.3$]; $\{\mathbb{IDEA}, \mathbb{ICE}\}$ [$p_r = 1.0, p_\varrho = 0.0$, *gpdf* = *permutation (frequency tables)*, *factorization* $\in \{$*(multivariate, AIC)*, *(multivariate, BIC)*$\}$, *operators* $\in \{$*splice*, (*splice*,*swap*)$\}$]; no mutation; with elitism [*replace all non–selected solutions*]).

- **$\mathbb{ICE}$ based on Bayesian chain factorizations**
  Only the use of the Bayesian chain factorization does not lead to polynomial scale–up behavior unless the perfect chain structure is provided. The number of evaluations required is even outside of the range we chose to plot in Figure 6.10 Again, the use of lower order linkage information does not lead to efficient exploitation of the additively decomposable problem structure and thus we will not investigate the use of Bayesian chain factorizations any further.
- **$\mathbb{ICE}$ based on multivariate factorizations**
  Contrary to when the normal gpdf is used, using the AIC metric in combination with permutation gpdfs *does* result in $\mathbb{ICE}$ algorithms that scale up polynomially. The reason for this is that in the case of the permutation gpdf, the number of parameters grows factorially instead of quadratically. As a result, the penalization for a certain multivariate factorization will be much larger when using permutation gpdfs. Allowing the use of the swap operator improves the scale–up behavior of $\mathbb{ICE}$ because lower–order decision errors can be corrected. Using the transfer operator improves the scale–up behavior even further. This is especially the case for the BIC metric as can be seen in Figures 6.10 and 6.11. The length of the subproblems is however quite small; each subproblem only has $4! = 24$ different solutions. As the length of the subproblems increases, the benefits of using the swap and transfer operators are expected to become even clearer.

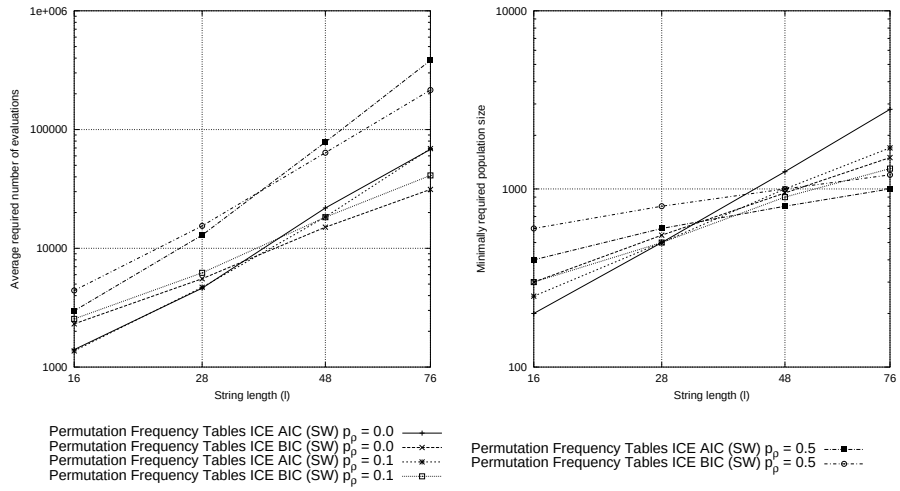


Figure 6.12: Scale–up behavior on the additively decomposable relative–ordering deceptive problem with subfunctions of length 4. The results are averaged over 30 runs and indicate the minimal computational requirements to find the optimal solution in all 30 runs. The results are plotted on a log–log scale. (truncation selection [$\tau = 0.3$]; $\mathbb{ICE}$ [$p_r = 1.0, p_\varrho \in \{0.0, 0.1, 0.5\}$, *gpdf* = *permutation (frequency tables)*, *factorization* $\in$ {*(multivariate, AIC)*, *(multivariate, BIC)*}, *operators* = (*splice*, *swap*)]; no mutation; with elitism [*replace all non–selected solutions*]).

- **Influence of random rescaling**
  The results in Figure 6.12 serve to indicate that the basis of problem structure exploitation in the $\mathbb{ICE}$ algorithms based on multivariate factorizations and permutation frequency tables is competent. The added use of random rescaling does not cause the scale–up behavior to become exponential, although it does cause the degree of the polynomial that describes the scale–up behavior with respect to the required number of evaluations to increase. Similar arguments hold for the representation in which default tables are used. With a cutoff value of $\xi = 0.3$, $\mathbb{ICE}$ algorithms based on the permutation gpdf and default tables work similar on relative–ordering additively decomposable optimization problems to $\mathbb{ICE}$ algorithms in which frequency tables are used.

- **Overall**
  In Appendix B Section B.2 (Figure B.9) an overview is given of the scale–up behavior of all $\mathbb{ICE}$ algorithms. The actual scale–up coefficients were computed with a least squares line fit on the log–log scale. It is important to note that although simple EA approaches scale up exponentially on the relative–ordering additively decomposable optimization problem, a few $\mathbb{ICE}$ algorithms are capable of subquadratic scale–up with respect to the average required number of evaluations. Unfortunately, the time require-

ments scale up worse than subquadratic. The reason for this is that the factorization selection algorithm already has a running time complexity of $\mathcal{O}(l^3)$. This represents an important trade–off between minimizing the number of required evaluations by improving the mechanism of induction in an EA to produce better solutions based on less previously evaluated solutions and the required running time.

Non–overlapping subfunctions of length 5

It seems from the results in Figure B.9 for the subfunctions of length 4 that the amount of penalization has an important role in determining the scale–up potential of $\mathbb{ICE}$. If we furthermore recall that the added benefit of using the swap operator increases with the length of the subfunctions, it is reasonable to decide that in the remainder of our experiments we will only consider i) the use of the AIC metric in combination with the permutation gpdf, frequency tables and the splice and swap operators, and ii) the use of the BIC metric in combination with the normal gpdf and the splice operator or the permutation gpdf with at least the swap operator allowed as well. In Figure 6.13 the scale–up behavior of a selection of these algorithms is shown with the probability of random rescaling set to 0. The results for all tested algorithms are tabulated in Appendix B Section B.2 (Figure B.10).

Normal gpdf

- **Overall**
  The variant of $\mathbb{ICE}$ that uses the normal gpdf is not capable of solving the problem at all for $p_\varrho = 0.5$. For $p_\varrho = 0.1$, the scale–up behavior is not effected too much. Although the scale–up behavior for this variant of $\mathbb{ICE}$ is actually not that bad, the constant difference with approaches based on the normal gpdf is quite large as can be seen in Figure 6.13. Thus, with respect to inductive capabilities and the resulting required number of evaluations, the use of the normal gpdf is not preferable.

Permutation gpdf

- **Normal $\mathbb{ICE}$ versus permutation $\mathbb{ICE}$**
  Although the use of permutation gpdfs leads to a better scale–up behavior, there is a negative side to their use. Because the permutation frequency tables are of maximum size $\kappa!$ and because we require $\mathcal{O}(\kappa \log \kappa)$ time to convert random keys of length κ to integers, the actual running time of the algorithms based on the permutation gpdf is much larger than when the normal gpdf is used. At $l = 50$ for instance, the best performing normal $\mathbb{ICE}$ runs at a factor of 27.13 times faster than the best performing permutation $\mathbb{ICE}$ on the additively decomposable test problem with subfunctions of length 5. Asymptotically speaking, the scale–up behavior of the different $\mathbb{ICE}$ variants is similar, but using permutation gpdfs has a smaller constant overhead. Thus, especially if the fitness function is time consuming, variants of $\mathbb{ICE}$ based on permutation gpdfs are preferable.

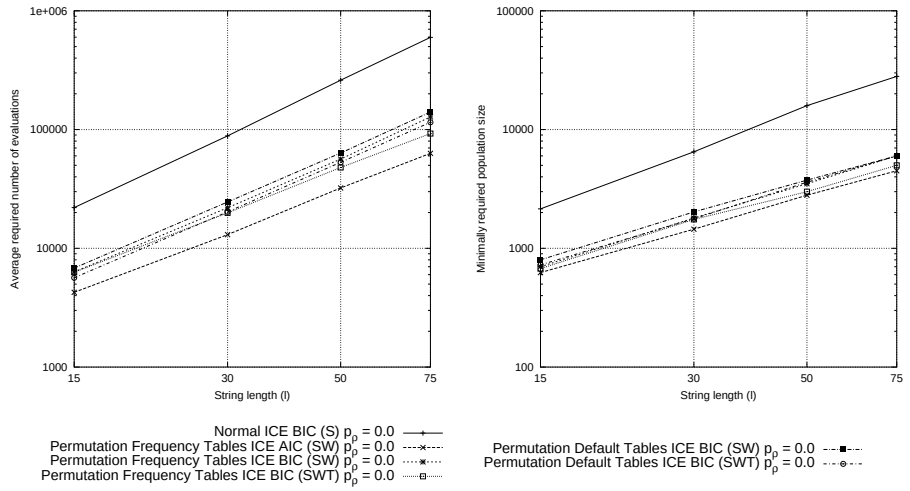


Figure 6.13: Scale–up behavior on the additively decomposable relative–ordering deceptive problem with subfunctions of length 5. The results are averaged over 30 runs and indicate the minimal computational requirements to find the optimal solution in all 30 runs. The results are plotted on a log–log scale. A selection of all tested algorithms is shown. The algorithms are the same as those used for the experiments with subfunctions of length 4, the parameter details of which can be found in figures 6.7 through 6.12.

- **AIC metric versus BIC metric**
  The best scale–up behavior is obtained if the permutation gpdf is used in combination with frequency tables and the AIC metric. This is a result of the smaller penalization in the AIC metric, which results in larger factors for smaller population sizes. This can lead to overly complex models, for which reason the BIC metric is often preferred. In our case however, it introduces a useful bias for correctly finding the index clusters, especially since the maximum factor size was limited to 7. Therefore, for a more generally applicable and less biased approach, we prefer the BIC metric.

- **Operators for learning multivariate factorizations**
  The use of the transfer operation shows a benefit over using only the splice and swap operations. However, although the scale–up behavior indeed improves, the constant time overhead is increased by allowing another operator in the greedy factorization selection algorithm. This is again representative of the trade–off between competence in induction and required running time. A rule–of–thumb is to only use the splice operator and the swap operator since both the swap operator and the transfer operator allow for the correction of lower–order decision errors in the greedy factorization selection process, but their joint availability does not improve the results much more than if only one of them is available. Yet, the additional time requirements are twice as large.

Overlapping subfunctions of length 4

In Appendix B Section B.2 (Figure B.11), the results for the additively partially overlapping relative–ordering deceptive problem are tabulated. The results show the minimal requirements on the algorithms to solve the problem optimally in all of 30 independent runs or the performance at $n = 10^5$.

Normal gpdf and permutation gpdf

- **Influence of random rescaling**
  No algorithm was capable of optimizing the largest problem without random rescaling. With only a small probability of random rescaling, the overlapping deceptive problem can already be solved optimally for $n \leq 10^5$ quite efficiently. As the length of the problem increases, the added use of random rescaling improves the results of the different $\mathbb{ICE}$ variants. Furthermore, increasing the probability of random rescaling improves the results further for all algorithms with the exception of the $\mathbb{ICE}$ algorithm in which default tables are used for the permutation gpdf. Moreover, the results on the relative–ordering additively decomposable deceptive optimization problems become worse as the probability of random rescaling is increased. Therefore, we suggest that a small p_ϱ should be used in general.

Permutation gpdf

- **Frequency tables versus default tables**
  Since $\mathbb{ICE}$ using default tables for the permutation gpdf in combination with a small probability of random rescaling gives the best results over all algorithms for the larger instance of the overlapping test problem, it is to be expected that this $\mathbb{ICE}$ variant is the most preferable for relative–ordering additively overlapping problems. Moreover, this approach also gave good results on the additively decomposable optimization problem. Combined with the fact that the use of default tables allows $\mathbb{ICE}$ to learn a more accurate probabilistic model for describing problem structure, this variant of $\mathbb{ICE}$ is arguably the best generally applicable and most flexible of all tested $\mathbb{ICE}$ algorithms.

6.5.4 Practitioner's summary

The algorithms that have experimentally been found to be the best at inducing and exploiting structural features in permutation problem are variants of $\mathbb{ICE}$. The use of default tables indicates a slightly larger overall requirement on the computational resources. This argument, combined with the advantage that default tables allow more complex models to compete in model selection leads us to conclude that the use of default tables with the BIC metric and a small probability of using random rescaling is the most effective allround variant of $\mathbb{ICE}$ if no a priori exploitable problem–specific information is available. To implement this variant of $\mathbb{ICE}$, the IDEA framework needs to be instantiated

with the greedy learning of multivariate factorizations. The details of this can be found in Section 2.6. In addition, the swap operator has to be implemented, the details of which can be found in Section 6.3.2. Moreover, the permutation gpdf should be used. The details of this gpdf regarding its estimation from data are given in Section 6.3.2. The parameters of the gpdf should be represented using default tables, the details of which can be found in Section 6.3.2 also. The drawing of samples does not need to be implemented since the block crossover operator should be used to obtain an $\mathbb{ICE}$ instance (Section 6.4).

With respect to the requirements on the population size and the number of evaluations, the $\mathbb{ICE}$ algorithms can outperform previously proposed EAs on additively decomposable and additively overlapping relative–ordering problems. This includes the OmeGA, as was shown for a larger additively overlapping problem than the one used in this chapter (Bosman and Thierens, 2001b). However, the time requirements for probabilistic modelling are significant. It is important to note that $\mathbb{ICE}$ algorithms based on the normal gpdf have considerable less overhead and run faster if we disregard the time requirements for function evaluations. However, the use of normal gpdfs leads to less efficient permutation–based induction. Therefore, if the fitness function is highly time consuming, permutation $\mathbb{ICE}$ is preferred, otherwise, normal $\mathbb{ICE}$ is preferred. Thus, normal $\mathbb{ICE}$ is an interesting alternative. To implement normal $\mathbb{ICE}$, the only ingredient that has to be changed is the gpdf, which should be the normal gpdf. The required details on this gpdf are given in Section 5.1.1.

The problems used for testing in this chapter are not real–world practical problems. Although $\mathbb{ICE}$ algorithms have shown superior performance on our benchmark problems, this does not guarantee that they will perform similarly superior on any real–world practical problem. Certainly, if the problem has an (near) additively decomposable structure, $\mathbb{ICE}$ algorithms are efficient problem solvers. However, because these algorithms use multivariate factorizations, they have quite a strong bias for such problems and may perform less good in the absence of such structural features. Therefore, we propose to set p_r to a lower value than 1.0 in general ($p_r = 0.5$ for instance). Furthermore, since OmeGA is also an EA that is capable of exploiting structural features but it performs inferior on our test problems, it is is probably less biased towards the structure in our test problems. Therefore, the OmeGA is a good third general alternative.

The most important remark is that for any practical optimization problem the best that one can do is to design a problem–specific recombination operator that is capable of efficiently exploiting the structure of the optimization problem. If such information is available and can be exploited, designing such a recombination operator should always be preferred. If it is not immediately clear how this can be done, the application of inductive optimization algorithms such as $\mathbb{ICE}$ can be used. On the one hand they can be used as an optimization tool. But for a specific practical optimization problem, it may on the other hand be even more interesting to closely observe the models that $\mathbb{ICE}$ constructs during optimization to thereby obtain additional insights into the dependencies in the problem, which may then help in designing a much faster, problem–specific recombination operator for the problem at hand.

6.6 Discussion and future research

We have shown in this chapter how multivariate factorizations for permutations can be estimated in a greedy fashion from data using the normal gpdf and the permutation gpdf. In the case of the permutation gpdf, we have also shown how this technique can be used in IDEAs to obtain polynomially scalable optimization of additively decomposable relative–ordering deceptive permutation optimization problems. Simple permutation EAs scale up exponentially on these problems. Furthermore, by replacing the probabilistic sampling method with a block–crossover operator, which yields the ICE algorithm, we have shown that the scale–up behavior can be improved further. In Chapter 4 we motivated the IDEA approach by showing that the difference between using a 2–parent crossover operator and a multivariate factorization is in the worst case (slightly) worse for the multivariate factorization but in the best case (slightly) better for the multivariate factorization. Hence we argued that one might just as well investigate means of inducing a good factorization instead of means of inducing a good crossover operator. In this Chapter we have used both crossover and multivariate factorizations adaptively. From our experimental results it has become clear that the balance between the performance of an adaptive crossover operator versus an adaptive multivariate factorization is tipped towards the crossover side in the case of permutation optimization.

Although good results have been obtained for additively decomposable problems, multivariate factorizations are not well–suited for problems with overlapping subfunctions. To this end, Bayesian factorizations are likely to be more appropriate. Although the ICE algorithm can no longer be used properly in combination with Bayesian factorizations, better overall results are expected to be obtained using Bayesian factorizations since they are capable of modelling more involved interactions between problem variables. Using the tools provided in this thesis, Bayesian factorizations can be estimated from data. However, it is to be expected that the use of a greedy learning algorithm that introduces one arc at a time will face lower order decision error problems in a similar fashion as for multivariate factorizations. To remedy this problem, new operators will be required in the greedy learning algorithm.

We have only used a limited number of test problems. Furthermore, in all our test problems, only the relative ordering of genes is important. It would also be interesting to investigate the applicability of our approaches to real–life optimization problems that also appear frequently in combinatorial optimization literature. Examples are the traveling salesman problem, vehicle routing and scheduling. Since the modelling capabilities of the algorithms in this thesis are limited to multivariate factorizations, it is quite likely that within only a few evaluations, our algorithms do not outperform simple permutation GA approaches. The dependencies between problem variables in a scheduling problem may be of a very high order. Furthermore, they may be overlapping or even structured in a more complex manner. It is to be expected in such a case that our newly proposed algorithms will only outperform simple EAs on average if

a large number of evaluations is allowed. The use of Bayesian factorizations, possibly with local structures, may therefore be more interesting to investigate on such real–life problems. An investigation of the performance of the proposed algorithms on a more involved test suite is interesting and of importance.

Exploiting regularities on the level of relative orderings may not be the most effective approach for different optimization problems. Although by copying a set of random keys we are sure that the relative ordering of corresponding loci remains intact, depending on the values for the other loci outside the group of copied random keys, the number of integers in between them in the decoded permutation may still change. We must realize that in order to perform effective recombination for a problem such as the TSP, the *direct* relative ordering in which the numbers in the permutations are placed next to one another is the most important linkage information. In other words, it is not the relative ordering in general that is important, but the *neighbor* information. For this reason, the use of random keys and the exploitation of relative ordering information does not help most efficiently in the optimization of problems such as the TSP. Thus, other probabilistic models for permutations that are capable of exploiting regularities in the space of neighbor information are likely to result in IDEAs that have a much better performance on problems such as the TSP. It would be interesting to develop an EA that is capable of detecting either type of permutation problem structure and subsequently use it in optimization.

Finally, we have argued that the use of a small probability of random rescaling is the most appropriate in general. However, if there are no overlapping subfunctions, the use of random rescaling only makes optimization more difficult. Therefore, it would be interesting to see whether the results can be improved by setting p_ϱ adaptively by for instance looking at the rate of success of applying random rescaling and by changing p_ϱ accordingly. The so–obtained ICE could offer a good basis of comparison for future IDEAs for permutation optimization such as the ones in which Bayesian factorizations are used.

'The best way to have a good idea is to have a lot of ideas.'
Linus Pauling

Multi–objective optimization

Multi–objective optimization problems naturally arise in many real–world situations. Multi–objective optimization differs from single–objective optimization in that a multiple of *objectives* are available that should be optimized simultaneously such that no expression of weights is available that allows us to combine the objectives in a single scalar objective to be minimized. Often, these multiple objectives are conflicting. An example of conflicting objectives that arises in industry, is when we want to maximize the quality of a product while at the same time we want to minimize the production costs of the product. Such conflicting objectives give rise to a key characteristic of multi–objective optimization problems, which is the existence of sets of solutions that cannot be ordered in terms of preference when only considering their objective function values. The goal is now to find a diverse and representative subset of optimal solutions instead of only a single one. In this chapter, we investigate how efficient multi–objective IDEAs can be constructed that are capable of finding a good diverse representation of all multi–objective optimal solutions.

The chapter is organized as follows. In Section 7.1, we present a few important multi–objective definitions. In Section 7.2 we discuss the goal in multi–objective optimization and discuss ways to measure multi–objective optimization performance. In Section 7.3 we discuss a few approaches to multi–objective optimization and point out some of the current state–of–the–art multi–objective EAs (MOEAs). In Section 7.4, we focus on the construction of competent IDEAs for multi–objective optimization. In Section 7.5 we verify the effectiveness of our proposed approaches in two different problem domains and compare them to two other well–known efficient multi–objective EAs. We end this chapter with a discussion of our results and some indications for future work in Section 7.6.

7.1 Multi–objective optimization definitions

To formalize the definition of multi–objective optimization, four relevant concepts exist. Assuming that the optimization problem consists of m objectives, we write $\mathfrak{G}(\mathbf{Z}) = (\mathfrak{G}_0(\mathbf{Z}), \mathfrak{G}_1(\mathbf{Z}), \ldots, \mathfrak{G}_{m-1}(\mathbf{Z}))$ and let $\boldsymbol{\mathcal{M}} = \{0, 1, \ldots, m-1\}$. Without loss of generality, we assume that we seek to minimize all objectives simultaneously. In this context, the four relevant concepts in multi–objective optimization are defined as follows:

1. **Pareto dominance**
   A solution $\mathbf{Z}^0$ is said to (Pareto) *dominate* a solution $\mathbf{Z}^1$ ($\mathbf{Z}^0 \succ \mathbf{Z}^1$) if and only if $\left(\forall i \in \boldsymbol{\mathcal{M}} : \mathfrak{G}_i(\mathbf{Z}^0) \leq \mathfrak{G}_i(\mathbf{Z}^1)\right) \wedge \left(\exists i \in \boldsymbol{\mathcal{M}} : \mathfrak{G}_i(\mathbf{Z}^0) < \mathfrak{G}_i(\mathbf{Z}^1)\right)$

2. **Pareto optimal**
   A solution $\mathbf{Z}^0$ is said to be *Pareto optimal* if and only if $\neg\exists \mathbf{Z}^1 : \mathbf{Z}^1 \succ \mathbf{Z}^0$

3. **Pareto optimal set**
   The set $\boldsymbol{\mathcal{P}_S}$ of all Pareto optimal solutions: $\boldsymbol{\mathcal{P}_S} = \{\mathbf{Z}^0 | \neg\exists \mathbf{Z}^1 : \mathbf{Z}^1 \succ \mathbf{Z}^0\}$

4. **Pareto optimal front**
   The set $\boldsymbol{\mathcal{P}_F}$ of all objective function values corresponding to the solutions in $\boldsymbol{\mathcal{P}_S}$: $\boldsymbol{\mathcal{P}_F} = \{(\mathfrak{G}_0(\mathbf{Z}^0), \mathfrak{G}_1(\mathbf{Z}^0), \ldots, \mathfrak{G}_{m-1}(\mathbf{Z}^0)) | \mathbf{Z}^0 \in \boldsymbol{\mathcal{P}_S}\}$

The Pareto optimal set $\boldsymbol{\mathcal{P}_S}$ is a definition of all trade–off optimal solutions in the parameter space. The Pareto optimal front $\boldsymbol{\mathcal{P}_F}$ is the same set of solutions, only regarded in the objective space. The size of either set can be infinite, in which case it is impossible to find the optimal set or front with a finite number of solutions.

7.2 Multi–objective optimization goals

In this section we discuss the goal in solving multi–objective optimization problems. In Section 7.2.1 we indicate that although the optimum of a multi–objective optimization problem is well defined, there is more than one goal to take into account when evaluating approximations to the optimum. In Section 7.2.2 we discuss a few important performance indicators and we discuss the subtlety in defining the goodness of an approximation.

7.2.1 Approximation sets, optimality and benchmarking

We only consider the subset of all non–dominated solutions that is contained in the final population that results from running a MOEA. We call such a subset an *approximation set* and denote it by $\mathcal{S}$. The size of the approximation set depends on the settings used to run the MOEA with.

Regardless of the size of $\boldsymbol{\mathcal{P}_S}$, we are interested in finding an approximation set of finite size that is a good approximate representation of $\boldsymbol{\mathcal{P}_S}$. Ideally, we

would like to obtain an approximation set that contains a selection of solutions from $\boldsymbol{\mathcal{P}_S}$ such that the solutions in the approximation set are as diverse as possible. However, we do not have access to $\boldsymbol{\mathcal{P}_S}$ beforehand. Therefore, we want to get close to $\boldsymbol{\mathcal{P}_S}$ but in such a way that the approximation set $\mathcal{S}$ that we find, is as diverse as possible, without compromising as much as possible the proximity of $\mathcal{S}$ with respect to $\boldsymbol{\mathcal{P}_S}$. Regarding this diversity, it is important to note that it depends on the mapping between the parameter space and the objective space whether a good spread of the solutions in the parameter space is also a good spread of the solutions in the objective space. However, it is common practice to search for a good diversity of the solutions in the objective space along the approximation set (Coello Coello, 1999) because a decision–maker will ultimately have to pick a single solution. Therefore, it is often best to present a wide variety of trade–off solutions for the specified objectives.

There is an inherent trade–off in the intuitive two–sided goal. However, this trade–off only exists if we assume that we are not able to find the optimal approximation set. The optimal approximation set is well defined if we assume a fixed size of the approximation set. The optimal approximation set is a selection of solutions from $\boldsymbol{\mathcal{P}_S}$ such that the solutions in the approximation set are as diverse as possible. Since the distance to the Pareto optimal front for any solution in the optimal approximation set is 0 and we assume a fixed size of the approximation set, optimality can now be obtained by optimizing only a single objective, which is diversity. In general, there are two ways to benchmark EAs. Either we know the optimum and determine the resources such as population size and number of evaluations that are required on average to obtain the optimum in a predefined percentage of all runs, or we fix the number of evaluations beforehand and determine the maximum score that the EAs obtain on average over all runs. The first way of benchmarking results in values for different EAs that can directly be compared to each other and determine whether one EA is a more competent optimizer than is another. This is also the case for multi–objective optimization since the optimal approximation set is well defined. The second way of benchmarking represents a more practical situation, since we usually do not assume that an unlimited number of function evaluations is available. For single–objective optimization, the objective value can directly be used as the score in this type of benchmark. In multi–objective optimization this is not the case due to the trade–off in the two–sided goal in multi–objective optimization that we have pointed out. This trade–off will be reflected in the score that we use to compare the results of the EAs in the benchmark. It is this type of benchmarking that we will investigate.

7.2.2 Performance indicators

In this section, we discuss performance indicators. A performance indicator is a function that, given an approximation set $\mathcal{S}$, returns a real value that indicates how good $\mathcal{S}$ is with respect to a certain feature that is measured by the performance indicator. Performance indicators are commonly used to determine the performance of a MOEA and to compare this performance with

other MOEAs if the number of evaluations is fixed beforehand. However, there are some limitations to the use of performance indicators. We first describe a few important performance indicators. Subsequently, we will discuss the limitations of these performance indicators.

Selected performance indicators

Since we are interested in performance as measured in the objective space, we define the distance between two multi–objective solutions $\mathbf{Z}^0$ and $\mathbf{Z}^1$ to be the Euclidean distance between their objective values $\mathfrak{G}(\mathbf{Z}^0)$ and $\mathfrak{G}(\mathbf{Z}^1)$:

$$d(\mathbf{Z}^0, \mathbf{Z}^1) = \sqrt{\sum_{i=0}^{m-1} (\mathfrak{G}_i(\mathbf{Z}^1) - \mathfrak{G}_i(\mathbf{Z}^0))^2} \tag{7.1}$$

If we only want to measure diversity, we can use the **FS** (*Front Spread*) indicator. This performance indicator was first used by Zitzler (1999). The **FS** indicator indicates the size of the objective space covered by an approximation set. A larger **FS** indicator value is preferable. The **FS** indicator for an approximation set $\mathcal{S}$ is defined to be the maximum Euclidean distance inside the smallest m–dimensional bounding–box that contains $\mathcal{S}$. This distance can be computed using the maximum distance among the solutions in $\mathcal{S}$ in each dimension separately:

$$\mathbf{FS}(\mathcal{S}) = \sqrt{\sum_{i=0}^{m-1} \max_{(\mathbf{Z}^0, \mathbf{Z}^1) \in \mathcal{S} \times \mathcal{S}} \{(\mathfrak{G}_i(\mathbf{Z}^0) - \mathfrak{G}_i(\mathbf{Z}^1))^2\}} \tag{7.2}$$

In combination with the **FS** indicator, it is also important to know how many points are available in the set of non–dominated solutions, because a larger set of trade–off points is more desirable. This quantity is called the **FO** (*Front Occupation*) indicator and was first used by Van Veldhuizen (Van Veldhuizen, 1999). A larger **FO** indicator value is preferable.

$$\mathbf{FO}(\mathcal{S}) = |\mathcal{S}| \tag{7.3}$$

The ultimate goal is to cover the Pareto optimal front. An intuitive way to define the distance between an approximation set $\mathcal{S}$ and the Pareto optimal front is to average the minimum distance between a solution and the Pareto optimal front over each solution in $\mathcal{S}$. We refer to this distance as the distance from a set of non–dominated solutions to the Pareto optimal front and it serves as proximity indicator, which we denote by $\boldsymbol{D}_{\mathcal{S} \rightarrow \boldsymbol{\mathcal{P}}_{\boldsymbol{F}}}$. This performance indicator was first used by Van Veldhuizen (Van Veldhuizen, 1999). A smaller value for this performance indicator is preferable.

$$\boldsymbol{D}_{\mathcal{S} \rightarrow \boldsymbol{\mathcal{P}}_{\boldsymbol{F}}}(\mathcal{S}) = \frac{1}{|\mathcal{S}|} \sum_{\mathbf{Z}^0 \in \mathcal{S}} \min_{\mathbf{Z}^1 \in \boldsymbol{\mathcal{P}}_{\boldsymbol{S}}} \{d(\mathbf{Z}^0, \mathbf{Z}^1)\} \tag{7.4}$$

An approximation set with a good $\boldsymbol{D}_{\mathcal{S}\rightarrow\boldsymbol{\mathcal{P}_F}}$ indicator value does not imply that a good diverse representation of the Pareto optimal set has been obtained, since the indicator only reflects how far away the obtained points are from the Pareto optimal front on average. An approximation set consisting of only a single solution can already have a low value for this indicator. To include the goal of diversity, the reverse of the $\boldsymbol{D}_{\mathcal{S}\rightarrow\boldsymbol{\mathcal{P}_F}}$ indicator is a better guideline for evaluating MOEAs. In the reverse distance indicator, we compute for each solution in the Pareto optimal set the distance to the closest solution in an approximation set $\mathcal{S}$ and take the average as the indicator value. We denote this indicator by $\boldsymbol{D}_{\boldsymbol{\mathcal{P}_F}\rightarrow\mathcal{S}}$ and refer to it as the distance from the Pareto optimal front to an approximation set. A smaller value for this performance indicator is preferable. In the definition of this indicator, we must realize that the Pareto optimal front may be continuous. For an exact definition, we therefore have to use a line integration over the entire Pareto front. For a 2–dimensional multi–objective problem we obtain the following expression:

$$\boldsymbol{D}_{\boldsymbol{\mathcal{P}_F}\rightarrow\mathcal{S}}(\mathcal{S}) = \int_{\boldsymbol{\mathcal{P}_F}} \min_{\mathbf{Z}^0\in\mathcal{S}}\{d(\mathbf{Z}^0,\mathbf{Z}^1)\}\mathrm{d}\mathfrak{G}(\mathbf{Z}^1) \tag{7.5}$$

In most practical test applications, it is easier to compute a uniformly sampled set of many solutions along the Pareto optimal front and to use this discretized representation of $\boldsymbol{\mathcal{P}_F}$ instead. A discretized version of the Pareto optimal front is also available if a discrete multi–objective optimization problem is being solved. In the discrete case, the $\boldsymbol{D}_{\mathcal{S}\rightarrow\boldsymbol{\mathcal{P}_F}}$ indicator is defined by:

$$\boldsymbol{D}_{\boldsymbol{\mathcal{P}_F}\rightarrow\mathcal{S}}(\mathcal{S}) = \frac{1}{|\boldsymbol{\mathcal{P}_S}|}\sum_{\mathbf{Z}^1\in\boldsymbol{\mathcal{P}_S}} \min_{\mathbf{Z}^0\in\mathcal{S}}\{d(\mathbf{Z}^0,\mathbf{Z}^1)\} \tag{7.6}$$

An illustration of the $\boldsymbol{D}_{\boldsymbol{\mathcal{P}_F}\rightarrow\mathcal{S}}$ indicator is presented in Figure 7.1. The $\boldsymbol{D}_{\boldsymbol{\mathcal{P}_F}\rightarrow\mathcal{S}}$ indicator represents both the proximity and the diversity goal in multi–objective optimization. The $\boldsymbol{D}_{\boldsymbol{\mathcal{P}_F}\rightarrow\mathcal{S}}$ indicator for an approximation set $\mathcal{S}$ is zero if and only if all points in $\boldsymbol{\mathcal{P}_F}$ are contained in $\mathcal{S}$ as well. Furthermore, a single solution from the Pareto optimal set will lead to the same $\boldsymbol{D}_{\boldsymbol{\mathcal{P}_F}\rightarrow\mathcal{S}}$ indicator as a more diverse set of solutions that has objective values that are slightly further away from the Pareto optimal front. Moreover, a similarly diverse approximation set of solutions that is closer to the Pareto optimal front, will have a lower $\boldsymbol{D}_{\boldsymbol{\mathcal{P}_F}\rightarrow\mathcal{S}}$ indicator value. However, an approximation set of solutions that is extremely diverse but far away from the Pareto optimal front, such as the non–dominated solutions of a randomly generated set of solutions, has a bad $\boldsymbol{D}_{\boldsymbol{\mathcal{P}_F}\rightarrow\mathcal{S}}$ indicator value. This underlines the important point that diversity is *not* equally important as is proximity because a larger diversity is often not hard to come by. What is important is the diversity *along* the objectives of a set of non–dominated solutions that is as close as possible to the Pareto optimal front.

A performance indicator that is closely related to the $\boldsymbol{D}_{\boldsymbol{\mathcal{P}_F}\rightarrow\mathcal{S}}$ indicator, is the hypervolume indicator by Knowles and Corne (Knowles and Corne, 2002). In the hypervolume indicator, a point in the objective space is picked such that

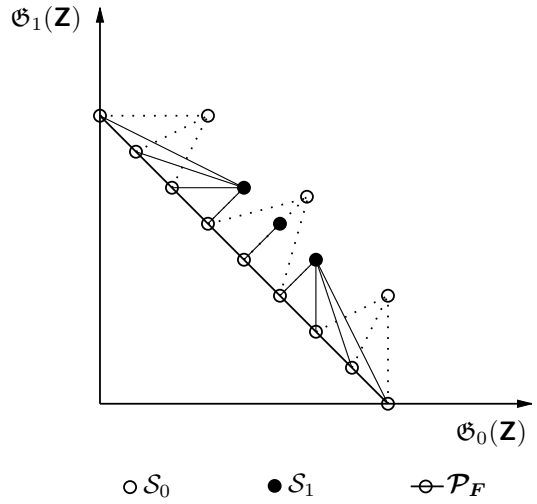


Figure 7.1: The approximation set $\mathcal{S}_1$ is closer to the (discretized) Pareto optimal front but has less diversity, while approximation set $\mathcal{S}_0$ is further away from the front but has greater diversity: both sets have approximately the same $\boldsymbol{D}_{\mathcal{P}_F \to \mathcal{S}}$ indicator value though.

it is dominated by all points in the approximation sets that need to be evaluated. The indicator value is then equal to the hypervolume of the multi–dimensional region enclosed by the approximation set and the picked reference point. This value is an indicator of the region in the objective space that is dominated by the approximation set. The main difference between the hypervolume indicator and the $\boldsymbol{D}_{\mathcal{P}_F \to \mathcal{S}}$ indicator is that for the hypervolume indicator a reference point has to be chosen. Different reference points lead to different indicator values. Moreover, different reference points can lead to indicator values that indicate a preference for different approximation sets. Since in the $\boldsymbol{D}_{\mathcal{P}_F \to \mathcal{S}}$ indicator the true Pareto optimal front is used, the $\boldsymbol{D}_{\mathcal{P}_F \to \mathcal{S}}$ indicator does not suffer from this drawback. Of course, a major drawback of the $\boldsymbol{D}_{\mathcal{P}_F \to \mathcal{S}}$ indicator is that in a real application the true Pareto optimal front is not known beforehand. In that case, the Pareto front of all approximation sets could be used as a substitute for the actual Pareto optimal front.

The relation between performance indicators and the comparison of MOEAs

If we want to use performance indicators to investigate the performance of a MOEA and compare it with other MOEAs, there are some important limitations to consider that have been proven by Zitzler, Laumanns, Thiele, Fonseca and da Fonseca (2002). These limitations are related to the extent to which performance indicators are capable of truly indicating whether one approximation set $\mathcal{S}^0$ is better than $\mathcal{S}^1$ in a certain sense. To this end, the concept of domi-

nation has to be extended to approximation sets. Zitzler et al. (2002) consider the following dominance relations for approximation sets:

1. **Strict Pareto dominance**
   An approximation set $\mathcal{S}^0$ is said to *strictly (Pareto) dominate* an approximation set $\mathcal{S}^1$ (denoted $\mathcal{S}^0 \succ\!\!\succ \mathcal{S}^1$) if and only if $\forall \mathbf{Z}^1 \in \mathcal{S}^1 : \left(\exists \mathbf{Z}^0 \in \mathcal{S}^0 : \left(\forall i \in \boldsymbol{\mathcal{M}} : \mathfrak{G}_i(\mathbf{Z}^0) < \mathfrak{G}_i(\mathbf{Z}^1)\right)\right)$
2. **Pareto dominance**
   An approximation set $\mathcal{S}^0$ is said to *(Pareto) dominate* an approximation set $\mathcal{S}^1$ (denoted $\mathcal{S}^0 \succ \mathcal{S}^1$) if and only if $\forall \mathbf{Z}^1 \in \mathcal{S}^1 : \left(\exists \mathbf{Z}^0 \in \mathcal{S}^0 : \mathbf{Z}^0 \succ \mathbf{Z}^1\right)$
3. **Better**
   An approximation set $\mathcal{S}^0$ is said to be *better* than an approximation set $\mathcal{S}^1$ (denoted $\mathcal{S}^0 \rhd \mathcal{S}^1$) if and only if $\mathcal{S}^0 \neq \mathcal{S}^1$ and $\left(\forall \mathbf{Z}^1 \in \mathcal{S}^1 : \left(\exists \mathbf{Z}^0 \in \mathcal{S}^0 : \left(\forall i \in \boldsymbol{\mathcal{M}} : \mathfrak{G}_i(\mathbf{Z}^0) \ \leq \mathfrak{G}_i(\mathbf{Z}^1)\right)\right)\right)$
4. **Weak Pareto dominance**
   An approximation set $\mathcal{S}^0$ is said to *weakly (Pareto) dominate* an approximation set $\mathcal{S}^1$ (denoted $\mathcal{S}^0 \succeq \mathcal{S}^1$) if and only if $\forall \mathbf{Z}^1 \in \mathcal{S}^1 : \left(\exists \mathbf{Z}^0 \in \mathcal{S}^0 : \left(\forall i \in \boldsymbol{\mathcal{M}} : \mathfrak{G}_i(\mathbf{Z}^0) \leq \mathfrak{G}_i(\mathbf{Z}^1)\right)\right)$
5. **Incomparable**
   An approximation set $\mathcal{S}^0$ is said to be *incomparable* to an approximation set $\mathcal{S}^1$ (denoted $\mathcal{S}^0 \parallel \mathcal{S}^1$) if and only if $\neg\left(\mathcal{S}^0 \succeq \mathcal{S}^1\right) \wedge \neg\left(\mathcal{S}^1 \succeq \mathcal{S}^0\right)$

It was shown by Zitzler et al. (2002) that for any finite combination of performance indicators such as the ones presented in the previous section, there is no function of these performance indicators that specifies for any two approximation sets $\mathcal{S}^0$ and $\mathcal{S}^1$ whether $\mathcal{S}^0 \rhd \mathcal{S}^1$ holds. Thus, using the terminology and definitions by Zitzler et al. (2002), we may not draw any conclusions regarding whether one approximation set is better than another approximation set on the basis of performance indicators such as the ones we have described so far.

