Centrum voor Wiskunde en Informatica

REPORT*RAPPORT*

Modeling of Genetic Algorithms with a Finite Population

C.H.M. van Kemenade

Software Engineering (SEN)

SEN-R9725 December 31, 1997

Report SEN-R9725
ISSN 1386-369X

CWI
P.O. Box 94079
1090 GB Amsterdam
The Netherlands

Modeling of Genetic Algorithms with a Finite Population

Cees H.M. van Kemenade
CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

ABSTRACT

Cross-competition between non-overlapping building blocks can strongly influence the performance of evolutionary algorithms. The choice of the selection scheme can have a strong influence on the performance of a genetic algorithm. This paper describes a number of different genetic algorithms, all involving elitism. Infinite population models are presented for each of these algorithms. A problem involving cross-competition is introduced and we show how we can make use of equivalence-classes to make an efficient tracing of the transmission-function models possible on this type of problems. By adding a small extension to the models it is possible to predict the qualitative behavior of finite population genetic algorithms on this type of problems also. Using this model the reliability of the different genetic algorithms and the influence of population sizing on the reliability is investigated.

1991 Mathematics Subject Classification: 68T05, 68T20
1991 Computing Reviews Classification System: F.2.m, G.1.6, I.2.8
Keywords and Phrases: genetic algorithms, selection schemes, transmission functions, tracing models
Note: Work carried out under theme SEN4 "Evolutionary Computation". Parts of this report were published in the proceedings of the International conference on Genetic Algorithm 1997 (Lansing, USA) and parts were published in the proceedings of the Nordic Workshop on Genetic Algorithms 1997 (Helsinki, Finland).

1. Introduction

Interesting optimization problems typically have a search space that is much too large to be searched fast by means of enumerative methods. Under these circumstances one is forced to make assumptions with respect to the structure of the search space. These assumptions can be used to create a more efficient search strategy for the problem at hand. The price to be paid is that the search strategy becomes tailored towards solving problems that have a search space where the applied assumptions are valid.

Genetic algorithms are a class of algorithms that use principles from evolution to implement a search strategy. The GA framework is widely applicable because it allows actual GA's to be tailored towards the specific problem that has to be handled. An instance of a genetic algorithm is obtained by choosing an actual representation for the individuals (candidate solutions), a set of genetic operators that will be used to create new individuals based upon a set of already existing parent individuals, and a selection scheme. A particular choice of the representation, the operators, and the selection scheme corresponds to a set of assumptions about the structure of the search space.

A possible explanation for the working of the GA is given by the building block hypothesis as introduced by Goldberg [Gol89]:

> "... so does a genetic algorithm seek near optimal performance through the juxtaposition of short, low-order, high-performance schemata, or building blocks."

In this paper an optimization problem is constructed that adheres to this hypothesis. For this problems the global optimum can be partitioned in a set of building blocks. These building blocks can be discovered independently and mixed afterwards. Note that we do not know in advance which bits are correlated.

A GA has two tasks to perform simultaneously. First it must grow the building blocks and next it must mix these building blocks in order to obtain the globally optimal solution. In order to get a reliable convergence to the optimal solution both processes have to perform well [TG93]. It appears to be of crucial importance to balance these two processes. Striking this balance is far from trivial. The growing process can be accelerated by increasing the selective pressure, but high selective pressures also lead to faster convergence and therefore leave less time for the mixing process to perform its task. Cross-competition arises when building blocks from different partitions compete with each other as opposed to the competition within the partitions [GDT93], where a partitioning divides the search space in a set of disjunct regions. In case of a binary search space such a partition can be described by means of a set of non-overlapping schemata `***f**fff*f*`, where `*` is the wild-card symbol and `f` represents a fixed allele (either 0 or 1). Two types of building blocks are especially sensitive to extinction due to cross-competition, i.e. building blocks resulting in a relatively small fitness benefit and building blocks that are relatively difficult to propagate by the crossover operator (the operator is biased). An example of a biased crossover operator is the one-point crossover. This operator is biased toward solutions involving schemata with a short defining length as these schemata are less likely to be disrupted by this operator. Biases in the crossover operators have been studied [Boo92, ECS89, SEO91].

We have developed transmission function models for a number of genetic algorithms [vK97a]. Tracing these probabilistic models corresponds to running a genetic algorithm with an infinite population. The results obtained by tracing these models can be regarded as a bound on the reliability of the corresponding real genetic algorithms. Tracing of probability models for a simple GA was done by Whitley [Whi92]. Kok and Floréen traced probabilities using a model based on bit-products and Walsh-products [KF95]. All these models correspond to genetic algorithms involving a infinite population. We have extended the transmission function models such that these can also model genetic algorithms with a finite population size [vK97b].

In this paper we investigate a set of genetic algorithms some of which involve elitism. In an elitist GA the parents can be propagated directly to subsequent generations, which contrast the generational GA where the parents are always discarded after producing enough offspring to populate the next generation. Theoretical analysis shows that a canonical GA will not converge to the global optimum in general, but a GA that maintains the best solution will converge [Rud94] and a GA involving elitism will converge to the optimum too [Rud96]. Maintaining the best individuals means keeping these individuals in the population without allowing them to reproduce anymore, while in case of elitism an individual is allowed to reproduce throughout its lifetime.

The behavior of the models of genetic algorithms are studied on a problem involving cross-competition By comparing the results for the different algorithms inferences can be made regarding why certain algorithms perform better.

2. Transmission models of selection schemes

We are going to consider the canonical genetic algorithm [Hol75], the generational genetic algorithm using tournament selection [Gol89], (μ, λ) and $(\mu + \lambda)$ selection [BHmS91, Rec94, Sch95], triple-competition (rat-race selection) [vK97b], and and elitist recombination [TG94]. Furthermore we are going to discuss the Breeder genetic algorithm by Mühlenbein et al. [MSV94], and the CHC algorithm by Eshelman [Esh91]

A selection scheme is assumed to select the parent pairs for generation G_{i+1} from the individuals in generation G_i. In case of a canonical genetic algorithm the parents taken from generation G_i are discarded and a fitness proportional selection is applied to their offspring. When using an elitist GA the parents of the previous generation can be selected also.

In order to describe the different models we are going to use transmission functions [Alt94]:

> "It is the relationship between the transmission function and the fitness function that determines GA performance. The transmission function "screens off" [Sal71, Bra90] the

effect of the choice of representation and operators, in that either affect the dynamics of the GA only through their effect on the transmission function."

The general form of transmission-selection recursion was used at least as early as 1970 by Slatkin [Sla70].

A transmission function describes the probability distribution of offspring from every possible pair of parents. For a binary genetic operator the transmission function looks like $T(i \leftarrow j,k)$ where j and k are the labels of the two parents and i is the label of the offspring. To be more specific, let $\mathcal{S}$ be the search space; then $T : \mathcal{S}^3 \to [0,1]$. As $T(i \leftarrow j,k)$ represents a distribution for fixed j and k we should have $\sum_i T(i, \leftarrow j,k) = 1$ for all $j,k \in \mathcal{S}$. For a symmetric operator we have the additional condition $T(i \leftarrow j,k) = T(i \leftarrow k,j)$.

We are going to use the transmission function model to model a diverse set of selection schemes including some schemes involving elitism. The following notation will be used. The distribution of the current population will be denoted by $\vec{x}$ and the newly generated population will be denoted as $\vec{x'}$. In order based selection schemes, such as for example tournament selection, it is the fitness-based rank in the population that determines the probability of being selected as a parent. Therefore we introduce the function $f_<(j,\vec{x})$ that computes the fraction of individuals in distribution $\vec{x}$ that have a fitness strictly smaller than the fitness of individual labeled j, and let $f_=(j,\vec{x})$ denote the proportion of individuals that have a fitness equal to f_j.