Although the result by Zitzler et al. (2002) is very important, its implications only apply to cases in which it is clear from a domination point of view that one approximation set is better than another approximation set. For instance, if $\mathcal{S}^0 \succ\!\!\succ \mathcal{S}^1$ holds, then $\mathcal{S}^0$ is truly preferable over $\mathcal{S}^1$. However, there are some important further aspects to consider that relate to the comparison of competent MOEAs. Even if $\mathcal{S}^0 \parallel \mathcal{S}^1$ holds, we could still prefer $\mathcal{S}^0$ over $\mathcal{S}^1$. Consider for instance the example in Figure 7.2. Following the definitions for comparing approximation sets by Zitzler et al. (2002), $\mathcal{S}^0 \parallel \mathcal{S}^1$ holds. However, $\mathcal{S}^0$ has many more non–dominated solutions and a much larger diversity than does $\mathcal{S}^1$. Even if $\mathcal{S}^1$ had only a single solution placed somewhere on the line between the current two solutions in $\mathcal{S}^1$, the two approximation sets would still be incomparable. Still, it is fair to say here that approximation set $\mathcal{S}^0$ is preferable.

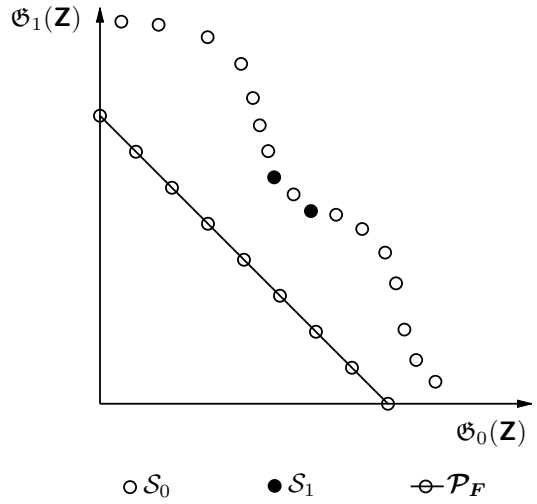


Figure 7.2: Although approximation sets $\mathcal{S}_0$ and $\mathcal{S}_1$ are very different and approximation set $\mathcal{S}_0$ has many more non–dominated solutions and a better diversity than $\mathcal{S}_1$, using the dominance criteria by Zitzler et al. (2002) $\mathcal{S}_0$ and $\mathcal{S}_1$ can only be classified as incomparable.

The class of incomparable approximation sets is very large and it can be argued that this class includes sets that may clearly be called preferable over other sets in the same class. It can furthermore be argued that this class is filled with pairs of approximation sets such that one approximation set of this pair is not clearly preferable over the other approximation set. This is for instance often the case if the two approximation sets intersect in the objective space and have a comparable diversity and size. Another example of pairs of approximation sets that are not easily said to be preferable over each other is given in Figure 7.3. This example represents the arguable statement that the class of incomparable approximation sets contains a large number of approximation sets that represent a true trade–off between the goals of proximity and diversity.

The existence of trade–off approximation sets in the class of incomparable approximation sets is very important when comparing MOEAs. As the efficiency of newly designed MOEAs increases, results such as the one in Figure 7.3 will become ever more likely to occur. Clearly, if two algorithms have the same emphasis on diversity preservation as they have on getting as close as possible to the Pareto optimal front, these algorithms will end up with approximation sets that are incomparable, unless one algorithm is truly less competent than the other. In this case testing the results of the algorithms using the categorization by Zitzler et al. (2002) will point out which algorithm is superior. However, if one algorithm places more emphasis on diversity preservation and the other algorithm places more emphasis on getting get as close as possible to the Pareto optimal front, results such as the ones in Figures 7.2 and 7.3 are likely to occur. The categorization by Zitzler et al. (2002) will in both cases point out that

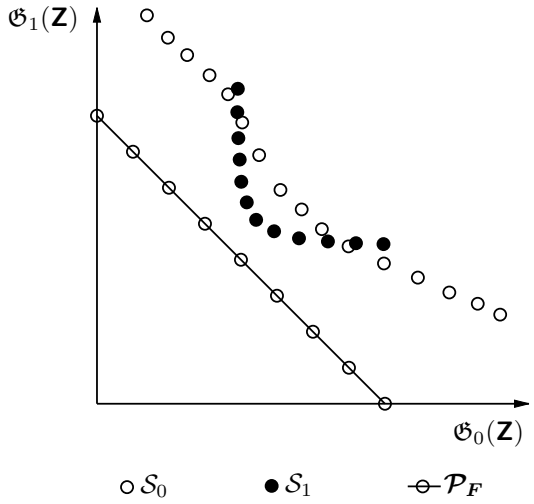


Figure 7.3: Whereas approximation set $\mathcal{S}_0$ is very diverse and has an overall good approximation of the Pareto optimal front, approximation set $\mathcal{S}_1$ is less diverse but has a better proximity with respect to the Pareto optimal front. These two approximation sets represent a trade–off in that, without a preference for diversity or proximity with respect to the Pareto optimal front, neither approximation set can be called preferable. Using the dominance criteria by Zitzler et al. (2002) $\mathcal{S}_0$ and $\mathcal{S}_1$ are classified as incomparable.

the algorithms are incomparable although it can be argued very plausibly in the case of Figure 7.2 that one result is less preferable than the other. In this case, performance indicators such as the ones that we have described can offer additional information. In the case of Figure 7.2 we will find that although the $\boldsymbol{D}_{\mathcal{S}\rightarrow\mathcal{P}_F}$ and $\boldsymbol{D}_{\mathcal{P}_F\rightarrow\mathcal{S}}$ indicators are relatively similar, the **FS** and **FO** indicators will be significantly better for $\mathcal{S}_0$ than for $\mathcal{S}_1$. This will lead us to conclude that $\mathcal{S}_0$ is indeed preferable. In the case of Figure 7.3 we will find that the $\boldsymbol{D}_{\mathcal{P}_F\rightarrow\mathcal{S}}$ and **FO** indicators are relatively similar, but the $\boldsymbol{D}_{\mathcal{S}\rightarrow\mathcal{P}_F}$ indicator is better for $\mathcal{S}_1$ whereas the **FS** indicator is better for $\mathcal{S}_0$. This will lead us to decide that neither approximation set is preferable.

Concluding, there is a good chance that two MOEAs are classified as being incomparable with respect to the definitions of Zitzler et al. (2002) unless a truly significant competence difference between the MOEAs exists. This limits the practical use of the dominance relations for approximation sets by Zitzler et al. (2002). Moreover, if the results of two MOEAs are classified as being incomparable, one MOEA may still be called more preferable than the other depending on the balance between proximity and diversity. On the other hand, if the two MOEAs are both competent, the trade–off that lies in the balance between proximity and diversity can cause the results of the two algorithms to be quite different or to be quite similar and yet we cannot clearly say that one MOEA is more preferable than the other. Furthermore, the combination of our

three selected performance indicators are well motivated and are a good choice to get a global indication of the performance of MOEAs.

7.3 Multi–objective optimization algorithms

There are generally speaking two ways in which multi–objective optimization problems can be tackled. One is by repeated optimization using a single solution. Another is by simultaneous optimization using a set of solutions. We first describe repeated optimization techniques and briefly point out the problems with this approach. We then describe simultaneous optimization and point out why EAs are good candidates for effective multi–objective optimization. Moreover, we describe two state–of–the–art MOEAs in more detail because they will play an important part in the experimental section of this chapter.

7.3.1 Repeated optimization

One approach is to repeatedly find a single new solution and construct the final result by making a selection from the obtained solutions. To find a single solution, a weighted aggregation approach is usually taken in which the different objectives are summed up to one single scalar objective, $\mathfrak{G}'(\mathbf{Z}) = \sum_{i=0}^{m-1} \lambda_i \mathfrak{G}_i(\mathbf{Z})$. The advantage of this approach is that available single–objective optimization approaches can directly be used to solve the single–objective problem repeatedly for different combination of weights λ_i. However, such repeated optimization may be inefficient because the trajectories followed for certain individual optimization trials may not differ all too much and the information used during optimization could have perhaps have been exploited more efficiently when processing all trajectories simultaneously. As a result, obtaining only a single solution every time may be inefficient. Furthermore, setting the weights is very difficult. Equidistant sets of combinations of weights do not need to lead to a neat diverse set of solutions along the Pareto front at all. A further important weakness of this approach is that not all parts of the Pareto optimal front can be covered using weighted aggregation (Das and Dennis, 1997; Jin, Olhofer and Sendhoff, 2001).

7.3.2 Simultaneous optimization

Another approach is to use a set of solutions that is iteratively altered. Since the optimum of a multi–objective optimization problem is a set that consists by definition of at least a single solution, approaches based on sets of solutions seem very plausible. In such approaches the important goal of the preservation of diversity along the front can efficiently be attended to by comparing the available solutions to each other during optimization. The notion of searching a space by maintaining a population of solutions is characteristic of EAs, which makes them natural candidates. A strongly increasing amount of research has

indeed been done in the field of evolutionary multi–objective optimization in recent years (Coello Coello, 1999; Zitzler, Deb, Thiele, Coello Coello and Corne, 2001) with very promising results, making evolutionary–based approaches currently one of the most generally applicable and successful approaches to multi–objective optimization. Two MOEAs are of specific interest, since they have been shown to be among the currently best performing MOEAs (Deb, Agrawal, Pratab and Meyarivan, 2000; Zitzler, Deb and Thiele, 2000). These MOEAs are the *Strength Pareto Evolutionary Algorithm* (SPEA) by Zitzler and Thiele (1999) and the second version of the *Non–Dominated Sorting Genetic Algorithm* (NSGA–II) by Deb et al. (2000). We will give a brief description of both algorithms.

SPEA

In SPEA, the non–dominated solutions found earlier during optimization are stored externally from the population. If the size of this external storage becomes too large, clustering is performed on the external storage in the objective space and some solutions are discarded from each cluster. Clustering is performed using the joining algorithm, which was earlier called the *average linkage method* in the context of multi–objective optimization (Morse, 1980). A distance threshold is specified beforehand and each solution in the external storage is initialized as a cluster. The algorithm then iteratively joins clusters by computing the distances between the clusters. A pair of clusters with a distance smaller than the threshold is then selected and joined. The distance between two clusters is computed as the average distance between pairs of solutions across the two clusters. Crossover and mutation are applied to solutions that are selected from both the population as well as the external storage. Selection is performed using tournament selection with a tournament size of 2. The most characteristic and profound item in SPEA is the way fitness is assigned before selection. Each solution in the external storage is assigned a strength proportional to the number of solutions it dominates in the population. Each solution in the population is assigned one plus the sum of the strengths of the solutions in the external storage that dominate it. The additional value of one is required to ensure that the externally stored solutions are always better (a lower strength is preferable). This mechanism prefers individuals near the Pareto front and distributes them at the same time. An example in two dimensions is given in Figure 7.4.

NSGA–II

In NSGA–II, no external storage is used. The selection phase consists of two parts. First, a preselection is made based on the *domination rank*. Rank 0 consists of all non–dominated solutions. Rank $i \geq 1$ consists of all non–dominated solutions when all ranks $0 \leq j < i$ are disregarded. These ranks are the basis of selection in NSGA–II. A solution with a lower rank is always preferred. An example of domination ranks is given in Figure 7.4. To promote diversity, a distance is associated with each solution. This distance is based on neighboring

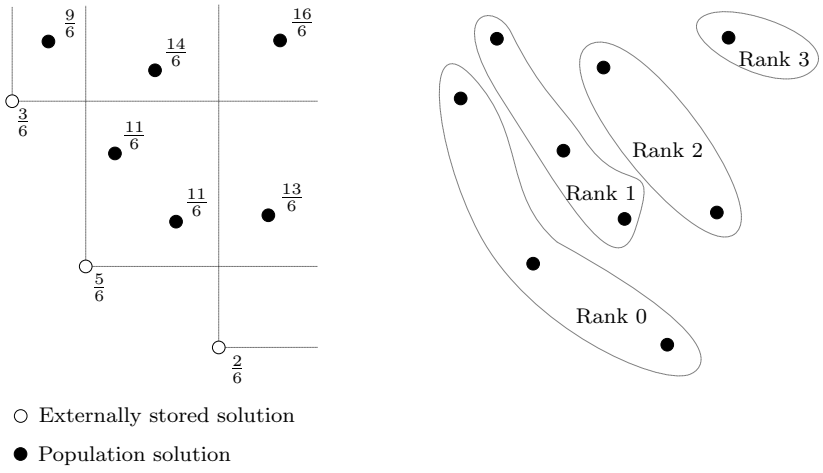


Figure 7.4: The fitness assignment in SPEA for selection (left) in two dimensions. The goal is to minimize both objectives. On the right, the division into layers of different domination ranks for the same set of solutions is shown. This rank–based division is the basis of selection in NSGA–II.

solutions that have the same domination rank. To compute the distance for a set of solutions with the same domination rank, the solutions are regarded in each objective separately. First, the distance of each solution is initialized to 0. Then, the solutions are sorted for each objective $0 \leq i < m$ anew. For each objective i, the solutions that appear first and last in the i–th sorting are given a distance of ∞ to give a preference to boundary solutions. The other solutions in the domination rank are assigned their currently computed distance plus the distance in the i–th objective between their two neighbors according to the i–th sorting. The preselection has size $\frac{1}{2}n$ and is obtained using truncation selection. This means that all solutions in consecutive ranks are selected until the addition of a complete rank would result in a preselection size larger than $\frac{1}{2}n$. Solutions from this last rank are selected using the diversity measure until exactly $\frac{1}{2}n$ solutions are in the preselection. Subsequently, tournament selection is applied to this preselection using the same comparison metric in which domination rank is always more important, to select pairs of parents that will undergo recombination. This procedure is repeated until exactly $\frac{1}{2}n$ new solutions have been generated. These offspring solutions replace the solutions in the population that were not selected to be in the preselection.

7.4 Multi–objective mixture–based IDEAs

The IDEA instances proposed so far in this thesis are only directly suited for single–objective optimization problems. To obtain instances that are suited for multi–objective optimization, we propose to instantiate two steps in the IDEA

framework differently from the single–objective instances. These two steps are the selection of genotypes and the building of a probabilistic model. The latter step will be restricted to models for mixture probability distributions. We shall first describe these two different instantiations in more detail. We then introduce a specific IDEA for multi–objective optimization using these new instantiations.

7.4.1 Balanced multi–objective truncation selection

To search for the Pareto–front, we must ensure pressure towards this goal. In the IDEA framework, a selection of $\lfloor \tau n \rfloor$ individuals is made, where τ is the truncation percentile and n is the population size. Making this selection on the basis of Pareto–dominance ensures pressure towards non–dominated solutions. There are different ways in which solutions can be selected. Two examples are given by the way in which selection is done by SPEA and NSGA–II as discussed in Section 7.3. In both these approaches, diversity preservation is also enforced, which ensures a good diverse selection along the front, which in turn stimulates the parallel exploration along the front. This has proven to be beneficial in multi–objective optimization.

We propose a simple but effective approach to selecting solutions that are both close to the front as well as a diverse representation of the available solutions. An important novelty of our approach is that the selection pressure towards diversity preservation and towards non–domination can be tuned using a single parameter. The selection operator we propose first computes the *domination count* of all individuals. The domination count of a solution equals the number of times the solution is dominated by other solutions in the population. Subsequently, a pre–selection $\boldsymbol{\mathcal{S}}^P$ is made of $\lfloor \delta\tau n \rfloor$ individuals ($\delta \in [1; \frac{1}{\tau}]$) by using truncation selection on the domination count (select the best $\lfloor \delta\tau n \rfloor$ solutions). However, if the solution with the largest domination count to end up in $\boldsymbol{\mathcal{S}}^P$ by truncation selection has a domination count of 0, *all* individuals with a domination count of 0 are selected instead, resulting in $|\boldsymbol{\mathcal{S}}^P| \geq \lfloor \delta\tau n \rfloor$. This ensures that once the search starts to converge onto a certain Pareto front, we enforce diversity over all of the available solutions on the front. The final selection $\boldsymbol{\mathcal{S}}$ is obtained from $\boldsymbol{\mathcal{S}}^P$ by using a nearest neighbor heuristic to promote diversity. First, an individual with a maximum value for an arbitrary objective is deleted from $\boldsymbol{\mathcal{S}}^P$ and added to $\boldsymbol{\mathcal{S}}$. To break ties, the remaining objectives are scanned for maximality in an arbitrary order. For all solutions in $\boldsymbol{\mathcal{S}}^P$, the nearest neighbor distance is computed to the single solution in $\boldsymbol{\mathcal{S}}$. The distance that we use is the Euclidean distance scaled to the sample range in each objective. The solution in $\boldsymbol{\mathcal{S}}^P$ with the *largest* distance is then deleted from $\boldsymbol{\mathcal{S}}^P$ and added to $\boldsymbol{\mathcal{S}}$. The distances in $\boldsymbol{\mathcal{S}}^P$ are updated by investigating whether the distance to the newly added point in $\boldsymbol{\mathcal{S}}$ is smaller than the currently stored distance. These last two steps are repeated until $\lfloor \tau n \rfloor$ solutions are in the final selection. This selection operator has a running time complexity of $\mathcal{O}(n^2)$. This is no worse than the minimum of $\mathcal{O}(n^2)$ for computing the domination counts which is required in all MOEAs. Pseudo–code for this selection operator is given in Figure 7.5.

```
DiversityPreservingSelection( δ, τ )
 1  for i ← 0 to n − 1 do
    1.1  count[P_i] ← 0
 2  for i ← 0 to n − 1 do
    2.1  for j ← 0 to n − 1 do
         2.1.1  if i ≠ j ∧ dominates(P_j, P_i)
                2.1.1.1  count[P_i] ← count[P_i] + 1
 3  s ← Sort(<, count)
 4  S^P ← ()
 5  i ← 0
 6  while i < ⌊δτn⌋ ∨ count[P_{s_i}] = 0 do
    6.1  S^P ← S^P ⊔ (P_{s_i})
    6.2  i ← i + 1
 7  j ← MaximumSolutionInArbitraryObjective(S^P)
 8  S ← (S^P_j)
 9  S^P ← S^P − (S^P_j)
10  for i ← 0 to |S^P| − 1 do
    10.1  distance[S^P_i] ← ScaledEuclideanDistance(S^P_i, S_0)
11  for i ← 1 to ⌊τn⌋ − 1 do
    11.1  j ← argmax_{k∈{0,1,...,|S^P|−1}} { distance[S^P_k] }
    11.2  S ← S ⊔ (S^P_j)
    11.3  S^P ← S^P − (S^P_j)
    11.4  for j ← 0 to |S^P| − 1 do
          11.4.1  d ← ScaledEuclideanDistance(S^P_j, S_i)
          11.4.2  if d < distance[S^P_j] then
                  11.4.2.1  distance[S^P_j] ← d
12  return(S)
```

Figure 7.5: Pseudo–code for the diversity preserving selection operator used in MIDEA instances of the IDEA framework. After computing the domination counts and truncation selection on the domination count to obtain a preselection of solutions, a single solution is picked from the preselection such that it is maximal in an arbitrary objective. This solution is added to the final selection Solutions are then iteratively picked from the preselection by taking the solution that has the largest distance to its nearest neighbor in the final selection. After adding this solution to the final selection, the distances are updated for the remaining solutions in the preselection.

7.4.2 Stimulating diversity with mixture probability distributions

Regarding the construction of new offspring solutions, we can use any of the techniques described earlier in this thesis to construct IDEAs for multi–objective optimization. However, the specific use of mixture probability distributions is expected to provide additional useful resources for the exploration of the multi–objective search space, since they are capable of processing relations between different features of the selected solutions. In a pilot study (Thierens and Bosman, 2001b), the use of mixture probability distributions without diversity preserving selection was already shown to be able to obtain a diverse covering of the front. The probability distributions were obtained by clustering the selected solutions. Although clustering can be done in the parameter space as well as in the objective space, clustering in the objective space is better suited for maintaining a diverse covering of the Pareto front. The use of mixture probability distributions obtained by clustering the objective space allows for a parallel exploration of the Pareto front through the different clusters. The results using only mixture probability distributions did however indicate that the use of additional diversity preservation mechanisms could improve the results because there is no guarantee that solutions on the borders of the Pareto front will be preserved.

7.4.3 A specific multi–objective mixture–based IDEA

Combined with diversity preserving selection, an IDEA based on mixture probability distributions should be an effective tool for multi–objective optimization. The use of mixture probability distributions obtained by means of clustering the objective space stimulates the generation of a diverse set of solutions. In addition, the diversity preserving selection operator combined with the elitism in monotonic IDEAs is capable of ensuring that this diversity will not get lost during optimization.

In the remainder of this thesis we shall refer to the specific IDEA instance that is obtained by using mixture probability distributions obtained by means of clustering the objective space and the diversity preserving selection operator, the *Multi–objective Mixture–based Iterated Density Estimation Evolutionary Algorithm* (MIDEA).

7.5 Experiments

We compare IDEA instances, amongst which MIDEA instances, to SPEA and NSGA–II on a test–suite of eight multi–objective optimization problems. We varied the dimensionality of these problems in order to get a total of sixteen problem instances to test the MOEAs on. The multi–objective optimization problems are described in Section 7.5.1 In Section 7.5.2 we present our experiment setup. In Section 7.5.3 we discuss the obtained results. Finally, in Section 7.5.4 we give a short summary for the EA practitioner.

7.5.1 Multi–objective optimization problems

Our test suite consists of problems with real–valued variables as well as with binary variables. In both cases we have used four different optimization problems and have used two different dimensionalities for these problems to obtain a total test suite size of 16 problems. In the following we give a brief description of the problems in our test suite.

Real–valued multi–objective optimization problems

A variety of test problems for real–valued variables has been proposed that may cause different types of problems for multi–objective optimization algorithms (Deb, 1999; Zitzler et al., 2000; Deb, Pratap and Meyarivan, 2001). From this set of problems, we have selected three problems that are commonly used to benchmark multi–objective optimization algorithms. The fourth real–valued test problem is a new test problem we have designed to test the performance of MOEAs if there are strong interactions between the problem variables. These problems represent a spectrum of multi–objective problem difficulty as they make it difficult for a multi–objective optimization algorithm to progress towards the global optimal front and to maintain a diverse spread of solutions due to properties such as discontinuous fronts and multi–modality. The problems with real–valued variables that we use in our experiments are all defined for two objectives. An overview of our test problems is given in Figure 7.6.

$\underline{ZDT_4}$
Function ZDT_4 was introduced by Zitzler et al. (2000). It is very hard to obtain the optimal front $\mathfrak{G}_1(\mathbf{Y}) = 1 - \sqrt{Y_0}$ in ZDT_4 since there are many local fronts. Moreover, the number of local fronts increases as we get closer to the Pareto optimal front. The main problem that a MOEA should be able to overcome to optimize this problem is thus strong multi–modality.

$\underline{ZDT_6}$
Function ZDT_6 was also introduced by Zitzler et al. (2000). The density of solutions in ZDT_6 increases as we move away from the Pareto optimal front. Furthermore, ZDT_6 has a non–uniform density of solutions *along* the Pareto optimal front as there are more solutions as $\mathfrak{G}_0(\mathbf{Y})$ goes up to 1. Therefore, a good diverse spread of solutions along the Pareto front is hard to obtain. The Pareto front for ZDT_6 is given by $\mathfrak{G}_1(\mathbf{Y}) = 1 - \mathfrak{G}_0(\mathbf{Y})^2$ with $\mathfrak{G}_0(\mathbf{Y}) \in [1 - e^{-1/3}; 1]$.

$\underline{CTP_7}$
Function CTP_7 was introduced by Deb et al. (2001). Its Pareto optimal front differs slightly from that of ZDT_4, but otherwise shares the multi–modal front problem. In addition, this problem has constraints in the objective space, which makes finding a diverse representation of the Pareto front more difficult since the Pareto front is discontinuous and it is hard to obtain an approximation of a front that has a few solutions in each feasible part of that front.

Name	Definition		Range
BT_1	**Minimize**	$(\mathfrak{G}_0(\mathbf{Y}), \mathfrak{G}_1(\mathbf{Y}))$	• $\mathsf{Y}_0 \in [0;1]$ • $\mathsf{Y}_i \in [-3;3]$ $(1 \leq i < l)$
	Where	• $\mathfrak{G}_0(\mathbf{Y}) = \mathsf{Y}_0$	
		• $\mathfrak{G}_1(\mathbf{Y}) = 1 - \mathfrak{G}_0(\mathbf{Y}) + 10^7 - \frac{100}{\left(10^{-5} + \sum_{i=1}^{l-1} \lvert \sum_{j=1}^{i} \mathsf{Y}_i \rvert\right)}$	
ZDT_4	**Minimize**	$(\mathfrak{G}_0(\mathbf{Y}), \mathfrak{G}_1(\mathbf{Y}))$	• $\mathsf{Y}_0 \in [0;1]$ • $\mathsf{Y}_i \in [-5;5]$ $(1 \leq i < l)$
	Where	• $\mathfrak{G}_0(\mathbf{Y}) = \mathsf{Y}_0$	
		• $\mathfrak{G}_1(\mathbf{Y}) = \gamma\left(1 - \sqrt{\frac{\mathfrak{G}_0(\mathbf{Y})}{\gamma}}\right)$	
		• $\gamma = 1 + 10(l\text{–}1) + \sum_{i=1}^{l-1}\left(\mathsf{Y}_i^2 - 10\cos(4\pi\mathsf{Y}_i)\right)$	
ZDT_6	**Minimize**	$(\mathfrak{G}_0(\mathbf{Y}), \mathfrak{G}_1(\mathbf{Y}))$	• $\mathsf{Y}_i \in [0;1]$ $(0 \leq i < l)$
	Where	• $\mathfrak{G}_0(\mathbf{Y}) = 1 - e^{-4\mathsf{Y}_0}\sin^6(6\pi\mathsf{Y}_0)$	
		• $\mathfrak{G}_1(\mathbf{Y}) = \gamma\left(1 - \left(\frac{\mathfrak{G}_0(\mathbf{Y})}{\gamma}\right)^2\right)$	
		• $\gamma = 1 + 9\left(\sum_{i=1}^{l-1}\frac{\mathsf{Y}_i}{9}\right)^{0.25}$	
CTP_7	**Minimize**	$(\mathfrak{G}_0(\mathbf{Y}), \mathfrak{G}_1(\mathbf{Y}))$	• $\mathsf{Y}_0 \in [0;1]$ • $\mathsf{Y}_i \in [-5;5]$ $(1 \leq i < l)$
	Where	• $\mathfrak{G}_0(\mathbf{Y}) = \mathsf{Y}_0$	
		• $\mathfrak{G}_1(\mathbf{Y}) = \gamma\left(1 - \frac{\mathfrak{G}_0(\mathbf{Y})}{\gamma}\right)$	
		• $\gamma = 1 + 10(l\text{–}1) + \sum_{i=1}^{l-1}\left(\mathsf{Y}_i^2 - 10\cos(4\pi\mathsf{Y}_i)\right)$	
	Such That	• $\cos(-\frac{5\pi}{100})\mathfrak{G}_1(\mathbf{Y}) - \sin(-\frac{5\pi}{100})\mathfrak{G}_0(\mathbf{Y}) \geq 40\lvert\sin(5\pi\left[\sin(-\frac{5\pi}{100})\mathfrak{G}_1(\mathbf{Y}) + \cos(-\frac{5\pi}{100})\mathfrak{G}_0(\mathbf{Y})\right])\rvert^6$	

Figure 7.6: Real–valued multi–objective optimization test problems.

BT_1
Function BT_1 has not been used before in the field multi–objective optimization. It differs from the other three functions in that it has multivariate (linear) interactions between the problem variables. Therefore, more complex factorizations are required to exploit these interactions, whereas all of the other problems are well–suited to be optimized using the univariate factorization. The Pareto optimal front is given by $\mathfrak{G}_1(\mathbf{Y}) = 1 - \mathsf{Y}_0$.

Binary multi–objective optimization problems

In Figure 7.7, we have specified four binary multi–objective optimization problems. Next to being binary, these problems are also multi–objective variants of well–known combinatorial optimization problems. The number of objectives for these problems is not restricted to two and is denoted by m.

It is important to note that we have used random instances for the combinatorial optimization problems. In the case of only a single objective, random instances may on average be easy for some combinatorial problems. However, in the case of multiple objectives, finding the Pareto front is usually much more difficult, even if efficient algorithms are available for the single–objective case (Ehrgott and Gandibleux, 2000). Therefore, the instances used in our test suite are not expected to be over–easy. Furthermore, the problems also serve to indicate differences between the different multi–objective algorithmic approaches other than the fact that dependencies between problem variables can be exploited. This relative performance of the algorithms may be well observed using our proposed test–suite. On the other hand, the degree of interaction between the problem variables in randomly generated problem instances may not be too large, which may cause optimization algorithms that regard the problem variables independently of each other to be the most efficient.

Maximum satisfiability
In the maximum satisfiability problem, we are given a propositional formula in conjunctive normal form. The goal is to satisfy as many clauses as possible. The solution string is a truth assignment to the involved literals. These formulas can be represented by a matrix in which row i specifies what literals appear either positive (1) or negative (-1) in clause i. In the multi–objective variant of this problem, we have m of such matrices and only a single solution to satisfy as many clauses as possible in each objective at the same time.

Knapsack
The multi–objective knapsack problem was first used to test MOEAs on by Zitzler and Thiele (1999). We are given m knapsacks with a specified capacity and n items. Each item can have a different weight and profit in every knapsack. Selecting item i in a solution implies placing it in every knapsack. A solution may not cause exceeding the capacity of any knapsack.

Name	**Definition**
MS (Maximum Satisfiability)	**Maximize** $(\mathfrak{G}_0(\mathbf{X}), \mathfrak{G}_1(\mathbf{X}), \ldots, \mathfrak{G}_{m-1}(\mathbf{X}))$ **Where** • $\forall_{i \in \boldsymbol{\mathcal{M}}} : \mathfrak{G}_i(\mathbf{X}) = \sum_{j=0}^{c_i - 1} \operatorname{sgn}\left(\left[\sum_{k=0}^{l-1} (C_i)_{jk} \otimes \mathsf{X}_k\right]\right)$ • $\operatorname{sgn}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$ • $\begin{array}{c\|cc} \otimes & 0 & 1 \\ \hline -1 & 1 & 0 \end{array}$ $\begin{array}{c\|cc} \otimes & 0 & 1 \\ \hline 0 & 0 & 0 \end{array}$ $\begin{array}{c\|cc} \otimes & 0 & 1 \\ \hline 1 & 0 & 1 \end{array}$
KN (Knapsack)	**Maximize** $(\mathfrak{G}_0(\mathbf{X}), \mathfrak{G}_1(\mathbf{X}), \ldots, \mathfrak{G}_{m-1}(\mathbf{X}))$ **Where** • $\forall_{i \in \boldsymbol{\mathcal{M}}} : \mathfrak{G}_i(\mathbf{X}) = \sum_{j=0}^{l-1} P_{ij} \mathsf{X}_j$ **Such That** • $\forall_{i \in \boldsymbol{\mathcal{M}}} : \sum_{j=0}^{l-1} W_{ij} \mathsf{X}_j \leq c_i$
SC (Set Covering)	**Minimize** $(\mathfrak{G}_0(\mathbf{X}), \mathfrak{G}_1(\mathbf{X}), \ldots, \mathfrak{G}_{m-1}(\mathbf{X}))$ **Where** • $\forall_{i \in \boldsymbol{\mathcal{M}}} : \mathfrak{G}_i(\mathbf{X}) = \sum_{j=0}^{l-1} C_{ij} \mathsf{X}_j$ **Such That** • $\forall_{i \in \boldsymbol{\mathcal{M}}} : \forall_{0 \leq j < r} : \sum_{k=0}^{l-1} (A_i)_{jk} \mathsf{X}_k \geq 1$
MST (Minimal Spanning Tree)	**Minimize** $(\mathfrak{G}_0(\mathbf{X}), \mathfrak{G}_1(\mathbf{X}), \ldots, \mathfrak{G}_{m-1}(\mathbf{X}))$ **Where** • $\forall_{i \in \boldsymbol{\mathcal{M}}} : \mathfrak{G}_i(\mathbf{X}) = \sum_{j=0}^{l-1} W_{ij} \mathsf{X}_j$ **Such That** • $\forall_{S \subseteq V} : \sum_{\mathsf{X}_j \in (S \times (V-S))} \mathsf{X}_j \geq 1$ • $\forall_{S \subseteq V} : \sum_{\mathsf{X}_j \in (S \times S)} \mathsf{X}_j \leq \lvert S \rvert - 1$

Figure 7.7: Binary multi–objective combinatorial optimization test problems.

Set covering
In the set covering problem, we are given l locations at which we can place some service at a specified cost. Furthermore, associated with each location is a set of regions $\subseteq \{0, 1, \ldots r-1\}$ that can be serviced from that location. The goal is to select locations such that *all* regions are serviced against minimal costs. In the multi–objective variant of set covering, m services are placed at a location. Each service however covers its own set of regions when placed at a certain location and has its own cost associated with a certain location. A binary solution indicates at which locations the services are placed.

Minimal spanning tree
In the minimal spanning tree problem we are given an undirected graph (V, E) such that each edge has a certain weight. We are interested in selecting edges $E_T \subseteq E$ such that (V, E_T) is a spanning tree. The objective is to find a spanning tree such that the weight of all its edges is minimal. In the multi–objective variant of this problem, each edge can have a different weight in each objective.

7.5.2 Experiment setup

Optimization problem dimensionalities

Real–valued multi–objective optimization problems
For the real–valued problems, we tested all algorithms with both $l = 10$ and $l = 100$ problem variables.

Binary multi–objective optimization problems
For the binary problems, we used test instances with $l = 100$ and $l = 1000$. For the maximum satisfiability problem, we generated the test instances by generating 2500 clauses for $l = 100$ and 12500 clauses for $l = 1000$ with a random number of literals between 1 and 5. For the knapsack problem, we generated instances by generating random weights in $[1; 10]$ and random profits in $[1; 10]$. The capacity of a knapsack was set at half of the total weight of all the items, weighted according to that knapsack objective. For set covering, the costs were generated at random in $[1; 10]$. We used 250 regions and 2500 regions to be serviced for $l = 100$ and $l = 1000$ respectively. We varied the problem difficulty through the region–location adjacency relation. This relation was generated by making each location adjacent to 70 and 50 randomly selected regions for $l = 100$ and $l = 1000$ respectively. Finally, for the minimum spanning tree problem, we used full graphs with 105 edges (15 vertices) and 1035 edges (46 vertices). The dimensionality of these problems is therefore not precisely 100 and 1000. The weights of the edges were generated randomly in $[1; 10]$.

Optimization problem constraints

Problems CTP_7, set covering, knapsack and minimal spanning tree have constraints. To deal with them, we can use a repair mechanism to transform infea-

sible solutions into feasible solutions. Another approach is based on the notion of constraint–domination introduced by Deb et al. (2001). This notion allows to deal with constrained multi–objective problems in a general fashion. A solution $\mathbf{Z}^0$ is said to *constraint–dominate* solution $\mathbf{Z}^1$ if any of the following is true:

1. Solution $\mathbf{Z}^0$ is *feasible* and solution $\mathbf{Z}^1$ is *infeasible*
2. Solutions $\mathbf{Z}^0$ and $\mathbf{Z}^1$ are both *infeasible*, but $\mathbf{Z}^0$ has a smaller overall constraint violation
3. Solutions $\mathbf{Z}^0$ and $\mathbf{Z}^1$ are both *feasible* and $\mathbf{Z}^0 \succ \mathbf{Z}^1$

The overall constraint violation is the amount by which a constraint is violated, summed over all constraints. We have used this principle for problems CTP_7 and set covering. For the knapsack problem, an elegant repair mechanism was proposed earlier by Zitzler and Thiele (1999). For the minimal spanning tree problem, the number of constraints grows exponentially with the problem size l. We therefore propose to use repair mechanisms for these latter two problems.

Knapsack repair mechanism
If a solution violates a constraint, the repair mechanism iteratively removes items until all constrains are satisfied. The order in which the items are investigated, is determined by the maximum profit/weight ratio. The items with the lowest profit/weight ratio are removed first.

Minimal spanning tree repair mechanism
First the edges are removed from the currently constructed graph and they are sorted according to their weight. Next, they are added to the graph such that no cycles are introduced. This is done by only allowing edges to be introduced *between* the connected components in the graph. If after this phase, the number of connected components has not been reduced to 1, all edges between the connected components are regarded in increasing weight and again the connected components are merged until a single component is left.

General algorithmic setup

We ran every algorithm 50 times on each problem. In any single run we chose to allow a maximum of $20 \cdot 10^3$ evaluations for the real–valued problems of dimensionality $l = 10$ and the binary problems of dimensionality $l = 100$ and a maximum of $100 \cdot 10^3$ evaluations for the real–valued problems of dimensionality $l = 100$ and the binary problems of dimensionality $l = 1000$. As a result of imposing the restriction of a maximum of evaluations, a value for the population size n exists for each MOEA such that the MOEA will perform best. For too large population sizes, the search will move towards a random search and for too small population sizes, there is not enough information to perform adequate model selection and induction. We therefore increased the population size in steps of 25 to find the best results. To actually select the best population size, we selected the result with the lowest value for the $\boldsymbol{D}_{\boldsymbol{\mathcal{P}_F} \rightarrow \mathcal{S}}$ indicator.

Algorithms

We tested a few variants of three MOEAs. In the following we will describe the details that are required in addition to the details given in earlier sections for constructing the actual MOEAs that we will use for testing.

SPEA
For SPEA, we used uniform crossover and one–point crossover with a probability of 0.8. Bit flipping mutation was used in combination with either of these recombination operators with a probability of 0.01. These settings were used previously by the SPEA authors (Zitzler et al., 2000). We allowed the size of the external storage in SPEA to become as large as the population size. For the real problems, we encoded every variable with 30 bits.

NSGA–II
For NSGA–II, we used the same crossover and mutation operators and the same encoding for the real variables.

MIDEA
For MIDEA, we used the leader clustering algorithm in the objective space such that four clusters were constructed on average. If the number of clusters becomes too large, the requirements for the population size increases in order to facilitate proper factorization selection in each cluster. We do not suggest that the number of clusters we use is optimal, but it will serve to indicate the effectiveness of parallel exploration along the Pareto front as well as diversity preservation. In each cluster, we either used the univariate factorization or we estimated a Bayesian factorization in the case of real variables. However, in the case of 100–dimensional real–valued problems, we allowed only at most a single parent for any variable. In the case of binary variables, we used the optimal dependency tree algorithm by Chow and Liu (1968) to estimate a tree factorization in each cluster. To further investigate the influence of the different components in the MIDEA algorithm, we also performed tests in which only a single cluster is used. Furthermore, we also replaced the use of estimating probability distributions by the use of one–point crossover and uniform crossover with mutation as used in the SPEA and NSGA–II algorithms. In the case of clustering in combination with the use of crossover operators, restricted mating was employed in order to ensure clustered exploration along the front. In restricted mating crossover, an offspring is produced using two parent solutions that are picked from the same cluster. For the truncation percentile, we used the rule of thumb by Mühlenbein and Mahnig (1999) and set τ to 0.3. Furthermore, for the comparison benchmarks, we set the diversity preservation parameter to $\delta = 1.5$, which was experimentally determined to give good results both with respect to diversity preservation and well as selective pressure. For an investigation of the influence of δ on the performance of MIDEA, we also varied δ and observed the results in some additional experiments, the results of which are reported below.

7.5.3 Results

To compare the MOEAs, we investigate their average performance with respect to performance indicators introduced in Section 7.2. The performance indicators that we use are the $\boldsymbol{D}_{\mathcal{P}_F \to \mathcal{S}}$ indicator, the **FS** indicator and the **FO** indicator. For the $\boldsymbol{D}_{\mathcal{P}_F \to \mathcal{S}}$ performance indicator, we used different sets to represent the Pareto optimal front for the real–valued optimization problems and the binary optimization problems. For the real–valued optimization problems we used a uniformly sampled set of 5000 solutions along the Pareto optimal front. Since we do not know the Pareto optimal front for the binary optimization problems, we used the Pareto front over all results obtained by all MOEAs.

For each of the performance indicators, we computed their average and standard deviation over the 50 runs to get an assessment of their performance. The averages are tabulated in Appendix B Section B.3 (Figures B.12 through B.17). The best results are written in boldface. For each algorithm, the type of recombination is indicated as a superscript. The $\mathbb{M}$IDEA algorithms are indicated by a single $\mathbb{M}$ symbol. For all tested $\mathbb{M}$IDEA algorithms, the subscript indicates whether only a single cluster was used or whether clustering was performed in either the parameter space or the objective space. The population sizes that led to the best performance, are tabulated in Appendix B Section B.3 (Figures B.18 and B.19). For the standard deviations, we refer the interested reader to a technical report (Bosman and Thierens, 2002c). Although the average behavior is the most interesting, the standard deviations are vital to determine whether the differences in the average behavior of the different algorithms are significant. To investigate these significances, we have performed Aspin–Welch–Satterthwaite (AWS) statistical hypothesis T–tests at a significance level of $\alpha = 0.05$. The AWS T–test is a statistical hypothesis test for the equality of means in which the equality of variances is not assumed. For each problem, we verified for each pair of algorithms whether the average obtained performance indicator values differ significantly. We assigned a value of 1 if an algorithm scored significantly better and a value of -1 if an algorithm scored significantly worse. We summed the so obtained matrices over all problems to get the statistically significant improvement matrices that are shown in Appendix B Section B.3 (Figures B.20 through B.22). We also computed the sum for each algorithm of its significant improvement values over all other algorithms to indicate the summed relative statistically significant performance of the algorithms. A less detailed summary of the statistical significance tests is shown in Figure 7.9. In this figure histograms are used to indicate the sum of the results of the statistical significance tests for each algorithm compared with all other algorithms. The histogram represents the sums for the real–valued problems and the combinatorial problems for the different tested dimensionalities and the average of these four sums.

Influence of problem dimensionality

The $\mathbb{M}$IDEA variants perform best in the case in which the dimensionality of the problem is larger ($l = 100$ for the real–valued problems, $l = 1000$ for the binary problems). This is most likely due to the more powerful diversity exploration

and preservation in MIDEA. As the dimensionality of the problem goes up, the parameter search space becomes larger. In the case of the binary combinatorial problems, the number of solutions in the objective space becomes larger as well. If clustering in the objective space is used in MIDEA, better results are obtained on average as the dimensionality of the problem increases. In Figure 7.8 the Pareto fronts over 50 runs for all algorithms are plotted on one problem from each problem class and dimensionality. The better diversity preservation and proper distribution of the points along the front can be seen clearly for the problems of larger dimensionality. For the lower dimensionality problems, better diversity preservation can also be observed, which is most exemplified by the fact that MIDEA obtains non–dominated solutions at the outer ends of the front for the knapsack problem with $l = 100$.

Influence of mixtures by clustering the objective space
The fact that the use of mixtures by clustering the objective space allows for enhanced diversity exploration and preservation, can also be observed by the difference between the spread obtained by MIDEA with crossover operators using only a single cluster versus the case in which on average four clusters are used. A wider spread of solutions is found when clustering is enabled. Furthermore, although clustering in the parameter space is a powerful approach to enhance the learning of probabilistic models, it does not lead to better results in multi–objective optimization.

Influence of the problem structure exploitation capabilities of IDEAs
On the BT_1 problem, modelling interactions in MIDEA leads to better results than those obtained by the other MOEAs. Thus, exploiting interactions can be beneficial in multi–objective optimization. For the BT_1 problem with $l = 10$, if we allow for $5 \cdot 10^5$ evaluations, the MIDEA variant that learns Bayesian factorizations is even capable of finding near optimal solutions whereas the other MOEAs were observed not at all to be able to produce comparable results.

The influence of δ and the trade–off between proximity and diversity
In our benchmarks, we have picked a specific value for δ. However, the δ parameter is a unique parameter that determines the balance between non–domination selection pressure and diversity preservation selection pressure. It is therefore important to also reflect on the influence of δ on the performance of MIDEA.

- **The influence of δ on different problems and selection strategies** We have performed additional tests using the MIDEA framework in which we have varied the value of δ from 1 to 3 in steps of 0.25 and have kept τ fixed at 0.3. To demonstrate the influence of using domination ranks as is done in the NSGA–II instead of using the domination count, we have also used domination ranks for determining the preselection. Figure 7.10 shows the resulting values on each problem for the four different performance indicators from Section 7.2.2 obtained with the population size that resulted in the best $\boldsymbol{D}_{\mathcal{P}_F \rightarrow \mathcal{S}}$ indicator value, averaged over 10 runs.

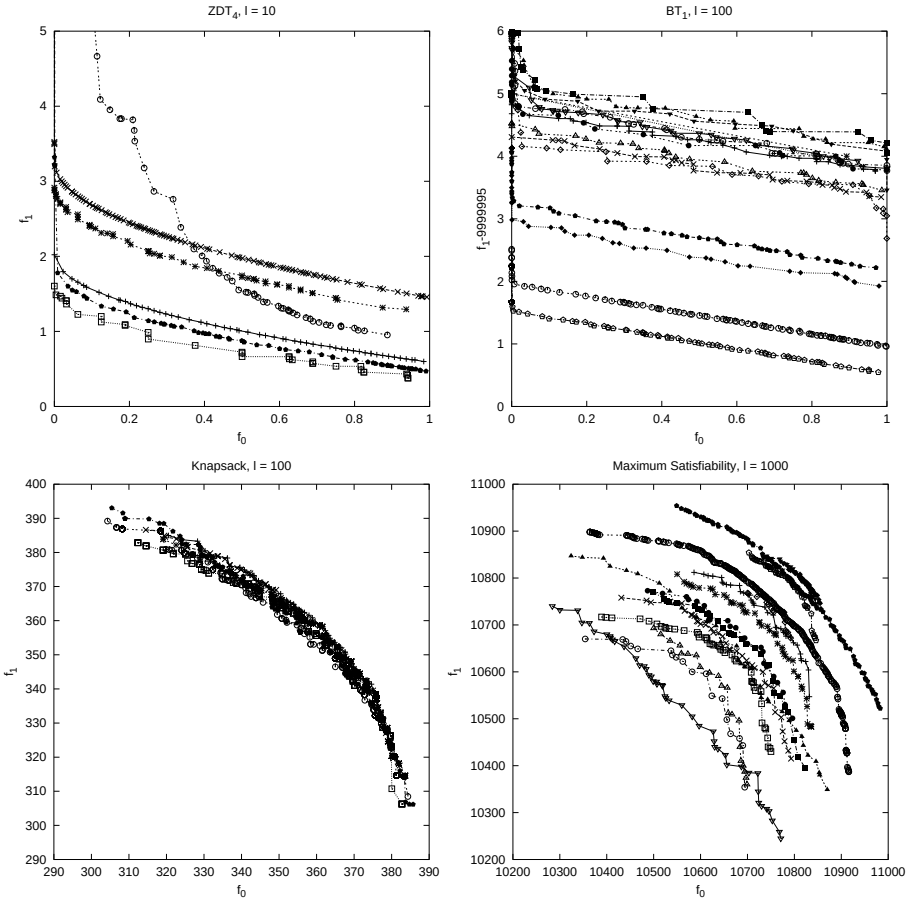


Figure 7.8: Pareto fronts over 50 runs on a few of the tested problems. For the ZDT_4 problem in 10 dimensions and the knapsack problem in 100 dimensions, only the SPEA, NSGA-II and MIDEA with clustering in the objective space and probabilistic model learning variants are shown. For the BT_1 problem in 100 dimensions and the maximum satisfiability problem in 1000 dimensions, the results for all algorithms are shown.

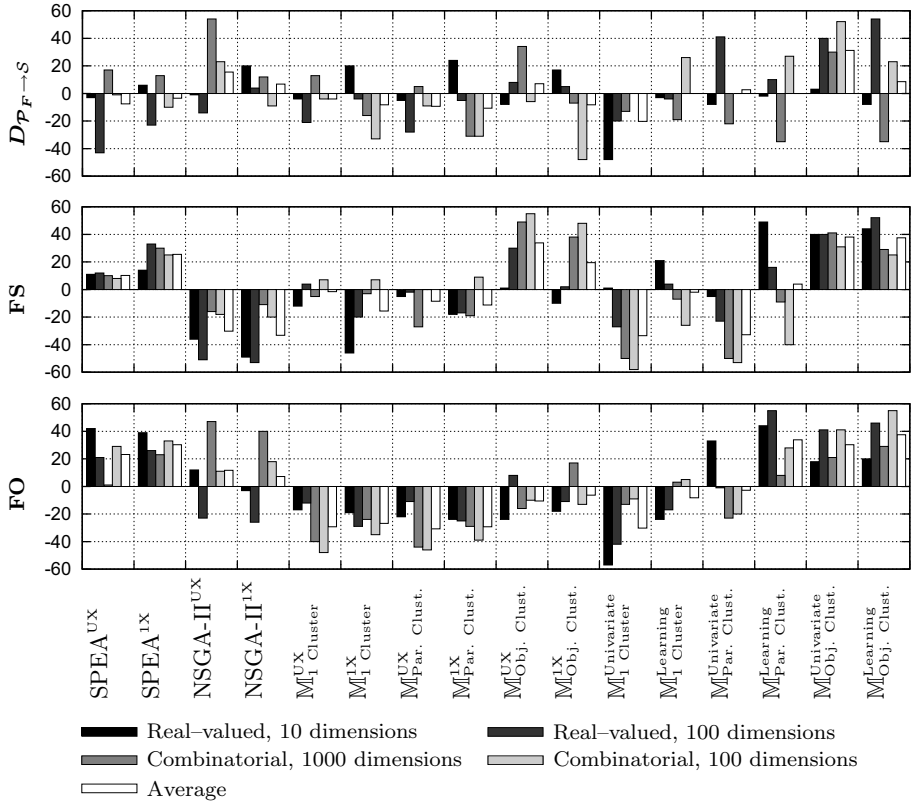


Figure 7.9: A summary of the results of the statistical hypothesis tests performed for each pair of algorithms. For each algorithm, the sum of the outcome of the statistical hypothesis tests is shown for the real–valued problems and the combinatorial problems for each dimensionality separately. Furthermore, the average of these values is also shown, which serves as a global indicator of the performance of an algorithm relative to the other tested algorithms.

The results for the two non–domination preselection approaches do not differ much. The behavior with respect to the different performance indicators on each problem is similar. For small values of δ, the ability to find solutions close to the Pareto optimal front worsens as δ is increased and thus more effort is spent on diversity preservation. This can be seen in the figure for the $\boldsymbol{D}_{\mathcal{S}\rightarrow\boldsymbol{\mathcal{P}}_{\boldsymbol{F}}}$ indicator value. Furthermore, the number of solutions on the front rapidly drops to lower values since elitism does not always maintain the non–dominated solutions. Still, the added effort spent on diversity does pay off in a certain way, since the diversity as measured by the front spread indicator increases as δ is increased. The most interesting results can be seen in the figure that displays the $\boldsymbol{D}_{\boldsymbol{\mathcal{P}}_{\boldsymbol{F}}\rightarrow\mathcal{S}}$ indicator value. For quite a large range of values for δ, the indicator value does not worsen, but sometimes becomes even better. Within this range, the trade–off between diversity preservation along the set of non–dominated solutions and the proximity of non–dominated solutions with respect to the Pareto optimal front is the most interesting. With respect to the performance indicator used, there is a certain optimal value. However, this performance indicator only reflects a certain balance between the two goals. Since the average distance to the front only worsens and the front spread only increases, most different settings for the algorithm do not outperform each other if we have no preference for these two goals. Outperformance can only be detected if δ becomes very large. In that case, the front spread increases slightly but the average distance of each point in the resulting approximation set to the Pareto optimal front increases very much. These observations regarding outperformance are confirmed by the results in Figure 7.11. The results in this figure show for the use of one–point crossover the most frequently occurring relation from the categorization of Zitzler et al. (2002) when comparing the approximation sets of a MOEA using one value for δ with another value for δ. Indeed, in almost all cases, the approximation sets are most frequently categorized as incomparable. Only for $\delta = 3.0$ there are some cases in which we can speak of true outperformance. Moreover, within this large set of incomparable MOEAs, the distance to the Pareto front worsens monotonically as δ is increased, but the front spread improves monotonically.

As we argued earlier in Section 7.2.2, the additional information based on the performance indicators leads us to conclude that there truly is no preference for any of the MOEAs that are classified as being incomparable. However, the classification in Figure 7.11 becomes less certain as the value of δ is increased. The reason for this is that not all combinations of approximation sets over the different runs are classified as being incomparable. As δ is increased, the frequency of the classification of being incomparable gets closer to the classification of being better or even being strictly dominating. If these frequencies become very close, we might already find one MOEA preferable over another. Intuitively, all of this is reflected by the large, relatively flat part in the $\boldsymbol{D}_{\boldsymbol{\mathcal{P}}_{\boldsymbol{F}}\rightarrow\mathcal{S}}$ performance indicator. We truly have no preference for any of the MOEAs that correspond

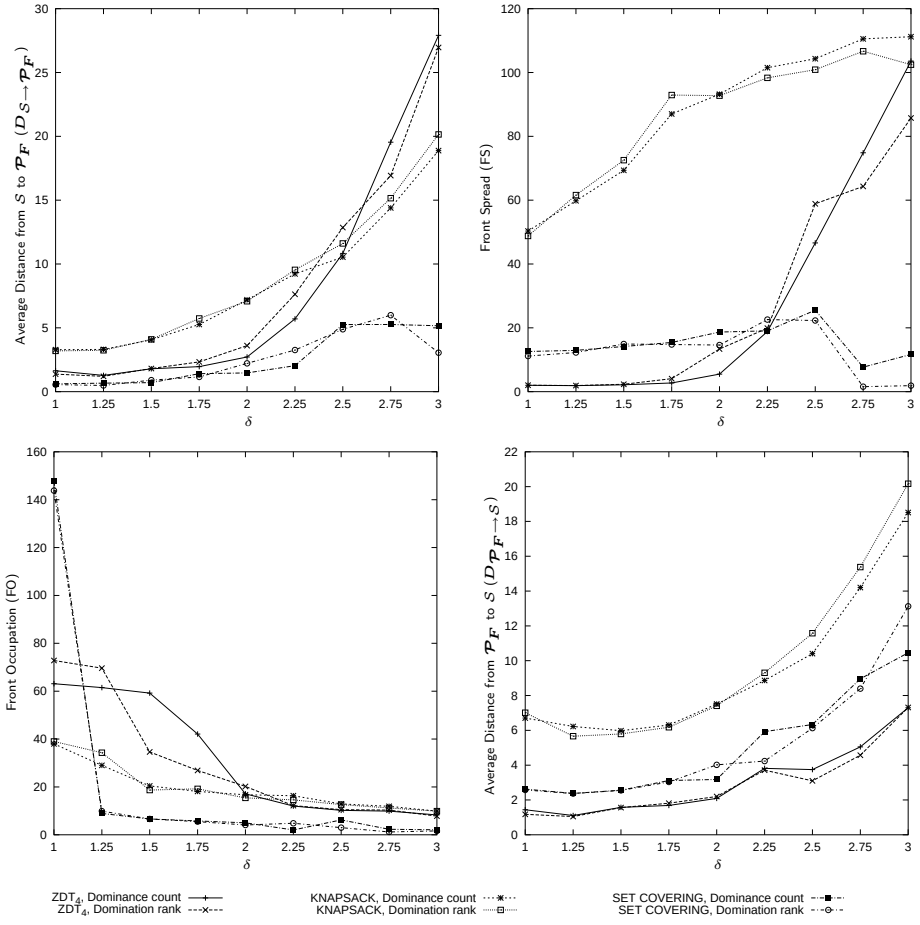


Figure 7.10: Results on ZDT_4 with $l = 10$, the knapsack and the set covering problems using one–point crossover and one cluster in $\mathbb{M}$IDEA. The results measured in four different performance indicators are shown as a function of δ.

Knapsack, one–point crossover

Knapsack, one–point crossover

δ	1.00	1.25	1.50	1.75	2.00	2.25	2.50	2.75	3.00
1.00	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
1.25	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
1.50	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
1.75	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\succ\succ$
2.00	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\succ\succ$
2.25	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
2.50	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
2.75	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
3.00	$\parallel$	$\parallel$	$\parallel$	$\prec\prec$	$\prec\prec$	$\parallel$	$\parallel$	$\parallel$	$\parallel$

Figure 7.11: Comparing all combinations of the results of using one–point crossover for all tested values of δ on the multi–objective knapsack problem with $l = 100$. The table entries are row–versus–column relations.

to this flat part in the graph. However, as this indicator value increases, a preference starts to be formed towards MOEAs with a lower value for the $\boldsymbol{D}_{\boldsymbol{\mathcal{P}}_{\boldsymbol{F}} \to \mathcal{S}}$ indicator. Indeed, this already happens for smaller values of δ than $\delta = 3$. This corresponds to the observation of the decrease of the frequency of the incomparable classification. In general, making modest choices on whether we spend more effort on diversity preservation or on proximity leads to MOEAs that have a performance such that we truly do not prefer one MOEA over another. Since for $\delta = 1$, the MIDEA framework is similar to the NSGA–II, in combination with the results obtained in our benchmark again argues that MIDEA is equally good as does the NSGA–II if we disregard the type of recombination operator.

- **The influence of δ over different recombination operators**
  In Figure 7.12 additional results are shown for the knapsack problem using different recombination methods. A similar behavior is observed for the different recombination operators as observed for only one–point crossover using different selection strategies on all three problems. However, one interesting additional phenomenon can be seen in the graphs in Figure 7.12. The results obtained for the approaches that use clustering in the objective space, have an *intrinsically* better performance with respect to the front spread performance indicator than any other method. For all values of δ, both the front spread as well as the distance to the Pareto optimal front are better if clustering is used to construct a mixture of univariate factorizations instead of the univariate factorization. This is confirmed only in part by the classification results in Figure 7.13. Indeed, for a larger variety of values for δ, the univariate factorization is outperformed by the mixture of univariate factorizations. However, based on the results provided by the performance indicators, one would expect the figure to show that for all values of δ, the MOEA that uses the univariate factor-

ization is outperformed by the MOEA that uses the mixture of univariate factorizations. However, the majority of the comparisons result in the classification of being incomparable. Whereas in the case when we were comparing one–point crossover with itself, the classifications of being incomparable were a result of approximation sets that are in most cases not preferable over one another such as in Figure 7.3, in this case the classifications of being incomparable are a result of cases such as the one in Figure 7.2. The performance indicators now show that we can truly speak of a preference for using the mixture of univariate factorizations over the univariate factorization for the multi–objective knapsack problem since they are all in favor of the mixture of univariate factorizations. The added use of clustering seems to lead to more advanced MOEAs than when clustering is not used to stimulate parallel exploration. This example serves to show that although there is an intrinsic trade–off in the choices that are to be made in the general framework, this does not imply that we cannot make some general choices that lead to intrinsically better MOEAs. On the other hand, within for instance the use of a mixture model, by changing the value of δ, again different choices for spending more or less effort on diversity preservation are made. Similar arguments and comparison classifications can be made to show again that the performance for the different goals in multi–objective optimization when using clustering in the objective space are mostly incomparable, which indicates that the trade–off between diversity preservation and proximity is still present, even if intrinsically better recombination operators are used.

There is one more interesting thing to be observed in Figure 7.12. The use of Bayesian tree factorizations is a more involved method than using univariate factorizations, since the latter approach is quite similar to using uniform crossover with a crossover probability of 1.0. However, the use of the univariate factorization is still capable of producing solutions that are closer to the Pareto optimal front. One of the reasons for this is that the approach based on estimating Bayesian tree factorizations is capable of generating solutions at locations in which the less involved recombination operators are not capable of generating new solutions. This can be seen in the figure for the front occupation, since the Bayesian tree factorization approach is capable of generating more non–dominated solutions than the less involved recombination operators. Also, a larger front–spread is obtained using the Bayesian tree factorization. As a result of this, a better approximation of the Pareto optimal front is more likely to be obtained, but more evaluations may be required because a larger set of solutions may be found than is possible using the less involved operators. This is another important aspect to consider when evaluating MOEAs.

- **Overall considerations regarding the influence of δ**
  Our choice of $\delta = 1\frac{1}{2}$ in the benchmark of $\mathbb{M}$IDEA may have indicated a less preferable performance with respect to getting close to the Pareto optimal front than using NSGA–II or SPEA. However, the performance with

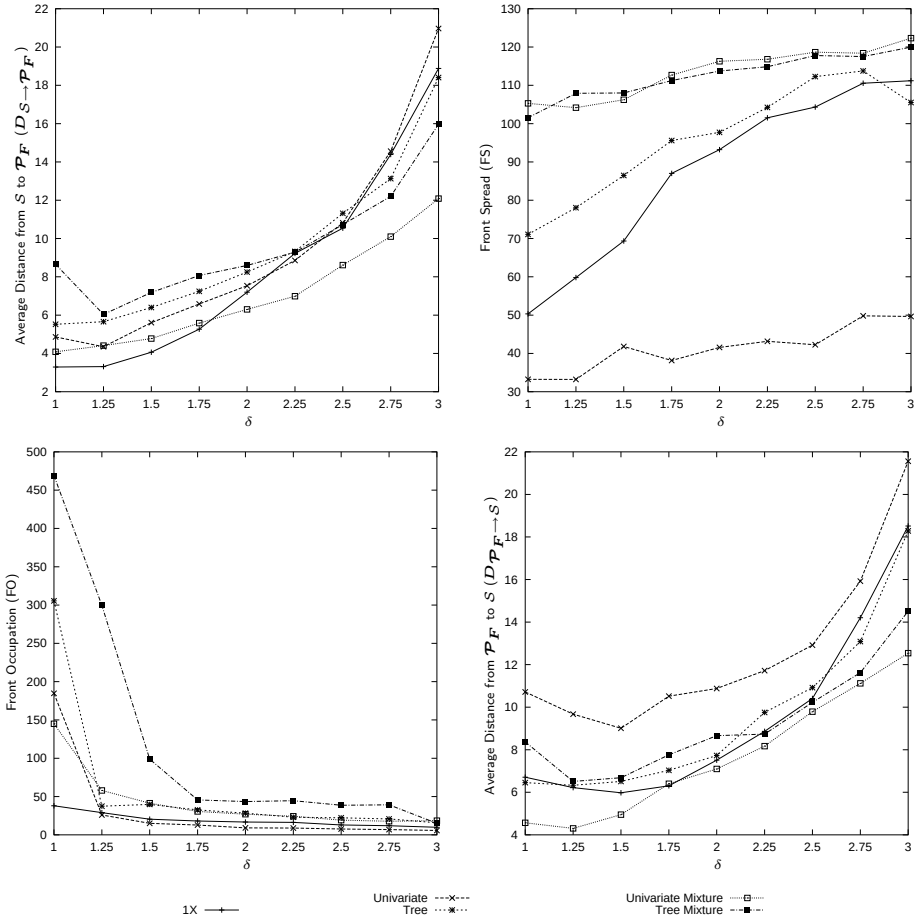
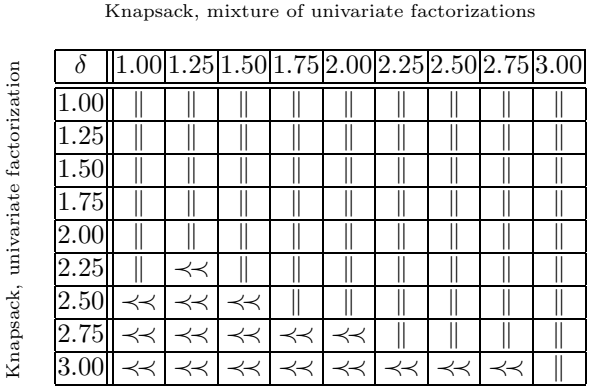


Figure 7.12: Additional results for the knapsack problem with $l = 100$ using different recombination operators in the example instance of the general elitist framework. Selection is based on the domination count. The results measured in four different performance indicators are shown.



δ	1.00	1.25	1.50	1.75	2.00	2.25	2.50	2.75	3.00
1.00	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
1.25	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
1.50	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
1.75	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
2.00	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
2.25	$\parallel$	$\prec\succ$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
2.50	$\prec\succ$	$\prec\succ$	$\prec\succ$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
2.75	$\prec\succ$	$\prec\succ$	$\prec\succ$	$\prec\succ$	$\prec\succ$	$\parallel$	$\parallel$	$\parallel$	$\parallel$
3.00	$\prec\succ$	$\prec\succ$	$\prec\succ$	$\prec\succ$	$\prec\succ$	$\prec\succ$	$\prec\succ$	$\prec\succ$	$\parallel$

Figure 7.13: Comparing all combinations of the results of using the univariate factorization for all tested values of δ with the result of using the mixture of univariate factorizations for all tested values of δ on the multi–objective knapsack problem with $l = 100$. The table entries are row–versus–column relations.

respect to diversity was in general better. Our additional experiments have indicated that the results of NSGA–II, SPEA and MIDEA with $\delta = 1\frac{1}{2}$ do not lead us to conclude that any of these MOEAs outperform any other of these MOEAs. The reason for this is that δ is a trade–off parameter that is unique to the field of multi–objective evolutionary optimization. Our choice of δ in our benchmark has led to a shift of the performance of MIDEA towards diversity preservation in such a way that the resulting MIDEA is not inferior to for instance NSGA–II, but performs differently with respect to the goals of proximity and diversity. Moreover, for $\delta = 1$, MIDEA performs similar to NSGA–II. Thus, if we momentarily disregard the instantiation of the recombination operator, the MIDEA approach is capable of obtaining results that are similar to the results of current state–of–the art MOEAs. However, the selection mechanism in MIDEA allows the user to shift the bias of the algorithm more towards the goal of proximity or the goal of diversity, which cannot be done by other currently available MOEAs. This makes MIDEA an interesting approach that offers a good new perspective on multi–objective evolutionary optimization.

7.5.4 Practitioner's summary

Our experimental results indicate that clustering the objective space to construct mixture probability distributions in MIDEAs leads to superior MOEAs, making our specific MIDEA instance based on mixture probability distributions truly an effective new tool for multi–objective optimization. Furtermore, NSGA–II is overall the most competitive. However, there is an added value to the use of MIDEA in that it is able to obtain and maintain a larger and more diverse Pareto front by parallel front exploration and diversity preserving

selection. The experiments underline these results as the front spread (Figures B.14 and B.15), front occupation (Figures B.16 and B.17) and the global Pareto fronts in Figure 7.8 indicate a better performance. This increased performance is also statistically significant, as can be seen in figures B.21 and B.22. The use of clustering to obtain mixture probability distributions clearly leads to a significant increase of performance in the preservation and exploration of diversity. Yet, using only a single cluster in combination with the diversity preserving selection method in MIDEA on average yields inferior results compared to NSGA–II with respect to the $\boldsymbol{D}_{\boldsymbol{\mathcal{P}_F}\rightarrow\mathcal{S}}$ performance indicator. However, the δ parameter in the diversity preservation scheme that we have used takes the focus of MIDEA further away from obtaining solutions as close as possible to the Pareto optimal front as δ is increased. It is to be expected that results similar to those obtained by the NSGA–II will be obtained if δ is set to 1 since in that case explicit diversity preservation is only performed if the number of non–dominated solutions exceeds the selection size, which is a similar approach as is used in NSGA–II.

Overall, MIDEA is a very good MOEA that could be applied to real–world problems. We suggest to set $\delta \in [1; 1\frac{1}{2}]$ and to first use simple factorizations such as the univariate factorization. If more time and function evaluations are available, more complex factorizations can be used as well. To implement MIDEA, the IDEA framework needs to be instantiated with the learning of mixture probability distributions by clustering. Details of various clustering algorithms are given in Section 2.7. For the estimation of a simpler probability distribution in each cluster, the desired way of learning a probabilistic model should be implemented (see Section 2.6). The gpdf to be used depends on the application, but for real–valued random variables, the normal gpdf can be used. The details of this gpdf regarding its estimation from data and the drawing of new samples are given in Section 5.1.1. Finally, selection should be performed using the diversity preserving selection operator (see Section 7.4.1).

7.6 Discussion and future research

Our results for multi–objective optimization using MIDEA instances indicate that the use of mixture probability distributions can indeed lead to more effective multi–objective optimization than when other evolutionary recombination and mutation operators are used. However, measuring the improvement in effectiveness must be done carefully. Our results have indicated that the use of mixture probability distributions that are obtained by clustering the objective space can lead to better MOEAs both in the sense of getting closer to the Pareto optimal front as well as in obtaining a more diverse approximation set. Using factorizations to further exploit problem structure in the form of dependencies between problem variables can lead to the generation of more solutions on a less preferred front. Although such an approximation set is a result that can be found more efficiently by estimating involved probability distributions instead of using classical recombination operators, such a result is intuitively less de-

sirable. More research is required to investigate this issue further. On the one hand it would be interesting to attempt to overcome this problem and ensure that the added complexity of the inductive capabilities of estimating probability distributions results in a more effective exploration towards the Pareto optimal front. On the other hand it would be interesting to investigate what type of (real–world) multi–objective optimization problems can be solved more efficiently using MIDEA instances because of difficulties such as non–linear dependencies between the problem variables.

The number of evaluations that were allowed in this benchmark, is relatively small. Although a good assessment is obtained in how good the tested approaches are at rapidly obtaining a good approximation of the global Pareto optimal front, the use of learning techniques to exploit problem structure usually requires a larger number of evaluations in order for their effectiveness to be displayed. This is even more so if mixtures of factorizations are used, since the number of solutions in each mixture component is smaller than the complete number of selected solutions. This behavior can be seen from the results since the $\boldsymbol{D}_{\boldsymbol{\mathcal{P}}_F \rightarrow \mathcal{S}}$ performance indicator is more often improved by using learning techniques if a single cluster is used than when multiple clusters are used. Thus, it would be interesting to see how the use of learning techniques further influences the results if more evaluations were to be allowed.

Regarding the clustering of the objective space, a similar question remains as for single–objective numerical optimization of how many clusters are truly required to obtain good results. The main difference with numerical optimization is that for multi–objective optimization, clustering is best done in the objective space. The number of objectives will not likely get extremely large because the number of solutions that is required to estimate a Pareto hyper–front grows enormously as the number of objectives increases. A not–all–too large number of clusters is therefore likely to always lead to good diverse results. Still, it would be interesting to investigate the impact on the diversity preserving and Pareto optimal front approaching capabilities as the number of clusters is increased.

In MIDEA, the δ parameter is used to control the combination of non–domination selection and diversity selection. This parameter also controls the combination of proximity and diversity that is offered to the recombination operator. It would be interesting to see the results if we have one δ parameter for determining the ratio between proximity and diversity for selection and elitism, and a different δ parameter for determining the ratio between proximity and diversity for the selection that is offered to the exploration part of the MOEA. In this case, finer grained control and insights would be obtained for determining the influence of added effort on diversity for exploitation and separately for exploration. In our example instance, these influences are linked together since the selected solutions are also used for elitism, which may quickly cause many non–dominated solutions to be lost as δ is increased.

Finally, it would be interesting to test MIDEA instances on problems with (only a few) more objectives and see how well the selection method remains capable of maintaining diversity in objective spaces with a larger dimensionality.

'Ideas... they have the power...'
Napoleon Hill

Discussion and conclusions

In this thesis, we have studied iterated density estimation evolutionary algorithms (IDEAs) as an improvement over classical evolutionary algorithms (EAs) for solving optimization problems. The main contribution of these IDEAs resides in the fact that by estimating a probability distribution over selected solutions and subsequently drawing new solutions from this distribution, an inductive tool is provided for online identification of features of the problem's structure. These features are then used to guide the search more efficiently towards the promising regions of the search space. IDEAs are relatively new to the field of evolutionary computation. In this thesis, we have motivated their use. Moreover, we have investigated the optimization performance of IDEAs in the fields of numerical optimization, permutation optimization and multi–objective optimization. In this chapter, we reflect on the use and applicability of IDEAs for enhanced evolutionary optimization, as well as on future perspectives.

Feasibility, limitations and assumptions for using probabilistic models

For discrete problems with a structure that is decomposable to some degree, IDEAs are capable of detecting and exploiting this structure to perform less disruptive and more constructive recombination than do classical EAs. The reason is that IDEAs exhibit an inductive bias that is adaptive since it resides in a probabilistic model estimated from the selected solutions. Classical recombination operators, in contrast, are not adaptive. Adaptivity in itself however, does not suffice. The model used must be capable of representing the problem's decomposable structure. In IDEAs often factorizations are used which may well match with the structure of the optimization problem at hand, thereby causing a boost to the optimization process.

However, some critical observations regarding the use of probabilistic models and model selection are in order. The most important observation relates to the issue of dimensionality. Some probabilistic models can in theory allow for very good estimates of the true probability distribution that underlies a given set of solutions. For real–valued data, the histogram gpdf is an example of such a model. For this gpdf however, the number of bins increases exponentially with the number of dimensions. If 10 bins are defined for each dimension, then using a joint histogram requires $10^3 = 1000$ bins for $l = 3$. To reliably estimate the probability for each bin reliably, an enormous number of solutions is required. Yet, we would like to be able to solve problems with many more variables efficiently and reliably. A similar observation applies to discrete random variables and the use of frequency tables. It is further important to note that although we do not assume any prior knowledge of a problem's structure, we are not trying to design algorithms that are capable of efficiently and reliably solving all optimization problems. If a problem's variables are all dependent on each other, such as a needle–in–a–haystack problem for all variables, there are simply no substructures that can be exploited by the search algorithm. Due to the curse of dimensionality, we then know that our IDEAs will require an exponential number of evaluations to solve the problem.

When stating that IDEAs are capable of extracting structural features of an optimization problem to achieve efficient and scalable evolutionary optimization, we implicitly assume that there is some decomposability in the problem that can be exploited and captured using a suitable probabilistic model that requires a tractable number of solutions to estimate its parameters. Under this assumption, using proper probabilistic models in IDEAs is a feasible approach to solving optimization problems. In fact, the approach allows for solving such problems more effectively than classical EAs in terms of the number of required evaluations and population size, even for problems of a high dimensionality, as observed in this thesis. It is thus not the dimensionality, but the required complexity of the model class that counts.

We have argued that, under the assumption that an optimization problem has structural features that can be captured and exploited using a tractable probabilistic model, IDEAs are able to perform more effective evolutionary optimization. The question remains, however, whether or not *learning* such a probabilistic model during optimization in an IDEA is also feasible. To this end, it must be tractable to adaptively fit a probabilistic model of bounded complexity to the problem's structure. Depending on the structure, a probabilistic model may or may not be adequate. An example is the univariate factorization. For this model, we know that for certain non–linear subproblems we cannot solve an additively decomposable optimization problem in polynomial time if the subproblems are of length larger than one. However, for any multivariate factorization or any Bayesian factorization, it can be shown analytically that we can solve additively decomposable optimization problems in polynomial time, provided that we can estimate the factorization reliably in polynomial time. Allowing for a more complex model requires more samples and a larger running time. Unfortunately, since we do not know the structure

of a given optimization problem, we cannot select beforehand the most efficient type of model to use. The most reasonable approach, therefore, is to use a model that can capture a good range of dependencies, such as the multivariate factorization or the Bayesian factorization, and to find the best possible instance for the selected type of model during optimization. Hence, under the mentioned assumptions, not only is model selection in IDEAs feasible, it is also required.

For the construction of a model–learning algorithm, once the type of probabilistic model to be learned is fixed, we need a selection criterion preferring one model instance over another. Whatever algorithm is used for model selection, the metric imposes a bias on the model selection algorithm towards a certain subclass of probabilistic models. With the penalization metrics that we have used, for example, we have explicitly enforced a bias towards simpler models. Note that this bias is in accordance with our assumption that the optimization problem that we are trying to solve has a certain degree of decomposability. The bias towards simpler models further enforces more efficient optimization behavior of the resulting IDEAs on problems that have a limited degree of interaction between their variables. Selecting the amount of regularization therefore is not only important for learning the model, it is also vital to the optimization behavior of the resulting IDEA. Upon investigating the behavior of IDEAs for a larger class of optimization problems, therefore, the influence of different model–selection techniques and metrics should not be neglected.

Overall conclusions and future perspective

The IDEA framework allows for elegant modelling of EAs that perform optimization based on density estimation. Within this framework, we have proposed and investigated various types of probability distribution to arrive at IDEA instances for numerical optimization, permutation optimization and multi–objective optimization. In our experiments, IDEAs have obtained better results than classical EAs. The probabilistic model learning techniques that we have used for learning probabilistic models constitute only a part of the arsenal of techniques known today in the statistical learning community. It would be interesting to see whether and how recent advances in statistical learning can be applied to optimization with IDEAs. An example of a technique that has recently gained popularity in classification, is the *support vector machine* (SVM) (Vapnik, 1995). The SVM method is capable of efficiently deriving structural features from the set of samples by mapping the samples onto a high–dimensional space in which linear techniques are used. The SVM method has further been shown to be quite effective in density estimation (Vapnik and Mukherjee, 1999).

In general, IDEAs can outperform classical EAs by using probabilistic techniques to induce and exploit features of the problem's structure. The key to effective optimization is an inductive bias that fits the structure of the problem under study. In an IDEA, the probabilistic model used constitutes an explicit, adaptive inductive bias. Depending on the model used, different types of inductive bias can be introduced and different types of structure can be exploited. If the optimization problem's structure has a degree of decomposability

that can be effectively modelled and induced from selected solutions, then an IDEA is an efficient algorithm for solving the problem. The same is actually true for the use of crossover operators. We have shown that crossover operators are closely related to factorizations. However, crossover operators are classically non–adaptive and therefore have a fixed bias. If we would have an adequate way of making crossover operators adaptive to the problem at hand, crossover–based EAs would also be rendered more efficient in solving optimization problems. The possibility to adapt the model to the optimization problem by inducing it from the selected solutions contributes to the attractiveness of approaches such as IDEA. To adapt the model in a meaningful way, induction must be used on previously evaluated solutions. Because EAs work with a population of solutions, they are natural candidates for optimization based on model adaptation by induction since all kinds of statistical methods can be used that work on collections of samples, such as the estimation of probability distributions.

EA research has led to the acknowledgement that for a fixed representation of solutions, the use of non–adaptive recombination operators such as the classical crossover operators will not lead to efficient optimization unless the bias introduced by the recombination operator fits the problem's structure. To arrive at more efficient optimization, therefore, either the representation has to be changed to fit the bias of the recombination operator, or the operator has to be changed to fit the structure of the optimization problem. The latter approach is the most commonly used, that is, problem–specific recombination operators are constructed. If we assume to have no prior knowledge of the optimization problem under study, however, it is impossible to do so. The only alternative then is to expand the capacity of the model used in the recombination operator so that it can be adapted to the optimization problem. In this thesis, the model used is a probabilistic one. Different types of model can alternatively be used, however. The main point is that model–based EAs are required to improve the general efficiency and applicability of EAs in BBO under the assumption of tractably exploitable features of a problem's structure.