2.1 Canonical Genetic algorithm

The canonical genetic algorithm is a generational GA using fitness proportional selection. This dynamical system that describes a transition from a current population to a new population is given by the formula [Alt94]:

$$x'_i = \sum_{j,k} T_o(i \leftarrow j,k) \frac{f_j f_k}{\bar{f}^2} x_j x_k,$$

where x_i is the frequency of the individual labeled i, and x'_i is its frequency during the next generation, f_i is the fitness of the individual labeled i, $\bar{f}$ is the average fitness, and T_o is the transmission function describing the actual interaction between genetic operators and representation. The probability that a parent of type j is selected is given by the formula $x_j f_j / \bar{f}$, so the factor that follows $T_o(i \leftarrow j,k)$ denotes the probability that the two parents have label j and k when applying fitness proportional selection.

2.2 Deterministic n-tournament selection

The only difference between the canonical genetic algorithm and the generational genetic algorithm with tournament selection is the method of selection. Therefore both models use the same transmission function $T_o(i \leftarrow j,k)$ to model interaction between genetic operators and representation of individuals. The only modification we need is to replace the factor corresponding to fitness proportional selection (i.e. $x_j f_j / \bar{f}$) by a factor that represents the probability of selecting a certain parent under tournament selection. This results in the formula:

$$x'_i = \sum_{j,k} T_o(i \leftarrow j,k) P^{(n)}_{tour}(j,\vec{x}) P^{(n)}_{tour}(k,\vec{x}),$$

where $P^{(n)}_{tour}(j,\vec{x})$ describes the probability that an individual with label j is selected from a population with distribution $\vec{x}$ during a n-tournament. A n-tournament selection is performed by choosing n individuals uniform at random from the population and selecting the one having the highest fitness. In case of a tie we assume that the individual which was chosen first wins. Given the distribution of the current population we can compute this probability by following the sequence of choice events that give the selected individual:

$$P^{(n)}_{tour}(j,\vec{x}) = \sum_{t=1}^{n} f_<(j,\vec{x})^{t-1} x_j \ (f_<(j,\vec{x}) + f_=(j,\vec{x}))^{n-t} .$$

The t^{th} term of this sum computes the probability that the first $t-1$ selected individual have a fitness lower than the individual with label j, the t^{th} individual has label j, and all subsequent $n-t$ individuals have a fitness smaller than or equal to the fitness of the individual with label j. The sum over all possible values computes the probability that an individual having label j wins the n-tournament. In case of a tie, i.e. there is more than one winner, the individual selected first is taken as the winner. We can also perform a random tie-breaking rule. This will result in the same distribution as there is no specific order in the selection of the individuals that participate in the tournament are selected.

2.3 Evolution strategy (μ,λ) and BGA selection

The evolutionary strategies date back to the '60 [BHmS91, Rec94, Sch95]. We are going to model the (μ,λ) and the $(\mu+\lambda)$ evolution strategy. When using (μ,λ) selection a parent population containing μ parents is used to generate λ offspring. To generate an offspring we select two parents uniform at random from the parent population and apply recombination The best μ offspring are used as the parents for the next generation, so $\lambda \geq \mu$, and as a rule of thumb one often uses $\lambda/\mu \approx 7$.

In order to describe this selection scheme we introduce the truncation operator $\mathrm{Tr} : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$. This operator takes a normalized population $\vec{p}$ and a parameter $\alpha \in [0,1]$ as input and returns a new normalized population contain the α fraction of best individuals out of the original population.

This operator selects a pivot individual i such that $f_<(i,\vec{p}) < 1-\alpha$ and $f_<(i,\vec{p}) + f_=(i,\vec{p}) \leq 1-\alpha$. If more than one value of i satisfies these constraints then we can make an arbitrary choice among these i. Now the operator $\mathrm{Tr}(\vec{p},\alpha)$ is defined as:

$$\mathrm{Tr}(\vec{p},\alpha)_j = \begin{cases} \frac{1}{\alpha} p_j & f_j > f_i \\ \frac{1}{\alpha} \frac{\alpha - (1 - f_<(i,\vec{p}) - f_=(i,\vec{p}))}{f_=(i,\vec{p})} & f_j = f_i \\ 0 & f_j < f_i \end{cases}$$

Given this truncation operator the formula for the (μ,λ) selection) is:

$$\vec{x'} = \mathrm{Tr}(\vec{y}, \frac{\mu}{\lambda}),$$

where the elements of $\vec{y}$ are given by the formula

$$y_i = \sum_{j,k} T_o(i \leftarrow j,k) x_j x_k.$$

Combining these last two formula's gives the complete transmission-function model.

The Breeder Genetic Algorithm is described by Mühlenbein et al. [MSV94]. It uses a $T\%$ truncation selection, which means that the $T\%$ best individuals of the current population are allowed to reproduce. Among these $T\%$ best the parents are selected uniform at random and a new population is generated. Typically T will be between 10 and 50. This selection scheme corresponds to the (μ,λ) selection where $\mu = \lambda T/100$.

When applying BGA the best individual found so far will always be retained. We do not model this type of elitism in the infinite population model.

2.4 Evolution strategy $(\mu+\lambda)$ and CHC selection

The $(\mu+\lambda)$ selection scheme is almost similar to the (μ,λ) selection scheme [BHmS91, Rec94, Sch95]. The difference is that in the $(\mu+\lambda)$ selection scheme the μ new parents are obtained by selecting the best μ individuals from both parents and offspring.

$$\vec{x'} = \mathrm{Tr}\left(\mathrm{Bl}(\vec{x},\vec{y},\frac{\mu}{\mu+\lambda}), \frac{\mu}{\mu+\lambda}\right)$$

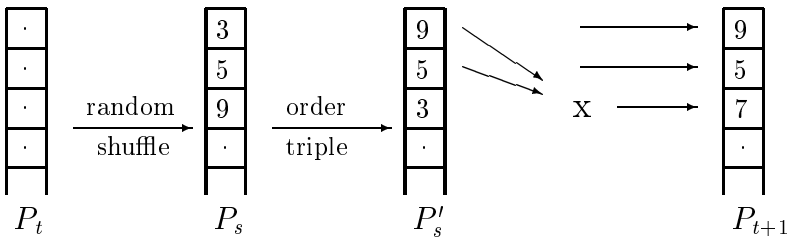

Figure 1: Schematic representation of triple-competition

where $\mathrm{Bl}(\vec{x}_1, \vec{x}_2, \beta)$ computes a blend of vectors $\vec{x}$ and $\vec{y}$ according to the formula

$$\mathrm{Bl}(\vec{x}_1, \vec{x}_2, \beta) = \beta\vec{x}_1 + (1-\beta)\vec{x}_2$$

Here the first term represents the density of the μ parents while the second term represented the contribution of the λ offspring. The vector $\vec{y}$ is again computed by the formula:

$$y_i = \sum_{j,k} T_o(i \leftarrow j,k) x_j x_k$$

The CHC algorithm is described by Eshelman [Esh91]. The CHC algorithm uses an unbiased selection of parents. Given a parents population of size N, a set of N offspring is produced. The next parent generation is obtained by selecting the N best amongst the N parents and their N offspring. This selection scheme corresponds to $(\mu + \lambda)$ selection, where $\mu = \lambda = N$.

An additional feature of the CHC selection scheme is the incest-prevention. CHC does incest prevention by computing the hamming distance between the two parents. If the hamming distance is below a certain threshold than the pair of parents is not allowed to reproduce. Typically CHC starts with a threshold $L/4$, where L is the length of the bit-string. We did not implement this incest prevention scheme in our model as it requires knowledge about the hamming-distances between the different types. But for problems where this type of information is available the modeling of the incest prevention will be relatively straightforward.