Various models now exist in EA research that have been used for optimization. The question now is not which model is the overall best or the most general: there always is a trade–off between a model's applicability to a specific problem and its applicability to a class of problems. The more general the model, the wider it is applicable, yet also the more resources are required for its proper induction thus the less efficient it becomes for a specific problem. The right questions to ask thus are which model is more suited for which application and to what extent can the new tools aid in solving practical, real–world problems. To answer these questions, the currently available model–based EAs will have to be applied to real–world problems and their performance will have to be carefully studied. These results for specific types of problem will yield insight in the applicability of specific models to specific optimization problems. More important even is the reason *why* certain models perform well and others do not for a specific type of problem. The answer to this question will, on the one hand, indicate how the models that have been used so far can be improved upon and stimulate further development along the lines of inducing models in EAs. On the

other hand, it will provide better insights into the *true* requirements for solving different types of real–world problem. The most interesting research topic will then be to find ways of turning the use of adaptive model–based recombination by induction into a helpful tool for constructing problem–specific operators to be used in EAs. Possibly, this could simply be obtained by using a restricted subclass of the original model class. But careful consideration of how the model is adapted by the EA may lead to a good understanding of dependencies of various types that the optimization problem's structure consists of. These dependencies can then be respected in a completely new problem–specific recombination operator. Certainly, probabilistic model–based EAs such as IDEA will play a prominent role in this research, but it will not be restricted to this type of EA: it will have to involve all adaptive model–based EAs. The main point is that, in the end, for solving a specific type of optimization problem efficiently, the best that one can possibly do is to introduce a good problem–specific inductive bias into the search algorithm.

'I think it would be a good idea.'
Mahatma Gandhi

Maximum likelihood univariate conditional gpdfs

In Section 2.4.2 we have described Bayesian factorizations. From that section, it has become clear that in order to use such factorizations, we need to be able to estimate and sample from univariate conditional gpdfs of the form $P_{\boldsymbol{\theta}}(Z_i|Z_{\boldsymbol{\pi}_i})$. In this appendix we derive the maximum likelihood estimation for the univariate conditional gpdf form of various gpdfs in the integer sample space and the real–valued sample space by computing the fraction of two maximum likelihood multivariate gpdfs according to the definition of the conditional gpdf (Section 2.4.2). We already know from Section 2.4.2 that these conditional gpdfs are actually probability distributions over Z_i. Assuming that we know how to work with the gpdf over a single random variable, if the functional form of the univariate conditional gpdf is closed, this directly implies that we can also work with the univariate conditional gpdf. We will indicate for the derived conditional gpdfs that they are indeed gpdfs over a single variable and that they are of a closed form.

A.1 Integer sample space

In this section, we present the maximum likelihood estimation for the univariate conditional form of the frequency gpdf.

A.1.1 Frequency

The maximum likelihood multivariate integer gpdf for random variables $X_{\boldsymbol{j}}$, is defined by (Anderson, 1958; Tatsuoka, 1971):

$$P^{\mathcal{F}}(X_{\boldsymbol{j}})(x_{\boldsymbol{j}}) \;=\; \frac{1}{|\boldsymbol{\mathcal{S}}|} \sum_{i=0}^{|\boldsymbol{\mathcal{S}}|-1} \begin{cases} 1 & \text{if } (\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{j}} = x_{\boldsymbol{j}} \\ 0 & \textit{otherwise} \end{cases} \tag{A.1}$$

Combining the definition of the conditional gpdf from Section 2.4.2 with equation A.1, we obtain:

$$P^{\mathcal{F}}(X_i|X_{\boldsymbol{\pi}_i})(x_{(i)\sqcup\boldsymbol{\pi}_i}) \;=\; \frac{\frac{1}{|\boldsymbol{\mathcal{S}}|}\sum_{j=0}^{|\boldsymbol{\mathcal{S}}|-1} \begin{cases} 1 & \text{if } (\boldsymbol{\mathcal{S}}_j)_{(i)\sqcup\boldsymbol{\pi}_i} = x_{(i)\sqcup\boldsymbol{\pi}_i} \\ 0 & \textit{otherwise} \end{cases}}{\frac{1}{|\boldsymbol{\mathcal{S}}|}\sum_{j=0}^{|\boldsymbol{\mathcal{S}}|-1} \begin{cases} 1 & \text{if } (\boldsymbol{\mathcal{S}}_j)_{\boldsymbol{\pi}_i} = x_{\boldsymbol{\pi}_i} \\ 0 & \textit{otherwise} \end{cases}} \tag{A.2}$$

Let $\phi^{\mathcal{F}}(\boldsymbol{\mathcal{S}}, \boldsymbol{j}, x_{\boldsymbol{j}}) \sqsubseteq \boldsymbol{\mathcal{S}}$ denote a filter on the sample collection $\boldsymbol{\mathcal{S}}$ such that its result contains exactly those samples in $\boldsymbol{\mathcal{S}}$ that are assigned the values $x_{\boldsymbol{j}}$ at positions $\boldsymbol{j}$:

$$\boldsymbol{\mathcal{S}}_k \in \phi^{\mathcal{F}}(\boldsymbol{\mathcal{S}}, \boldsymbol{j}, x_{\boldsymbol{j}}) \leftrightarrow (\boldsymbol{\mathcal{S}}_k)_{\boldsymbol{j}} = x_{\boldsymbol{j}} \tag{A.3}$$

We then obtain:

$$P^{\mathcal{F}}(X_i|X_{\boldsymbol{\pi}_i})(x_{(i)\sqcup\boldsymbol{\pi}_i}) = \tag{A.4}$$

$$\frac{1}{|\phi^{\mathcal{F}}(\boldsymbol{\mathcal{S}}, \boldsymbol{\pi}_i, x_{\boldsymbol{\pi}_i})|} \sum_{j=0}^{|\phi^{\mathcal{F}}(\boldsymbol{\mathcal{S}}, \boldsymbol{\pi}_i, x_{\boldsymbol{\pi}_i})|-1} \begin{cases} 1 & \text{if } (\phi^{\mathcal{F}}(\boldsymbol{\mathcal{S}}, \boldsymbol{\pi}_i, x_{\boldsymbol{\pi}_i})_j)_i = x_i \\ 0 & \textit{otherwise} \end{cases}$$

The univariate conditional gpdf for the discrete integer case is thus of a closed form, since it is again a frequency count, only now over a special selection of all available samples. For each $i \in \boldsymbol{\mathcal{L}}$ we can thus make a table of size $|\Omega^i|$ and compute the average frequency count for all but one of the values in Ω^i in the sample collection $\phi^{\mathcal{F}}(\boldsymbol{\mathcal{S}}, \boldsymbol{j}, x_{\boldsymbol{j}})$. The probability for the last value is given by $1-$ the sum over the probabilities for the other values in Ω^i.

A.2 Real–valued sample space

In this section, we present the maximum likelihood estimation for the univariate conditional form of the histogram gpdf, the normal gpdf, the normal kernels gpdf and the normal mixture gpdf. Since the normal kernels gpdf is an extreme case of the normal mixture gpdf, we will only derive the univariate conditional mixture gpdf.

A.2.1 Histogram

For the histogram gpdf, we have already indicated a bijective mapping to the discrete integer gpdf in Section 5.1.1. Therefore, an application of the maximum likelihood univariate conditional histogram gpdf can directly be obtained by using the result for the discrete integer univariate conditional gpdf in Section A.1.

A.2.2 Normal

Let $\hat{\boldsymbol{\Sigma}}$ and $\hat{\boldsymbol{\mu}}$ be the maximum likelihood estimates for the symmetric covariance matrix and mean vector of the normal gpdf over all random variables $\boldsymbol{\mathcal{Y}}$. Furthermore, let $\hat{\boldsymbol{\Sigma}}^{\boldsymbol{j}}$ denote the matrix that is obtained by taking the rows and columns indicated by the vector $\boldsymbol{j} \sqsubseteq \boldsymbol{\mathcal{L}}$ from $\hat{\boldsymbol{\Sigma}}$, that is, $(\hat{\boldsymbol{\Sigma}}^{\boldsymbol{j}})_{ik} = \hat{\boldsymbol{\Sigma}}_{\boldsymbol{j}_i \boldsymbol{j}_k}$. For convenience we shall allow ourselves to write $\hat{\boldsymbol{\Sigma}}^{\boldsymbol{j}}_{ij} = (\hat{\boldsymbol{\Sigma}}^{\boldsymbol{j}})_{ij}$. Furthermore, it is common practice to define $\hat{\boldsymbol{W}}^{\boldsymbol{j}}$ to be the inverse matrix of $\hat{\boldsymbol{\Sigma}}^{\boldsymbol{j}}$, that is, $\hat{\boldsymbol{W}}^{\boldsymbol{j}} = (\hat{\boldsymbol{\Sigma}}^{\boldsymbol{j}})^{-1}$. Matrix $\hat{\boldsymbol{W}}^{\boldsymbol{j}}$ is known as the *precision matrix*. The maximum likelihood multivariate normal gpdf for random variables $Y_{\boldsymbol{j}}$, is (Anderson, 1958; Tatsuoka, 1971):

$$P^{\mathcal{N}}(Y_{\boldsymbol{j}})(y_{\boldsymbol{j}}) = \frac{(2\pi)^{-\frac{|\boldsymbol{j}|}{2}}}{(\det \hat{\boldsymbol{\Sigma}}^{\boldsymbol{j}})^{\frac{1}{2}}} e^{-\frac{1}{2}(y_{\boldsymbol{j}} - \hat{\boldsymbol{\mu}}_{\boldsymbol{j}})^T \hat{\boldsymbol{W}}^{\boldsymbol{\alpha}} (y_{\boldsymbol{j}} - \hat{\boldsymbol{\mu}}_{\boldsymbol{j}})} = \tag{A.5}$$

Combining the definition of the conditional gpdf from Section 2.4.2 with equation A.5, we obtain:

$$P^{\mathcal{N}}(Y_i | Y_{\boldsymbol{\pi}_i})(y_{(i) \sqcup \boldsymbol{\pi}_i}) \; = \tag{A.6}$$

$$\frac{\frac{(2\pi)^{-\frac{|\boldsymbol{\pi}_i|+1}{2}}}{(\det \hat{\boldsymbol{\Sigma}}^{(i)\sqcup\boldsymbol{\pi}_i})^{\frac{1}{2}}} e^{-\frac{1}{2}(y_{(i)\sqcup\boldsymbol{\pi}_i} - \hat{\boldsymbol{\mu}}_{(i)\sqcup\boldsymbol{\pi}_i})^T \hat{\boldsymbol{W}}^{(i)\sqcup\boldsymbol{\pi}_i} (y_{(i)\sqcup\boldsymbol{\pi}_i} - \hat{\boldsymbol{\mu}}_{(i)\sqcup\boldsymbol{\pi}_i})}}{\frac{(2\pi)^{-\frac{|\boldsymbol{\pi}_i|}{2}}}{(\det \hat{\boldsymbol{\Sigma}}^{\boldsymbol{\pi}_i})^{\frac{1}{2}}} e^{-\frac{1}{2}(y_{\boldsymbol{\pi}_i} - \hat{\boldsymbol{\mu}}_{\boldsymbol{\pi}_i})^T \hat{\boldsymbol{W}}^{\boldsymbol{\pi}_i} (y_{\boldsymbol{\pi}_i} - \hat{\boldsymbol{\mu}}_{\boldsymbol{\pi}_i})}} = \alpha e^{\beta}$$

We may determine α and β separately to find the resulting expression for the univariate conditional normal gpdf. First, we determine α:

$$\alpha \; = \; \frac{(2\pi)^{-\frac{|\boldsymbol{\pi}_i|+1}{2}}}{(\det \hat{\boldsymbol{\Sigma}}^{(i)\sqcup\boldsymbol{\pi}_i})^{\frac{1}{2}}} \frac{(\det \hat{\boldsymbol{\Sigma}}^{\boldsymbol{\pi}_i})^{\frac{1}{2}}}{(2\pi)^{-\frac{|\boldsymbol{\pi}_i|}{2}}} \; = \; \frac{1}{\sqrt{\frac{\det \hat{\boldsymbol{\Sigma}}^{(i)\sqcup\boldsymbol{\pi}_i}}{\det \hat{\boldsymbol{\Sigma}}^{\boldsymbol{\pi}_i}}} \sqrt{2\pi}} \tag{A.7}$$

Second, we rewrite β as follows:

$$\beta = \begin{array}{l} \frac{1}{2}\Big[-(y_{(i)\sqcup\boldsymbol{\pi}_i} - \hat{\boldsymbol{\mu}}_{(i)\sqcup\boldsymbol{\pi}_i})^T \hat{\boldsymbol{W}}^{(i)\sqcup\boldsymbol{\pi}_i} (y_{(i)\sqcup\boldsymbol{\pi}_i} - \hat{\boldsymbol{\mu}}_{(i)\sqcup\boldsymbol{\pi}_i}) \\ \quad +(y_{\boldsymbol{\pi}_i} - \hat{\boldsymbol{\mu}}_{\boldsymbol{\pi}_i})^T \hat{\boldsymbol{W}}^{\boldsymbol{\pi}_i} (y_{\boldsymbol{\pi}_i} - \hat{\boldsymbol{\mu}}_{\boldsymbol{\pi}_i}) \Big] \end{array} = \tag{A.8}$$

$$\begin{array}{l} \frac{1}{2}\Bigg[-\left(\sum_{j=0}^{|\boldsymbol{\pi}_i|-1}\left(\sum_{k=0}^{|\boldsymbol{\pi}_i|-1}\left(y_{(\boldsymbol{\pi}_i)_k} - \hat{\boldsymbol{\mu}}_{(\boldsymbol{\pi}_i)_k}\right)\hat{\boldsymbol{W}}^{(i)\sqcup\boldsymbol{\pi}_i}_{(k+1)(j+1)}\right)\left(y_{(\boldsymbol{\pi}_i)_j} - \hat{\boldsymbol{\mu}}_{(\boldsymbol{\pi}_i)_j}\right)\right) \\ -(y_i - \hat{\boldsymbol{\mu}}_i)\sum_{k=0}^{|\boldsymbol{\pi}_i|-1}(y_{(\boldsymbol{\pi}_i)_k} - \hat{\boldsymbol{\mu}}_{(\boldsymbol{\pi}_i)_k})\hat{\boldsymbol{W}}^{(i)\sqcup\boldsymbol{\pi}_i}_{(k+1)0} \\ -(y_i - \hat{\boldsymbol{\mu}}_i)^2\hat{\boldsymbol{W}}^{(i)\sqcup\boldsymbol{\pi}_i}_{00} \\ -(y_i - \hat{\boldsymbol{\mu}}_i)\sum_{j=0}^{|\boldsymbol{\pi}_i|-1}(y_{(\boldsymbol{\pi}_i)_j} - \hat{\boldsymbol{\mu}}_{(\boldsymbol{\pi}_i)_j})\hat{\boldsymbol{W}}^{(i)\sqcup\boldsymbol{\pi}_i}_{0(j+1)} \\ +\left(\sum_{j=0}^{|\boldsymbol{\pi}_i|-1}\left(\sum_{k=0}^{|\boldsymbol{\pi}_i|-1}\left(y_{(\boldsymbol{\pi}_i)_k} - \hat{\boldsymbol{\mu}}_{(\boldsymbol{\pi}_i)_k}\right)\left(\hat{\boldsymbol{W}}^{\boldsymbol{\pi}_i}_{kj}\right)\right)\left(y_{(\boldsymbol{\pi}_i)_j} - \hat{\boldsymbol{\mu}}_{(\boldsymbol{\pi}_i)_j}\right)\right) \Bigg] \end{array} =$$

$$\begin{array}{l} \frac{1}{2}\Bigg[-\left(\sum_{j=0}^{|\boldsymbol{\pi}_i|-1}\left(\sum_{k=0}^{|\boldsymbol{\pi}_i|-1}\left(y_{(\boldsymbol{\pi}_i)_k} - \hat{\boldsymbol{\mu}}_{(\boldsymbol{\pi}_i)_k}\right)\hat{\boldsymbol{W}}^{(i)\sqcup\boldsymbol{\pi}_i}_{(k+1)(j+1)}\right)\left(y_{(\boldsymbol{\pi}_i)_j} - \hat{\boldsymbol{\mu}}_{(\boldsymbol{\pi}_i)_j}\right)\right) \\ -2y_i\sum_{k=0}^{|\boldsymbol{\pi}_i|-1}(y_{(\boldsymbol{\pi}_i)_k} - \hat{\boldsymbol{\mu}}_{(\boldsymbol{\pi}_i)_k})\hat{\boldsymbol{W}}^{(i)\sqcup\boldsymbol{\pi}_i}_{(k+1)0} \\ +2\hat{\boldsymbol{\mu}}_i\sum_{k=0}^{|\boldsymbol{\pi}_i|-1}(y_{(\boldsymbol{\pi}_i)_k} - \hat{\boldsymbol{\mu}}_{(\boldsymbol{\pi}_i)_k})\hat{\boldsymbol{W}}^{(i)\sqcup\boldsymbol{\pi}_i}_{(k+1)0} \\ -y_i^2\hat{\boldsymbol{W}}^{(i)\sqcup\boldsymbol{\pi}_i}_{00} \\ +2y_i\hat{\boldsymbol{\mu}}_i\hat{\boldsymbol{W}}^{(i)\sqcup\boldsymbol{\pi}_i}_{00} \\ -\hat{\boldsymbol{\mu}}_i^2\hat{\boldsymbol{W}}^{(i)\sqcup\boldsymbol{\pi}_i}_{00} \\ +\left(\sum_{j=0}^{|\boldsymbol{\pi}_i|-1}\left(\sum_{k=0}^{|\boldsymbol{\pi}_i|-1}\left(y_{(\boldsymbol{\pi}_i)_k} - \hat{\boldsymbol{\mu}}_{(\boldsymbol{\pi}_i)_k}\right)\left(\hat{\boldsymbol{W}}^{\boldsymbol{\pi}_i}_{kj}\right)\right)\left(y_{(\boldsymbol{\pi}_i)_j} - \hat{\boldsymbol{\mu}}_{(\boldsymbol{\pi}_i)_j}\right)\right) \Bigg] \end{array} =$$

$$\underset{\text{def}}{=} \left\{ \begin{array}{l} a_0 = -\left(\sum_{j=0}^{|\boldsymbol{\pi}_i|-1}\left(\sum_{k=0}^{|\boldsymbol{\pi}_i|-1}\left(y_{(\boldsymbol{\pi}_i)_k} - \hat{\boldsymbol{\mu}}_{(\boldsymbol{\pi}_i)_k}\right)\hat{\boldsymbol{W}}^{(i)\sqcup\boldsymbol{\pi}_i}_{(k+1)(j+1)}\right)\left(y_{(\boldsymbol{\pi}_i)_j} - \hat{\boldsymbol{\mu}}_{(\boldsymbol{\pi}_i)_j}\right)\right) \\ a_1 = -2\sum_{k=0}^{|\boldsymbol{\pi}_i|-1}(y_{(\boldsymbol{\pi}_i)_k} - \hat{\boldsymbol{\mu}}_{(\boldsymbol{\pi}_i)_k})\hat{\boldsymbol{W}}^{(i)\sqcup\boldsymbol{\pi}_i}_{(k+1)0} \\ a_2 = 2\hat{\boldsymbol{\mu}}_i\sum_{k=0}^{|\boldsymbol{\pi}_i|-1}(y_{(\boldsymbol{\pi}_i)_k} - \hat{\boldsymbol{\mu}}_{(\boldsymbol{\pi}_i)_k})\hat{\boldsymbol{W}}^{(i)\sqcup\boldsymbol{\pi}_i}_{(k+1)0} \\ a_3 = -\hat{\boldsymbol{W}}^{(i)\sqcup\boldsymbol{\pi}_i}_{00} \\ a_4 = 2\hat{\boldsymbol{\mu}}_i\hat{\boldsymbol{W}}^{(i)\sqcup\boldsymbol{\pi}_i}_{00} \\ a_5 = -\hat{\boldsymbol{\mu}}_i^2\hat{\boldsymbol{W}}^{(i)\sqcup\boldsymbol{\pi}_i}_{00} \\ a_6 = \left(\sum_{j=0}^{|\boldsymbol{\pi}_i|-1}\left(\sum_{k=0}^{|\boldsymbol{\pi}_i|-1}\left(y_{(\boldsymbol{\pi}_i)_k} - \hat{\boldsymbol{\mu}}_{(\boldsymbol{\pi}_i)_k}\right)\left(\hat{\boldsymbol{W}}^{\boldsymbol{\pi}_i}_{kj}\right)\right)\left(y_{(\boldsymbol{\pi}_i)_j} - \hat{\boldsymbol{\mu}}_{(\boldsymbol{\pi}_i)_j}\right)\right) \\ \gamma = -a_3 \\ \delta = -a_1 - a_4 \\ \eta = -a_0 - a_2 - a_5 - a_6 \end{array} \right\}$$

$$\frac{1}{2}(a_0+a_1y_i+a_2+a_3y_i^2+a_4y_i+a_5+a_6) \;=\; -\frac{1}{2}(\gamma y_i^2+\delta y_i+\eta) \;=\; \frac{-(y_i^2+\frac{\delta}{\gamma}y_i+\frac{\eta}{\gamma})}{2\frac{1}{\gamma}}$$

So that when we realize that $(y_i-\breve{\mu}_i)^2 = y_i - 2\breve{\mu}_i y_i + \breve{\mu}_i^2$, we find that if we have $(\frac{\delta}{-2\gamma})^2 = \frac{\eta}{\gamma}$ (which can be checked to be the case), we have that $\breve{\mu}_i = \sqrt{\frac{\eta}{\gamma}} = \frac{\delta}{-2\gamma}$. The expression for the univariate conditional normal gpdf we have derived so far is now given by:

$$P^{\mathcal{N}}(Y_i|Y_{\boldsymbol{\pi}_i})(y_{(i)\sqcup\boldsymbol{\pi}_i}) = \frac{1}{\sqrt{\frac{\det\,\hat{\boldsymbol{\Sigma}}^{(i)\sqcup\boldsymbol{\pi}_i}}{\det\,\hat{\boldsymbol{\Sigma}}^{\boldsymbol{\pi}_i}}}\sqrt{2\pi}} e^{\frac{(y_i-\breve{\mu}_i)^2}{2\frac{1}{\gamma}}} \tag{A.9}$$

The conditional gpdf is of the closed form $1/(\breve{\sigma}_i\sqrt{2\pi})\exp(-(y_i-\breve{\mu}_i)^2/2\breve{\sigma}_i^2)$ if we have that $\breve{\sigma}_i = \sqrt{\gamma^{-1}} = (\det\,\hat{\boldsymbol{\Sigma}}^{(i)\sqcup\boldsymbol{\pi}_i})/(\det\,\hat{\boldsymbol{\Sigma}}^{\boldsymbol{\pi}_i})$. The latter is indeed the case since $\sqrt{\gamma^{-1}} = \sqrt{1/\hat{\boldsymbol{W}}_{00}^{(i)\sqcup\boldsymbol{\pi}_i}}$, which can be checked using Cramer's rule from linear algebra to be equal to the fraction of the determinants. As a result, the univariate conditional normal gpdf is a one–dimensional normal gpdf that is defined as follows:

$$P^{\mathcal{N}}(Y_i|Y_{\boldsymbol{\pi}_i})(y_{(i)\sqcup\boldsymbol{\pi}_i}) = \frac{1}{(\breve{\sigma}_i\sqrt{2\pi})} e^{\frac{-(y_i-\breve{\mu}_i)^2}{2\breve{\sigma}_i^2}} \tag{A.10}$$

$$\text{where } \begin{cases} \breve{\sigma}_i \;=\; \frac{1}{\sqrt{\hat{\boldsymbol{W}}_{00}^{(i)\sqcup\boldsymbol{\pi}_i}}} \\ \breve{\mu}_i \;=\; \frac{\hat{\boldsymbol{\mu}}_i\hat{\boldsymbol{W}}_{00}^{(i)\sqcup\boldsymbol{\pi}_i} - \sum_{j=0}^{|\boldsymbol{\pi}_i|-1}(y_{(\boldsymbol{\pi}_i)_j}-\hat{\boldsymbol{\mu}}_{(\boldsymbol{\pi}_i)_j})\hat{\boldsymbol{W}}_{(j+1)0}^{(i)\sqcup\boldsymbol{\pi}_i}}{\hat{\boldsymbol{W}}_{00}^{(i)\sqcup\boldsymbol{\pi}_i}} \end{cases}$$

A.2.3 Normal kernels and normal mixture

The normal kernels gpdf can be seen as an extreme case of the normal mixture gpdf. We can represent either gpdf for a selection of the random variables $Y_{\boldsymbol{j}}$ and w mixture gpdf components as follows:

$$P_{\boldsymbol{\theta}}^{\mathcal{N}_M}(Z_{\boldsymbol{j}}) = \sum_{i=0}^{w-1}\beta_i P_{\boldsymbol{\theta}_i}^{\mathcal{N}}(Y_{\boldsymbol{j}}) \tag{A.11}$$

In the case of the normal kernels gpdf, we have that $w = |\boldsymbol{\mathcal{S}}|$ and the covariance matrices are only to be adapted to the sample range in each dimension. For the normal mixture gpdf, the covariance matrices are estimated from the data and the number of mixture components is usually quite small compared to the number of sample points. To obtain an expression for the univariate conditional mixture gpdf, we combine the definition of the conditional gpdf from Section 2.4.2 with equation A.11 and obtain:

$$P_{\boldsymbol{\theta}}^{\mathcal{N}_M}(Y_i|Y_{\boldsymbol{\pi}_i})(y_{(i)\sqcup\boldsymbol{\pi}_i}) \;=\; \frac{\sum_{j=0}^{w-1}\beta_j P_{\boldsymbol{\theta}^j}^{\mathcal{N}}(Y_{(i)\sqcup\boldsymbol{\pi}_i})}{\sum_{j=0}^{w-1}\beta_j P_{\boldsymbol{\theta}^j}^{\mathcal{N}}(Y_{\boldsymbol{\pi}_i})} \tag{A.12}$$

The denominator in equation A.12 is constant given the values for random variables $Y_{\boldsymbol{\pi}_i}$. Again using the definition of the conditional gpdf from Section 2.4.2, we may continue to write:

$$P_{\boldsymbol{\theta}}^{\mathcal{N}_M}(Y_i|Y_{\boldsymbol{\pi}_i})(y_{(i)\sqcup\boldsymbol{\pi}_i}) \;=\; \sum_{j=0}^{w-1} \frac{\beta_j P_{\boldsymbol{\theta}^j}^{\mathcal{N}}(Y_{\boldsymbol{\pi}_i})}{P_{\boldsymbol{\theta}}^{\mathcal{N}_M}(Y_{\boldsymbol{\pi}_i})(y_{\boldsymbol{\pi}_i})} P_{\boldsymbol{\theta}^j}^{\mathcal{N}}(Y_i|Y_{\boldsymbol{\pi}_i}) \tag{A.13}$$

The resulting expression is thus a weighted sum of univariate conditional gpdfs. We can now use the univariate conditional result in equation A.10 for the normal gpdf to complete the derivation:

$$P_{\boldsymbol{\theta}}^{\mathcal{N}_M}(Y_i|Y_{\boldsymbol{\pi}_i})(y_{(i)\sqcup\boldsymbol{\pi}_i}) \;=\; \sum_{j=0}^{w-1} \breve{\beta}_i^j \frac{1}{(\breve{\sigma}_i^j\sqrt{2\pi})} e^{\frac{-(y_j-\breve{\mu}_i^j)^2}{2(\breve{\sigma}_i^j)^2}} \tag{A.14}$$

where
$$\begin{cases} \breve{\beta}_i^j \;=\; \dfrac{\beta_j \frac{(2\pi)^{-\frac{|\boldsymbol{\pi}_i|}{2}}}{(\det\,\boldsymbol{\Sigma}^{j,\boldsymbol{\pi}_i})^{\frac{1}{2}}} e^{-\frac{1}{2}(y_{\boldsymbol{\pi}_i}-\boldsymbol{\mu}^j_{\boldsymbol{\pi}_i})^T(\boldsymbol{\Sigma}^{j,\boldsymbol{\pi}_i})^{-1}(y_{\boldsymbol{\pi}_i}-\boldsymbol{\mu}^j_{\boldsymbol{\pi}_i})}}{\sum_{q=0}^{w-1}\beta_q \frac{(2\pi)^{-\frac{|\boldsymbol{\pi}_i|}{2}}}{(\det\,\boldsymbol{\Sigma}^{q,\boldsymbol{\pi}_i})^{\frac{1}{2}}} e^{-\frac{1}{2}(y_{\boldsymbol{\pi}_i}-\boldsymbol{\mu}^q_{\boldsymbol{\pi}_i})^T(\boldsymbol{\Sigma}^{q,\boldsymbol{\pi}_i})^{-1}(y_{\boldsymbol{\pi}_i}-\boldsymbol{\mu}^q_{\boldsymbol{\pi}_i})}} \\ \breve{\sigma}_i^j \;=\; \dfrac{1}{\left(\left(\boldsymbol{\Sigma}^{j,(i)\sqcup\boldsymbol{\pi}_i}\right)^{-1}\right)_{00}} \\ \breve{\mu}_i^j \;=\; \dfrac{\boldsymbol{\mu}_0^j\left(\left(\boldsymbol{\Sigma}^{j,(i)\sqcup\boldsymbol{\pi}_i}\right)^{-1}\right)_{00} - \sum_{q=0}^{|\boldsymbol{\pi}_i|-1}(y_{(\boldsymbol{\pi}_i)_q}-\boldsymbol{\mu}^j_{(\boldsymbol{\pi}_i)_q})\left(\left(\boldsymbol{\Sigma}^{j,(i)\sqcup\boldsymbol{\pi}_i}\right)^{-1}\right)_{q0}}{\left(\left(\boldsymbol{\Sigma}^{j,(i)\sqcup\boldsymbol{\pi}_i}\right)^{-1}\right)_{00}} \end{cases}$$

The univariate conditional normal mixture gpdf is thus of a closed form as it contains w mixture components, each of which is multiplied by a one–dimensional normal gpdf, making the result a normal mixture probability distribution over random variable Y_i. Computing the mixture components is however quite a computationally intensive task. Therefore, using normal mixture gpdfs in the estimation of Bayesian factorizations from data will require a significantly larger amount of time than using for instance only the normal gpdf.

'An idea is never given to you without you being given the power to make it reality. You must, nevertheless, suffer for it.'

Richard Bach

Tabulated experimental results

This appendix contains tabulated results for the three main different types of optimization problem that were tackled with IDEAs in chapters 5, 6 and 7. The details of the experimental setup for each main type of optimization problem are given in the respective chapters.

This appendix is organized as follows. In Section B.1 we present tabulated results of the numerical optimization experiments described in Chapter 5. In Section B.2 we present tabulated results of the permutation optimization experiments described in Chapter 6. Finally, in Section B.3 we present tabulated results of the multi–objective optimization experiments described in Chapter 7.

B.1 Numerical optimization

$l = 5$						
EA	**Griewank**			**Michalewicz**		
	n	$\overline{fitness}$	$\overline{eval.}$	n	$\overline{fitness}$	$\overline{eval.}$
sGA^{1X}	30100	0.001230	10^6	8050	−4.687658	210382
sGA^{UX}	16000	0.006009	10^6	7300	−4.687658	286361
$\mathrm{IDEA}^{\mathrm{Binary\ Tree\ Bayesian}}$	15600	0.000000	675228	1300	−4.687658	27810
$\mathrm{IDEA}^{\mathrm{Normal\ Univariate}}$	100	0.004310	541709	16600	−4.664754	1004385
$\mathrm{IDEA}^{\mathrm{Normal\ Bayesian}}$	140	0.006942	661559	23000	−4.663060	1005161
$\mathrm{IDEA}^{\mathrm{Normal\ Joint}}$	150	0.005940	472115	15400	−4.662717	1007252
$\mathrm{IDEA}^{\mathrm{Normal\ Univariate\ Mixture}}$	380	0.024612	985262	1300	−4.687658	73451
$\mathrm{IDEA}^{\mathrm{Normal\ Bayesian\ Mixture}}$	450	0.022360	736983	1750	−4.687658	91493
$\mathrm{IDEA}^{\mathrm{Normal\ Joint\ Mixture}}$	710	0.021228	962298	2000	−4.687658	138457
$\mathrm{IDEA}^{\mathrm{Normal\ Kernels},\alpha=1}_{\mathrm{Univariate}}$	4200	0.000000	250068	4500	−4.687658	1000216
$\mathrm{IDEA}^{\mathrm{Normal\ Kernels},\alpha=10}_{\mathrm{Univariate}}$	2800	0.000000	10^6	30000	−4.687646	10^6
$\mathrm{IDEA}^{\mathrm{Normal\ Kernels},\alpha=1}_{\mathrm{Joint}}$	10^5	2.461451	10^6	5400	−4.679305	1003584
$\mathrm{IDEA}^{\mathrm{Normal\ Kernels},\alpha=10}_{\mathrm{Joint}}$	70	0.013574	10^6	22000	−4.687638	1007664
$\mathrm{IDEA}^{\mathrm{Normal\ Mixture}}_{\mathrm{Univariate}}$	2400	0.000000	303803	2100	−4.687658	44906
$\mathrm{IDEA}^{\mathrm{Normal\ Mixture}}_{\mathrm{Joint}}$	1100	0.000221	830311	60	−4.023693	1000025
$\mathrm{IDEA}^{\mathrm{Histogram}}_{\mathrm{Univariate}}$	250	0.000000	237603	2550	−4.687658	192045
$\mathrm{ES}^{1\ \mathrm{Variance}}$	310	0.000000	105555	**60**	**−4.687658**	**11568**
$\mathrm{ES}^{l\ \mathrm{Variances}}$	350	0.000000	440615	80	−4.687658	23096
$\mathrm{ES}^{\mathrm{Covariance\ Matrix}}$	570	0.000000	803358	80	−4.687658	24440
$\mathbb{R}\mathrm{CG}$	—	0.061579	709450	—	−4.624676	270707
$\mathrm{GLIDE}^{\tau_G=0.05,\mathrm{Normal}}_{\mathrm{Univariate}}$	**50**	**0.000000**	**26835**	70	−4.687658	58094
$\mathrm{GLIDE}^{\tau_G=0.05,\mathrm{Normal}}_{\mathrm{Bayesian}}$	60	0.000000	36239	120	−4.687658	118459
$\mathrm{GLIDE}^{\tau_G=0.05,\mathrm{Normal}}_{\mathrm{Joint}}$	70	0.000000	38627	100	−4.687658	80662
$\mathrm{GLIDE}^{\tau_G=0.05,\mathrm{Normal}}_{\mathrm{Univariate\ Mixture}}$	100	0.000000	53412	130	−4.687658	44828
$\mathrm{GLIDE}^{\tau_G=0.05,\mathrm{Normal}}_{\mathrm{Bayesian\ Mixture}}$	100	0.000000	50375	350	−4.687658	229100
$\mathrm{GLIDE}^{\tau_G=0.05,\mathrm{Normal}}_{\mathrm{Joint\ Mixture}}$	140	0.000000	77763	300	−4.687658	165770
$\mathrm{GLIDE}^{\tau_G=0.25,\mathrm{Normal}}_{\mathrm{Univariate}}$	50	0.000000	39294	70	−4.687658	110792
$\mathrm{GLIDE}^{\tau_G=0.25,\mathrm{Normal}}_{\mathrm{Bayesian}}$	50	0.000000	37535	90	−4.687658	133713
$\mathrm{GLIDE}^{\tau_G=0.25,\mathrm{Normal}}_{\mathrm{Joint}}$	70	0.000000	52878	80	−4.687658	124765
$\mathrm{GLIDE}^{\tau_G=0.25,\mathrm{Normal}}_{\mathrm{Univariate\ Mixture}}$	90	0.000000	77981	100	−4.687658	88270
$\mathrm{GLIDE}^{\tau_G=0.25,\mathrm{Normal}}_{\mathrm{Bayesian\ Mixture}}$	100	0.000000	80981	150	−4.687658	147420
$\mathrm{GLIDE}^{\tau_G=0.25,\mathrm{Normal}}_{\mathrm{Joint\ Mixture}}$	140	0.000000	119489	140	−4.687658	121529

Figure B.1: Results on Griewank's function and Michalewicz's function in $l = 5$ dimensions for all tested algorithms. The best results are indicated in bold.

$l = 5$						
EA	**Rosenbrock**			**Summation Cancellation**		
	n	$\overline{fitness}$	$\overline{eval.}$	n	$\overline{fitness}$	$\overline{eval.}$
sGA^{1X}	31000	0.165161	10^6	24000	3406056.258683	10^6
sGA^{UX}	47000	0.462513	10^6	3250	9991625.116575	203766
$\text{IDEA}^{\text{Binary}}_{\text{Tree Bayesian}}$	26000	1.620513	10^6	4000	9991625.116575	231161
$\text{IDEA}^{\text{Normal}}_{\text{Univariate}}$	100000	1.708365	10^6	8500	9999999.996842	10^6
$\text{IDEA}^{\text{Normal}}_{\text{Bayesian}}$	100000	1.380745	10^6	300	10000000.000000	23890
$\text{IDEA}^{\text{Normal}}_{\text{Joint}}$	100000	1.561144	10^6	**150**	**10000000.000000**	**11068**
$\text{IDEA}^{\text{Normal}}_{\text{Univariate Mixture}}$	42000	0.259529	10^6	1050	10000000.000000	144938
$\text{IDEA}^{\text{Normal}}_{\text{Bayesian Mixture}}$	46000	0.025127	10^6	600	10000000.000000	56340
$\text{IDEA}^{\text{Normal}}_{\text{Joint Mixture}}$	26700	0.003951	834151	600	10000000.000000	51625
$\text{IDEA}^{\text{Normal Kernels},\alpha=1}_{\text{Univariate}}$	280	0.004286	10^6	13000	667604.825340	10^6
$\text{IDEA}^{\text{Normal Kernels},\alpha=10}_{\text{Univariate}}$	480	0.103085	10^6	29000	70952.444693	10^6
$\text{IDEA}^{\text{Normal Kernels},\alpha=1}_{\text{Joint}}$	230	0.002065	10^6	6250	371520.271624	10^6
$\text{IDEA}^{\text{Normal Kernels},\alpha=10}_{\text{Joint}}$	7500	0.003806	10^6	20000	63731.685431	10^6
$\text{IDEA}^{\text{Normal Mixture}}_{\text{Univariate}}$	6250	0.044026	623266	4500	692.547496	10^6
$\text{IDEA}^{\text{Normal Mixture}}_{\text{Joint}}$	250	0.061835	530872	100	6000065.241993	10^6
$\text{IDEA}^{\text{Histogram}}_{\text{Univariate}}$	150	0.593096	6139	400	10000000.000000	64356
$\text{ES}^{\text{1 Variance}}$	30	0.000202	694962	30	10000000.000000	37578
$\text{ES}^{l\text{ Variances}}$	90	0.000028	982575	30	10000000.000000	62715
$\text{ES}^{\text{Covariance Matrix}}$	60	0.000001	140466	20	10000000.000000	42706
$\mathbb{R}$CG	—	0.000000	8474	—	3855170.945988	693250
$\text{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Univariate}}$	30	0.000000	8729	430	9999999.744276	727435
$\text{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Bayesian}}$	**30**	**0.000000**	**4438**	190	10000000.000000	199856
$\text{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Joint}}$	40	0.000000	7477	140	10000000.000000	143155
$\text{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Univariate Mixture}}$	50	0.000000	10911	410	9999999.989328	671733
$\text{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Bayesian Mixture}}$	50	0.000000	6787	400	10000000.000000	485244
$\text{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Joint Mixture}}$	40	0.000000	5937	440	10000000.000000	500081
$\text{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Univariate}}$	20	0.000000	7856	290	9999997.376604	10^6
$\text{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Bayesian}}$	20	0.000000	7332	210	10000000.000000	751962
$\text{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Joint}}$	20	0.000000	7317	140	10000000.000000	477266
$\text{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Univariate Mixture}}$	20	0.000000	6615	260	9999990.727742	10^6
$\text{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Bayesian Mixture}}$	20	0.000000	7249	320	9999998.908201	10^6
$\text{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Joint Mixture}}$	20	0.000000	6694	330	9999998.345713	10^6

Figure B.2: Results on Rosenbrock's function and the summation cancellation function in $l = 5$ dimensions for all tested algorithms. The best results are indicated in bold.

$l = 25$						
EA	**Griewank**			**Michalewicz**		
	n	$\overline{fitness}$	$\overline{eval.}$	n	$\overline{fitness}$	$\overline{eval.}$
sGA^{1X}	20000	1.026956	10^6	20000	−24.597163	10^6
sGA^{UX}	6500	0.000126	10^6	9000	−24.070203	10^6
$\mathrm{IDEA}^{\text{Binary}}_{\text{Tree Bayesian}}$	10500	0.000000	988183	17000	−24.519908	10^6
$\mathrm{IDEA}^{\text{Normal}}_{\text{Univariate}}$	140	0.000000	8139	80	−23.731830	298732
$\mathrm{IDEA}^{\text{Normal}}_{\text{Bayesian}}$	170	0.000000	9746	120	−23.816976	433399
$\mathrm{IDEA}^{\text{Normal}}_{\text{Joint}}$	850	0.000000	46682	280	−20.936572	10^6
$\mathrm{IDEA}^{\text{Normal}}_{\text{Univariate Mixture}}$	190	0.000000	10736	170	−19.029565	247190
$\mathrm{IDEA}^{\text{Normal}}_{\text{Bayesian Mixture}}$	260	0.000000	14351	500	−17.315832	588250
$\mathrm{IDEA}^{\text{Normal}}_{\text{Joint Mixture}}$	1250	0.000000	69140	450	−14.818623	10^6
$\mathrm{IDEA}^{\text{Normal Kernels},\alpha=1}_{\text{Univariate}}$	9500	0.000003	10^6	60000	−21.514443	10^6
$\mathrm{IDEA}^{\text{Normal Kernels},\alpha=10}_{\text{Univariate}}$	12000	0.000183	10^6	100000	−15.874657	10^6
$\mathrm{IDEA}^{\text{Normal Kernels},\alpha=1}_{\text{Joint}}$	30	0.032261	10^6	7500	−18.007642	10^6
$\mathrm{IDEA}^{\text{Normal Kernels},\alpha=10}_{\text{Joint}}$	420	0.032997	10^6	41000	−13.695497	10^6
$\mathrm{IDEA}^{\text{Normal Mixture}}_{\text{Univariate}}$	450	0.000000	28100	170	−23.137773	620678
$\mathrm{IDEA}^{\text{Normal Mixture}}_{\text{Joint}}$	300	2.766037	10^6	15000	−10.287409	10^6
$\mathrm{IDEA}^{\text{Histogram}}_{\text{Univariate}}$	210	0.000000	15084	160	−23.569571	464217
$\mathrm{ES}^{1\text{ Variance}}$	50	0.000000	33440	**1100**	**−24.633195**	**569360**
$\mathrm{ES}^{l\text{ Variances}}$	80	0.000000	129552	950	−24.621152	10^6
$\mathrm{ES}^{\text{Covariance Matrix}}$	90	0.000000	146565	1000	−24.621624	10^6
$\mathbb{R}$CG	—	**0.000000**	**7615**	—	−14.186312	441433
$\mathrm{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Univariate}}$	20	0.000000	10418	20	−19.064770	666077
$\mathrm{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Bayesian}}$	30	0.000000	13235	30	−20.121662	629649
$\mathrm{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Joint}}$	60	0.000000	55951	100	−19.164625	10^6
$\mathrm{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Univariate Mixture}}$	50	0.000000	16247	150	−17.971096	927916
$\mathrm{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Bayesian Mixture}}$	50	0.000000	10671	140	−18.246196	731313
$\mathrm{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Joint Mixture}}$	60	0.000000	37417	650	−16.768625	10^6
$\mathrm{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Univariate}}$	30	0.000000	25262	20	−18.188244	925082
$\mathrm{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Bayesian}}$	30	0.000000	24691	20	−19.814443	368731
$\mathrm{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Joint}}$	50	0.000000	47220	90	−18.932534	10^6
$\mathrm{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Univariate Mixture}}$	30	0.000000	19295	90	−17.437226	770746
$\mathrm{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Bayesian Mixture}}$	30	0.000000	17815	90	−19.699357	688178
$\mathrm{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Joint Mixture}}$	60	0.000000	71161	350	−16.337914	10^6

Figure B.3: Results on Griewank's function and Michalewicz's function in $l = 25$ dimensions for all tested algorithms. The best results are indicated in bold.

$l = 25$						
EA	**Rosenbrock**			**Summation Cancellation**		
	n	$\overline{fitness}$	$\overline{eval.}$	n	$\overline{fitness}$	$\overline{eval.}$
sGA^{1X}	20000	23.023205	10^6	10000	101.715586	873223
sGA^{UX}	15000	23.038780	10^6	2600	2256.327854	907637
$\text{IDEA}^{\text{Binary}}_{\text{Tree Bayesian}}$	15000	22.214878	10^6	5000	15768.927066	10^6
$\text{IDEA}^{\text{Normal}}_{\text{Univariate}}$	15000	22.908086	10^6	2750	7388.304274	10^6
$\text{IDEA}^{\text{Normal}}_{\text{Bayesian}}$	16000	22.247199	10^6	13000	7888281.961947	10^6
$\text{IDEA}^{\text{Normal}}_{\text{Joint}}$	20000	22.231879	10^6	**1150**	**10000000.000000**	**223767**
$\text{IDEA}^{\text{Normal}}_{\text{Univariate Mixture}}$	11000	22.903072	958223	2300	7768.135946	10^6
$\text{IDEA}^{\text{Normal}}_{\text{Bayesian Mixture}}$	20000	22.272257	10^6	12000	7090775.716263	10^6
$\text{IDEA}^{\text{Normal}}_{\text{Joint Mixture}}$	1250	21.073810	10^6	2000	10000000.000000	393720
$\text{IDEA}^{\text{Normal Kernels},\alpha=1}_{\text{Univariate}}$	4300	17.178668	10^6	2400	388.183912	10^6
$\text{IDEA}^{\text{Normal Kernels},\alpha=10}_{\text{Univariate}}$	14000	21.505943	10^6	4500	52.505755	10^6
$\text{IDEA}^{\text{Normal Kernels},\alpha=1}_{\text{Joint}}$	1250	12.901411	10^6	2200	390.412829	10^6
$\text{IDEA}^{\text{Normal Kernels},\alpha=10}_{\text{Joint}}$	6250	23.102531	10^6	8500	75.580032	10^6
$\text{IDEA}^{\text{Normal Mixture}}_{\text{Univariate}}$	3000	1.047507	10^6	50	4.963893	10^6
$\text{IDEA}^{\text{Normal Mixture}}_{\text{Joint}}$	900	0.727705	10^6	10000	15.799390	10^6
$\text{IDEA}^{\text{Histogram}}_{\text{Univariate}}$	11000	22.412371	10^6	900	223193.316574	10^6
$\text{ES}^{\text{1 Variance}}$	10	1.460191	10^6	400	165344.032102	10^6
$\text{ES}^{l\text{ Variances}}$	40	0.004546	957920	70	4181755.798606	10^6
$\text{ES}^{\text{Covariance Matrix}}$	20	0.000601	803634	70	7593966.192454	10^6
$\mathbb{R}$CG	—	**0.000000**	**56253**	—	235.136680	375119
$\text{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Univariate}}$	30	0.000000	88192	170	4627.027957	10^6
$\text{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Bayesian}}$	40	0.000000	60125	170	6485926.439209	10^6
$\text{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Joint}}$	20	0.000000	94064	420	376076.190825	10^6
$\text{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Univariate Mixture}}$	60	0.000000	98870	170	3860.382808	10^6
$\text{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Bayesian Mixture}}$	80	0.000000	107679	220	383909.826545	10^6
$\text{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Joint Mixture}}$	60	0.000000	133637	520	36033.026775	10^6
$\text{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Univariate}}$	20	0.000000	70851	110	7132.729447	10^6
$\text{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Bayesian}}$	20	0.000000	69090	90	295293.443831	10^6
$\text{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Joint}}$	20	0.000000	116603	170	13813.983046	10^6
$\text{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Univariate Mixture}}$	40	0.000000	149490	120	3581.259446	10^6
$\text{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Bayesian Mixture}}$	40	0.000000	143372	140	10859.179775	10^6
$\text{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Joint Mixture}}$	60	0.000000	263868	290	852.931234	10^6

Figure B.4: Results on Rosenbrock's function and the summation cancellation function in $l = 25$ dimensions for all tested algorithms. The best results are indicated in bold.

$l = 100$						
EA	**Griewank**			**Michalewicz**		
	n	$\overline{fitness}$	$\overline{eval.}$	n	$\overline{fitness}$	$\overline{eval.}$
$\mathrm{sGA}^{\mathrm{1X}}$	14000	21.555610	10^6	14000	−93.933613	10^6
$\mathrm{sGA}^{\mathrm{UX}}$	6250	1.258145	10^6	3000	−92.479503	10^6
$\mathrm{IDEA}^{\mathrm{Binary}}_{\mathrm{Tree\ Bayesian}}$	*Time limit exceeded*			*Time limit exceeded*		
$\mathrm{IDEA}^{\mathrm{Normal}}_{\mathrm{Univariate}}$	**260**	**0.000000**	**30492**	40	−68.886636	70904
$\mathrm{IDEA}^{\mathrm{Normal}}_{\mathrm{Bayesian}}$	350	0.000000	40054	70	−65.147652	507010
$\mathrm{IDEA}^{\mathrm{Normal}}_{\mathrm{Joint}}$	7500	0.000000	842409	2000	−32.605695	10^6
$\mathrm{IDEA}^{\mathrm{Normal}}_{\mathrm{Univariate\ Mixture}}$	300	0.000000	34524	110	−49.335770	10^6
$\mathrm{IDEA}^{\mathrm{Normal}}_{\mathrm{Bayesian\ Mixture}}$	450	0.000000	49999	200	−41.908569	780142
$\mathrm{IDEA}^{\mathrm{Normal}}_{\mathrm{Joint\ Mixture}}$	7000	0.000000	783318	4250	−30.371235	10^6
$\mathrm{IDEA}^{\mathrm{Normal\ Kernels},\alpha=1}_{\mathrm{Univariate}}$	11000	0.006834	10^6	44000	−38.532014	10^6
$\mathrm{IDEA}^{\mathrm{Normal\ Kernels},\alpha=10}_{\mathrm{Univariate}}$	8000	0.008543	10^6	9000	−28.773093	10^6
$\mathrm{IDEA}^{\mathrm{Normal\ Kernels},\alpha=1}_{\mathrm{Joint}}$	100	0.062471	10^6	25000	−29.805404	10^6
$\mathrm{IDEA}^{\mathrm{Normal\ Kernels},\alpha=10}_{\mathrm{Joint}}$	480	0.651136	10^6	25000	−27.996997	10^6
$\mathrm{IDEA}^{\mathrm{Normal\ Mixture}}_{\mathrm{Univariate}}$	650	0.000000	81818	130	−87.313509	299682
$\mathrm{IDEA}^{\mathrm{Normal\ Mixture}}_{\mathrm{Joint}}$	6250	210.992172	10^6	7500	−22.263911	10^6
$\mathrm{IDEA}^{\mathrm{Histogram}}_{\mathrm{Univariate}}$	390	0.000000	61930	110	−82.030702	608463
$\mathrm{ES}^{\mathrm{1\ Variance}}$	110	0.000000	145640	**1300**	**−99.499577**	$\mathbf{10^6}$
$\mathrm{ES}^{l\ \mathrm{Variances}}$	60	0.000000	397380	50	−32.004584	10^6
$\mathrm{ES}^{\mathrm{Covariance\ Matrix}}$	40	0.000000	302300	50	−29.902677	10^6
$\mathbb{R}\mathrm{CG}$	—	1617.093918	325946	—	−42.952690	650003
$\mathrm{GLIDE}^{\tau_G=0.05,\mathrm{Normal}}_{\mathrm{Univariate}}$	30	0.000000	74304	60	−50.478533	10^6
$\mathrm{GLIDE}^{\tau_G=0.05,\mathrm{Normal}}_{\mathrm{Bayesian}}$	50	0.000000	110448	40	−55.639026	10^6
$\mathrm{GLIDE}^{\tau_G=0.05,\mathrm{Normal}}_{\mathrm{Joint}}$	330	0.000000	803892	330	−47.157100	10^6
$\mathrm{GLIDE}^{\tau_G=0.05,\mathrm{Normal}}_{\mathrm{Univariate\ Mixture}}$	50	0.000000	90345	90	−53.351580	10^6
$\mathrm{GLIDE}^{\tau_G=0.05,\mathrm{Normal}}_{\mathrm{Bayesian\ Mixture}}$	180	0.000000	476169	80	−58.424541	10^6
$\mathrm{GLIDE}^{\tau_G=0.05,\mathrm{Normal}}_{\mathrm{Joint\ Mixture}}$	1900	0.151935	10^6	40	−44.821068	10^6
$\mathrm{GLIDE}^{\tau_G=0.25,\mathrm{Normal}}_{\mathrm{Univariate}}$	20	0.000000	101972	40	−50.705480	10^6
$\mathrm{GLIDE}^{\tau_G=0.25,\mathrm{Normal}}_{\mathrm{Bayesian}}$	50	0.000000	256170	20	−54.930641	10^6
$\mathrm{GLIDE}^{\tau_G=0.25,\mathrm{Normal}}_{\mathrm{Joint}}$	340	0.000019	10^6	330	−44.341350	10^6
$\mathrm{GLIDE}^{\tau_G=0.25,\mathrm{Normal}}_{\mathrm{Univariate\ Mixture}}$	50	0.000000	261983	80	−51.239764	10^6
$\mathrm{GLIDE}^{\tau_G=0.25,\mathrm{Normal}}_{\mathrm{Bayesian\ Mixture}}$	200	0.000000	10^6	60	−59.596523	10^6
$\mathrm{GLIDE}^{\tau_G=0.25,\mathrm{Normal}}_{\mathrm{Joint\ Mixture}}$	1800	258.631246	10^6	30	−43.836324	10^6

Figure B.5: Results on Griewank's function and Michalewicz's function in $l = 100$ dimensions for all tested algorithms. The best results are indicated in bold.

$l = 100$						
EA	**Rosenbrock**			**Summation Cancellation**		
	n	$\overline{fitness}$	$\overline{eval.}$	n	$\overline{fitness}$	$\overline{eval.}$
sGA^{1X}	17500	1413.923152	10^6	14000	3.821750	10^6
sGA^{UX}	5000	105.992052	10^6	2000	0.863864	10^6
$IDEA^{Binary}_{Tree\ Bayesian}$	*Time limit exceeded*			*Time limit exceeded*		
$IDEA^{Normal}_{Univariate}$	6000	97.198780	881909	310	9.178545	10^6
$IDEA^{Normal}_{Bayesian}$	5000	96.618794	675442	1150	46.425887	10^6
$IDEA^{Normal}_{Joint}$	12000	98.238924	10^6	5900	8431073.897552	10^6
$IDEA^{Normal}_{Univariate\ Mixture}$	4000	97.206950	590530	340	9.560386	10^6
$IDEA^{Normal}_{Bayesian\ Mixture}$	11000	96.569325	10^6	1250	18.562170	10^6
$IDEA^{Normal}_{Joint\ Mixture}$	16000	98.443210	10^6	**6200**	**8460648.155723**	$\mathbf{10^6}$
$IDEA^{Normal\ Kernels,\alpha=1}_{Univariate}$	4000	121.660861	10^6	150	1.508652	10^6
$IDEA^{Normal\ Kernels,\alpha=10}_{Univariate}$	9000	141.039766	10^6	22000	0.765696	10^6
$IDEA^{Normal\ Kernels,\alpha=1}_{Joint}$	1250	125.193529	10^6	850	3.922830	10^6
$IDEA^{Normal\ Kernels,\alpha=10}_{Joint}$	4050	606.977392	10^6	5000	1.740747	10^6
$IDEA^{Normal\ Mixture}_{Univariate}$	3000	16.606148	10^6	18000	0.615149	10^6
$IDEA^{Normal\ Mixture}_{Joint}$	250	99.000000	10^6	6250	1.442994	10^6
$IDEA^{Histogram}_{Univariate}$	3500	96.847875	935860	260	3.760914	10^6
$ES^{1\ Variance}$	900	95.101859	10^6	310	2.817394	10^6
$ES^{l\ Variances}$	40	81.155691	10^6	40	2.613287	10^6
$ES^{Covariance\ Matrix}$	10	61.423842	10^6	40	4.461414	10^6
$\mathbb{R}CG$	—	0.797325	301536	—	6.008094	845607
$GLIDE^{\tau_G=0.05,Normal}_{Univariate}$	20	0.000000	511406	40	6.112191	10^6
$GLIDE^{\tau_G=0.05,Normal}_{Bayesian}$	**20**	**0.000000**	**257235**	100	7.491924	10^6
$GLIDE^{\tau_G=0.05,Normal}_{Joint}$	20	40.539283	10^6	450	15.911143	10^6
$GLIDE^{\tau_G=0.05,Normal}_{Univariate\ Mixture}$	60	0.000000	908640	80	5.651255	10^6
$GLIDE^{\tau_G=0.05,Normal}_{Bayesian\ Mixture}$	80	0.000000	950771	100	5.555018	10^6
$GLIDE^{\tau_G=0.05,Normal}_{Joint\ Mixture}$	20	0.000000	807933	60	2.646361	10^6
$GLIDE^{\tau_G=0.25,Normal}_{Univariate}$	30	0.000000	10^6	30	4.849281	10^6
$GLIDE^{\tau_G=0.25,Normal}_{Bayesian}$	30	0.000000	927440	60	6.847906	10^6
$GLIDE^{\tau_G=0.25,Normal}_{Joint}$	20	42.391048	10^6	30	2.780007	10^6
$GLIDE^{\tau_G=0.25,Normal}_{Univariate\ Mixture}$	30	0.000000	914461	80	4.213135	10^6
$GLIDE^{\tau_G=0.25,Normal}_{Bayesian\ Mixture}$	30	0.000008	871303	70	5.133669	10^6
$GLIDE^{\tau_G=0.25,Normal}_{Joint\ Mixture}$	20	0.000023	993732	20	2.873178	10^6

Figure B.6: Results on Rosenbrock's function and the summation cancellation function in $l = 100$ dimensions for all tested algorithms. The best results are indicated in bold.

Statistically Significant Improvement Matrix	sGA^{1X}	sGA^{UX}	IDEA$^{\text{Binary}}_{\text{Tree Bayesian}}$	IDEA$^{\text{Normal}}_{\text{Univariate}}$	IDEA$^{\text{Normal}}_{\text{Bayesian}}$	IDEA$^{\text{Normal}}_{\text{Joint}}$	IDEA$^{\text{Normal}}_{\text{Univariate Mixture}}$	IDEA$^{\text{Normal}}_{\text{Bayesian Mixture}}$	IDEA$^{\text{Normal}}_{\text{Joint Mixture}}$	IDEA$^{\text{Normal Kernels},\alpha=1}_{\text{Univariate}}$	IDEA$^{\text{Normal Kernels},\alpha=10}_{\text{Univariate}}$	IDEA$^{\text{Normal Kernels},\alpha=1}_{\text{Joint}}$	IDEA$^{\text{Normal Kernels},\alpha=10}_{\text{Joint}}$	IDEA$^{\text{Normal Mixture}}_{\text{Univariate}}$	IDEA$^{\text{Normal Mixture}}_{\text{Joint}}$	IDEA$^{\text{Histogram}}_{\text{Univariate}}$
sGA^{1X}	0	2	1	-2	-2	-2	-4	-5	-6	-2	1	-2	4	-2	2	-2
sGA^{UX}	-2	0	0	-2	-3	-4	-5	-6	-6	-2	2	3	5	-3	2	-6
IDEA$^{\text{Binary}}_{\text{Tree Bayesian}}$	-1	0	0	0	-5	-5	-2	-4	-6	-2	0	0	2	-4	0	-6
IDEA$^{\text{Normal}}_{\text{Univariate}}$	2	2	0	0	-1	1	1	-2	0	4	4	6	8	0	6	-5
IDEA$^{\text{Normal}}_{\text{Bayesian}}$	2	3	5	1	0	2	6	5	2	4	5	6	8	0	7	3
IDEA$^{\text{Normal}}_{\text{Joint}}$	2	4	5	-1	-2	0	-1	-1	1	1	5	6	8	-5	7	-4
IDEA$^{\text{Normal}}_{\text{Univariate Mixture}}$	4	5	2	-1	-6	1	0	-2	1	4	6	8	8	-2	6	-3
IDEA$^{\text{Normal}}_{\text{Bayesian Mixture}}$	5	6	4	2	-5	1	2	0	1	4	8	7	8	-1	7	6
IDEA$^{\text{Normal}}_{\text{Joint Mixture}}$	6	6	6	0	-2	-1	-1	-1	0	5	9	7	9	-4	8	0
IDEA$^{\text{Normal Kernels},\alpha=1}_{\text{Univariate}}$	2	2	2	-4	-4	-1	-4	-4	-5	0	12	2	9	-3	6	-7
IDEA$^{\text{Normal Kernels},\alpha=10}_{\text{Univariate}}$	-1	-2	0	-4	-5	-5	-6	-8	-9	-12	0	-5	4	-6	2	-7
IDEA$^{\text{Normal Kernels},\alpha=1}_{\text{Joint}}$	2	-3	0	-6	-6	-6	-8	-7	-7	-2	5	0	9	-4	4	-7
IDEA$^{\text{Normal Kernels},\alpha=10}_{\text{Joint}}$	-4	-5	-2	-8	-8	-8	-8	-8	-9	-9	-4	-9	0	-4	4	-10
IDEA$^{\text{Normal Mixture}}_{\text{Univariate}}$	2	3	4	0	0	5	2	1	4	3	6	4	4	0	3	1
IDEA$^{\text{Normal Mixture}}_{\text{Joint}}$	-2	-2	0	-6	-7	-7	-6	-7	-8	-6	-2	-4	-4	-3	0	-8
IDEA$^{\text{Histogram}}_{\text{Univariate}}$	2	6	6	5	-3	4	3	-6	0	7	7	7	10	-1	8	0
ES$^{1 \text{ Variance}}$	10	12	12	6	2	6	6	4	8	12	12	10	12	5	11	5
ES$^{l \text{ Variances}}$	8	10	12	4	0	2	4	1	3	8	12	10	12	2	12	1
ES$^{\text{Covariance Matrix}}$	10	10	10	4	0	1	4	4	5	8	12	12	12	2	12	4
$\mathbb{R}$CG	2	-1	2	-4	-4	-1	-4	-3	-1	2	4	5	6	2	7	-1
GLIDE$^{\tau_G=0.05,\text{Normal}}_{\text{Univariate}}$	8	7	6	1	-1	4	1	2	6	10	12	12	12	5	12	4
GLIDE$^{\tau_G=0.05,\text{Normal}}_{\text{Bayesian}}$	8	8	8	2	-1	5	1	2	5	11	12	12	12	4	12	3
GLIDE$^{\tau_G=0.05,\text{Normal}}_{\text{Joint}}$	8	8	8	4	-2	2	3	-1	4	10	12	12	12	0	12	1
GLIDE$^{\tau_G=0.05,\text{Normal}}_{\text{Univariate Mixture}}$	8	7	6	0	-2	4	0	1	6	10	12	11	12	5	12	2
GLIDE$^{\tau_G=0.05,\text{Normal}}_{\text{Bayesian Mixture}}$	7	8	8	2	-1	4	0	1	4	10	12	12	12	4	12	2
GLIDE$^{\tau_G=0.05,\text{Normal}}_{\text{Joint Mixture}}$	6	7	8	1	-2	2	-1	-3	3	8	10	6	12	2	12	-1
GLIDE$^{\tau_G=0.25,\text{Normal}}_{\text{Univariate}}$	8	7	6	-1	-2	4	-2	-1	5	10	12	11	12	4	12	1
GLIDE$^{\tau_G=0.25,\text{Normal}}_{\text{Bayesian}}$	8	8	8	2	-2	5	0	0	5	10	12	12	12	3	12	1
GLIDE$^{\tau_G=0.25,\text{Normal}}_{\text{Joint}}$	6	7	7	1	-2	1	-2	-2	3	10	12	10	12	0	12	-2
GLIDE$^{\tau_G=0.25,\text{Normal}}_{\text{Univariate Mixture}}$	8	7	6	-1	-2	4	-2	0	6	10	12	10	12	4	12	1
GLIDE$^{\tau_G=0.25,\text{Normal}}_{\text{Bayesian Mixture}}$	8	8	7	0	-2	3	0	0	3	10	12	12	12	4	12	1
GLIDE$^{\tau_G=0.25,\text{Normal}}_{\text{Joint Mixture}}$	4	5	6	-1	-2	0	-4	-3	2	8	10	6	10	2	10	-1

Figure B.7: Number of times an improvement was found to be statistically significant, summed over all tested problems. The numbers in a single row indicate the summed number of significantly better or worse results compared to the algorithms in the different columns. The results are continued in figure B.8.

Statistically Significant Improvement Matrix	$\text{ES}^{1\text{ Variance}}$	$\text{ES}^{l\text{ Variances}}$	$\text{ES}^{\text{Covariance Matrix}}$	$\mathbb{R}$CG	$\text{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Univariate}}$	$\text{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Bayesian}}$	$\text{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Joint}}$	$\text{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Univariate Mixture}}$	$\text{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Bayesian Mixture}}$	$\text{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Joint Mixture}}$	$\text{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Univariate}}$	$\text{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Bayesian}}$	$\text{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Joint}}$	$\text{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Univariate Mixture}}$	$\text{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Bayesian Mixture}}$	$\text{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Joint Mixture}}$
sGA^{1X}	-10	-8	-10	-2	-8	-8	-8	-8	-7	-6	-8	-8	-6	-8	-8	-4
sGA^{UX}	-12	-10	-10	1	-7	-8	-8	-7	-8	-7	-7	-8	-7	-7	-8	-5
$\text{ID}\mathbb{E}\text{A}^{\text{Binary}}_{\text{Tree Bayesian}}$	-12	-12	-10	-2	-6	-8	-8	-6	-8	-8	-6	-8	-7	-6	-7	-6
$\text{ID}\mathbb{E}\text{A}^{\text{Normal}}_{\text{Univariate}}$	-6	-4	-4	4	-1	-2	-4	0	-2	-1	1	-2	-1	1	0	1
$\text{ID}\mathbb{E}\text{A}^{\text{Normal}}_{\text{Bayesian}}$	-2	0	0	4	1	1	2	2	1	2	2	2	2	2	2	2
$\text{ID}\mathbb{E}\text{A}^{\text{Normal}}_{\text{Joint}}$	-6	-2	-1	1	-4	-5	-2	-4	-4	-2	-4	-5	-1	-4	-3	0
$\text{ID}\mathbb{E}\text{A}^{\text{Normal}}_{\text{Univariate Mixture}}$	-6	-4	-4	4	-1	-1	-3	0	0	1	2	0	2	2	0	4
$\text{ID}\mathbb{E}\text{A}^{\text{Normal}}_{\text{Bayesian Mixture}}$	-4	-1	-4	3	-2	-2	1	-1	-1	3	1	0	2	0	0	3
$\text{ID}\mathbb{E}\text{A}^{\text{Normal}}_{\text{Joint Mixture}}$	-8	-3	-5	1	-6	-5	-4	-6	-4	-3	-5	-5	-3	-6	-3	-2
$\text{ID}\mathbb{E}\text{A}^{\text{Normal Kernels},\alpha=1}_{\text{Univariate}}$	-12	-8	-8	-2	-10	-11	-10	-10	-10	-8	-10	-10	-10	-10	-10	-8
$\text{ID}\mathbb{E}\text{A}^{\text{Normal Kernels},\alpha=10}_{\text{Univariate}}$	-12	-12	-12	-4	-12	-12	-12	-12	-12	-10	-12	-12	-12	-12	-12	-10
$\text{ID}\mathbb{E}\text{A}^{\text{Normal Kernels},\alpha=1}_{\text{Joint}}$	-10	-10	-12	-5	-12	-12	-12	-11	-12	-6	-11	-12	-10	-10	-12	-6
$\text{ID}\mathbb{E}\text{A}^{\text{Normal Kernels},\alpha=10}_{\text{Joint}}$	-12	-12	-12	-6	-12	-12	-12	-12	-12	-12	-12	-12	-12	-12	-12	-10
$\text{ID}\mathbb{E}\text{A}^{\text{Normal Mixture}}_{\text{Univariate}}$	-5	-2	-2	-2	-5	-4	0	-5	-4	-2	-4	-3	0	-4	-4	-2
$\text{ID}\mathbb{E}\text{A}^{\text{Normal Mixture}}_{\text{Joint}}$	-11	-12	-12	-7	-12	-12	-12	-12	-12	-12	-12	-12	-12	-12	-12	-10
$\text{ID}\mathbb{E}\text{A}^{\text{Histogram}}_{\text{Univariate}}$	-5	-1	-4	1	-4	-3	-1	-2	-2	1	-1	-1	2	-1	-1	1
$\text{ES}^{1\text{ Variance}}$	0	3	2	2	-2	-4	1	-2	-1	2	-2	-1	3	0	0	4
$\text{ES}^{l\text{ Variances}}$	-3	0	-3	0	-4	-6	-2	-4	-2	-1	-4	-4	-1	-4	-2	-1
$\text{ES}^{\text{Covariance Matrix}}$	-2	3	0	0	-4	-5	-2	-4	-2	0	-3	-4	0	-2	-2	0
$\mathbb{R}$CG	-2	0	0	0	-7	-9	-6	-3	-6	-4	-5	-6	-2	-5	-4	-4
$\text{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Univariate}}$	2	4	4	7	0	-4	2	4	2	7	6	0	7	9	5	10
$\text{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Bayesian}}$	4	6	5	9	4	0	5	7	10	12	9	8	11	10	10	11
$\text{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Joint}}$	-1	2	2	6	-2	-5	0	1	-1	6	-1	-3	10	2	3	8
$\text{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Univariate Mixture}}$	2	4	4	3	-4	-7	-1	0	-1	5	3	-4	5	8	2	10
$\text{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Bayesian Mixture}}$	1	2	2	6	-2	-10	1	1	0	7	2	-3	4	4	6	9
$\text{GLIDE}^{\tau_G=0.05,\text{Normal}}_{\text{Joint Mixture}}$	-2	1	0	4	-7	-12	-6	-5	-7	0	-5	-6	-2	-3	-2	8
$\text{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Univariate}}$	2	4	3	5	-6	-9	1	-3	-2	5	0	-5	6	0	1	7
$\text{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Bayesian}}$	1	4	4	6	0	-8	3	4	3	6	5	0	8	6	5	10
$\text{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Joint}}$	-3	1	0	2	-7	-11	-10	-5	-4	2	-6	-8	0	-2	-3	6
$\text{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Univariate Mixture}}$	0	4	2	5	-9	-10	-2	-8	-4	3	0	-6	2	0	-3	10
$\text{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Bayesian Mixture}}$	0	2	2	4	-5	-10	-3	-2	-6	2	-1	-5	3	3	0	9
$\text{GLIDE}^{\tau_G=0.25,\text{Normal}}_{\text{Joint Mixture}}$	-4	1	0	4	-10	-11	-8	-10	-9	-8	-7	-10	-6	-10	-9	0

Figure B.8: Statistically significant improvement matrix continued from figure B.7.

B.2 Permutation optimization

AIC Metric			$\mathcal{O}(l^{x.xx})$		
$\mathbb{ICE}$	Oper.	p_ϱ	n	$\overline{\text{Eval.}}$	$\overline{\text{Time}}$
N	S	0.0	—	—	—
N	S	0.1	—	—	—
N	S	0.5	—	—	—
P	S	0.0	1.54	2.23	4.26
P	S	0.1	1.26	2.26	4.40
P	S	0.5	0.63	3.27	5.41
P	SW	0.0	1.70	2.53	5.00
P	SW	0.1	1.24	2.51	5.03
P	SW	0.5	0.58	3.13	4.26
P	SWT	0.0	1.58	2.33	4.76
P	SWT	0.1	1.46	2.33	4.84
P	SWT	0.5	0.74	3.05	5.79
PDT	S	0.0	1.22	1.96	4.20
PDT	S	0.1	1.04	2.08	4.32
PDT	S	0.5	0.98	2.19	4.39
PDT	SW	0.0	1.33	2.12	4.99
PDT	SW	0.1	1.10	1.93	4.61
PDT	SW	0.5	1.96	2.73	5.85
PDT	SWT	0.0	1.10	1.72	4.34
PDT	SWT	0.1	0.89	1.70	4.24
PDT	SWT	0.5	1.96	3.31	6.54

BIC Metric			$\mathcal{O}(l^{x.xx})$		
$\mathbb{ICE}$	Oper.	p_ϱ	n	$\overline{\text{Eval.}}$	$\overline{\text{Time}}$
N	S	0.0	1.02	1.73	3.32
N	S	0.1	1.45	2.45	4.16
N	S	0.5	1.61	4.93	5.16
P	S	0.0	1.53	2.22	3.95
P	S	0.1	1.07	2.00	4.15
P	S	0.5	0.81	2.88	5.04
P	SW	0.0	1.03	1.69	4.05
P	SW	0.1	0.96	1.81	4.10
P	SW	0.5	0.44	2.51	3.49
P	SWT	0.0	1.03	1.58	3.87
P	SWT	0.1	0.93	1.74	4.02
P	SWT	0.5	1.34	2.94	5.57
PDT	S	0.0	1.00	1.81	4.00
PDT	S	0.1	1.17	2.13	4.35
PDT	S	0.5	0.43	2.59	4.57
PDT	SW	0.0	1.47	2.08	4.58
PDT	SW	0.1	1.14	2.00	4.45
PDT	SW	0.5	1.00	2.36	5.23
PDT	SWT	0.0	1.18	1.78	4.15
PDT	SWT	0.1	0.74	1.58	3.98
PDT	SWT	0.5	1.00	2.78	5.54

Figure B.9: Results on the additively decomposable relative–ordering deceptive problem ($\kappa^{\boldsymbol{\iota}} = 4$). The experimentally determined polynomial scale–up coefficients with respect to the population size n, the average required evaluations and the average actual running time for the tested $\mathbb{ICE}$ algorithms are shown. The algorithms are indicated by N for the use of normal gpdfs, P for the use of permutation gpdfs using frequency tables and PDT for the use of permutation gpdfs using default tables. The factorization learning operations are indicated by S for the splice operation, W for the swap operation and T for the transfer operation.

AIC Metric			$\mathcal{O}(l^{x.xx})$		
$\mathbb{ICE}$	Oper.	p_ϱ	n	Eval.	Time
P	SW	0.0	1.23	1.68	4.07
P	SW	0.1	1.42	2.15	4.58
P	SW	0.5	1.65	4.14	7.08

BIC Metric			$\mathcal{O}(l^{x.xx})$		
$\mathbb{ICE}$	Oper.	p_ϱ	n	Eval.	Time
N	S	0.0	1.61	2.06	3.70
N	S	0.1	1.72	2.79	4.33
N	S	0.5	—	—	—
P	SW	0.0	1.33	1.86	4.84
P	SW	0.1	1.43	1.69	4.51
P	SW	0.5	2.13	3.87	6.84
P	SWT	0.0	1.23	1.68	4.08
P	SWT	0.1	1.33	1.71	4.01
P	SWT	0.5	1.75	2.77	5.61
PDT	SW	0.0	1.25	1.87	4.71
PDT	SW	0.1	1.71	2.21	5.13
PDT	SW	0.5	3.80	4.36	8.19
PDT	SWT	0.0	1.32	1.87	4.80
PDT	SWT	0.1	1.81	2.28	4.88
PDT	SWT	0.5	3.32	4.45	8.29

Figure B.10: Results on the additively decomposable relative–ordering deceptive problem ($\kappa^\iota = 5$). The polynomial scale–up coefficients with respect to the population size n, the average required evaluations and the average actual running time for the tested $\mathbb{ICE}$ algorithms are shown. The algorithms are indicated by N for the use of normal gpdfs, P for the use of permutation gpdfs using frequency tables and PDT for the use of permutation gpdfs using default tables. The factorization learning operations are indicated by S for the splice operation, W for the swap operation and T for the transfer operation.

AIC Metric			$\lvert\boldsymbol{\iota}\rvert = \mathbf{3}, \boldsymbol{\kappa^\iota} = \mathbf{4}$			$\lvert\boldsymbol{\iota}\rvert = \mathbf{6}, \boldsymbol{\kappa^\iota} = \mathbf{4}$		
$\mathbb{ICE}$	Oper.	p_ϱ	n	$\overline{\text{subs}}$	$\overline{\text{Eval.}}$	n	$\overline{\text{subs}}$	$\overline{\text{Eval.}}$
N	S	0.0	100000	2.13	613340	100000	3.84	1453352
N	S	0.1	100000	2.70	1054346	100000	4.50	2216364
N	S	0.5	100000	2.97	4335060	100000	5.33	38633217
P	SW	0.0	21000	3.00	222893	100000	5.20	1654022
P	SW	0.1	1900	3.00	21022	16000	6.00	328135
P	SW	0.5	1500	3.00	18982	11000	6.00	442256

BIC Metric			$\lvert\boldsymbol{\iota}\rvert = \mathbf{3}, \boldsymbol{\kappa^\iota} = \mathbf{4}$			$\lvert\boldsymbol{\iota}\rvert = \mathbf{6}, \boldsymbol{\kappa^\iota} = \mathbf{4}$		
$\mathbb{ICE}$	Oper.	p_ϱ	n	$\overline{\text{subs}}$	$\overline{\text{Eval.}}$	n	$\overline{\text{subs}}$	$\overline{\text{Eval.}}$
N	S	0.0	100000	2.17	643674	100000	4.07	1579354
N	S	0.1	100000	2.70	1066013	100000	4.87	2585035
N	S	0.5	100000	2.87	4111057	100000	5.40	45423314
P	SW	0.0	21000	3.00	250336	100000	5.00	1224018
P	SW	0.1	1600	3.00	19088	22000	6.00	576436
P	SW	0.5	1500	3.00	22520	18000	6.00	1116807
P	SWT	0.0	19000	3.00	210534	100000	5.00	1259755
P	SWT	0.1	1400	3.00	16998	20000	6.00	538037
P	SWT	0.5	1200	3.00	16759	12000	6.00	798334
PDT	SW	0.0	70000	3.00	684962	100000	5.00	1374018
PDT	SW	0.1	4750	3.00	42999	18000	6.00	302783
PDT	SW	0.5	1500	3.00	13586	80000	6.00	1116019
PDT	SWT	0.0	50000	3.00	473512	100000	5.00	1416019
PDT	SWT	0.1	4000	3.00	35371	13000	6.00	202301
PDT	SWT	0.5	1500	3.00	12641	70000	6.00	1050020

Figure B.11: Results for all algorithms on the overlapping deceptive permutation problem with $\kappa^\iota = 4$; $\overline{\text{subs}}$ stands for the average number of subfunctions solved optimally. The abbreviations are the same as those in Figures B.9 and B.10.