2.5 Triple-competition selection

The triple-competition selection is an elitist genetic algorithm [vK97b]. It uses a tournament-like selection of the parents and parents are propagated to the next generation. Figure 1 shows how generation P_{t+1} is generated from generation P_t. A single box corresponds to an individual, the number in the box is its fitness and a stack of such boxes corresponds to a population. The first step involves a random shuffle of the individuals in population P_t resulting in a randomly ordered population P_s. The population P_s is partitioned in sets of three individuals and each triple is ordered such that the best individuals are on top, resulting in an intermediate population P'_s. Next the two top-ranked individuals of each triple are allowed to recombine to generate a single offspring. The two parents and the offspring are added to the population P_{t+1}, so only the lowest ranked individual of each triple is replaced. During a single step of this algorithm $N/3$ offspring are generated, where N is the size of the population. This algorithm can be modeled by means of the following equation:

$$x'_i = \frac{1}{3}\sum_{j,k} T_o(i \leftarrow j,k) P(j,k,\vec{x}) + \frac{2}{3}\sum_{k} P(i,k,\vec{x})$$

where the first sum corresponds the distribution of the newly generated offspring and the second sum corresponds to the distribution of the surviving parents. The distribution of the new population,

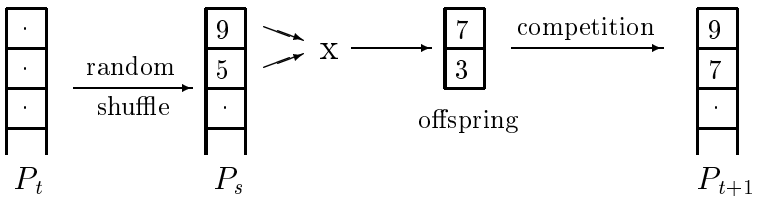

Figure 2: Schematic representation of Elitist recombination

denoted by $\vec{x'}$, consists for one third out of newly generated offspring and two third of surviving parents. The function $P(j,k,\vec{x})$ computes the probability that the individuals j and k are selected as parents, where the individual labeled j is selected first. In order to compute this probability we must differentiate between the two cases ($f_j = f_k$) and ($f_j \neq f_k$), and for each case we again must differentiate between the cases that the third individual is smaller or equal to the worst out of j and k. The following formula computes this term

$$P(j,k,\vec{x}) = \begin{cases} 3\ (f_<(j,\vec{x}) + f_=(j,\vec{x})/3)x_j x_k & \text{if } (f_j = f_k) \\ 3\ (f_<(\mathrm{Sm}(j,k),\vec{x}) + f_=(\mathrm{Sm}(j,k),\vec{x})/2)x_j x_k & \text{otherwise} \end{cases}$$

Here the function $\mathrm{Sm}(j,k)$ selects the label corresponding to the individual having the smallest fitness out of j, and k. The multiplier 3 in both formulas corresponds to the number of possible orders of the three involved individuals given that j is selected before k.

Triple-competition selection as defined above involves elitism as the best individual is always transferred to the next generation. But it is still possible that newly produced offspring has a lower fitness than the individual that is replaced by this new offspring. For the purpose of comparison the strict elitist triple-competition selection is introduced. This selection scheme is similar to the normal triple-competition selection except for the additional rule that an individual will never be replaced by a worse individual. A single operator application also requires knowledge about the fitness of the third parent and therefore $T_o(i \leftarrow j,k,l)$ should be used.

2.6 Elitist recombination

Elitist recombination [TG94] selects parents by doing a random pairing of individuals. As all parents have the exactly same probability of being selected. This corresponds to a uniform/unbiased selection of parents, where earch parent is selected exactly once. Figure 2 shows how the next population P_{t+1} is produced from the current population P_t. Just like triple-competition selection a random shuffle is applied. The resulting population G_s is partitioned in a set of adjacent pairs and for each pair apply the recombination operator in order to obtain two offspring. Next a competition is held amongst the two offspring and their two parents, and the best two out of these four are propagated to the next population P_{t+1}. In Figure 2 one parent, having fitness 9, and one offspring, having fitness 7, is propagated to population P_{t+1}. When using this algorithm When using elitist recombination it is possible, but not guaranteed that parents will survive, so elitism is only used in the case that the offspring is inferior to its parents. The dynamical system representing elitist recombination is:

$$x'_i = \sum_{j,k} T_{er}(i \leftarrow j,k)x_j x_k.$$

The selection of individuals that enter the next population is not visible anymore in this model. This selection mechanism consists of a local competition between parents and their direct offspring and therefore is located inside the transmission function T_{er}. Outside the transmission function we can not observe anymore which where the parents.

Here we are going to study a generalization of elitist recombination that creates a fixed number of offspring $n \geq 2$. In the original definition of elitist recombination one uses $n = 2$. Creation of a number of offspring and accepting only the best few has been suggested by Altenberg [Alt94] under the name soft brood selection. For the elitist recombination and its generalization a modified transmission function $T_{er}(i \leftarrow j,k)$ is used as the selection of survivors is done by means of a local competition between parents and offspring. This in contrast to the usual selection mechanisms that operate on the complete population. Such a local selection scheme can be modeled by modifying the original transmission function $T_o(i \leftarrow j,k)$ that describes the interaction between the operators and the representation used. Each column of the new transmission function $T_{er}(i \leftarrow j,k)$ can be computed independently. Assuming that we have C different types with labels in the range 0 to $C-1$ the transmission function $T_o(i \leftarrow j,k)$ can be represented by a matrix having C rows and $C \times C$ columns. Let $\overrightarrow{t_o^{jk}}$ denote the column with index $jC+k$ of the matrix $T_o(i \leftarrow j,k)$. This column represents the probability distribution of the offspring when applying recombination to a parent of respectively type j and type k.

In the rest of this section we are going to make extensive use of the binomial distribution, which will be denoted as

$$\mathrm{Bin}(n,k,p) = \left(\begin{array}{c} n \\ k \end{array} \right) p^k (1-p)^{n-k}$$

where n is the total number of experiments, k is the number of successful outcomes and p is the probability of a successful outcome.

During the computations we are going to make extensive use of the auxilary function $P_{accept}(P_<, P_=, n, a)$ that computes the probability that an offspring individual is accepted. Accepting an offspring means that the offspring is amongst the best two during the tournament between the two parents and their offspring. The parameters of $P_{accept}(P_<, P_=, n, a)$ are $P_<$ which denotes the proportion of offspring having smaller fitness than the individual under consideration, $P_=$ denotes the proportion of offspring having equal fitness, n denotes the number of additional offspring generated, and a denotes the number of offspring that can be accepted apart from the current offspring. This function can be computed by first conditioning on the number of superior offspring detected followed by a conditioning on the number of offspring having equal fitness

$$P_{accept}(P_<, P_=, n, a) = \sum_{l=0}^{a} \mathrm{Bin}(n, l, 1-P_<-P_=) \sum_{m=0}^{n-l} \mathrm{Bin}\left(n-l, m, \frac{P_=}{P_<+P_=}\right) \min\{1, \frac{a-l+1}{m+1}\}$$

In this formula l denotes the number of offspring having a fitness larger than the fitness of the individual under consideration, and m denotes the number of offspring that have exactly the same fitness. The first binomial distribution in this formula computes the probability that l out of the n other offspring have a higher fitness than the current individual. The proportion of these individuals is $(1-P_<-P_=)$. The second binomial computes the probability that m out of $n-l$ offspring have a fitness equal to the individual under consideration. As we know that all $n-l$ offspring have a fitness lower or equal to the fitness of the current individual the proportion of offspring that has equal fitness is $P_=/(P_<+P_=)$. Given the values of l and m the probability that the current individual is accepted will be equal to $\min\{(a-l+1)/(m+1)\}$. This auxilary function forms the basis of our further computations.