B.3 Multi–objective optimization

	$D_{\mathcal{P}_F \to \mathcal{S}}$							
EA	BT_1^{10}	ZDT_4^{10}	ZDT_6^{10}	CTP_7^{10}	BT_1^{100}	ZDT_4^{100}	ZDT_6^{100}	CTP_7^{100}
SPEA^{UX}	$100 \cdot 10^5$	4.62	0.193	7.97	$100 \cdot 10^5$	470	7.64	499
SPEA^{1X}	$100 \cdot 10^5$	3.90	0.172	7.31	$100 \cdot 10^5$	447	7.06	476
NSGA-II^{UX}	$100 \cdot 10^5$	4.39	0.303	7.25	$100 \cdot 10^5$	360	5.99	348
NSGA-II^{1X}	$100 \cdot 10^5$	**1.40**	0.328	**3.32**	$100 \cdot 10^5$	297	6.59	303
$\mathbb{M}_{1\text{ Cluster}}^{UX}$	$100 \cdot 10^5$	4.43	0.358	6.63	$100 \cdot 10^5$	374	6.72	378
$\mathbb{M}_{1\text{ Cluster}}^{1X}$	$100 \cdot 10^5$	1.89	0.291	4.13	$100 \cdot 10^5$	336	6.81	345
$\mathbb{M}_{\text{Par. Clust.}}^{UX}$	$100 \cdot 10^5$	4.01	0.368	6.42	$100 \cdot 10^5$	400	6.98	394
$\mathbb{M}_{\text{Par. Clust.}}^{1X}$	$100 \cdot 10^5$	1.65	0.298	3.77	$100 \cdot 10^5$	332	7.01	340
$\mathbb{M}_{\text{Obj. Clust.}}^{UX}$	$100 \cdot 10^5$	3.98	0.354	7.27	$100 \cdot 10^5$	311	5.96	326
$\mathbb{M}_{\text{Obj. Clust.}}^{1X}$	$100 \cdot 10^5$	2.03	0.311	3.95	$100 \cdot 10^5$	328	6.74	335
$\mathbb{M}_{1\text{ Cluster}}^{\text{Univariate}}$	$100 \cdot 10^5$	14.0	1.08	16.5	$100 \cdot 10^5$	774	3.06	875
$\mathbb{M}_{1\text{ Cluster}}^{\text{Learning}}$	$100 \cdot 10^5$	11.2	**0.00239**	15.3	$100 \cdot 10^5$	597	**0.434**	600
$\mathbb{M}_{\text{Par. Clust.}}^{\text{Univariate}}$	$999 \cdot 10^4$	5.36	0.798	7.93	$100 \cdot 10^5$	168	3.70	192
$\mathbb{M}_{\text{Par. Clust.}}^{\text{Learning}}$	$999 \cdot 10^4$	14.0	0.159	17.1	$100 \cdot 10^5$	416	0.470	523
$\mathbb{M}_{\text{Obj. Clust.}}^{\text{Univariate}}$	$100 \cdot 10^5$	5.00	0.306	8.64	$100 \cdot 10^5$	157	4.60	**161**
$\mathbb{M}_{\text{Obj. Clust.}}^{\text{Learning}}$	$\mathbf{998 \cdot 10^4}$	11.5	0.287	12.6	$\mathbf{100 \cdot 10^5}$	**144**	1.30	165

Figure B.12: Average of the $D_{\mathcal{P}_F \to \mathcal{S}}$ performance indicator on all real–valued problems.

	$D_{\mathcal{P}_F \to \mathcal{S}}$							
EA	MS^{100}	KN^{100}	SC^{100}	MST^{105}	MS^{1000}	KN^{1000}	SC^{1000}	MST^{1035}
SPEA^{UX}	6.10	5.14	2.93	1.31	147	67.3	442	3.39
SPEA^{1X}	7.15	5.14	2.99	1.39	237	83.1	376	3.09
NSGA-II^{UX}	**4.52**	**4.22**	**1.79**	1.09	147	56.1	185	3.55
NSGA-II^{1X}	8.03	5.40	2.64	1.36	250	90.5	254	3.18
$\mathbb{M}_{1\text{ Cluster}}^{UX}$	8.18	5.62	2.13	1.54	194	76.5	241	4.37
$\mathbb{M}_{1\text{ Cluster}}^{1X}$	10.5	6.65	2.49	1.52	326	136	364	3.96
$\mathbb{M}_{\text{Par. Clust.}}^{UX}$	7.79	6.17	2.10	1.68	191	78.9	233	4.54
$\mathbb{M}_{\text{Par. Clust.}}^{1X}$	10.5	7.66	3.00	1.64	317	138	355	4.14
$\mathbb{M}_{\text{Obj. Clust.}}^{UX}$	9.01	5.01	2.12	**0.906**	213	59.3	254	4.63
$\mathbb{M}_{\text{Obj. Clust.}}^{1X}$	12.7	6.17	2.69	1.07	359	146	410	4.32
$\mathbb{M}_{1\text{ Cluster}}^{\text{Univariate}}$	8.35	9.95	1.92	1.95	124	105	**85.9**	5.58
$\mathbb{M}_{1\text{ Cluster}}^{\text{Learning}}$	16.5	6.52	2.39	1.82	147	73.1	136	2.90
$\mathbb{M}_{\text{Par. Clust.}}^{\text{Univariate}}$	9.21	8.70	2.18	2.28	136	98.2	91.6	4.95
$\mathbb{M}_{\text{Par. Clust.}}^{\text{Learning}}$	14.9	7.47	2.90	1.56	110	88.6	141	2.86
$\mathbb{M}_{\text{Obj. Clust.}}^{\text{Univariate}}$	9.07	5.34	2.01	1.15	**26.5**	**22.3**	129	2.41
$\mathbb{M}_{\text{Obj. Clust.}}^{\text{Learning}}$	16.5	7.23	5.38	1.39	113	64.0	472	**2.13**

Figure B.13: Average of the $D_{\mathcal{P}_F \to \mathcal{S}}$ performance indicator on all combinatorial problems.

Front Spread **FS**								
EA	BT_1^{10}	ZDT_4^{10}	ZDT_6^{10}	CTP_7^{10}	BT_1^{100}	ZDT_4^{100}	ZDT_6^{100}	CTP_7^{100}
SPEA$^{\mathrm{UX}}$	225	51.4	5.22	44.9	2.06	692	1.85	733
SPEA$^{\mathrm{1X}}$	369	55.8	5.26	46.3	2.31	736	3.02	773
NSGA-II$^{\mathrm{UX}}$	179	3.60	1.09	1.76	0.413	35.2	0.756	29.3
NSGA-II$^{\mathrm{1X}}$	23.4	8.93	1.03	1.31	1.02	33.4	0.665	13.9
$\mathbb{M}^{\mathrm{UX}}_{\mathrm{1\ Cluster}}$	655	8.55	2.90	39.1	2.18	395	3.43	365
$\mathbb{M}^{\mathrm{1X}}_{\mathrm{1\ Cluster}}$	78.6	2.46	1.92	1.41	2.27	94.0	1.40	88.6
$\mathbb{M}^{\mathrm{UX}}_{\mathrm{Par.\ Clust.}}$	357	12.2	5.05	4.85	2.11	384	3.10	345
$\mathbb{M}^{\mathrm{1X}}_{\mathrm{Par.\ Clust.}}$	199	2.45	5.33	1.66	2.31	129	1.53	93.1
$\mathbb{M}^{\mathrm{UX}}_{\mathrm{Obj.\ Clust.}}$	685	40.8	4.11	41.8	2.15	740	4.75	737
$\mathbb{M}^{\mathrm{1X}}_{\mathrm{Obj.\ Clust.}}$	262	3.38	3.94	58.9	2.29	359	2.30	371
$\mathbb{M}^{\mathrm{Univariate}}_{\mathrm{1\ Cluster}}$	293	70.8	1.15	84.7	1.82	393	0.180	347
$\mathbb{M}^{\mathrm{Learning}}_{\mathrm{1\ Cluster}}$	$129 \cdot 10^1$	84.9	3.00	87.4	2.12	635	2.20	342
$\mathbb{M}^{\mathrm{Univariate}}_{\mathrm{Par.\ Clust.}}$	$508 \cdot 10^1$	24.0	2.47	28.8	2.19	231	0.05	306
$\mathbb{M}^{\mathrm{Learning}}_{\mathrm{Par.\ Clust.}}$	$112 \cdot 10^2$	142	5.15	116	1.91	577	7.01	588
$\mathbb{M}^{\mathrm{Univariate}}_{\mathrm{Obj.\ Clust.}}$	$209 \cdot 10^1$	90.4	**5.29**	114	2.45	636	**8.10**	619
$\mathbb{M}^{\mathrm{Learning}}_{\mathrm{Obj.\ Clust.}}$	$\mathbf{164 \cdot 10^2}$	**197**	3.68	**188**	**3.28**	$\mathbf{175 \cdot 10^1}$	3.97	$\mathbf{183 \cdot 10^1}$

Figure B.14: Average of the **FS** performance indicator on all real–valued problems.

Front Spread **FS**								
EA	MS^{100}	KN^{100}	SC^{100}	MST^{105}	MS^{1000}	KN^{1000}	SC^{1000}	MST^{1035}
SPEA$^{\mathrm{UX}}$	109	69.5	**64.6**	30.6	288	254	631	52.1
SPEA$^{\mathrm{1X}}$	123	82.6	51.0	32.5	399	308	**636**	50.8
NSGA-II$^{\mathrm{UX}}$	110	71.8	16.1	26.3	370	288	144	33.7
NSGA-II$^{\mathrm{1X}}$	129	76.6	12.8	23.9	364	291	107	36.6
$\mathbb{M}^{\mathrm{UX}}_{\mathrm{1\ Cluster}}$	114	82.0	19.6	19.8	389	347	164	37.1
$\mathbb{M}^{\mathrm{1X}}_{\mathrm{1\ Cluster}}$	122	81.4	17.3	21.5	421	301	154	44.3
$\mathbb{M}^{\mathrm{UX}}_{\mathrm{Par.\ Clust.}}$	99.4	69.0	18.1	19.2	374	326	170	36.2
$\mathbb{M}^{\mathrm{1X}}_{\mathrm{Par.\ Clust.}}$	111	77.3	16.7	20.6	420	327	135	44.2
$\mathbb{M}^{\mathrm{UX}}_{\mathrm{Obj.\ Clust.}}$	173	114	21.3	**37.3**	**706**	**585**	319	**77.3**
$\mathbb{M}^{\mathrm{1X}}_{\mathrm{Obj.\ Clust.}}$	**169**	98.9	20.5	32.2	681	521	249	76.4
$\mathbb{M}^{\mathrm{Univariate}}_{\mathrm{1\ Cluster}}$	68.3	39.4	16.2	17.4	111	106	11.9	22.7
$\mathbb{M}^{\mathrm{Learning}}_{\mathrm{1\ Cluster}}$	123	83.8	18.4	17.8	153	199	142	36.9
$\mathbb{M}^{\mathrm{Univariate}}_{\mathrm{Par.\ Clust.}}$	73.8	48.8	15.0	15.2	85.0	126	18.9	25.5
$\mathbb{M}^{\mathrm{Learning}}_{\mathrm{Par.\ Clust.}}$	124	83.2	15.8	19.4	171	179	24.8	34.7
$\mathbb{M}^{\mathrm{Univariate}}_{\mathrm{Obj.\ Clust.}}$	167	**108**	23.8	26.3	559	485	168	55.1
$\mathbb{M}^{\mathrm{Learning}}_{\mathrm{Obj.\ Clust.}}$	162	104	20.6	24.1	579	544	143	46.2

Figure B.15: Average of the **FS** performance indicator on all combinatorial problems.

Front Occupation **FO**								
EA	BT_1^{10}	ZDT_4^{10}	ZDT_6^{10}	CTP_7^{10}	BT_1^{100}	ZDT_4^{100}	ZDT_6^{100}	CTP_7^{100}
SPEA$^{\mathrm{UX}}$	**60.9**	99.0	50.0	43.5	49.8	27.6	18.7	26.7
SPEA$^{\mathrm{1X}}$	38.7	**187**	49.6	43.2	48.8	27.4	29.3	26.8
NSGA-II$^{\mathrm{UX}}$	5.42	59.7	47.5	59.3	100	5.80	6.00	4.00
NSGA-II$^{\mathrm{1X}}$	29.5	32.7	31.2	9.98	75.0	5.00	6.60	3.00
$\mathbb{M}_{1\ \mathrm{Cluster}}^{\mathrm{UX}}$	9.92	41.7	8.06	9.00	14.4	12.8	14.4	12.6
$\mathbb{M}_{1\ \mathrm{Cluster}}^{\mathrm{1X}}$	13.4	30.3	6.52	11.9	16.5	7.10	6.64	5.94
$\mathbb{M}_{\mathrm{Par.\ Clust.}}^{\mathrm{UX}}$	7.46	25.4	8.02	18.2	15.4	12.9	15.2	12.4
$\mathbb{M}_{\mathrm{Par.\ Clust.}}^{\mathrm{1X}}$	9.78	24.7	7.80	11.9	17.5	7.20	8.12	6.68
$\mathbb{M}_{\mathrm{Obj.\ Clust.}}^{\mathrm{UX}}$	13.9	10.0	8.48	8.62	19.1	20.0	19.6	21.7
$\mathbb{M}_{\mathrm{Obj.\ Clust.}}^{\mathrm{1X}}$	9.94	31.4	7.32	15.6	17.4	12.2	9.76	12.2
$\mathbb{M}_{1\ \mathrm{Cluster}}^{\mathrm{Univariate}}$	5.74	6.88	4.90	4.14	36.7	6.9	2.55	3.20
$\mathbb{M}_{1\ \mathrm{Cluster}}^{\mathrm{Learning}}$	6.06	8.36	**258**	4.96	13.1	5.25	**369**	3.75
$\mathbb{M}_{\mathrm{Par.\ Clust.}}^{\mathrm{Univariate}}$	29.6	98.8	30.0	**82.0**	33.4	69.4	3.70	18.3
$\mathbb{M}_{\mathrm{Par.\ Clust.}}^{\mathrm{Learning}}$	52.7	65.4	104	69.2	**149**	105	92.0	**112**
$\mathbb{M}_{\mathrm{Obj.\ Clust.}}^{\mathrm{Univariate}}$	12.5	68.7	56.3	34.0	64.5	**106**	27.7	78.9
$\mathbb{M}_{\mathrm{Obj.\ Clust.}}^{\mathrm{Learning}}$	30.1	26.4	197	32.1	111	50.8	163	43.0

Figure B.16: Average of the **FO** performance indicator on all real–valued problems.

Front Occupation **FO**								
EA	MS^{100}	KN^{100}	SC^{100}	MST^{105}	MS^{1000}	KN^{1000}	SC^{1000}	MST^{1035}
SPEA$^{\mathrm{UX}}$	96.4	49.8	27.2	49.6	49.4	49.5	26.2	50.0
SPEA$^{\mathrm{1X}}$	96.4	87.6	27.7	124	49.9	49.7	26.5	98.6
NSGA-II$^{\mathrm{UX}}$	200	**175**	150	200	35.9	33.1	7.50	250
NSGA-II$^{\mathrm{1X}}$	99.9	**175**	**212**	200	42.9	37.0	7.20	249
$\mathbb{M}_{1\ \mathrm{Cluster}}^{\mathrm{UX}}$	32.2	18.4	7.80	63.8	17.6	16.6	4.40	14.4
$\mathbb{M}_{1\ \mathrm{Cluster}}^{\mathrm{1X}}$	41.9	28.7	10.3	62.2	18.9	15.3	5.70	21.8
$\mathbb{M}_{\mathrm{Par.\ Clust.}}^{\mathrm{UX}}$	49.1	20.7	8.00	41.2	15.0	16.9	4.50	15.5
$\mathbb{M}_{\mathrm{Par.\ Clust.}}^{\mathrm{1X}}$	58.0	30.2	8.72	58.2	19.0	15.2	4.10	26.1
$\mathbb{M}_{\mathrm{Obj.\ Clust.}}^{\mathrm{UX}}$	73.3	30.2	11.8	61.1	28.5	27.6	6.7	24.8
$\mathbb{M}_{\mathrm{Obj.\ Clust.}}^{\mathrm{1X}}$	71.3	62.9	17.2	352	34.4	18.5	5.90	27.2
$\mathbb{M}_{1\ \mathrm{Cluster}}^{\mathrm{Univariate}}$	111	14.3	10.2	105	39.3	16.4	28.4	15.7
$\mathbb{M}_{1\ \mathrm{Cluster}}^{\mathrm{Learning}}$	227	34.7	13.5	42.8	19.9	40.7	6.43	286
$\mathbb{M}_{\mathrm{Par.\ Clust.}}^{\mathrm{Univariate}}$	167	17.7	9.84	18.1	41.4	27.8	4.00	17.0
$\mathbb{M}_{\mathrm{Par.\ Clust.}}^{\mathrm{Learning}}$	130	70.7	13.1	54.0	106	49.9	5.86	325
$\mathbb{M}_{\mathrm{Obj.\ Clust.}}^{\mathrm{Univariate}}$	132	38.1	19.3	**299**	108	57.2	**255**	41.6
$\mathbb{M}_{\mathrm{Obj.\ Clust.}}^{\mathrm{Learning}}$	**390**	164	34.9	93.5	**548**	**67.0**	30.3	$\mathbf{105 \cdot 10^1}$

Figure B.17: Average of the **FO** performance indicator on all combinatorial problems.

	Population Size n							
EA	BT_1^{10}	ZDT_4^{10}	ZDT_6^{10}	CTP_7^{10}	BT_1^{100}	ZDT_4^{100}	ZDT_6^{100}	CTP_7^{100}
SPEA$^{\mathrm{UX}}$	50	50	25	25	25	25	25	25
SPEA$^{\mathrm{1X}}$	25	100	25	25	25	25	25	25
NSGA-II$^{\mathrm{UX}}$	200	200	100	100	100	200	200	150
NSGA-II$^{\mathrm{1X}}$	200	375	75	300	75	200	150	300
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{UX}}$	75	100	25	25	100	125	200	125
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{1X}}$	100	450	25	300	125	325	100	175
$\mathbb{M}_{\text{Par. Clust.}}^{\mathrm{UX}}$	175	75	25	100	75	125	150	175
$\mathbb{M}_{\text{Par. Clust.}}^{\mathrm{1X}}$	125	450	25	275	150	175	100	175
$\mathbb{M}_{\text{Obj. Clust.}}^{\mathrm{UX}}$	225	25	25	25	125	200	200	300
$\mathbb{M}_{\text{Obj. Clust.}}^{\mathrm{1X}}$	150	475	25	725	125	200	100	150
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{Univariate}}$	150	50	75	50	100	75	375	50
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{Learning}}$	150	75	425	75	175	100	700	100
$\mathbb{M}_{\text{Par. Clust.}}^{\mathrm{Univariate}}$	175	125	175	125	225	150	450	150
$\mathbb{M}_{\text{Par. Clust.}}^{\mathrm{Learning}}$	400	250	275	250	200	150	550	125
$\mathbb{M}_{\text{Obj. Clust.}}^{\mathrm{Univariate}}$	275	125	200	125	250	200	800	200
$\mathbb{M}_{\text{Obj. Clust.}}^{\mathrm{Learning}}$	450	200	250	150	225	300	400	250

Figure B.18: Population sizes used for the real–valued problems.

	Population Size n							
EA	MS^{100}	KN^{100}	SC^{100}	MST^{105}	MS^{1000}	KN^{1000}	SC^{1000}	MST^{1035}
SPEA$^{\mathrm{UX}}$	50	25	25	25	25	25	25	25
SPEA$^{\mathrm{1X}}$	50	50	25	100	25	25	25	50
NSGA-II$^{\mathrm{UX}}$	200	175	150	200	200	250	200	250
NSGA-II$^{\mathrm{1X}}$	100	175	250	200	150	200	150	250
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{UX}}$	100	150	225	1250	175	200	100	350
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{1X}}$	100	175	125	1200	150	150	150	375
$\mathbb{M}_{\text{Par. Clust.}}^{\mathrm{UX}}$	125	175	225	625	125	225	100	375
$\mathbb{M}_{\text{Par. Clust.}}^{\mathrm{1X}}$	125	275	125	950	125	125	100	525
$\mathbb{M}_{\text{Obj. Clust.}}^{\mathrm{UX}}$	175	175	100	950	175	250	125	450
$\mathbb{M}_{\text{Obj. Clust.}}^{\mathrm{1X}}$	150	175	150	675	200	125	100	425
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{Univariate}}$	325	250	350	2700	650	500	150	475
$\mathbb{M}_{1\text{ Cluster}}^{\mathrm{Learning}}$	425	400	500	1800	1000	700	500	1500
$\mathbb{M}_{\text{Par. Clust.}}^{\mathrm{Univariate}}$	275	250	400	875	650	475	500	400
$\mathbb{M}_{\text{Par. Clust.}}^{\mathrm{Learning}}$	675	450	550	625	900	500	700	825
$\mathbb{M}_{\text{Obj. Clust.}}^{\mathrm{Univariate}}$	250	200	400	900	475	600	475	775
$\mathbb{M}_{\text{Obj. Clust.}}^{\mathrm{Learning}}$	500	450	850	1950	750	1250	1900	1250

Figure B.19: Population sizes used for the combinatorial problems.

$D_{\mathcal{P}_F \to \mathcal{S}}$																
Statistically Significant Improvement Matrix	SPEA^{UX}	SPEA^{1X}	NSGA-II^{UX}	NSGA-II^{1X}	$\mathbb{M}^{UX}_{1\text{ Cluster}}$	$\mathbb{M}^{1X}_{1\text{ Cluster}}$	$\mathbb{M}^{UX}_{\text{Par. Clust.}}$	$\mathbb{M}^{1X}_{\text{Par. Clust.}}$	$\mathbb{M}^{UX}_{\text{Obj. Clust.}}$	$\mathbb{M}^{1X}_{\text{Obj. Clust.}}$	$\mathbb{M}^{\text{Univariate}}_{1\text{ Cluster}}$	$\mathbb{M}^{\text{Learning}}_{1\text{ Cluster}}$	$\mathbb{M}^{\text{Univariate}}_{\text{Par. Clust.}}$	$\mathbb{M}^{\text{Learning}}_{\text{Par. Clust.}}$	$\mathbb{M}^{\text{Univariate}}_{\text{Obj. Clust.}}$	$\mathbb{M}^{\text{Learning}}_{\text{Obj. Clust.}}$
SPEA^{UX}	0	-6	-8	-2	1	-1	0	0	-4	-2	4	1	-2	-1	-9	-1
SPEA^{1X}	6	0	-8	-3	-3	1	-3	3	-4	2	6	-1	-1	1	-9	-1
NSGA-II^{UX}	8	8	0	3	11	5	10	4	4	3	7	3	0	2	-7	1
NSGA-II^{1X}	2	3	-3	0	3	10	4	9	-2	9	4	-2	0	1	-9	-2
$\mathbb{M}^{UX}_{1\text{ Cluster}}$	-1	3	-11	-3	0	0	6	1	-5	-1	5	1	0	1	-8	-4
$\mathbb{M}^{1X}_{1\text{ Cluster}}$	1	-1	-5	-10	0	0	3	3	-5	0	2	-2	-4	-1	-11	-3
$\mathbb{M}^{UX}_{\text{Par. Clust.}}$	0	3	-10	-4	-6	-3	0	-1	-4	-4	6	-1	1	-1	-9	-4
$\mathbb{M}^{1X}_{\text{Par. Clust.}}$	0	-3	-4	-9	-1	-3	1	0	-4	2	2	-4	-3	-3	-10	-4
$\mathbb{M}^{UX}_{\text{Obj. Clust.}}$	4	4	-4	2	5	5	4	4	0	4	4	3	-1	2	-8	0
$\mathbb{M}^{1X}_{\text{Obj. Clust.}}$	2	-2	-3	-9	1	0	4	-2	-4	0	2	-5	-4	-1	-10	-2
$\mathbb{M}^{\text{Univariate}}_{1\text{ Cluster}}$	-4	-6	-7	-4	-5	-2	-6	-2	-4	-2	0	-7	-5	-8	-10	-9
$\mathbb{M}^{\text{Learning}}_{1\text{ Cluster}}$	-1	1	-3	2	-1	2	1	4	-3	5	7	0	-2	0	-9	-3
$\mathbb{M}^{\text{Univariate}}_{\text{Par. Clust.}}$	2	1	0	0	0	4	-1	3	1	4	5	2	0	-1	-4	-5
$\mathbb{M}^{\text{Learning}}_{\text{Par. Clust.}}$	1	-1	-2	-1	-1	1	1	3	-2	1	8	0	1	0	-7	-2
$\mathbb{M}^{\text{Univariate}}_{\text{Obj. Clust.}}$	9	9	7	9	8	11	9	10	8	10	10	9	4	7	0	5
$\mathbb{M}^{\text{Learning}}_{\text{Obj. Clust.}}$	1	1	-1	2	4	3	4	4	0	2	9	3	5	2	-5	0

Figure B.20: Number of times an improvement was found to be statistically significant in the $D_{\mathcal{P}_F \to \mathcal{S}}$ performance indicator, summed over all tested problems. The numbers in a single row indicate the summed number of significantly better or worse results compared to the algorithms in the different columns.

Front Spread **FS**																
Statistically Significant Improvement Matrix	SPEA^{UX}	SPEA^{1X}	NSGA-II^{UX}	NSGA-II^{1X}	$\mathbb{M}^{UX}_{1\ Cluster}$	$\mathbb{M}^{1X}_{1\ Cluster}$	$\mathbb{M}^{UX}_{Par.\ Clust.}$	$\mathbb{M}^{1X}_{Par.\ Clust.}$	$\mathbb{M}^{UX}_{Obj.\ Clust.}$	$\mathbb{M}^{1X}_{Obj.\ Clust.}$	$\mathbb{M}^{Univariate}_{1\ Cluster}$	$\mathbb{M}^{Learning}_{1\ Cluster}$	$\mathbb{M}^{Univariate}_{Par.\ Clust.}$	$\mathbb{M}^{Learning}_{Par.\ Clust.}$	$\mathbb{M}^{Univariate}_{Obj.\ Clust.}$	$\mathbb{M}^{Learning}_{Obj.\ Clust.}$
SPEA^{UX}	0	-7	9	8	3	5	6	4	-4	0	11	3	13	2	-6	-6
SPEA^{1X}	7	0	16	14	8	11	11	9	-2	2	11	7	13	5	-5	-5
NSGA-II^{UX}	-9	-16	0	5	-12	-6	-9	-7	-15	-14	3	-9	1	-6	-14	-13
NSGA-II^{1X}	-8	-14	-5	0	-10	-7	-7	-9	-16	-14	1	-7	0	-6	-16	-15
$\mathbb{M}^{UX}_{1\ Cluster}$	-3	-8	12	10	0	4	5	5	-12	-8	9	0	10	-2	-15	-13
$\mathbb{M}^{1X}_{1\ Cluster}$	-5	-11	6	7	-4	0	-2	-1	-14	-14	5	-3	4	-1	-15	-14
$\mathbb{M}^{UX}_{Par.\ Clust.}$	-6	-11	9	7	-5	2	0	0	-12	-7	9	0	8	-3	-14	-11
$\mathbb{M}^{1X}_{Par.\ Clust.}$	-4	-9	7	9	-5	1	0	0	-11	-12	6	-2	4	-2	-15	-12
$\mathbb{M}^{UX}_{Obj.\ Clust.}$	4	2	15	16	12	14	12	11	0	7	11	9	14	6	1	1
$\mathbb{M}^{1X}_{Obj.\ Clust.}$	0	-2	14	14	8	14	7	12	-7	0	9	5	11	2	-5	-4
$\mathbb{M}^{Univariate}_{1\ Cluster}$	-11	-11	-3	-1	-9	-5	-9	-6	-11	-9	0	-12	-2	-14	-15	-16
$\mathbb{M}^{Learning}_{1\ Cluster}$	-3	-7	9	7	0	3	0	2	-9	-5	12	0	12	-4	-12	-13
$\mathbb{M}^{Univariate}_{Par.\ Clust.}$	-13	-13	-1	0	-10	-4	-8	-4	-14	-11	2	-12	0	-12	-15	-16
$\mathbb{M}^{Learning}_{Par.\ Clust.}$	-2	-5	6	6	2	1	3	2	-6	-2	14	4	12	0	-8	-11
$\mathbb{M}^{Univariate}_{Obj.\ Clust.}$	6	5	14	16	15	15	14	15	-1	5	15	12	15	8	0	-2
$\mathbb{M}^{Learning}_{Obj.\ Clust.}$	6	5	13	15	13	14	11	12	-1	4	16	13	16	11	2	0

Figure B.21: Number of times an improvement was found to be statistically significant in the FS performance indicator, summed over all tested problems. The numbers in a single row indicate the summed number of significantly better or worse results compared to the algorithms in the different columns.

Front Occupation **FO**																
Statistically Significant Improvement Matrix	SPEA^{UX}	SPEA^{1X}	NSGA-II^{UX}	NSGA-II^{1X}	$\mathbb{M}^{UX}_{1\ Cluster}$	$\mathbb{M}^{1X}_{1\ Cluster}$	$\mathbb{M}^{UX}_{Par.\ Clust.}$	$\mathbb{M}^{1X}_{Par.\ Clust.}$	$\mathbb{M}^{UX}_{Obj.\ Clust.}$	$\mathbb{M}^{1X}_{Obj.\ Clust.}$	$\mathbb{M}^{Univariate}_{1\ Cluster}$	$\mathbb{M}^{Learning}_{1\ Cluster}$	$\mathbb{M}^{Univariate}_{Par.\ Clust.}$	$\mathbb{M}^{Learning}_{Par.\ Clust.}$	$\mathbb{M}^{Univariate}_{Obj.\ Clust.}$	$\mathbb{M}^{Learning}_{Obj.\ Clust.}$
SPEA^{UX}	0	-3	2	4	13	14	15	14	13	12	12	8	9	-8	-4	-8
SPEA^{1X}	3	0	1	3	16	16	16	16	16	14	14	8	10	-4	-2	-6
NSGA-II^{UX}	-2	-1	0	1	8	8	9	8	7	5	10	3	4	-4	-2	-7
NSGA-II^{1X}	-4	-3	-1	0	7	9	8	8	7	5	11	2	3	-8	-6	-9
$\mathbb{M}^{UX}_{1\ Cluster}$	-13	-16	-8	-7	0	-3	0	0	-11	-7	1	-2	-9	-13	-16	-13
$\mathbb{M}^{1X}_{1\ Cluster}$	-14	-16	-8	-9	3	0	1	0	-8	-11	3	-2	-5	-13	-14	-14
$\mathbb{M}^{UX}_{Par.\ Clust.}$	-15	-16	-9	-8	0	-1	0	-1	-11	-8	2	-3	-8	-15	-16	-14
$\mathbb{M}^{1X}_{Par.\ Clust.}$	-14	-16	-8	-8	0	0	1	0	-9	-12	2	-2	-6	-15	-16	-14
$\mathbb{M}^{UX}_{Obj.\ Clust.}$	-13	-16	-7	-7	11	8	11	9	0	1	5	1	-3	-13	-14	-15
$\mathbb{M}^{1X}_{Obj.\ Clust.}$	-12	-14	-5	-5	7	11	8	12	-1	0	8	4	-3	-11	-12	-12
$\mathbb{M}^{Univariate}_{1\ Cluster}$	-12	-14	-10	-11	-1	-3	-2	-2	-5	-8	0	-6	-9	-11	-15	-12
$\mathbb{M}^{Learning}_{1\ Cluster}$	-8	-8	-3	-2	2	2	3	2	-1	-4	6	0	2	-7	-6	-11
$\mathbb{M}^{Univariate}_{Par.\ Clust.}$	-9	-10	-4	-3	9	5	8	6	3	3	9	-2	0	-11	-7	-8
$\mathbb{M}^{Learning}_{Par.\ Clust.}$	8	4	4	8	13	13	15	15	13	11	11	7	11	0	5	-3
$\mathbb{M}^{Univariate}_{Obj.\ Clust.}$	4	2	2	6	16	14	16	16	14	12	15	6	7	-5	0	-4
$\mathbb{M}^{Learning}_{Obj.\ Clust.}$	8	6	7	9	13	14	14	14	15	12	12	11	8	3	4	0

Figure B.22: Number of times an improvement was found to be statistically significant in the FO performance indicator, summed over all tested problems. The numbers in a single row indicate the summed number of significantly better or worse results compared to the algorithms in the different columns.

Bibliography

Akaike, H. (1973). Information theory and an extension of the maximum likelihood principle, *in* P. N. Petrov and F. Csaki (eds), *Proceedings of the Second International Symposium on Information Theory*, Akademia Kiado, Budapest, pp. 267–281.

Anderson, T. W. (1958). *An Introduction to Multivariate Statistical Analysis*, John Wiley & Sons Inc., New York, New York.

Bäck, T. and Schwefel, H.-P. (1993). An overview of evolutionary algorithms for parameter optimization, *Evolutionary Computation* **1**(1): 1–23.

Bagley, J. D. (1967). *The behavior of adaptive systems which employ genetic and correlation algorithms*, PhD thesis, University of Michigan, Ann Arbor, Michigan.

Baluja, S. and Caruana, R. (1995). Removing the genetics from the standard genetic algorithm, *in* A. Prieditis and S. Russell (eds), *Proceedings of the Twelfth International Conference on Machine Learning*, Morgan Kauffman, Madison, Wisconsin, pp. 38–46.

Baluja, S. and Davies, S. (1997). Using optimal dependency–trees for combinatorial optimization: Learning the structure of the search space, *in* D. H. Fisher (ed.), *Proceedings of the 1997 International Conference on Machine Learning*, Morgan Kauffman, Madison, Wisconsin, pp. 30–38.

Barron, A. R. and Cover, T. M. (1991). Minimum complexity density estimation., *IEEE Transactions on Information Theory* **37**: 1034–1054.

Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization, *ORSA Journal on Computing* **6**: 154–160.

Bernardo, J. M. and Smith, A. F. M. (1994). *Bayesian Theory*, John Wiley & Sons Inc., New York, New York.

Bilmes, J. (1997). A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden markov models, Technical Report ICSI TR-97-021, Department of Electrical Engineering, University of Berkeley, Berkeley, California.

Bosman, P. A. N. and Thierens, D. (1999a). Linkage information processing in distribution estimation algorithms, *in* W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela and R. E. Smith (eds), *Proceedings of the Genetic and Evolutionary Computation Conference GECCO–1999*, Morgan Kaufmann, San Francisco, California, pp. 60–67.

Bosman, P. A. N. and Thierens, D. (1999b). On the modelling of evolutionary algorithms, *in* E. Postma and M. Gyssens (eds), *Proceedings of the Eleventh Belgium–Netherlands Conference on Artificial Intelligence BNAIC–1999*, pp. 67–74.

Bosman, P. A. N. and Thierens, D. (2000a). Continuous iterated density estimation evolutionary algorithms within the IDEA framework, *in* M. Pelikan, H. Mühlenbein and A. O. Rodriquez (eds), *Proceedings of the Optimization by Building and Using Probabilistic Models OBUPM Workshop at the Genetic and Evolutionary Computation Conference GECCO–2000*, Morgan Kaufmann, San Francisco, California, pp. 197–200.

Bosman, P. A. N. and Thierens, D. (2000b). Expanding from discrete to continuous estimation of distribution algorithms: The IDEA, *in* M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo and H.-P. Schwefel (eds), *Parallel Problem Solving from Nature – PPSN VI*, Springer–Verlag, Berlin, pp. 767–776.

Bosman, P. A. N. and Thierens, D. (2000c). Negative log–likelihood and statistical hypothesis testing as the basis of model selection in IDEAs, *in* A. Feelders (ed.), *Proceedings of the Tenth Belgium–Netherlands Conference on Machine Learning*, Tilburg University, Tilburg, pp. 109–116.

Bosman, P. A. N. and Thierens, D. (2001a). Advancing continuous IDEAs with mixture distributions and factorization selection metrics, *in* M. Pelikan and K. Sastry (eds), *Proceedings of the Optimization by Building and Using Probabilistic Models OBUPM Workshop at the Genetic and Evolutionary Computation Conference GECCO–2001*, Morgan Kaufmann, San Francisco, California, pp. 208–212.

Bosman, P. A. N. and Thierens, D. (2001b). Crossing the road to efficient IDEAs for permutation problems, *in* L. Spector, E. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, M. H. G. S. Pezeshk and E. Burke (eds), *Proceedings of the Genetic and Evolutionary Computation Conference – GECCO–2001*, Morgan Kaufmann, San Francisco, California, pp. 219–226.

Bosman, P. A. N. and Thierens, D. (2001c). Exploiting gradient information in continuous iterated density estimation evolutionary algorithms, *in* B. Kröse, M. de Rijke, G. Schreiber and M. van Someren (eds), *Proceedings of the Thirteenth Belgium–Netherlands Artificial Intelligence Conference BNAIC–2001*, pp. 69–76.

Bosman, P. A. N. and Thierens, D. (2001d). New IDEAs and more ICE by learning and using unconditional permutation factorizations., *Late–Breaking Papers of the Genetic and Evolutionary Computation Conference GECCO–2001*, pp. 16–23.

Bosman, P. A. N. and Thierens, D. (2002a). Multi–objective optimization with diversity preserving mixture–based iterated density estimation evolutionary algorithms, *International Journal of Approximate Reasoning* **31**: 259–289.

Bosman, P. A. N. and Thierens, D. (2002b). Permutation optimization by iterated estimation of random keys marginal product factorizations, *in* J. J. Merelo, P. Adamidis, H.-G. Beyer, J.-J. Fernández-Villicañas and H.-P. Schwefel (eds), *Parallel Problem Solving from Nature – PPSN VII*, Springer–Verlag, Berlin, pp. 331–340.

Bosman, P. A. N. and Thierens, D. (2003). The balance between proximity and diversity in multi–objective evolutionary algorithms, *IEEE Transactions on Evolutionary Computation*. Accepted for publication.

Bosman, P. and Thierens, D. (2002c). A thorough documentation of obtained results on real-valued continuous and combinatorial multi-objective optimization problems using diversity preserving mixture-based iterated density estimation evolutionary algorithms, Technical report UU-CS-2002-52, Institute of Information and Computing Sciences, Utrecht University, Utrecht.

Brandt, S. (1970). *Statistical and Computation Methods in Data Analysis*, North-Holland, Amsterdam.

Buntine, W. (1994). Operations for learning with graphical models, *Journal of Artificial Intelligence Research* **2**: 159–225.

Buntine, W. (1996). A guide to the literature on learning probabilistic networks from data, *IEEE Transactions On Knowledge And Data Engineering* **8**: 195–210.

Casella, G. and Berger, R. L. (1990). *Statistical Inference*, Wadsworth & Brooks/Cole, Belmont, California.

Chernoff, H. (1954). On the distribution of the likelihood ratio, *Annals of Mathematical Statistics* **25**: 573–578.

Chow, C. K. and Liu, C. N. (1968). Approximating discrete probability distributions with dependence trees, *IEEE Transactions on Information Theory* **14**: 462–467.

Coello Coello, C. A. (1999). A comprehensive survey of evolutionary-based multiobjective optimization techniques, *Knowledge and Information Systems.* **1**(3): 269–308.

Cover, T. M. and Thomas, J. A. (1991). *Elements of Information Theory*, John Wiley & Sons Inc., New York, New York.

Darwin, C. (1859). *On the Origin of Species*, John Murray, London.

Das, I. and Dennis, J. (1997). A closer look at the drawbacks of minimizing weighted sums of objectives for pareto set generation in multicriteria optimization problems, *Structural Optimization* **14**(1): 63–69.

Davis, L. (1985). Applying adaptive algorithms to epistatic domains, *in* A. Joshi (ed.), *Proceedings of the Nineth International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, Los Angeles, California, pp. 162–164.

Dawid, A. P. and Lauritzen, S. L. (1993). Hyper Markov laws in the statistical analysis of decomposable graphical models, *Annals of Statistics* **21**: 1272–1317.

de Bonet, J. S., Isbell, C. and Viola, P. (1996). MIMIC: Finding optima by estimating probability densities, *in* M. C. Mozer, M. I. Jordan and T. Petsche (eds), *Advances in Neural Information Processing*, Vol. 9, The MIT Press, Cambridge, Massachusetts, p. 424.

de Jong, K. A. (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, PhD thesis, University of Michigan, Ann Arbor, Michigan.

Deb, K. (1999). Multi-objective genetic algorithms: Problem difficulties and construction of test problems, *Evolutionary Computation* **7**(3): 205–230.

Deb, K., Agrawal, S., Pratab, A. and Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II, *in* M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo and H.-P. Schwefel (eds), *Parallel Problem Solving from Nature – PPSN VI*, Springer–Verlag, Berlin, pp. 849–858.

Deb, K. and Goldberg, D. E. (1994). Sufficient conditions for deception in arbitrary binary functions, *Annals of Mathematics and Artificial Intelligence* **10**(4): 385–408.

Deb, K., Pratap, A. and Meyarivan, T. (2001). Constrained test problems for multi–objective evolutionary optimization, *in* E. Zitzler, K. Deb, L. Thiele, C. A. Coello Coello and D. Corne (eds), *First International Conference on Evolutionary Multi–Criterion Optimization*, Springer–Verlag, Berlin, pp. 284–298.

DeGroot, M. H. (1970). *Optimal Statistical Decisions*, McGraw–Hill, New York, New York.

Dempster, A. P., Laird, N. M. and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm, *Journal of the Royal Statistic Society* **Series B 39**: 1–38.

Edmonds, J. (1967). Optimum branchings, *Journal of Research of the National Bureau of Standards* **71B**: 233–240. Reprinted in Math. of the Decision Sciences, *Amer. Math. Soc. Lectures in Appl. Math.*, 11:335–345, 1968.

Edwards, D. (1995). *Introduction to Graphical Modelling*, Springer–Verlag, Berlin.

Efron, B. and Tibshirani, R. (1991). Statistical data analysis in the computer age, *Science* **253**: 390–395.

Ehrgott, M. and Gandibleux, X. (2000). An annotated bibliography of multi–objective combinatorial optimization, Technical Report 62/2000, Fachbereich Mathematik, Universität Kaiserslautern, Kaiserslautern.

Etxeberria, R. and Larrañaga, P. (1999). Global optimization using bayesian networks, *in* A. A. O. R. M. R. S. Ortiz and R. S. Hermida (eds), *Proceedings of the Second Symposium on Artificial Intelligence CIMAF–1999*, Institute of Cybernetics, Mathematics and Physics, pp. 332–339.

Feller, W. (1968). *An Introduction to Probability Theory and its Applications, Volume 1*, John Wiley & Sons Inc., New York, New York.

Fogel, D. B. (1992). *Evolving Artificial Intelligence*, PhD thesis, University of California, San Diego, California.

Fogel, L. J., Owens, A. J. and Walsh, M. J. (1966). *Artificial Intelligence through Simulated Evolution*, John Wiley & Sons Inc., New York, New York.

Frantz, D. R. (1972). *Non–linearities in genetic adaptive search*, PhD thesis, University of Michigan, Ann Arbor, Michigan.

Friedman, N. and Goldszmidt, M. (1996). Learning Bayesian networks with local structure, *in* E. Horvits and F. Jensen (eds), *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence UAI–1996*, Morgan Kaufmann, San Francisco, California, pp. 252–262.

Fung, R. M. and Crawford, S. L. (1990). A system for induction of probabilistic models, *Eighth National Conference on Artificial Intelligence*, American Association for Artificial Intelligence, Boston, Massachusetts, pp. 762–779.

Gallagher, M., Fream, M. and Downs, T. (1999). Real–valued evolutionary optimization using a flexible probability density estimator, *in* W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela and R. E. Smith (eds), *Proceedings of the Genetic and Evolutionary Computation Conference GECCO–1999*, Morgan Kaufmann, San Francisco, California, pp. 840–846.

Geiger, D. (1992). An entropy-based learning algorithm of Bayesian conditional trees, *in* D. Dubois, M. P. Wellman, B. D'Ambrosio and P. Smets (eds), *Proceedings of the Eighth Conference on Uncertainty in Artificial Intelligence UAI–1992*, Morgan Kaufmann, San Mateo, California, pp. 92–97.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learing*, Addison Wesley, Reading, Massachusetts.

Goldberg, D. E. and Lingle, Jr., R. (1985). Alleles, loci, and the traveling salesman problem, *in* J. J. Grefenstette (ed.), *Proceedings of the First International Conference on Genetic Algorithms and their Applications*, Lawrence Erlbaum Associates, Pittsburgh, Pennsylvania, pp. 154–159.

Goldberg, D. E., Deb, K., Kargupta, H. and Harik, G. (1993). Rapid, accurate optimization of difficult problems using fast messy genetic algorithms, *in* S. Forrest (ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, California, pp. 56–64.

Goldberg, D. E., Korb, B. and Deb, K. (1989). Messy genetic algorithms: Motivation, analysis and first results, *Complex Systems* **10**(5): 385–408.

Harik, G. (1999). Linkage learning via probabilistic modeling in the ECGA, Technical Report 99010, IlliGAL, University of Illinois, Urbana, Illinois.

Harik, G. and Goldberg, D. E. (2000). Linkage learning through probabilistic expression, *Comp. methods in applied mechanics and engineering* **186**: 295–310.

Harik, G., Cantú-Paz, E., Goldberg, D. E. and Miller, B. L. (1999). The gambler's ruin problem, genetic algorithms, and the sizing of populations, *Evolutionary Computation* **7**(3): 231–253.

Harik, G., Lobo, F. and Goldberg, D. E. (1998). The compact genetic algorithm, *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, IEEE Press, Piscataway, New Jersey, pp. 523–528.

Harik, G. R. (1997). *Learning Gene Linkage to Efficiently Solve Problems of Bounded Difficulty Using Genetic Algorithms*, PhD thesis, University of Michigan, Ann Arbor, Michigan.

Harik, G. R. and Goldberg, D. E. (1997). Learning linkage, *in* R. K. Belew and M. D. Vose (eds), *Foundations of Genetic Algorithms 4*, Morgan Kaufmann, San Francisco, California, pp. 247–262.

Hartigan, J. (1975). *Clustering Algorithms*, John Wiley & Sons, Inc., New York, New York.

Heckerman, D. and Geiger, D. (1995). Learning Bayesian networks: A unification for discrete and Gaussian domains, *in* P. Besnard and S. Hanks (eds), *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence UAI–1995*, Morgan Kaufmann, San Mateo, California, pp. 274–284.

Hestenes, M. R. and Stiefel, E. (1952). Methods of conjugate gradients for solving linear systems, *Journal of Research of the National Bureau of Standards* **6**(49): 409–436.

Holland, J. H. (1975). *Adaptation in Natural and Artifical Systems*, University of Michigan Press, Ann Arbor, Michigan.

Jin, Y., Olhofer, M. and Sendhoff, B. (2001). Dynamic weighted aggregation for evolutionary multi-objective optimization: Why does it work and how?, *in* L. Spector, E. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, M. H. G. S. Pezeshk and E. Burke (eds), *Proceedings of the Genetic and Evolutionary Computation Conference – GECCO–2001*, Morgan Kaufmann, San Francisco, California, pp. 1042–1049.

Kargupta, H. (1995). *SEARCH, Polynomial Complexity, And The Fast Messy Genetic Algorithm*, PhD thesis, University of Illinois, Urbana, Illinois.

Kargupta, H. (1996). The gene expression messy genetic algorithm, *in* S. Forrest (ed.), *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, IEEE Press, Piscataway, New Jersey, pp. 631–636.

Kargupta, H. (2000). The genetic code–like transformations and their effect on learning functions, *in* M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo and H.-P. Schwefel (eds), *Parallel Problem Solving from Nature – PPSN VI*, Springer–Verlag, Berlin, pp. 99–108.

Kargupta, H. and Bandyopadhyay, S. (1998). Further experimentations on the scalability of the GEMGA, *in* A. E. Eiben, T. Bäck, M. Schoenauer and H.-P. Schwefel (eds), *Parallel Problem Solving from Nature – PPSN V*, Springer–Verlag, Berlin, pp. 315–324.

Kargupta, H., Deb, K. and Goldberg, D. E. (1992). Ordering genetic algorithms and deception, *in* R. Männer and B. Manderick (eds), *Parallel Problem Solving from Nature – PPSN II*, Springer–Verlag, Berlin, pp. 47–56.

Kass, R. E. and Raftery, A. E. (1995). Bayes factors and model uncertainty, *Journal of the American Statistical Association* **90**: 773–795.

Kendall, M. G. and Stuart, A. (1967). *The Advanced Theory of Statistics, Volume 2, Inference And Relationship*, Charles Griffin & Company Limited, London.

Knjazew, D. (2000). Application of the fast messy genetic algorithm to permutation and scheduling problems, Technical Report 2000022, IlliGAL, University of Illinois, Urbana, Illinois.

Knjazew, D. and Goldberg, D. (2000). Large–scale permutation optimization with the ordering messy genetic algorithm, *in* M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. Merelo and H.-P. Schwefel (eds), *Parallel Problem Solving from Nature – PPSN VI*, Springer–Verlag, Berlin, pp. 631–640.

Knowles, J. and Corne, D. (2002). On metrics for comparing non–dominated sets, *Proceedings of the 2002 Congress on Evolutionary Computation CEC 2002*, IEEE Press, Piscataway, New Jersey, pp. 666–674.

Knuth, D. E. (1981). *Semi–numerical Algorithms*, Vol. 2 of *The Art of Computer Programming*, Addison Wesley, Reading, Massachusetts.

Koza, J. R. (1992). *Genetic Programming: on the Programming of Computers by Means of Natural Selection*, The MIT Press.

Kullback, S. (1959). *Information Theory and Statistics*, John Wiley & Sons Inc., New York, New York.

Larrañaga, P. and Lozano, J. A. (2001). *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation.*, Kluwer Academic, London.

Larrañaga, P., Etxeberria, R., Lozano, J. A. and Peña, J. M. (2000). Optimization in continuous domains by learning and simulation of gaussian networks., *in* M. Pelikan, H. Mühlenbein and A. O. Rodriquez (eds), *Proceedings of the Optimization by Building and Using Probabilistic Models OBUPM Workshop at the Genetic and Evolutionary Computation Conference GECCO–2000*, Morgan Kaufmann, San Francisco, California, pp. 201–204.

Lauritzen, S. L. (1996). *Graphical Models*, Clarendon Press, Oxford.

Lobo, F. G., Deb, K., Goldberg, D. E., Harik, G. R. and Wang, L. (1998). Compressed introns in a linkage learning genetic algorithm, *in* W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba and R. Riolo (eds), *Genetic Programming 1998: Proceedings of the Third Annual Conference*, Morgan Kauffman, San Francisco, California, pp. 551–558.

Mahalanobis, P. C. (1930). On tests and measures of groups divergence I, *Journal of the Asiatic Society of Benagal* **26**: 541.

Meila, M. and Jordan, M. I. (1998). Estimating dependency structure as a hidden variable, *in* M. I. J. M. J. Kearns and S. A. Solla (eds), *Advances in Neural Information Processing Systems*, Vol. 10, The MIT Press, Cambridge, Massachusetts, pp. 584–590.

Morse, J. N. (1980). Reducing the size of the nondominated set: Pruning by clustering, *Computers and Operations Research* **7**(1–2): 55–66.

Mühlenbein, H. and Mahnig, T. (1999). FDA – a scalable evolutionary algorithm for the optimization of additively decomposed functions, *Evolutionary Computation* **7**(4): 353–376.

Mühlenbein, H. and Paaß, G. (1998). From recombination of genes to the estimation of distributions I. binary parameters, *in* A. E. Eiben, T. Bäck, M. Schoenauer and H.-P. Schwefel (eds), *Parallel Problem Solving from Nature – PPSN V*, Springer–Verlag, Berlin, pp. 178–187.

Mühlenbein, H., Mahnig, T. and Ochoa, A. (1999). Schemata, distributions and graphical models in evolutionary optimization, *Journal of Heuristics* **5**: 215–247.

Ochoa, A., Mühlenbein, H. and Soto, M. (2000). A factorized distribution algorithm using single connected Bayesian networks, *in* M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo and H.-P. Schwefel (eds), *Parallel Problem Solving from Nature – PPSN VI*, Springer–Verlag, Berlin, pp. 787–796.

Oliver, I. M., Smith, D. J. and Holland, J. R. C. (1987). A study of permutation crossover operators on the traveling salesman problem, *in* J. J. Grefenstette (ed.), *Proceedings of the Second International Conference on Genetic Algorithms and their Applications*, Lawrence Erlbaum Associates, Cambridge, Massachusetts, pp. 224–230.

Pelikan, M. and Goldberg, D. E. (2000). Genetic algorithms, clustering, and the breaking of symmetry, *in* M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo and H.-P. Schwefel (eds), *Parallel Problem Solving from Nature – PPSN VI*, Springer–Verlag, Berlin, pp. 385–394.

Pelikan, M. and Goldberg, D. E. (2001). Escaping hierarchical traps with competent genetic algorithms, *in* L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon and E. Burke (eds), *Proceedings of the GECCO–2001 Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, San Francisco, California, pp. 511–518.

Pelikan, M. and Mühlenbein, H. (1999). The bivariate marginal distribution algorithm, *in* R. Roy, T. Furuhashi, K. Chawdry and K. Pravir (eds), *Advances in Soft Computing - Engineering Design and Manufacturing*, Springer–Verlag, Berlin, pp. 521–535.

Pelikan, M., Goldberg, D. E. and Cantú-Paz, E. (1999). BOA: The Bayesian optimization algorithm, *in* W. Banzhaf, J. Daida, A. Eiben, M. Garzon, V. Honavar, M. Jakiela and R. Smith (eds), *Proceedings of the GECCO–1999 Genetic and Evolutionary Computation Conference*, Morgan Kauffman, San Francisco, California, pp. 525–532.

Pelikan, M., Goldberg, D. E. and Sastry, K. (2001). Bayesian optimization algorithm, decision graphs and occam's razor, *in* L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo,

S. Pezeshk, M. H. Garzon and E. Burke (eds), *Proceedings of the GECCO–2001 Genetic and Evolutionary Computation Conference*, Morgan Kauffman, San Francisco, California, pp. 519–526.

Pelikan, M., Goldberg, D. E. and Tsutsui, S. (2002). Combining the strengths of the Bayesian optimization algorithm and adaptive evolution strategies, *in* W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke and N. Jonoska (eds), *Proceedings of the GECCO–2002 Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, San Francisco, California, pp. 512–519.

Poland, W. B. and Shachter, R. D. (1994). Three approaches to probability model selection, *in* R. L. de Mantaras and D. Poole (eds), *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence UAI–1994*, Morgan Kaufmann, San Francisco, California, pp. 478–483.

Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P. (1992). *Numerical Recipies in C: The Art of Scientific Computing*, Cambridge University Press, Cambridge, Massachusetts.

Priebe, C. E. (1994). Adaptive mixtures, *Journal of the American Statistical Association* **89**(427): 796–806.

Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der Biologischen Evolution*, Frommann–Holzboog, Stuttgart.

Rissanen, J. (1987). Stochastic complexity., *Journal of the Royal Statistical Society series B.* **49**(3): 223–239 & 252–265.

Robles, V., de Miguel, P. and Larrañaga, P. (2001). Solving the traveling salesman problem with EDAs, *in* P. Larrañaga and J. Lozano (eds), *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation.*, Kluwer Academic, London.

Rudlof, S. and Köppen, M. (1996). Stochastic hill climbing with learning by vectors of normal distributions, *in* T. Furuhashi (ed.), *Proceedings of the First Online Workshop on Soft Computing (WSC1)*, Nagoya University, Nagoya, Japan, pp. 60–70.

Santana, R., Ochoa, A. and Soto, M. R. (2001). The mixture of trees factorized distribution algorithm, *in* L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon and E. Burke (eds), *Proceedings of the GECCO–2001 Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, San Francisco, California, pp. 543–550.

Schwarz, G. (1978). Estimating the dimension of a model, *Annals of Statistics* **6**(2): 461–464.

Schwefel, H.-P. (1965). *Kybernetische evolution als strategie der experimentelen forschung in der strömungstechnik*, Master's thesis, Technical University of Berlin, Berlin.

Schwefel, H.-P. (1981). *Numerical Optimization of Computer Models*, John Wiley & Sons Inc., New York, New York.

Sclove, S. L. (1994). Small– and large–sample statistical model–selection criteria, *in* P. Cheeseman and R. Oldford (eds), *Selecting Models from Data: Artificial Intelligence and Statistics IV*, Springer–Verlag, Berlin, pp. 31–39.

Sebag, M. and Ducoulombier, A. (1998). Extending population–based incremental learning to continuous search spaces, *in* A. E. Eiben, T. Bäck, M. Schoenauer and H.-P. Schwefel (eds), *Parallel Problem Solving from Nature – PPSN V*, Springer–Verlag, Berlin, pp. 418–427.

Servet, I., Trave-Massuyes, L. and Stern, D. (1997). Telephone network traffic overloading diagnosis and evolutionary computation technique, *in* J. K. Hao, E. Lutton, E. Ronald, M. Schoenauer and D. Snyers (eds), *Proceedings of Artificial Evolution '97*, Springer–Verlag, Berlin, pp. 137–144.

Shin, S.-Y., Cho, D.-Y., and Zhang, B.-T. (2001). Function optimization with latent variable models, *in* A. Ochoa, H. Mühlenbein, T. English and P. Larrañaga (eds), *Proceedings of the Third International Symposium on Adaptive Systems ISAS–2001 — Evolutionary Computation and Probabilistic Graphical Models*, Institute of Cybernetics, Mathematics and Physics, pp. 145–152.

Soto, M. and Ochoa, A. (2000). A factorized distribution algorithm based on polytrees, *Proceedings of the 2000 Congress on Evolutionary Computation CEC00*, IEEE Press, Piscataway, New Jersey, pp. 232–237.

Tanner, M. A. (1992). *Tools for Statistical Inference*, Springer–Verlag, Berlin.

Tatsuoka, M. M. (1971). *Multivariate Analysis: Techniques for Educational and Psychological Research*, John Wiley & Sons Inc., New York, New York.

Thierens, D. (1995). *Analysis and Design of Genetic Algorithms*, PhD thesis, University of Leuven, Leuven, Belgium.

Thierens, D. and Bosman, P. A. N. (2001a). Multi–objective optimization with iterated density estimation evolutionary algorithms using mixture models, *in* A. Ochoa, H. Mühlenbein, T. English and P. Larrañaga (eds), *Proceedings of the Third International Symposium on Adaptive Systems ISAS–2001 — Evolutionary Computation and Probabilistic Graphical Models*, Institute of Cybernetics, Mathematics and Physics, pp. 129–136.

Thierens, D. and Bosman, P. A. N. (2001b). Multi-objective mixture-based iterated density estimation evolutionary algorithms, *in* L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo,

S. Pezeshk, M. H. Garzon and E. Burke (eds), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, Morgan Kaufmann, San Francisco, California, pp. 663–670.

Trivedi, K. (1982). *Probability & Statistics with Reliability, Queuing and Computer Science Applications*, Prentice-Hall, Englewood Cliffs, New Jersey.

Tsutsui, S., Pelikan, M. and Goldberg, D. E. (2001). Evolutionary algorithm using marginal histogram in continuous domain, *in* M. Pelikan and K. Sastry (eds), *Proceedings of the Optimization by Building and Using Probabilistic Models OBUPM Workshop at the Genetic and Evolutionary Computation Conference GECCO–2001*, Morgan Kaufmann, San Francisco, California, pp. 230–233.

van Kemenade, C. H. M. (1998). Building block filtering and mixing, *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, IEEE Press, Piscataway, New Jersey, pp. 505–510.

Van Veldhuizen, D. A. (1999). *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations.*, PhD thesis, Graduate School of Engineering of the Air Force Institute of Technology, WPAFB, Ohio.

Vapnik, V. (1995). *The Nature of Statistical Learning Theory*, Springer–Verlag, Berlin.

Vapnik, V. and Mukherjee, S. (1999). Support vector method for multivariate density estimation, *in* D. Touretzky, M. Mozer and M. Hasselmo (eds), *Neural Information Processing Systems*, The MIT Press, Cambridge, Massachusetts, pp. 659–666.

Wallace, C. S. and Freeman, P. R. (1987). Estimation and inference by compact encoding, *Journal of the Royal Statistical Society B* **49**(3): 240–265.

Whitley, D., Starkweather, T. and Fuquay, D. (1989). Scheduling problems and traveling salesman: The genetic edge recombination operator, *in* J. D. Schaffer (ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufman, San Mateo, California.

Yin, X. and Germay, N. (1993). A fast genetic algorithm with sharing scheme using cluster analysis methods in multimodal function optimization, *Proceedings of the 1993 International Conference on Artificial Neural Nets and Genetic Algorithms*, Inssbruck, Austria, pp. 450–457.

Zhang, B.-T. and Shin, S.-Y. (2000). Bayesian evolutionary optimization using helmholtz machines, *in* M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo and H.-P. Schwefel (eds), *Parallel Problem Solving from Nature – PPSN VI*, Springer–Verlag, Berlin, pp. 827–836.

Zitzler, E. (1999). *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications*, PhD thesis, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland.

Zitzler, E. and Thiele, L. (1999). Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach, *IEEE Transactions on Evolutionary Computation* **3**(4): 257–271.

Zitzler, E., Deb, K. and Thiele, L. (2000). Comparison of multiobjective evolutionary algorithms: Empirical results, *Evolutionary Computation* **8**(2): 173–195.

Zitzler, E., Deb, K., Thiele, L., Coello Coello, C. A. and Corne, D. (2001). *Evolutionary Multi-Criterion Optimization First International Conference, EMO 2001 (Proceedings)*, Springer–Verlag, Berlin.

Zitzler, E., Laumanns, M., Thiele, L., Fonseca, C. M. and da Fonseca, V. G. (2002). Why quality assessment of multiobjective optimizers is difficult, *in* W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke and N. Jonoska (eds), *Proceedings of the GECCO–2002 Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, San Francisco, California, pp. 666–674.

Samenvatting

In de evolutionaire rekenkunde (*Evolutionary Computation* (EC)) wordt de natuurlijke evolutie gebruikt als een metafoor, op basis waarvan optimalisatie algoritmen geconstrueerd worden. In dergelijke evolutionaire algoritmen (*Evolutionary Algorithms* (EAs)) zijn abstracties van de meest belangrijke onderdelen van de natuurlijke evolutie geïmplementeerd.

Natuurlijke evolutie kan gezien worden als een efficiënt en parallel optimalisatie proces dat gebaseerd is op het principe van natuurlijke selectie en genetische overerfing. Natuurlijke selectie vindt plaats doordat betere organismen een grotere kans hebben om te overleven, hetgeen ook wel bekend staat als *survival of the fittest*. Door te paren produceren organismen die overleven nieuwe organismen. Als belangrijke eigenschappen van de ouders geërfd worden, zullen de nieuwe organismen ook een grotere kans hebben om te overleven.

Ieder organisme in de natuur heeft een unieke DNA codering. DNA kan gezien worden als een string of sequentie van informatie die alles beschrijft wat er fysiek gezien te weten is over het organisme dat gecodeerd wordt. Een dergelijke string wordt ook wel het *genotype* genoemd, terwijl de fysieke vorm van het organisme het *fenotype* genoemd wordt. Het fenotype bepaalt hoe goed een organisme zich kan gedragen in zijn omgeving, hetgeen de *fitness* van het organisme wordt genoemd. Het ontstaan van nieuwe organismen door het paren van ouders, kan gezien worden als een proces waarin genetisch materiaal geërfd wordt door een nieuw organisme. Vanuit dit standpunt is het maken van een nieuw genotype equivalent aan het maken van een combinatie van de genotypes van de ouders. Dit proces wordt ook wel *recombinatie* genoemd.

Vanuit een optimalisatie standpunt kunnen de fenotypes gezien worden als oplossingen voor een optimalisatie probleem. De bijbehorende genotypes zijn coderingen waarmee de recombinatie operator in het EA werkt. Het doel van het EA is het vinden van een genotype met de best mogelijke fitness. Dit wordt bewerkstelligd door een verzameling van genotypes te gebruiken. Deze verzameling wordt ook wel een *populatie* genoemd. Met behulp van selectie in een EA worden de betere genotypes uit de populatie gekozen op basis van hun fitness. Vervolgens wordt recombinatie toegepast om een verzameling van nieuwe genotypes te construeren. De nieuwe genotypes worden volgens een bepaald schema opgenomen in de huidige populatie zodanig dat de omvang van de populatie niet toeneemt. Dit proces van selectie en recombinatie wordt herhaald totdat aan een zeker terminatie criterium voldaan is.

Het genetisch algoritme (*Genetic Algorithm* (GA)) is een van de meest bekende EAs. In het klassieke GA is het genotype een binaire string. Een klassieke recombinatie operator voor GAs is *uniform crossover* (UX). Het construeren van een nieuw genotype met een *crossover* operator vindt in het algemeen plaats door verschillende delen van het genotype direct te erven van een ouder genotype. Het nieuwe genotype is dus een directe *mix* van het genetisch materiaal van de ouders. In uniform crossover wordt voor ieder bit afzonderlijk random bepaald of het bit geërfd moet worden van de eerste ouder of van de tweede ouder. Bij gebruik van deze recombinatie operator is de kans zeer klein dat een bepaalde combinatie van bits, die in een ouder genotype voorkomt, in zijn geheel wordt geërfd door een nieuw genotype. Hierdoor is deze operator niet geschikt voor het oplossen van optimalisatie problemen waarin groepen van bitposities een niet–lineaire contributie hebben aan de fitness van een genotype. Deze bitposities hangen sterk samen omdat de optimale combinatie van bits voor deze bitposities niet een combinatie is van suboplossingen die optimaal zijn. Hierdoor is het waarschijnlijker dat een optimale combinatie van bits voor dergelijke bitposities bewaard wordt door selectie en recombinatie dan dat deze combinatie geconstrueerd wordt door recombinatie. In het ergste geval is een dergelijke niet–lineaire fitness functie *deceptief* voor een GA. Een deceptieve functie heeft de eigenschap dat het GA misleid wordt zodat het suboptimum van de niet–lineaire functie gevonden wordt, behalve wanneer de kans erg groot is dat de recombinatie operator de bitposities van de niet–lineaire functie tegelijkertijd uit een enkele ouder overbrengt naar het nieuwe genotype. Voor recombinatie operatoren zonder deze eigenschap kan dit gedrag enkel worden tegen gegaan door de populatie veel groter te maken.

Hoeveel groter de populatie precies gemaakt moet worden, werd voor het eerst gekwantificeerd door Thierens (1995). Als de fitness een som is van functies van verzamelingen van mutueel exclusieve bitposities, zeggen we dat de fitness functie *additief decomponeerbaar* is. Thierens (1995) liet zien dat voor additief decomponeerbare problemen de populatie omvang exponentieel moet toenemen als het aantal deceptieve subfuncties toeneemt, wanneer de niet–lineaire samenhang van de subfuncties niet gerespecteerd wordt door de recombinatie operator. Dit resultaat heeft direct tot gevolg dat om dergelijke structuren in optimalisatie problemen te kunnen uitbuiten, om zo de exponentiële toename van de populatie omvang tegen te gaan, er een automatische identificatie van de structuur van het probleem gedaan zal moeten worden tijdens (evolutionaire) optimalisatie. Harik et al. (1999) hebben aangetoond dat wanneer de bitcombinaties voor de subfuncties niet met grote kans verstoord worden tijdens recombinatie, de populatie omvang slechts schaalt in de omvang van de wortel van het aantal subfuncties. Aldus, gegeven een competent, polynomiaal mechanisme voor het extraheren van de structuur van een probleem tijdens optimalisatie, zodanig dat deze informatie te gebruiken is in een goede recombinatie operator, moet het mogelijk zijn om beter schaalbare EAs te construeren.

Het belangrijkste onderzoeksonderwerp in dit proefschrift is het gebruik van automatische identificatie van de structuur van een probleem om de inductieve mogelijkheden van evolutionaire algoritmen te verbeteren zodat uiteindelijk

beter schaalbare en efficiëntere evolutionaire algoritmen ontworpen kunnen worden. Om dergelijke automatische identificatie te bewerkstelligen, richten we ons in dit proefschrift op het onderzoeken in welke mate kansverdelingen gebruikt kunnen worden. Het onderzoeksgebied van het schatten van kansverdelingen biedt een variëteit aan effectieve middelen om afhankelijkheden tussen random variabelen te modelleren om zo een goede beschrijving van data te verkrijgen. Dergelijke middelen bieden daarom de mogelijkheid om de decomponeerbare structuur van een probleem tijdens optimalisatie te detecteren en te modelleren. Door de recombinatie operator te vervangen door het schatten van de kansverdeling van de geselecteerde oplossingen en het vervolgens trekken van nieuwe oplossingen van deze kansverdeling, wordt een geïtereerd dichtheid–schattend evolutionair algoritme (*Iterated Density–Estimation Evolutionary Algorithm* (IDEA)) verkregen. Dergelijke algoritmen zijn vrij nieuw. Verschillende varianten zijn bekend onder verschillende acronymen die meestal gerelateerd zijn aan een specifiek type van kansverdeling dat gebruikt wordt.

In dit proefschrift onderzoeken we de bruikbaarheid van kansverdelingen in EAs voor het oplossen van verschillende soorten optimalisatie problemen. De technieken voor het schatten van kansverdelingen waartoe we ons beperken, zijn het gretig leren van multivariate en Bayesiaanse factorisaties en het leren van gemengde gefactoriseerde kansverdelingen door middel van clusteren. De optimalisatie problemen waarvoor we specifiek nieuwe dichtheid–schattende EAs ontwerpen en testen, beslaan de velden van numerieke optimalisatie, permutatie optimalisatie en de optimalisatie van problemen waarin er meer dan één doel is.

In hoofdstuk 1 wordt een algemene inleiding gegeven in optimalisatie en inductie. Ook wordt in dit hoofdstuk de doelstelling van het proefschrift beschreven en wordt een overzicht gegeven van de inhoud van de rest van het proefschrift.

In hoofdstuk 2 geven we een overzicht van de concepten uit de theorie van de kansrekening die relevant zijn voor het onderzoek in dit proefschrift. Kansverdelingen en probabilistische modellen worden in hun algemeenheid besproken, alsmede dichtheidsfuncties en factorisaties. Voorts worden ook eigenschappen van kansverdelingen zoals *likelihood* en entropie besproken. Vervolgens worden de methoden die in dit proefschrift gebruikt worden om factorisaties en gemengde factorisaties te schatten, behandeld.

In hoofdstuk 3 wordt het veld van de evolutionaire algoritmen beschreven en een gedetailleerde beschrijving van het GA gegeven. Verder wordt een algemene achtergrond op het gebied van GAs gegeven door een drietal resultaten over de werking van het GA te bespreken. Deze resultaten tonen de beperkingen van het klassieke GA aan zoals eerder in deze samenvatting genoemd.

In hoofdstuk 4 tonen we eerst aan dat het gebruik van gefactoriseerde kansverdelingen in plaats van crossover operatoren tot resultaten kunnen leiden die net zo goed zijn als wanneer de best mogelijke crossover operator gebruikt wordt. Daarna definiëren we het IDEA raamwerk waarin het gebruik van kansverdelingen in EAs geformaliseerd wordt. In de rest van het hoofdstuk wordt een overzicht gegeven van dergelijke algoritmen in de huidige gepubliceerde literatuur. Algoritmen van het IDEA type werden eerst voorgesteld voor binaire genotypes. In de eerste voorstellen werden relatief simpele gefactoriseerde

kansverdelingen gebruikt. Om er voor te zorgen dat polynomiale schaalbaarheid bereikt wordt voor additief decomponeerbare optimalisatie problemen, kan de maximale orde van interactie die in een factorisatie toegestaan is, niet beperkt worden. Een empirische validatie hiervan wordt ook gegeven in dit hoofdstuk.

In hoofdstuk 5 wordt de toepassing van IDEAs op numerieke optimalisatie problemen beschreven. Om dit te bewerkstelligen, worden reële continue kansverdelingen geschat. Zowel Bayesiaanse als multivariate factorisaties worden gebruikt, alsmede gemengde gefactoriseerde kansverdelingen door middel van clusteren. De kansverdelingen die gebruikt worden om de factorisaties op te baseren, zijn de normale verdeling, de normale kernels verdeling, de normale gemengde verdeling en de histogram verdeling. Alhoewel goede resultaten behaald kunnen worden met reële continue kansverdelingen, met name wanneer de dimensionaliteit van het probleem toeneemt, zijn er ook tekortkomingen in het gebruik van IDEAs voor numerieke optimalisatie. Het grootste probleem is dat eventuele lokale gradiënt informatie niet gebruikt wordt. Als convergentie zich begint voor te doen op een gebied in de buurt van het optimum, is er een grote kans dat het optimum zelf niet gevonden zal worden omdat er geen nieuwe oplossingen getrokken worden van de geschatte kansverdeling buiten de regio waarin de huidige geselecteerde oplossingen zich bevinden. Echter, het gebruik van reële continue IDEAs voor numerieke optimalisatie is een goede manier om globaal de structuur te extraheren en de aanwezigheid van interessante regionen in de zoekruimte te identificeren. Vanuit de regionen die door een IDEA geïdentificeerd worden, kan dan een lokale zoekprocedure gestart worden die de lokale gradiënt informatie gebruikt om snel en efficiënt een lokaal optimum te vinden. Deze hybridisatie van het IDEA raamwerk met het gebruik van een klassieke geconjugeerde gradiënten methode hebben we het GLIDE–EA (*Gradient Leveraged Iterated Density–Estimation Evolutionary Algorithm*) genoemd. We hebben aangetoond dat GLIDE in staat is om zeer goede resultaten te behalen op verscheidene moeilijke numerieke optimalisatie problemen.

Hoofdstuk 6 is gewijd aan het schatten van kansverdelingen voor permutaties om evolutionaire permutatie optimalisatie te verbeteren. Door een reële codering te gebruiken van permutaties, kunnen de IDEAs uit hoofdstuk 5 direct worden toegepast. Echter, vanwege de grote graad van redundantie die een dergelijke codering met zich meebrengt, worden de afhankelijkheden tussen de variabelen niet efficiënt gemodelleerd en verwerkt door het EA. Daarom stellen we voor om kansverdelingen direct over de permutaties te schatten en om deze informatie te gebruiken om nieuwe IDEAs te construeren. In dit hoofdstuk laten we zien hoe multivariate factorisaties geschat kunnen worden voor permutaties en hoe deze informatie gebruikt kan worden in IDEAs om nieuwe, efficiënte EAs te construeren voor permutatie optimalisatie. Een experimentele validatie geeft aan dat het schalings gedrag van de resulterende IDEAs polynomiaal is voor deceptieve additieve decomponeerbare permutatie optimalisatie problemen. Voor klassieke permutatie gebaseerde EAs is het schalings gedrag exponentieel.

In hoofdstuk 7 laten we zien hoe efficiënte instanties van het IDEA raamwerk gemaakt kunnen worden voor de optimalisatie van problemen met meerdere doelen. De beste variant van deze instanties draagt de specifieke naam MIDEA

(*Multi–Objective Mixture–Based* IDEA). In MIDEA worden gemengde kansverdelingen gebruikt om de constructie te stimuleren van een grote diversiteit aan oplossingen die van vergelijkbare kwaliteit zijn in het optimaliseren van meerdere doelen. Een speciale diversiteit behoudende selectie operator wordt gebruikt om er zeker van te zijn dat deze diversiteit niet verloren gaat. Evolutionaire algoritmen behoren tot de beste algoritmen voor meerdoelige optimalisatie omdat ze gebaseerd zijn op een populatie. Hierdoor zijn ze direct geschikt voor het vinden van meerdere oplossingen van gelijke kwaliteit zonder dat de zoek procedure opnieuw uitgevoerd hoeft te worden om nieuwe oplossingen te vinden. De MIDEA instanties zijn minstens zo efficiënt als de huidige beste meerdoelige EAs. MIDEAs presteren met name uitstekend in het verkrijgen en behouden van een diverse– en omvangrijke verzameling van oplossingen die allen even goed zijn in het optimaliseren van meerdere doelen. Dit is empirisch geverifiëerd in dit hoofdstuk door gebruik te maken van een verzameling van acht testproblemen.

Dit proefschrift wordt afgesloten in hoofdstuk 8 door te reflecteren op de toepasbaarheid van het schatten van kansverdelingen in de evolutionaire optimalisatie. Daarnaast worden ook richtlijnen voor verder onderzoek besproken.

In appendix A worden schattingen van een maximale likelihood gegeven voor verschillende typen van kansverdelingen in de conditionele vorm die nodig is voor het schatten van Bayesiaanse factorisaties.

In appendix B staan in tabelvorm de resultaten van de experimenten waarin IDEAs en andere EAs zijn gebruikt om de drie verschillende types van optimalisatie problemen op te lossen zoals gepresenteerd in hoofdstukken 5, 6 en 7.

'Long is the road from conception to completion.'
Molière

Acknowledgements

Being a Ph.D. student is much like embarking on a journey into uncertainty. Without proper guidance, the direction may stay unclear and landmarks may never be found. Such is the nature of research, as beforehand you never know for sure what you will discover. It is only by experience and insights that a firm course can be plotted. For the steadvast direction and the many results that I was able to obtain on my own journey into uncertainty, I owe a great thanks to my daily supervisor Dirk Thierens. Dirk, your knowledge, experience and great insights into Evolutionary Computation have been invaluable to me, my ideas and the completion of this thesis. I really enjoyed our many meetings and discussions and hope that our teamwork will endure for many years to come.

The Ph.D. journey ends with the writing of a thesis. Indeed, as stated in the quote above, the road from conception to completion is a long one. At times I found myself to be much like Bilbo Baggins, singing to myself: "The road goes ever on and on…". Fortunately, the road has now ended, and it has resulted in the completion of this thesis. I would like to thank my promotores Linda van der Gaag and Jan van Leeuwen for taking their time to provide detailed comments and suggestions that have helped to improve the organisation and readability of this thesis. I would also like to thank Steven van Dijk for proof reading the first chapters of the first version. Finally, thanks go out to the reading committee, Gusz Eiben, Pedro Larrañaga, Han La Poutré, Arno Siebes and Joos Vandewalle, for investing their time in reading my thesis and for using their expertise knowledge in evaluating it.

There are a few people in the research community that I have met during my Ph.D. journey that I would specifically like to address here. One of the nice aspects of being a Ph.D. student is that you can go to international conferences and meet with other researchers in the field. One of the interesting (and sometimes somewhat painful) aspects to researching IDEAs the past few years has been that during my Ph.D. project, the research on this type of algorithm has blossemed. As such it hasn't always been easy to ensure uniqueness of the research. Next to the interesting talks and discussions on IDEA–related topics I would like to thank Martin Pelikan for mainly staying in the binary domain during his Ph.D. research and thereby ensuring that we would be staying out of each other's way as much as possible. I'm sure we'll be running into each other again at conferences to come and I hope we'll have many more interesting discussions that will help keep our research apart! I would also like to mention

Franz Rothlauf. Meeting with you at a conference is always something to look forward to because it is always fun and always leads to nice discussions about both EC and non-EC topics. Finally, a special thanks goes out to Boštjan Likar. We shared an office during your visit at the Utrecht University. I remember that you wanted to try out each and every type of cooky the supermarket on campus had to offer and I sincerely enjoyed this curiosity of yours! Thank you for your support and enthusiasm with respect to my music in that same period. It is great that you and Tanja are in the same research community so that we can meet every now and then at a medical computer science conference.

Finally, I'd like to thank my family and friends for their support throughout the years. Specifically, many thanks go out to my parents for their continuous support in every aspect of my life. A special place is also reserved for my best friend Manodj Mathoera, who has sided with me for almost fifteen years now, making music every step of the way whenever possible. Thank you for helping in keeping the music alive, always.