A column of the matrix describing $T_o(i \leftarrow j,k)$ represents the probability density function (also called the mass function) over the space of all possible offspring i for a given pair of parents j and k. Let f_i denote the fitness of individual i and label the parents such that $f_J \geq f_K$, then the offspring can be divided among three sets:

$$\begin{aligned} S_I &= \{i \in \mathcal{S} : f_i \geq f_J\} \\ S_{II} &= \{i \in \mathcal{S} : f_K \leq f_i < f_J\} \\ S_{III} &= \{i \in \mathcal{S} : f_i < f_K\} \end{aligned}$$

Given a column $\overrightarrow{t_o^{jk}}$ of $T_o(i \leftarrow j,k)$ we can compute the corresponding column from $T_{er}(i \leftarrow j,k)$. In order to do so we are first going to compute unnormalized distribution of the offspring which will be denoted by $\vec{o}$ and the distribution of the surviving parents denoted by $\vec{s}$.

The probability than an offspring of type i is obtained by recombination is given by t_i^{jk}. The probability that the offspring is also accepted when applying elitist recombination with n offspring for each pair of parents depends on the set S_x, where $x = I, II$ or III, that offspring i belongs to. If $i \in S_I$ then the offspring is accepted if at most one of the other offspring has a higher fitness,

$$o_i = t_i^{jk} n P_{accept}(f_<(i, \overrightarrow{t_o^{jk}}), f_=(i, \overrightarrow{t_o^{jk}}), n-1, 1)$$

where $f_<(i, \overrightarrow{t_o^{jk}})$ again denotes the fraction of individuals in the distribution $\overrightarrow{t_o^{jk}}$ that have a lower fitness than an individual of type i. If $i \in S_{II}$ then the probability that an offspring is accepted depends on the number of offspring in region S_I and can be computed according to the formula

$$o_i = t_i^{jk} n \sum_{l=0}^{1} \mathrm{Bin}(n-1, l, 1 - f_<(J, \overrightarrow{t_o^{jk}}))\ P_{accept}\left(\frac{f_<(i, \overrightarrow{t_o^{jk}})}{f_<(J, \overrightarrow{t_o^{jk}})}, \frac{f_=(i, \overrightarrow{t_o^{jk}})}{f_<(J, \overrightarrow{t_o^{jk}})}, n-l-1, 0\right).$$

Here l denotes the number of other offspring located in the region S_I, the binomial distribution gives the probability of having l offspring in this region, and P_{accept} computes the probability that the offspring of type i is accepted given that $n-l-1$ other offspring also have a fitness below f_J. At most one offspring from region S_{II} will be selected. If $i \in S_{III}$ then $o_i = 0$ as the individual i will always be rejected.

Next we have to compute the distribution of the surviving parents, denoted by $\vec{s}$. Parent K will only be retained if all offspring belongs to to set S_{III}, so

$$s_K = \mathrm{Bin}(n, n, f_<(K, \overrightarrow{t_o^{jk}}))$$

Parent J is retained when all offspring is in $S_{II} + S_{III}$ or when one offspring is in set S_I and all other offspring are in set S_{III}. This probability is computed by the following formula

$$s_J = \mathrm{Bin}(n, n, f_<(J, \overrightarrow{t_o^{jk}})) + \mathrm{Bin}(n, n-1, f_<(J, \overrightarrow{t_o^{jk}}))\ \mathrm{Bin}\left(n-1, n-1, \frac{f_<(K, \overrightarrow{t_o^{jk}})}{f_<(J, \overrightarrow{t_o^{jk}})}\right)$$

Here the first term corresponds to the case that all offspring has fitness lower than f_J, and the second term corresponds to the case that exactly one offspring a fitness larger than or equal to f_J will all other offspring is located in region S_{III}. In this case the superior offspring will replace parent K instead of parent J.

The vector $\vec{o} + \vec{s}$ describes the unnomalized probability distribution of the two offspring that will be propagated to the next generation. Using these two vectors the column with index $jC + k$ of $T_er(i \leftarrow j,k)$ is given by the formula

$$\overrightarrow{t_{er}^{jk}} = \frac{1}{2}(\vec{o} + \vec{s})$$

By applying this procedure to every column of $T_o(i \leftarrow j,k)$ we obtain the matrix representing $T_er(i \leftarrow j,k)$. An efficient implementation of this computation will require $\mathcal{O}(C^3)$ time where $C = |\mathcal{S}|$, which is the cardinality of the search-space.

3. Infinite population models

We assume that a single application of the crossover operator produces one offspring. The transmission function model computes the expected distribution of this offspring, given the distribution of their parents.

Due to the law of the large numbers the transmission function model can also be used to predict the behavior of a genetic algorithm with an infinite population size. To see this let us assume that we have a population of size N, and let $X_j^{(i)}$ be equal to 1 if sample i is of type j, and 0 otherwise. We know that $X_1, X_2, ...$ are independent identically distributed random variables and $\mathbf{E}X_i = \mu$, where μ is the probability of a individual of type i in the offspring distribution. The strong law of large numbers now states

$$\frac{1}{n}\sum_{i=1}^{n} X_i \to \mu \text{ almost surely , as } n \to \infty$$

Using this fact we now that the proportion of individuals of type i will converge to the proportion predicted by the distribution, as $n \to \infty$, so the actual distribution of an infinitely large offspring population will correspond to the distribution predicted by the transmission function model.

Based on the transmission function model we can model the evolution of a genetic algorithm using such an infinite population size by iterated application of this model. Let us denote a single application of the transmission function model by means of the operator $\mathcal{F} : P \to P$. The initial population of a GA is usually obtained by drawing a uniform random sample. Lets call this distribution G_0. Using a transmission function model we can compute the distribution of an infinite population after one step of evolution by means of the formula $G_1 = \mathcal{F}(G_0)$. As G_1 is a distribution it is a valid input for $\mathcal{F}$ so the distribution of an infinite population after two generations is $G_2 = \mathcal{F}(G_1) = \mathcal{F}(\mathcal{F}(G_0))$, or more generaly after t generations is $G_t = \mathcal{F}^t(G_0)$, where the superscript t denotes iterated application of the function.

4. Cross-competition problem

When applying the building block assumption we assume that the global optimum is composed of a set of building blocks that can be grown independently and mixed afterwards. The requirement that building blocks can be grown independently does not mean that the building blocks will not compete with one another to get more copies. Such a competition between non-overlapping building blocks is called cross-competition As cross-competition reduces the diversity of building blocks within the population it can strongly influence the reliability of a GA.

In order to study this kind of effects we use a mixture of a counting-ones function of length l (also known as ONE-MAX function) and a deceptive trap function of length d [Gol89], so a single individual is represented by a string of $l + d$ bits. The bits can be partitioned in the two sets O and D. The first partition is the counting-ones part. The fitness contribution of this partition is represented by the formula, $f_O(b) = u_O(b)$, where $u_O(b)$ is a function of unitation. This function counts the number of 1-bits in the partition O of string b. The fitness of the deceptive part is given by the formula,

$$f_D(b) = \begin{cases} \alpha d & \text{if } u_D(b) = 0 \\ u_D(b) & \text{otherwise} \end{cases}$$

where $\alpha > 1$. The global optimum of the deceptive part contains only 0-bits, which results in a fitness contribution of αd.

The fitness of a string is determined by the sum of the fitness values of the two partitions ($f = f_O + f_D$). The global optimal solution contains 1-bits in the partition O and 0-bits in partition D. This solution has a fitness of $l + \alpha d$. The string containing 1-bits only has a fitness of $l + d$.

The actual linkage of the bits of the different partitions is unknown, so the loci occupied by a certain partitions can be spread over the bit-string. Instances of the defined problem class contains $l + d$ building blocks of length 1 and one building block of order d represented by the optimal schema for partition D.