It is funny how most of the time the people we love the most are mentioned last. One could argue that these people are already aware of your love and devotion towards them and need not to be placed front and center. I would like to look upon it from a "save the best for last" point of view. If anyone deserves the label "best" in all of this, it is Tanja. You have always played the most important role in my life and bring out the best in me. Funny detail is that the day I ended my graduation phase and embarked on this Ph.D. journey is the same day our joint journey began. Since then, we have been through so many things together that it is hard to imagine that it has only been four and a half years. There is not a day that goes by that I do not enjoy living with you in our own house on which we spent nine months of rebuilding. It is thanks to you that I have been able to put anything and everything into perspective and proceed no matter what, simply because you are able to bring meaning to the word "life". I can't wait to see where our journey will take us next, but I'm sure I'll enjoy it every step of the way. I'll be there for you as much as I possibly can, just as you have done and still do for me every day. Together we will write the pages of our neverending story...

Utrecht, april 2003,
Peter A.N. Bosman

Curriculum Vitae

Personalia

Naam	Peter Alexander Nicolaas Bosman
Geboortedatum	25 december 1975
Geboorteplaats	Utrecht
Nationaliteit	Nederlandse

Opleiding

1988-1994	Atheneum aan het Spectrum College te Utrecht
1994-1998	Doctoraal informatica (cum laude) aan de Universiteit Utrecht

Onderzoeks ervaring

1998-2003	Assistent in Opleiding (AiO) aan het Informatica Instituut van de Universiteit Utrecht (promotiedatum: 20 mei 2003)
2000	Lid van het programma committee van de *Optimization by Building and Using Probabilistic Models OBUPM Workshop* gehouden bij de *Genetic and Evolutionary Computation Conference GECCO–2000*.
2001	Lid van het programma committee van de *Optimization by Building and Using Probabilistic Models OBUPM Workshop* gehouden bij de *Genetic and Evolutionary Computation Conference GECCO–2001*.
2001	Reviewer voor *IEEE Transactions on Evolutionary Computation*
2002	Lid van het programma committee van de *Genetic and Evolutionary Computation Conference GECCO–2002*
2003	Lid van het programma committee van de *Genetic and Evolutionary Computation Conference GECCO–2003*

Prijzen

1998	Laureaat van de CIVI afstudeerprijs voor de Industrie 1998 in de discipline informatica en technische informatica
1999	Laureaat van de tweede prijs uitgeloofd door het NGI voor beste nationale afstudeerscriptie in 1998 in de informatica

Publications

Journals

P. A. N. Bosman and D. Thierens. The balance between proximity and diversity in multi–objective evolutionary algorithms. *IEEE Transactions on Evolutionary Algorithms*, 2003. Accepted for publication.

P. A. N. Bosman and D. Thierens. Multi–objective optimization with diversity preserving mixture–based iterated density estimation evolutionary algorithms. *International Journal of Approximate Reasoning*, **31**(3), pages 259–289, 2002.

Conference proceedings

P. A. N. Bosman and D. Thierens. On the modelling of evolutionary algorithms. In E. Postma and M. Gyssens, editors, *Proceedings of the Eleventh Belgium–Netherlands Conference on Artificial Intelligence BNAIC–1999*, pages 67–74, 1999.

P. A. N. Bosman and D. Thierens. Linkage information processing in distribution estimation algorithms. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO–1999*, pages 60–67. Morgan Kaufmann, 1999.

P. A. N. Bosman and D. Thierens. Continuous iterated density estimation evolutionary algorithms within the IDEA framework. In M. Pelikan, H. Mühlenbein, and A. O. Rodriquez, editors, *Proceedings of the Optimization by Building and Using Probabilistic Models OBUPM Workshop at the Genetic and Evolutionary Computation Conference GECCO–2000*, pages 197–200. Morgan Kaufmann, 2000.

P. A. N. Bosman and D. Thierens. Expanding from discrete to continuous estimation of distribution algorithms: The IDEA. In M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. J. Merelo, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature – PPSN VI*, pages 767–776. Springer Verlag, 2000.

P. A. N. Bosman and D. Thierens. Negative log–likelihood and statistical hypothesis testing as the basis of model selection in IDEAs. In A. Feelders, editor, *Proceedings of the Tenth Belgium–Netherlands Conference on Machine Learning*, pages 109–116. Tilburg University, 2000.

P. A. N. Bosman and D. Thierens. Advancing continuous IDEAs with mixture distributions and factorization selection metrics. In M. Pelikan and K. Sastry, editors, *Proceedings of the Optimization by Building and Using Probabilistic Models OBUPM Workshop at the Genetic and Evolutionary Computation Conference GECCO–2001*, pages 208–212. Morgan Kaufmann, 2001.

P. A. N. Bosman and D. Thierens. Crossing the road to efficient IDEAs for permutation problems. In L. Spector, E.D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, M. H. Garzon S. Pezeshk, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference – GECCO–2001*, pages 219–226. Morgan Kaufmann, 2001.

P. A. N. Bosman and D. Thierens. New IDEAs and more ICE by learning and using unconditional permutation factorizations. In *Late–Breaking Papers of the Genetic and Evolutionary Computation Conference GECCO–2001*, pages 16–23, 2001.

P. A. N. Bosman and D. Thierens. Exploiting gradient information in continuous iterated density estimation evolutionary algorithms. In B. Kröse, M. de Rijke, G. Schreiber, and M. van Someren, editors, *Proceedings of the thirteenth Belgium–Netherlands Artificial Intelligence Conference BNAIC–2001*, pages 69–76, 2001.

P. A. N. Bosman and D. Thierens. Permutation Optimization by Iterated Estimation of Random Keys Marginal Product Factorizations. In J. J. Merelo, P. Adamidis, H.-G. Beyer, J.-J. Fernández-Villicañas and H.-P. Schwefel (eds), Problem Solving from Nature – PPSN VII, pages 331–340. Springer Verlag, 2002.

D. Thierens and **P. A. N. Bosman**. Multi-objective mixture-based iterated density estimation evolutionary algorithms. In L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 663–670. Morgan Kaufmann, 2001.

D. Thierens and **P. A. N. Bosman**. Multi-objective optimization with iterated density estimation evolutionary algorithms using mixture models. In A. Ochoa, H. Mühlenbein, T. English, and P. Larrañaga, editors, *Proceedings of the Third International Symposium on Adaptive Systems ISAS–2001 — Evolutionary Computation and Probabilistic Graphical Models*, pages 129–136. Institute of Cybernetics, Mathematics and Physics, 2001.

Technical Reports

P. A. N. Bosman. EA visualizer tutorial. Technical report UU-CS-1999-20, Institute of Information and Computing Sciences, Utrecht University, Utrecht, 1999.

P. A. N. Bosman and D. Thierens. An algorithmic framework for density estimation based evolutionary algorithms. Technical report UU-CS-1999-46, Institute of Information and Computing Sciences, Utrecht University, Utrecht, 1999.

P. A. N. Bosman and D. Thierens. IDEAs based on the normal kernels probability density function. Technical report UU-CS-2000-11, Institute of Information and Computing Sciences, Utrecht University, Utrecht, 2000.

P. A. N. Bosman and D. Thierens. Mixed IDEAs. Technical report UU-CS-2000-45, Institute of Information and Computing Sciences, Utrecht University, Utrecht, 2000.

P. A. N. Bosman and D. Thierens. Random keys on ICE: marginal product factorized probability distributions in permutation optimization. Technical report UU-CS-2002-54, Institute of Information and Computing Sciences, Utrecht University, Utrecht, 2002.

P. A. N. Bosman and D. Thierens. A thorough documentation of obtained results on real-Valued continuous and combinatorial multi–objective optimization problems using diversity preserving mixture–based iterated density estimation evolutionary algorithms. Technical report UU-CS-2002-52, Institute of Information and Computing Sciences, Utrecht University, Utrecht, 2002.

Index

additively decomposable, 7, 57
allele, 43
allele swapping probability, 49
attractor, 56

bitcounting, 6
black–box optimization, 6
BMDA, 88
BOA, 89
building block, 53
building–block hypothesis, 52

cGA, 84
cluster, 34
COMIT, 88
compact permutation, 128
complexity penalization, 24
conditional
    gpdf, 17
    independence, 18
conjugate gradient, 112
    random restart, 112
constraints, 2
crossover
    one–point, 49
    uniform, 49
curse of dimensionality, 107

deceptive trap, 55
decision
    graph, 87
    tree, 87
default table, 146
descendent, 18
dimensionality, 3
DNA, 42

domain, 2
domination
    count, 183
    rank, 181

EBNA, 89
ECGA, 86
EGNA, 92
elitism, 79
entropy, 20
    differential, 20
    relative, 20
Euclidean distance, 34
event, 12
evolution strategies, 46, 102
evolutionary
    algorithm, 42
    computation, 42
    programming, 46
Expectation Maximization algorithm, 38

factorization, 15, 83
    Bayesian, 87
    graph, 16
    marginal product, 16, 85
    multivariate, 76
    univariate, 17
feasible, 2
fitness, 42
    function, 43
forest, 88
frequency probability mass function, 14

gene, 43

generalization, 24
generalized probability density function, 14, 16
generation, 44
genetic
  algorithm, 46
    building block filtering, 63
    fast messy, 62
    gene expression messy, 63
    linkage learning, 62
    messy, 62
  programming, 46
genetics, 42
genotype, 42
  space, 43
genotype space, 43
GLIDE, 112
global optimization, 4
globally optimal
  solution, 4
  value, 4
gpdf, 14, 16
gradient descent, 112

histogram fixed–width, 106
hybrid, 45

𝕀CE, 153
IDEA, 7, 78
independence, 16
index
  cluster vector, 56
  vector, 56
induction, 6
infeasible, 2
integer
  permutation genotype, 128
  sample space, 12

k–means partitioning algorithm, 36
Kullback–Leibler divergence, 20

leader partitioning algorithm, 35
LFDA, 89
likelihood, 19
  extended, 24
line minimization, 112
linkage, 55
  tight, 55
local
  optimization, 4
  structure, 148
locally optimal
  solution, 4
  value, 4
locus, 43

Mahalanobis distance, 35
marginalization, 14
maximization, 3
maximum likelihood, 19
𝕄IDEA, 185
MIMIC, 87, 157
minimal entropy, 20
minimization, 3
mixture
  component, 32, 92
  probability distribution, 92
mixture probability distributions, 32
model
  fitting, 15
  selection, 15
multi–modal optimization, 5
mutation, 42

natural evolution, 42
needle–in–a–haystack, 55
neighborhood, 4
node
  partition vector, 16, 85
  vector, 16, 85
normal
  gpdf, 96
  kernels gpdf, 99
  mixture gpdf, 103

objective
  function, 3
  space, 3
Occam's razor, 24
offspring, 42
OmeGA, 130

one–max, 6
optimal solution, 3
optimality, 3
optimization
    function, 3
    value, 3
overfitting, 24

PADA, 91
parameter, 2
    space, 2, 4
parent vector, 87
parents, 44
partition, 34
PBIL, 83
phenotype, 42
population, 44
precision matrix, 97, 213
probabilistic model, 15
probability
    density function, 14
    mass function, 14
    measure, 12
problem variables, 3

random
    keys, 129
    rescaling, 153
real–valued sample space, 12
recombination, 42
regularization, 24, 27
relative–position–of–value semantics, 141
relative–value–at–position semantics, 140
RKGA, 130

sample space, 12
schema, 50
    growth equation, 52
    theorem, 52
schemata, 50
search space, 2
selection, 42
    factorization, 76
    model, 77
    natural, 42
    proportionate, 47
    tournament, 48
    truncation, 48
selection pressure, 48
simulation sampling, 99
solution, 2
species, 42
support vector machine, 207

termination criterion, 44
tournament size, 48
truncation percentile, 48

UMDA, 84

variable, 2

List of notations, symbols and pseudo–code conventions

Notations

$S = \{\bullet, \bullet, \ldots, \bullet\}$	A set S with $\|S\|$ elements. The order of the elements in a set is *irrelevant*.
$\|S\|$	The number of elements in set S.
$\emptyset$	The empty set.
$s \in S = \begin{cases} true & \text{if } s \text{ is in } S \\ false & \text{otherwise} \end{cases}$	Element query.
$S^0 \subseteq S^1 = \forall s \in S^0 : s \in S^1$	Subset query.
$S^0 \subset S^1 = S^0 \subseteq S^1 \wedge \|S^0\| < \|S^1\|$	Proper subset query.
$S^0 \cup S^1 = \{s \| s \in S^0 \vee s \in S^1\}$	Union of sets, each element appears only once in the result, $\|S^0 \cup S^1\| \leq \|S^0\| + \|S^1\|$.
$S^0 \cap S^1 = \{s \| s \in S^0 \wedge s \in S^1\}$	Intersection of sets, each element appears only once in the result, $\|S^0 \cap S^1\| \leq \|S^0\|, \|S^0 \cap S^1\| \leq \|S^1\|$.
$S^0 - S^1 = \{s \| s \in S^0 \wedge s \notin S^1\}$	Difference of sets, exclusion of elements.
$\boldsymbol{v} = (\boldsymbol{v}_0, \boldsymbol{v}_1, \ldots, \boldsymbol{v}_{\|\boldsymbol{v}\|-1})$	A vector $\boldsymbol{v}$ of $\|\boldsymbol{v}\|$ elements. The order of the elements in a vector is *relevant*.
$\|\boldsymbol{v}\|$	The number of components in vector $\boldsymbol{v}$.
$\boldsymbol{v_j} = (\boldsymbol{v}_{(\boldsymbol{j}_0)}, \boldsymbol{v}_{(\boldsymbol{j}_1)}, \ldots, \boldsymbol{v}_{(\boldsymbol{j}_{\|\boldsymbol{j}\|-1})})$	Constructing a vector from an existing vector (usually subvector), $\forall i \in \{0, 1, \ldots, \|\boldsymbol{j}\| - 1\} : \boldsymbol{j}_i \in \{0, 1, \ldots, \|\boldsymbol{v}\| - 1\}$.
$()$	The empty vector.
$v \in \boldsymbol{v} = \exists 0 \leq i < \|\boldsymbol{v}\| : v = \boldsymbol{v}_i$	Element query.
$\boldsymbol{v}^0 \sqsubseteq \boldsymbol{v}^1 = \forall 0 \leq i < \|\boldsymbol{v}^0\| : \boldsymbol{v}_i^0 \in \boldsymbol{v}^1$	Subvector query.
$\boldsymbol{v}^0 \sqsubset \boldsymbol{v}^1 = \boldsymbol{v}^0 \sqsubseteq \boldsymbol{v}^1 \wedge \|\boldsymbol{v}^0\| < \|\boldsymbol{v}^1\|$	Proper subvector query.
$v_{\boldsymbol{j}} = (v_{\boldsymbol{j}_0}, v_{\boldsymbol{j}_1}, \ldots, v_{\boldsymbol{j}_{\|\boldsymbol{j}\|-1}})$	Shorthand for a vector, $\|v_{\boldsymbol{j}}\| = \|\boldsymbol{j}\|$.

$\boldsymbol{v}^0 \sqcup \boldsymbol{v}^1 = (\boldsymbol{v}^0_0, \boldsymbol{v}^0_1, \ldots, \boldsymbol{v}^0_{\lvert\boldsymbol{v}^0\rvert-1}, \boldsymbol{v}^1_0, \boldsymbol{v}^1_1, \ldots, \boldsymbol{v}^1_{\lvert\boldsymbol{v}^1\rvert-1})$	Splice operation on vectors, each element appears exactly as often as it appears in the input, $\lvert\boldsymbol{v}^0 \sqcup \boldsymbol{v}^1\rvert = \lvert\boldsymbol{v}^0\rvert + \lvert\boldsymbol{v}^1\rvert$.
$\boldsymbol{v}^0 \sqcap \boldsymbol{v}^1 = \boldsymbol{v}^0_{\boldsymbol{j}}$ such that $\forall 1 \leq i < \lvert\boldsymbol{j}\rvert : \lvert\boldsymbol{v}^0\rvert > \boldsymbol{j}_i > \boldsymbol{j}_{i-1} \geq 0$ and $\forall 0 \leq i < \lvert\boldsymbol{v}^0\rvert : \boldsymbol{v}^0_i \in \boldsymbol{v}^1 \leftrightarrow i \in \boldsymbol{j}$	Intersection operation on vectors, each element of $\boldsymbol{v}^0$ is preserved if and only if it also appears in $\boldsymbol{v}^1$, $\lvert\boldsymbol{v}^0 \sqcap \boldsymbol{v}^1\rvert \leq \lvert\boldsymbol{v}^0\rvert$. The relative ordering in $\boldsymbol{v}^0$ is also preserved.
$\boldsymbol{v}^0 - \boldsymbol{v}^1 = \boldsymbol{v}^0_{\boldsymbol{j}}$ such that $\forall 1 \leq i < \lvert\boldsymbol{j}\rvert : \lvert\boldsymbol{v}^0\rvert > \boldsymbol{j}_i > \boldsymbol{j}_{i-1} \geq 0$ and $\forall 0 \leq i < \lvert\boldsymbol{v}^0\rvert : \boldsymbol{v}^0_i \notin \boldsymbol{v}^1 \leftrightarrow i \in \boldsymbol{j}$	Difference operation on vectors, each element of $\boldsymbol{v}^0$ is preserved if and only if it doesn't appear in $\boldsymbol{v}^1$. The relative ordering in $\boldsymbol{v}^0$ is also preserved.
$dy_{\boldsymbol{a}} = \prod_{i=0}^{\lvert\boldsymbol{a}\rvert-1} dy_{\boldsymbol{a}_i}$	Shorthand for the multivariate derivative.
$P(\boldsymbol{\mathcal{Z}})$	A probability distribution for random variables $\boldsymbol{\mathcal{Z}}$, $P(\boldsymbol{\mathcal{Z}}) : \Omega \rightarrow \mathbb{R}$. Because $P(\boldsymbol{\mathcal{Z}})$ denotes a function, we write $P(\boldsymbol{\mathcal{Z}})(\boldsymbol{z})$ to get the function value for any $\boldsymbol{z} \in \Omega$.
$P(Z_{\boldsymbol{j}})$	The function that is obtained from $P(\boldsymbol{\mathcal{Z}})$ (by marginalization), such that $P(Z_{\boldsymbol{j}})(\boldsymbol{z}) = \Pr(Z_{\boldsymbol{j}} = \boldsymbol{z})$, $Z_{\boldsymbol{j}} \sqsubseteq \boldsymbol{\mathcal{Z}}$. It is a probability distribution over random variables $Z_{\boldsymbol{j}}$.
$P_{\mathcal{M}}(Z_{\boldsymbol{j}})$	The probability distribution associated with a probabilistic model $\mathcal{M}$.
$P_{\varsigma}(Z_{\boldsymbol{j}})$	The probability distribution associated with a probabilistic model $\mathcal{M} = (\varsigma, \boldsymbol{\theta})$ for which the parameters $\boldsymbol{\theta}$ are estimated in a predefined way ($\boldsymbol{\theta} \xleftarrow{fit} \varsigma$).
$\hat{}$	Indicates parameter estimates (such as $\hat{\boldsymbol{\theta}}$ and $\hat{\boldsymbol{\mu}}$). In the case of $\hat{P}$ it means that the parameters of probability distribution $\hat{P}$ are estimated from the data.
$\boldsymbol{\theta} \xleftarrow{fit} \varsigma$	Indication of the fact that the parameters $\boldsymbol{\theta}$ of a probabilistic model that are implied by a structure ς, are obtained by estimation in a predefined way.

Symbols

$\mathbb{A}_i$	The domain of the i–th gene variable, representing the possible alleles that can appear at locus i.
α	Smoothness control parameter for the normal kernels gpdf; determines the variance in each dimension of each normal kernel, adaptive to the range of the data: $\alpha \cdot \text{range}_i / \lvert\boldsymbol{\mathcal{S}}\rvert$
$\mathbb{B}$	The set of binary integer values $\mathbb{B} = \{0, 1\}$.
$\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_{k-1})$	The mixing coefficient in the mixture probability distribution; β_i is the weight of the i–th mixture component in the sum. Also the mixing coefficients of the normal gpdfs in the normal mixture gpdf.
$\boldsymbol{c}^i = (\boldsymbol{c}^i_0, \boldsymbol{c}^i_1, \ldots, \boldsymbol{c}^i_{\lvert\boldsymbol{c}^i\rvert-1})$	The i–th cluster contains $\lvert\boldsymbol{c}^i\rvert - 1$ indices that refer to the samples in the sample vector $\boldsymbol{\mathcal{S}}$, $(\boldsymbol{c}^i)_j = \boldsymbol{c}^i_j$, $\forall 0 \leq j < \lvert\boldsymbol{c}^i\rvert : \boldsymbol{c}^i_j \sqsubseteq (0, 1, \ldots \lvert\boldsymbol{\mathcal{S}}\rvert - 1)$.
$\boldsymbol{\mathcal{C}} = (\boldsymbol{c}^0, \boldsymbol{c}^1, \ldots, \boldsymbol{c}^{\lvert\boldsymbol{\mathcal{C}}\rvert-1})$	The vector of clusters.
$\mathfrak{C}$	The constraint satisfaction function that returns *feasible* or *infeasible* depending on whether a solution is feasible or not, $\mathfrak{C} : \mathbb{P} \rightarrow \{\textit{feasible}, \textit{infeasible}\}$.
d	The number of problem variables.
d	The Kullback–Leibler divergence function for two probability distributions, $d(P_{\mathcal{M}^0}(\boldsymbol{\mathcal{Z}}) \,\lvert\rvert\, P_{\mathcal{M}^1}(\boldsymbol{\mathcal{Z}}))$
$\boldsymbol{\mathcal{D}} = (0, 1, \ldots, d-1)$	A vector containing d numbers, $\boldsymbol{\mathcal{D}}_i = i$.
$\mathbb{D}_i$	The domain of the i–th problem variable.
$\mathfrak{D}$	The genotype decoding function $\mathfrak{D} : \mathbb{G} \rightarrow \mathbb{P}$.
$\mathfrak{F}$	The optimization function $\mathfrak{F} : \mathbb{P} \rightarrow \mathbb{O}$.
$\boldsymbol{\iota} = (\boldsymbol{\iota}_0, \boldsymbol{\iota}_1, \ldots, \boldsymbol{\iota}_{\lvert\boldsymbol{\iota}\rvert-1})$	The vector of vectors describing the indices of the genes that the $\lvert\boldsymbol{\iota}\rvert$ subfunctions pertain to in an additively composed fitness function $\left(\mathfrak{G}(\mathsf{Z}_{\boldsymbol{\mathcal{L}}}) = \sum_{i=0}^{\lvert\boldsymbol{\iota}\rvert-1} g_i(\mathsf{Z}_{\boldsymbol{\iota}_i})\right)$.
$\mathbb{G}$	The space of all possible genotypes (also called the genotype space).
$\mathfrak{G}$	The fitness function $\mathfrak{G} : \mathbb{G} \rightarrow \mathbb{O}, \mathfrak{G} = \mathfrak{F} \circ \mathfrak{D}$.

Γ	A probability measure on sample space Ω, $\Gamma\colon \mathcal{P}(\Omega) \to \mathbb{R}$. $\Gamma(A)$ is the probability of event A, that is $\Pr(\boldsymbol{\mathcal{Z}} \in A)$.
Γ^D	Probability measure for the integer sample space.
Γ^R	Probability measure for the real–valued sample space.
Γ^P	Probability measure for the integer permutation sample space.
h	The entropy function for probability distributions, $h(P_{\mathcal{M}}(\boldsymbol{\mathcal{Z}}))$.
$\boldsymbol{h}$	Binary or relative order schema; a vector of length $\lvert\boldsymbol{h}\rvert$ consisting of 0, 1 and $\star$ symbols in the binary case and $0, 1, \ldots, \lvert\boldsymbol{h}\rvert - 1$ and ! symbols in the relative permutation case.
k	The number of components in the mixture probability distribution.
$\boldsymbol{\mathcal{K}} = (0, 1, \ldots, k-1)$	A vector containing k numbers, $\boldsymbol{\mathcal{K}}_i = i$.
l	The number of variables in the genotype encoding of a solution, which equals the number of random variables.
$\mathfrak{L}$	The likelihood function for probability distributions, $\mathfrak{L}(P_{\boldsymbol{\mathcal{M}}}(\boldsymbol{\mathcal{Z}}), \boldsymbol{\mathcal{S}})$.
$\boldsymbol{\mathcal{L}} = (0, 1, \ldots, l-1)$	A vector containing l numbers, $\boldsymbol{\mathcal{L}}_i = i$.
m	Number of objectives in a multi–objective optimization problem.
$\mathcal{M} = (\varsigma, \boldsymbol{\theta})$	A probabilistic model consisting of a structure ς and a vector $\boldsymbol{\theta}$ of parameters.
$\boldsymbol{\mu}$	Vector of means in the normal gpdf.
$\boldsymbol{\mu}^i$	Vector of means in the i–th normal gpdf component in the normal mixture gpdf.
n	The number of genotypes in the population of an evolutionary algorithm.
$\mathbb{N}$	The set of all non–negative integer values $\mathbb{N} = \{0, 1, 2, \ldots\}$.
$\boldsymbol{\nu} = (\boldsymbol{\nu}_0, \boldsymbol{\nu}_1, \ldots, \boldsymbol{\nu}_{\lvert\boldsymbol{\nu}\rvert-1})$	The probabilistic model structure for a multivariate factorization. A vector of mutually exclusive vectors such that $\boldsymbol{\nu}_i$ is a vector indicating the random variables $Z_{\boldsymbol{\nu}_i}$ that are combined in a gpdf in the factorization.

O_i	Permutation random variable.
$\boldsymbol{\mathcal{O}} = (O_0, O_1, \ldots, O_{l-1}) = O_{\boldsymbol{\mathcal{L}}}$	A vector containing l permutation random variables, $\boldsymbol{\mathcal{O}}_i = O_i$.
$\mathbb{O}$	The objective space of an optimization problem containing all possible optimization values of an optimization problem.
Ω	The domain of random variables $\boldsymbol{\mathcal{Z}}$: the collection of all combinations of values that can be assigned to random variables $\boldsymbol{\mathcal{Z}}$.
$\Omega^{\boldsymbol{j}}$	The domain of random variables $Z_{\boldsymbol{j}}$: the collection of all combinations of values that can be assigned to random variables $Z_{\boldsymbol{j}}$.
$perm$	The permutation function that returns the set of all possible permutations for a given a vector.
p_r	The probability with which recombination is applied. If no recombination should be performed, each offspring is an exact copy of some parent.
p_m	The probability with which mutation is applied. Often interpreted as a probability of randomly changing a certain part of a genotype.
p_ϱ	The probability with which random rescaling is applied in $\mathbb{ICE}$ to a block of random keys genes that is copied in the block transfer recombination operator in $\mathbb{ICE}$.
$P^{\mathcal{F}}_{\boldsymbol{\theta}}(X_{\boldsymbol{j}})$	Frequency gpdf for integer random variables $X_{\boldsymbol{j}} \sqsubseteq \boldsymbol{\mathcal{X}}$.
$P^{\mathcal{F}}_{\boldsymbol{\theta}}(O_{\boldsymbol{j}})$	Frequency gpdf for integer permutation random variables $O_{\boldsymbol{j}} \sqsubseteq \boldsymbol{\mathcal{O}}$.
$P^{\mathcal{N}}_{(\boldsymbol{\mu}_{\boldsymbol{j}}, \boldsymbol{\Sigma}^{\boldsymbol{j}})}(Y_{\boldsymbol{j}})$	Normal gpdf for real–valued random variables $Y_{\boldsymbol{j}} \sqsubseteq \boldsymbol{\mathcal{Y}}$.
$P^{\mathcal{N}_K}_{(\boldsymbol{\mathcal{S}}, \boldsymbol{\Sigma}^{\boldsymbol{j}})}(Y_{\boldsymbol{j}})$	Normal kernels gpdf for real–valued random variables $Y_{\boldsymbol{j}} \sqsubseteq \boldsymbol{\mathcal{Y}}$.
$P^{\mathcal{N}_M}_{\left((\boldsymbol{\mu}^0_{\boldsymbol{j}}, \boldsymbol{\Sigma}^{0,\boldsymbol{j}}), \ldots, (\boldsymbol{\mu}^{w-1}_{\boldsymbol{j}}, \boldsymbol{\Sigma}^{w-1,\boldsymbol{j}}), \boldsymbol{\beta}\right)}(Y_{\boldsymbol{j}})$	Normal mixture gpdf for real–valued random variables $Y_{\boldsymbol{j}} \sqsubseteq \boldsymbol{\mathcal{Y}}$.
$P^{\mathcal{H}}_{(\boldsymbol{\beta}_{\boldsymbol{j}}, \boldsymbol{\pi})}(Y_{\boldsymbol{j}})$	Histogram gpdf for real–valued random variables $Y_{\boldsymbol{j}} \sqsubseteq \boldsymbol{\mathcal{Y}}$.

$\boldsymbol{\mathcal{P}_S}$	Pareto optimal set. The set of all Pareto optimal solutions.
$\boldsymbol{\mathcal{P}_F}$	Pareto optimal front. The set of all objective function values corresponding to the solutions in $\boldsymbol{\mathcal{P}_S}$.
$\mathcal{P}$	The powerset or powervector function that returns, given a set S or a vector $\boldsymbol{v}$, a set containing *all* subsets of S or a vector containing *all* subvectors of $\boldsymbol{v}$ respectively.
$\boldsymbol{\mathcal{P}}$	The population in an evolutionary algorithm (vector of n genotypes).
$\mathbb{P}$	The parameter space of an optimization problem (also called search space or domain).
$\boldsymbol{\pi} = (\boldsymbol{\pi}_0, \boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_{l-1})$	The probabilistic model structure for a Bayesian factorization. A vector of l vectors such that $Z_{\boldsymbol{\pi}_i}$ are the random variables that random variable Z_i is conditioned on.
R_i	Random keys gene variable.
$\mathbf{R} = (\mathsf{R}_0, \mathsf{R}_1, \ldots, \mathsf{R}_{l-1}) = \mathsf{R}_{\boldsymbol{\mathcal{L}}}$	A genotype (vector) containing l random keys gene variables, $\mathbf{R}_i = \mathsf{R}_i$.
$\mathbb{R}$	The set of all real values.
$\boldsymbol{\mathcal{S}} = (\boldsymbol{\mathcal{S}}_0, \boldsymbol{\mathcal{S}}_1, \ldots, \boldsymbol{\mathcal{S}}_{\lvert\boldsymbol{\mathcal{S}}\rvert-1})$	The vector of sample points. This vector is also the vector of selected solutions in the IDEA framework, in which case $\boldsymbol{\mathcal{S}} \sqsubseteq \boldsymbol{\mathcal{P}}$.
$\boldsymbol{\mathcal{S}}_i = ((\boldsymbol{\mathcal{S}}_i)_0, (\boldsymbol{\mathcal{S}}_i)_1, \ldots, (\boldsymbol{\mathcal{S}}_i)_{l-1})$	Each sample point (solution) is a genotype, which is a vector of length l.
$\boldsymbol{\mathcal{S}^j} = (\boldsymbol{\mathcal{S}}_0^{\boldsymbol{j}}, \boldsymbol{\mathcal{S}}_1^{\boldsymbol{j}}, \ldots, \boldsymbol{\mathcal{S}}_{\lvert\boldsymbol{\mathcal{S}}\rvert-1}^{\boldsymbol{j}})$	A vector containing a truncated version of each sample point in $\boldsymbol{\mathcal{S}}$; for each sample point only the dimensions indicated by $\boldsymbol{j}$ are in $\boldsymbol{\mathcal{S}^j}$, that is $(\boldsymbol{\mathcal{S}^j})_i = \boldsymbol{\mathcal{S}}_i^{\boldsymbol{j}} = (\boldsymbol{\mathcal{S}}_i)_{\boldsymbol{j}}$.
$\boldsymbol{\mathcal{S}}^P$	Preselection set in the diversity preserving selection operator in MIDEA, $\boldsymbol{\mathcal{S}} \sqsubseteq \boldsymbol{\mathcal{S}}^P \sqsubseteq \boldsymbol{\mathcal{P}}$.
$\mathfrak{S}$	The space of all possible probabilistic model structures. This definition is dependent on the context that defines the class of structures. For example: all possible Bayesian factorizations over all random variables $\boldsymbol{\mathcal{Z}}$.
$\acute{\boldsymbol{\mathcal{S}}}$	Sample points that serve as centers for the normal gpdfs in the normal kernels gpdf.

σ	The sorting function that, given a vector $\boldsymbol{o}$, returns a vector $\boldsymbol{p} = \sigma(\boldsymbol{o})$ that represents the sorting of $\boldsymbol{o}$ in ascending order, so $\boldsymbol{o}_{\boldsymbol{p}_0} < \boldsymbol{o}_{\boldsymbol{p}_1} < \ldots < \boldsymbol{o}_{\boldsymbol{p}_{\lvert\boldsymbol{o}\rvert-1}}$.
ς	The structure of a probabilistic model $\mathcal{M}$.
$\boldsymbol{\Sigma}$	Symmetric covariance matrix in the normal gpdf.
$\boldsymbol{\Sigma}^i$	Symmetric covariance matrix in the i–th normal gpdf component in the normal mixture gpdf.
$\boldsymbol{\Sigma}^{\boldsymbol{j}}$	Submatrix of symmetric covariance matrix in the normal gpdf obtained by taking only the rows and columns indicated by $\boldsymbol{j} \sqsubseteq \boldsymbol{\mathcal{L}}$, that is $(\boldsymbol{\Sigma}^{\boldsymbol{j}})_{ik} = \boldsymbol{\Sigma}_{\boldsymbol{j}_i \boldsymbol{j}_k}$.
$\boldsymbol{\Sigma}^{i,\boldsymbol{j}}$	Submatrix of symmetric covariance matrix in the i–th normal gpdf component in the normal mixture gpdf obtained by taking only the rows and columns indicated by $\boldsymbol{j} \sqsubseteq \boldsymbol{\mathcal{L}}$, that is $(\boldsymbol{\Sigma}^{i,\boldsymbol{j}})_{ik} = \boldsymbol{\Sigma}^i_{\boldsymbol{j}_i \boldsymbol{j}_k}$.
τ	Truncation percentile in truncation selection, $\tau \in [\frac{1}{n}, 1]$, used by default in monotonic IDEAs.
τ_G	Percentile of solutions to which at the end of a generation conjugate gradient optimization is applied in GLIDE.
$\boldsymbol{\theta}$	The parameters of a probabilistic model $\mathcal{M}$.
w	The number of components in the normal mixture gpdf.
$\boldsymbol{W}$	Precision matrix, the inverse of the symmetric covariance matrix $\boldsymbol{\Sigma}$, that is $\boldsymbol{W} = \boldsymbol{\Sigma}^{-1}$. Notations $\boldsymbol{W}^i$, $\boldsymbol{W}^{\boldsymbol{j}}$ and $\boldsymbol{W}^{i,\boldsymbol{j}}$ are precision matrix notations corresponding to the analogous notations for $\boldsymbol{\Sigma}$.
$\boldsymbol{\mathcal{W}} = (0, 1, \ldots, w-1)$	A vector containing w numbers, $\boldsymbol{\mathcal{W}}_i = i$.
X_i	Binary gene variable.
X_i	Binary random variable.
$\mathbf{X} = (\mathsf{X}_0, \mathsf{X}_1, \ldots, \mathsf{X}_{l-1}) = \mathsf{X}_{\boldsymbol{\mathcal{L}}}$	A genotype (vector) containing l binary gene variables, $\mathbf{X}_i = \mathsf{X}_i$.
$\boldsymbol{\mathcal{X}} = (X_0, X_1, \ldots, X_{l-1}) = X_{\boldsymbol{\mathcal{L}}}$	A vector containing l binary random variables, $\boldsymbol{\mathcal{X}}_i = X_i$.

Y_i	Real gene variable.
$\mathbf{Y} = (\mathsf{Y}_0, \mathsf{Y}_1, \ldots, \mathsf{Y}_{l-1}) = \mathsf{Y}_{\boldsymbol{\mathcal{L}}}$	A genotype (vector) containing l real–valued gene variables, $\mathbf{Y}_i = \mathsf{Y}_i$.
Y_i	Real random variable.
$\boldsymbol{\mathcal{Y}} = (Y_0, Y_1, \ldots, Y_{l-1}) = Y_{\boldsymbol{\mathcal{L}}}$	A vector containing l real–valued random variables, $\boldsymbol{\mathcal{Y}}_i = Y_i$.
Z_i	Gene variable of unspecified type.
$\mathbf{Z} = (\mathsf{Z}_0, \mathsf{Z}_1, \ldots, \mathsf{Z}_{l-1}) = \mathsf{Z}_{\boldsymbol{\mathcal{L}}}$	A genotype (vector) containing l gene variables of unspecified type, $\mathbf{Z}_i = \mathsf{Z}_i$.
Z_i	Random variable of unspecified type.
$\boldsymbol{\mathcal{Z}} = (Z_0, Z_1, \ldots, Z_{l-1}) = Z_{\boldsymbol{\mathcal{L}}}$	A vector containing l random variables of unspecified type, $\boldsymbol{\mathcal{Z}}_i = Z_i$.
$\mathbb{Z}$	The set of all integer values $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$.
ζ_i	Problem variable used to characterize $\mathfrak{F}$.
!	Wildcard in relative order schemata, indicate that these symbols may be replaced using exactly those integers in $\{0, 1, \ldots, k-1\}$ that have not yet been used in a relative order schema of length k.
$\prec$	Optimization ordering of $\mathbb{O}$. A solution $A \in \mathbb{P}$ is globally optimal if $\mathfrak{C}(A) = f\text{easible} \wedge \neg\exists B \in \mathbb{P} : \mathfrak{C}(B) = \textit{feasible} \wedge \mathfrak{F}(A) \prec \mathfrak{F}(B)$. Also denotes Pareto dominance.
$\preceq$	Weak–Pareto dominance relation for approximation sets.
$\prec\!\prec$	Strict Pareto dominance relation for approximation sets.
$\lhd$	Relation indicating one approximation set is better than another.
$\parallel$	Incomparable relation for approximation sets.
$\sim$	Notation for incomparability of two non identical optimization values ($\mathfrak{F}(A) \neq \mathfrak{F}(B)$). Both optimization values are equally preferred, $\mathfrak{F}(A) \sim \mathfrak{F}(B)$.
$\star$	Wildcard in binary schemata, matches both 0 and 1 symbols.
$\bigstar$	A binary schema of length l consisting only of $\star$ symbols, $\bigstar = \bigsqcup_{i=0}^{l-1}(\star)$.

Pseudo–code conventions

Programming constructions	
$\leftarrow$	An assignment is written $A \leftarrow X$ (assigning X to A).
Algorithms	Algorithms appear with capital letters, eg. ALGORITHM().
Reserved words	Reserved words appear in boldface, such as **while**.
Scope	The scope of grouping statements is related to the indentation and numbering of lines. The scope of a grouping statement in line l is $l.*$.
Composite	Fields of composite variables are written in italics and use brackets [] to identify the composite variable. If variable A is of a composite type describing length, height and weight, the use of these fields appears as $length[A]$, $height[A]$, $weight[A]$.

Elementary data types	
boolean	Truth value, element of $\{\boldsymbol{false}, \boldsymbol{true}\}$
bit	Element of $\{0, 1\}$
integer	Element of $\mathbb{Z}$
real	Element of $\mathbb{R}$

Data structures	
array	Typed array, memory block with fixed size. • Starting index: 0 • Notation for indexing: $A[i]$. • Notation for determining the length of an array: $\|A\|$.
vector	Typed dynamic array, memory block with non–fixed size. • Starting index: 0 • Notation for indexing: V_i. • Notation for determining the number of elements in a vector: $\|V\|$.

Default algorithms	
RANDOM(A)	Returns an element from set A drawn from a uniform probability distribution defined over A.