This problem has been designed to compare the mixing capabilities of different genetic algorithms when confronted with a problem containing many building blocks of different size. The cross-competition between the building blocks of the non-overlapping partitions O and D is investigated.

5. First estimates of reliability

For the problem presented in section 4 we do not know anything about the linkage of the bits from the building blocks. Therefore we are going to use a uniform crossover operator, as this operator does not have a positional bias [ECS89].

An optimal schema for the deceptive part is difficult to propagate to a subsequent generation. Let $p_{D,1}$ denote the probability that a bit from partition D has value 1. Let us assume that the two parents are chosen uniform at random from this population. When applying uniform crossover to these parents the probability that the offspring contains the optimal schema for the deceptive part is

$$(1 - p_{D,1})^d,$$

where d is the order of the building block. For a random initial population we have $p_{D,1} = 0.5$, so the probability of introducing instances of the optimal schema for the deceptive part is rather small. Generation of instances of this schema is further complicated deceptiveness present in partition D, as all bits tend to converge to their sub-optimal value. Due to these building blocks, that can be combined easily, the value of $p_{D,1}$ tends to increase. In order to find the global optimum we either have to increase the rate at which the instances of the optimal schema for the deceptive part are discovered, or we have to introduce a possibility to retain the strings that match this schema.

The canonical GA uses fitness proportional selection. The expected number of copies of the optimal schema for the deceptive part is dependent upon the value of α and the average fitness. A large value of α is required to get many instances of the optimal schema for the deceptive part. The crossover rate should be adjusted such that the expected fraction of strings that match the optimal schema for the deceptive part in the next generation is maximized. This proportion of instances is predicted by the formula:

$$P_{bb}(t+1) = (1 - p_c)P'_{bb}(t) + p_c(1 - P'_{D,1})^d,$$

where $P'_{bb}(t)$ denotes the proportion of strings matching a the optimal schema in generation t after selection, p_c is the crossover rate, and $P'_{D,1}$ is the proportion of 1-bits in partition D after selection. Isolating p_c results in

$$P_{bb}(t+1) = P'_{bb}(t) + p_c((1 - P'_{D,1})^d - P'_{bb}(t)).$$

This formula is linear in p_c, so the largest $P_{bb}(t+1)$ is obtained if p_c is 0 or 1. Which value is optimal depends on the sign of $((1 - P'_{D,1})^d - P_{bb}(t))$. Note that this result is about the optimal value of p_c during a single generation when the goal is to increase the proportion of building block bb. The above result will also hold when mutation is introduced in the above equation. The case $p_c = 0$ correspond to a GA doing selection and mutation only.

When applying a generational t-tournament selection the best individual will get t times as many copies as the median individual when ranked on fitness. The number of copies an individual obtains is only dependent on its fitness-based rank in the population. If $\alpha > (d+l)/d$ then the highest ranked individuals will all match the optimal schema for the deceptive part. If α gets below this bound the GA with tournament selection can still solve the problem, but a larger value of the tournament size is needed. Also if d increases a larger tournament size is required.

The elitist recombination algorithm does not have any parameters that can be adjusted. This GA uses elitism to preserve parents when the offspring is inferior.

6. Finite population models

In previous section an infinite population model based on the transmission function model is described. Such GA's having an infinite population can be described by a deterministic model, so even though we are using a probabilistic algorithm, the outcome when applying an GA with an infinite population is completely deterministic. When applying a practical GA we know that independent runs of the GA can result in different outcomes and we often apply a GA multiple times in order see whether different runs converge to different points in the search-space. This discrepancy between the deterministic

behavior of genetic algorithms with infinite population size and probabilistic behavior of practical GA's is a consequence of the limited population size.

Due to this limited population size the actual distribution of the population can deviate a lot from the distribution predicted by the transmission function model. This deviation might result in a GA that follows a trajectory different from the one followed by the infinite population GA. Also the point a specific finite GA trail converges to might be different from the fixed point the infinite population GA converges to. These discrepancies do not always have to be a negative influence. There are problem instances where the fixed point of the infinite population model corresponds to a suboptimal solution. In such cases it still might be possible for a finite population GA to locate the optimal solution with a certain probability. So an increase of population size does not always have to lead to a better performance.

In order to build a finite population model we have to correct our model to compensate for the differences in the distribution of the actual population and the distribution predicted by the infinite population model. Here we are going to assume that we can partition the set of all types in $\mathcal{S}$ in two sub-sets, S and $\overline{S}$. These two sets are complementary and their union covers the complete search-space $\mathcal{S}$. These sets will be chosen in such a way that elements of S contain certain parts of the optimal solution, while the elements of $\overline{S}$ do not contain these parts. We are going to assume that elements of set S are much more likely to produce the global optimal solution than the elements of set $\overline{S}$. In the next section we are going to give an example of a problem-instance and the corresponding choice for the set S, and $\overline{S}$.

As elements of set S are relatively likely to result in convergence to the global optimum it is important to know what fraction of the population is part of this set. This proportion might differ between different runs, but it will be possible to estimate the probabilities that a certain fraction of the population is contained in set S in generation t. The probability that k individuals out of a set of λ offspring belong to set S is given by the binomial distribution $\mathrm{Bin}(\lambda, k, p)$ where p denotes the expected proportion of types that belong to set S. Given that k out of N offspring belong to set S we can compute the actual proportion as $p' = k/N$. For large populations p' will be close to p, but for small population sizes the difference between these two might be rather larger. Of course the same procedure can be repeated by splitting the set S over two sets S_1 and $\overline{S_1}$ and again using the binomial distribution to estimate the probabilities of the different sizes of these two sub-sets.

Here we are going to discus the model in case that the complete search-space $\mathcal{S}$ is divided over two partitions. It will be rather straightforward to modify the model to use more complex partitions. First we are going to develop the general model for a finite population genetic algorithm. In order to do so we are going to introduce three auxilary functions $\mathcal{F}_o$, $\mathcal{F}_s$, and $\mathcal{F}$. The actual implementation of these three functions for the different genetic algorithms will be discussed in the next section.

Now we know the distribution of the individuals in S as a function of the population size we can generate a branching tree. The distribution can be used to compute the probabilities of the different branches of the tree. An example of a branching tree when tracing a single set S is shown in Figure 3. The root of the tree correspond to the distribution used to draw the initial population denoted by G_0. The first branch corresponds to the case that no individual in the population is contained in S, the second branch corresponds to case that exactly 1 individual in the population belongs to set S, and the branch with label k corresponds to case that k individuals belong to set S. The probability of branch k is denoted by p_k. Given that the population size is λ we can compute the p_i using the binomial distribution and as we know that exactly one branch should be taken we see that $\sum_{k=0}^{\lambda} p_k = 1$.

The root of the branching tree is the expected distribution of the initial population. Given that the actual proportion of individuals in the initial distribution of G_0 is p we can generate a distribution corresponding to k out of λ individuals belonging to set S. In order to do so we compute the actual proportion $p' = k/\lambda$. The proportion of all types that belong to set S are multiplied by factor p'/p, while the proportions of the other types are multiplied by a factor $(1-p')/(1-p)$. So the relative proportions of individuals that belong to the same set will remain unchanged, but the relative proportions of individuals that belong to different sets changes by this procedure. The dotted lines in

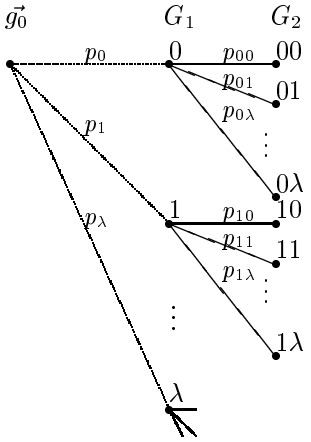

Figure 3: Picture of a branching tree for a finite-population model

the branching tree denote actual distributions of G_0 obtained from the expected distribution of this first generation. This modified distribution is again a normalized distribution. Let $\mathcal{G} : \mathbb{R} \times P \to P$ denote the function that performs this operation, where the real parameter denotes the fraction of individuals that should belong to set S, and P denotes a distribution over the search space. Application $\mathcal{G}$ to a normalized population will result in a normalized population and

$$\sum_{k=0}^{\lambda} \mathrm{Bin}(\lambda, k, p)\mathcal{G}\left(\frac{k}{\lambda}, \vec{g}\right) = \vec{g},$$

where the sum runs over all $\lambda + 1$ possible values of the real parameter p of $\mathcal{G}(p, \vec{g})$. To prove this let us take a single element g_i from distribution $\vec{g}$. Let us assume that type i belongs to set S. In that case the formula becomes $\sum_{k=0}^{\lambda} \mathrm{Bin}(\lambda, k, p)\frac{k}{\lambda}g_i = g_i$. After some reordering we obtain the formula $\sum_{k=0}^{\lambda} \mathrm{Bin}(\lambda, k, p)k = \lambda p$ which is a known identity. The same result hold for an element of set $\overline{S}$, when using the additional identity $\mathrm{Bin}(\lambda, k, p) = \mathrm{Bin}(\lambda, \lambda - k, 1 - p)$.

In Figure 3 we use dotted and solid lines. The dotted lines correspond transitions from the initial population to the population in generation 1. The distribution of the zeroth generation is given by $\vec{g_0}$, which typically corresponds to a uniform distribution over the complete search space $\mathcal{S}$. The solid lines correspond to the possible transitions that lead from generation t to generation $t + 1$, where $t > 0$. We have to discriminate between these two transitions as the transition from G_0 to G_1 just involves the initial population, while the later transitions might involve both surviving parents and newly generated offspring. In cases where we do not have any surviving parents, e.g. a generational genetic algorithm, both transitions will be identical.

Now let us assume that the population of parents contains μ individuals that are used to produce λ offspring. The transition process that corresponds to a single dotted line is shown in Figure 4. The distribution of the parents in the initial population of the transition that leads to node k of generation G_1 can be computed by the formula

$$\overrightarrow{\mathrm{Par}_0} = \mathcal{G}(\frac{k}{\mu}, \vec{g_0}),$$

which is denoted by the symbol $\mathcal{G}_k$ in Figure 4. The probability that of this transition is

$$p_k = \mathrm{Bin}(\mu, k, p),$$

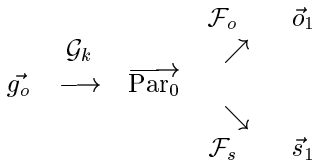

Figure 4: Processes underlying the transitions denoted by the dotted lines in the branching tree. These transitions compute distribution of the new offspring and the surviving parents of generation G_1.

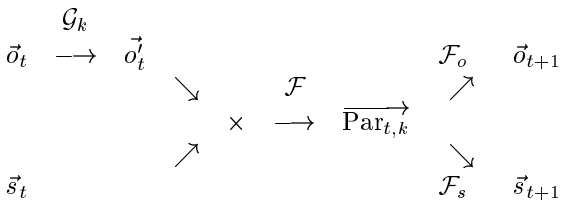

Figure 5: Processes underlying transitions denoted by the solid lines in the branching tree. These transitions compute the new offspring and surviving parents of generation G_{t+1} when given the distribution of survivors and offspring of generation G_t.

where k was the proportion of individuals from $\vec{g_0}$ that belongs to set S. Application of a transmission function will only tell us the expected distribution of the offspring. Taking a finite sample according to a distribution will give us an realization population. The distribution is likely to differ from the distribution it was taken from. We have to account for these differences in distribution in order to build a model for genetic algorithms with finite populations. We have to discriminate between newly generated offspring and surviving parents. The distribution of the surviving parents is already determined during earlier transitions. We introduce two additional functions. The first function is $\mathcal{F}_o : P \to P$ is the part of the transmission that produces the offspring, it takes a parent distribution as its input and generates the distribution of the offspring sample, so $\vec{o_1} = \mathcal{F}_o(\overrightarrow{\text{Par}_0})$, where $\overrightarrow{\text{Par}_0}$ corresponds to the distribution of the parents in the corresponding transition. The second function is $\mathcal{F}_s : P \to P$. It computes the distribution of the surviving parents, so $\vec{s_1} = \mathcal{F}_s(\overrightarrow{\text{Par}_0})$.

Figure 5 shows the actual computation that corresponds to the transitions denoted by a solid line. We have to discriminate between surviving parents $\vec{s}_t$ and offspring $\vec{o}_t$. Again let k denote the number of offspring that belongs to the set S, which ranges from 0 to λ. A distribution corresponding to k offspring that belong to set S is obtained by applying $\vec{o}'_t = \mathcal{G}\left(\frac{k}{\lambda}, \vec{o_t}\right)$. Given an actual realization of the offspring $\vec{o}'_t$ the parents and the offspring have to be merged to create the actual parents population. Let us introduce the function $\mathcal{F} : P \times P \to P$ to denote this process. Again the actual implementation of $\mathcal{F}$ will depend on the type of genetic algorithm that is modelled. So the parents population is obtained by means of the computation

$$\overrightarrow{\text{Par}_{t,k}} = \mathcal{F}(s_{t-1,x}, \mathcal{G}(\frac{k}{\lambda}, o_{t-1,x})).$$

Again the distribution of the parents and the offspring is obtained by means of the computation

$$o_{t,i} = \mathcal{F}_o(\overrightarrow{\text{Par}_{t,k}}) \quad \text{and} \quad s_{t,i} = \mathcal{F}_s(\overrightarrow{\text{Par}_{t,k}})$$

The probability that we end up in this node is given by

$$p_{t,k} = p_x \text{Bin}(\mu, k, p),$$

where p_x is the product of the probabilities of the sequence of transitions that led to the node $P_{t,x}$ that is the predecessor of the node under consideration.

The expected distribution of the population $\vec{g_t}$ is given by the formula:

$$\sum_{k=0}^{\lambda} p_{t,k} P_{t,k}$$

A complete evolution will involve $(N+1)^G$ branches, where N is the size of the population and G is the number of generations. It is not feasible to follow all these branches. In order to obtain a feasible computation the set of followed branches is limited to those branches that have a probability above a certain small threshold δ. A set of subsequent branches that are not followed will be combined into a single branch together with the nearest branch that is followed. The fraction of individuals in set S is assigned a weighted average according to the formula $k' = \sum_{k=l}^{m} p_{xk} p_x k$, where x is the label of the node in the preceding generation where the branching originates from, l and m denote the range of the branches that are combined to a single branch, and p_x denotes the probability that the evolution ends in node x. The resulting value of k' does not have to be integer anymore in the case that branches are merged. This probability of the joined branch is set to $\sum_{k=l}^{m} p_{xk}$. If none of the branches has a probability above the threshold than the net effect will be that the infinite population model is traced.

Given the lower bound on the probability of a traced branch we know that the total number of followed branches of a single generation is smaller than $\lfloor \frac{1}{\delta} \rfloor$. The maximal width of the tree at generation G_i is n^i, so the upper bound on total number of branches followed is given by the formula

$$UB = \sum_{i=0}^{i=G} \min\{\lfloor \frac{1}{\delta} \rfloor, n^i\} \leq 1 + \lfloor \frac{1}{\delta} \rfloor G$$

A tighter upper bound is obtained by observing that the $n^i <= \lfloor \frac{1}{\delta} \rfloor$ for generations i such that $i < g$ where $g = {}^n\log(\lfloor \frac{1}{\delta} \rfloor)$. Using this we get

$$UB = n^{\alpha} / ln(n) + \beta \lfloor \frac{1}{\delta} \rfloor .$$

where $\alpha = \max\{g+1, G\}$, and $\beta = \min\{0, G-g\}$.

Each branch corresponds to a constant amount of computation as each branch corresponds to a single computation of both $\mathcal{F}$ and $\mathcal{G}$. Therefore these bounds are also the bound on the total amount of computation.

Next we are going to discuss the implementation of the functions $\mathcal{F}_o$, $\mathcal{F}_s$, and $\mathcal{F}$ for the different evolutionary algorithms. In case an evolutionary algorithm does not involve elitism the actual implementation of $\mathcal{F}_s$ does not really matter. In these cases we will assume $\mathcal{F}_s(\vec{p}) = \vec{0}$.

6.1 Generational genetic algorithms

In case of the generational GA's the $\mathcal{F}_o$ that computes the distribution of the newly generated offspring corresponds to the transmission function model as described in section 3. When a generational model is applied there are no surviving parents, therefore the function that computes the surviving parents $\mathcal{F}_s(\vec{p}) = \vec{0}$, and the function that combines surviving parents and newly generated offspring will return the distribution of the offspring in for the generational GA's, $\mathcal{F}(\vec{s}, \vec{o}) = \vec{o}$. The number of offspring is equal to the number of parents, so $\mu = \lambda = N$, where N is the population size.

6.2 (μ, λ) and BGA selection

In case of the (μ, λ)-selection all information with respect to the distribution of the low-fitness individuals gets lost due to the application of the truncation function $\mathrm{Tr}(\vec{p}, \alpha)$. Therefore it is not possible to apply the same procedure as we used for the generational genetic algorithms. In order to obtain

a finite population model we have to apply the truncation function after selecting the branches. In this case we have to store the distribution of the population before application of the truncation step in each of the nodes. We can do this by defining $\mathcal{F}_o : P \to P$ as the operator that returns the intermediate population $\vec{y}$, see section 2.3. The function $\mathcal{F}$ will be implemented as follows

$$\mathcal{F}(\vec{s}, \vec{o}) = \mathrm{Tr}(\vec{o}, \frac{\mu}{\lambda}),$$

so the distribution of $\vec{s}$ does not play a role during the application of $\mathcal{F}$.

6.3 $(\mu + \lambda)$ and CHC selection
In case of the $(\mu+\lambda)$-selection both the truncation and the blending function used in section 2.4 requires a special treatment. The Blending function has to be delayed as the branching only applies to the newly created offspring and not to all individuals in the population. Again we define $\mathcal{F}_o : P \to P$ as the operator that returns the intermediate population $\vec{y}$, see section 2.3. Furthermore we have $\mathcal{F}_s(\vec{s}) = \vec{s}$, so it returns the parents and

$$\mathcal{F}(\vec{s}, \vec{o}) = \mathrm{Tr}\left(\mathrm{Bl}(\vec{s}, \vec{o}, \frac{\mu}{\mu+\lambda}), \frac{\mu}{\mu+\lambda})\right).$$

6.4 Triple-competition selection
In case of the triple-competition selection we can split the formula given in section 2.5 in the two parts. The first part corresponds to the distribution of the offspring

$$\mathcal{F}_o = \sum_{j,k} T_o(i \leftarrow j, k) P(j, k, \vec{x})$$

while the second part corresponds to the distribution of the parents

$$\mathcal{F}_s = \sum_{k} P(i, k, \vec{x})$$

After selecting the branch taken we have to blend the two parts.

$$\mathcal{F}(\vec{s}, \vec{o}) = \mathrm{Bl}(\vec{s}, \vec{o}, \frac{2}{3})).$$

The number of newly generated offspring that belongs to set S will vary between 0 and $N/3$.

6.5 Elitist recombination
In section 2.6 where elitist recombination was described, we already differentiate between newly generated offspring and surviving parents, denoted by respectively $\vec{o}$ and $\vec{s}$. So we get $\mathcal{F}_s = \frac{\vec{s}}{|\vec{s}|}$, $\mathcal{F}_o = \frac{\vec{o}}{|\vec{o}|}$ and $\mathcal{F}$ merges these two distributions in the right proportions. The only reason for temporarily splitting up the two parts is that the proportion of individuals amongst the offspring is described by a distribution, while the proportion amongst the parents is already fixed in an earlier stage of the process. In order to apply the function $\mathcal{G}_k$ we need to have a population $\vec{o}$ with an integer number of individuals in it. Therefore we round the proportion of offspring to the nearest integer which is given by the formula $\mathrm{Round}(N|\vec{o}|)$. The blending of surviving parents and newly generated offspring will also be done using this modified fraction of the offspring population.

7. Using equivalence classes

In order to trace the models we must implement the transmission functions $T(i \leftarrow j, k)$ that have been defined in section 2. Normally the transmission function is defined as an operator over the complete search space $T : \mathcal{S}^3 \to [0, 1]$. Tracing the evolution of such a dynamical system will require a lot of

computation. A single application involve $|\mathcal{S}|^3$ computations, where $|\mathcal{S}|$ denotes the cardinality of set $\mathcal{S}$. When tracing the evolution for the problem we are studying here we can get a much more efficient method by mapping the original search space $\mathcal{S}$ to a more compact space $\mathcal{V}$ where each element of $\mathcal{V}$ represents an equivalence class containing a set of elements from $\mathcal{S}$ and then to define the transmission function on the space of equivalence classes $\mathcal{V}$. The elements of an equivalence class should satisfy two conditions in order to apply the transmission function models to $\mathcal{V}$ instead of $\mathcal{S}$:

(1) all elements of an equivalence class should have the same fitness, and

(2) the distribution over the elements of an equivalence class should be known and constant.

The fitness should be the same such that all individuals in an equivalence class behave identical under selection. The distribution should be constant such that transmission function remains constant during the evolution. Of course we must adjust the distribution of the initial population in order to account for the differences in cardinality of the sets that are represented by the elements of $\mathcal{V}$.

In the rest of this paper we are studying problem that consist of a concatenation of functions of unitation. The fitness of a function of unitation is solely determined by the number of 1-bits in the string. Assuming that the problem consist of a concatenation of n functions of unitation where sub-function i has a length of l_i bits The total length of now becomes $l = \sum_{i=1}^{n} l_i$.

Now we introduce the space $\mathcal{V}$ having cardinality $|\mathcal{V}| = \prod_{j=1}^{n}(l_i + 1)$. A mapping from $\mathcal{S}$ into $\mathcal{V}$ is obtained by the following formula

$$F : \mathcal{S} \to \mathcal{V} = \sum_{i=1}^{n} u_n \prod_{j=1}^{i-1}(l_j + 1),$$

where u_i is the number of 1-bits in part i and l_i is the maximal number of 1-bits in part i, and we assume that $\prod_{i=1}^{0} v = 1$. Given an element $v \in \mathcal{V}$ we can compute the number of 1-bits in part k using the formula

$$\alpha(v,k) = \frac{v}{\prod_{i=1}^{k-1}(l_i+1)} \bmod (l_k + 1)$$

When using a random initial population, which corresponds to a uniform distribution over $\mathcal{S}$ the distribution in $\mathcal{V}$ computed as follows.

$$P(v) = \frac{1}{|\mathcal{S}|} \prod_{j=1}^{n} \begin{pmatrix} l_j \\ \alpha(v,j) \end{pmatrix}$$

For a single sub-function the probability that the offspring contains i 1-bits given parents that contain respectively j and k 1-bits is given by the given by the following recursive formula

$$\begin{array}{rrl} p(j,k,i,l) = & (1-\frac{j}{l})(1-\frac{k}{l}) & p(j,k,i,l-1)+ \\ & (1-\frac{j}{l})\frac{k}{l}\frac{1}{2} & (p(j,k-1,i,l-1)+p(j,k-1,i-1,l-1))+ \\ & \frac{j}{l}(1-\frac{k}{l})\frac{1}{2} & (p(j-1,k,i,l-1)+p(j-1,k,i-1,l-1))+ \\ & \frac{j}{l}\frac{k}{l} & p(j-1,k-1,i-1,l-1) \end{array}$$

where $p(j,k,i,n)$ represents the probability that an offspring string with i 1-bits is obtained from two parent string having respectively j and k 1-bits where l is the number of bits in this sub-function. The boundary conditions are $p(1,1,1,1) = p(0,0,0,1) = 1$, $p(1,0,1,1) = p(0,1,1,1) = p(1,0,0,1) = p(0,1,0,1) = \frac{1}{2}$, $p(0,0,1,1) = p(1,1,0,1) = 0$, and when $j < 0$, $k < 0$, or $i < 0$ we have $p(j,k,i,n) = 0$. By the symmetry of the uniform crossover we have $p(j,k,i,n) = p(k,j,i,n)$.

The transmission function $T(i \leftarrow j,k)$ can be represented by a $(n \times n^2)$-matrix of transmission probabilities where n denotes the cardinality of $|\mathcal{V}|$. As the individual partitions can evolve independently

of one another under uniform crossover we can take the products of the probabilities of each of the outcomes, which results in the formula.

$$T_o[i, j \cdot k] = \prod_{m=1}^{n} p(\alpha(i,m), \alpha(j,m), \alpha(k,m), l_m).$$

Modeling by means of equivalence classes ignores all details with respect to the distribution of one-bits over all loci that are bundled. In fact we are assuming that the probability of a 1-bit is independent of the locus. This will not be a limitation when tracing a genetic algorithm with an infinite population, and even when using genetic algorithms with a small population this will hold when averaging over a large number of runs. But within a single run the probability of finding a 1-bit at a specific locus can easily deviate from the expected probability due to drift. As this type of deviation is ignored when using the equivalence-classes we are in fact modeling a genetic algorithm without drift.

8. Results

Using the models we investigate the behavior of the different selection schemes when applied to the problem described in section 4. Because of differences in the characteristics of the different schemes, as described in section 5, different problem instances are needed when investigating cross-competition problems. All models use the same crossover rate $p_c = 1$. For the generational GA and the GA with 2-tournament selection we also did some experiments using $p_c = 0.8$. No mutation operator is applied because this operator is primarily meant to introduce alleles that got lost. As an infinite population does not lose alleles due to sampling errors we do not need a mutation operator.

Tracing such models corresponds to running the genetic algorithm with an infinite population. If the optimum has a decreasing probability when tracing the model then a run of the corresponding real genetic algorithm will not converge reliably to this optimum either. It is possible that a real GA with a finite population does find the optimum due to random effects, but it will not converge to the optimum with a high probability. Therefore we interpret results from the models as an upper bound on the reliability of the corresponding real GA's.

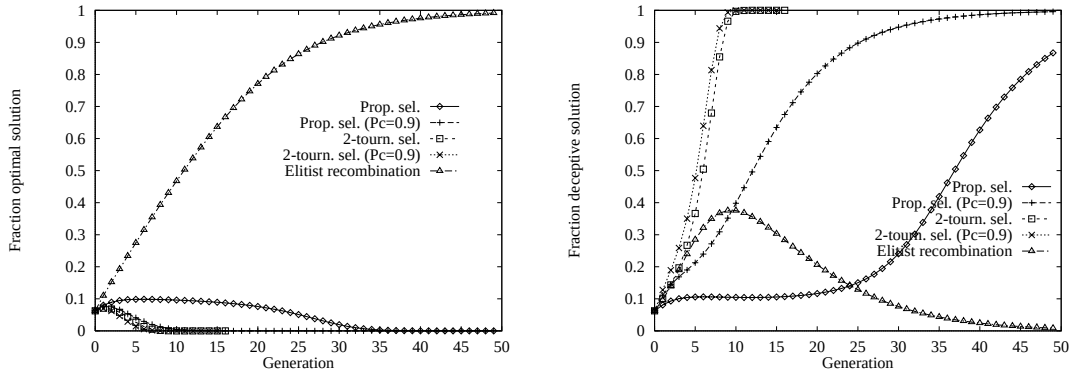

Figure 6: Problem instance with $\alpha = 1.6$, $d = 4$, and $l = 4$. Fraction strings matching the optimal schema (left) and the deceptive schema (right) for the deceptive part.

Figure 6 shows the fraction of strings that match the optimal schema for the deceptive part (left graph) and the fraction of strings that match the deceptive schema (right graph) for a problem with parameters $\alpha = 1.6$, $d = 4$, and $l = 4$. For all algorithms the fraction of strings matching the deceptive schema increases initially. Elitist recombination converges to the global optimum, the other algorithms do not.

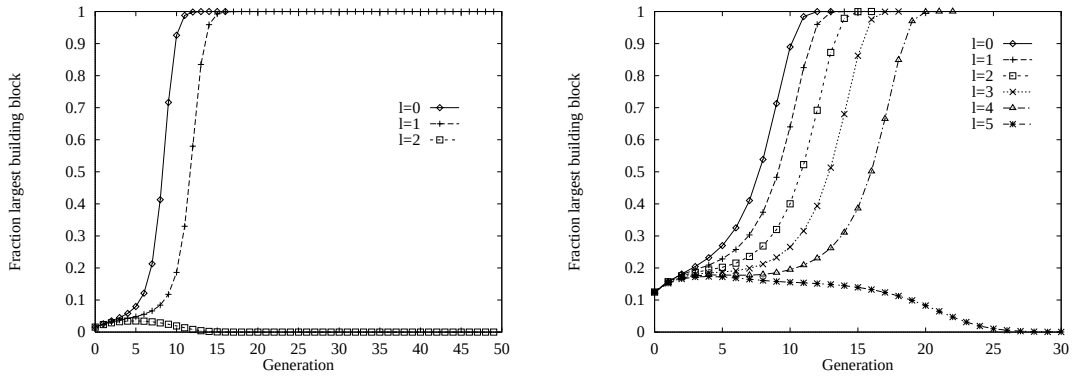

Figure 7: Fraction of strings containing building block of order d for proportional selection when $\alpha = 2.7$, $d = 6$, and $l = 0, 1,$ or 2 (left) and for 2-tournament selection when $\alpha = 1.1$, $d = 3$, and $l = 0, 1, 2, 3, 4,$ or 5 (right).

For the other experiments a fixed value is chosen both for α and for the order of the deceptive part, while the size of the counting one partition is varied. The evolution of the fraction of strings matching the optimal schema for the deceptive part is observed. If the population converges to the optimal schema in the deceptive part then it is likely to converge to the global optimal solution.

When using fitness proportional selection a large value of α is required in order to obtain enough copies of good solutions. The left graph of Figure 7 shows the results when applying a canonical GA to a problem having $d = 6$, $l = 0 - 2$ and $\alpha = 2.7$. If $\alpha \leq 2.55$ then the canonical GA did not converge at all. If the value of $l = 1$ the GA converges slower then in the case $l = 0$. If $l > 1$ then the GA is not able to locate the optimum anymore. The cross-competition between the one-bit building blocks in the partition O and the optimal schema of the deceptive part is too strong to keep instances of this schema in the population. When $p_c = 0.8$ the model of the canonical GA always converges to the non-optimal solution.

When using 2-tournament selection we can not process high order building blocks. The right graph of Figure 7 shows the results for a problem with $d = 3$, $l = 0 - 5$, and $\alpha = 1.1$. If the value of l is increased the GA converges slower. When $l = 5$ the GA does not converge to the optimum anymore. When $p_c = 0.8$ the model of the 2-tournament selection always converges to the non-optimal solution.

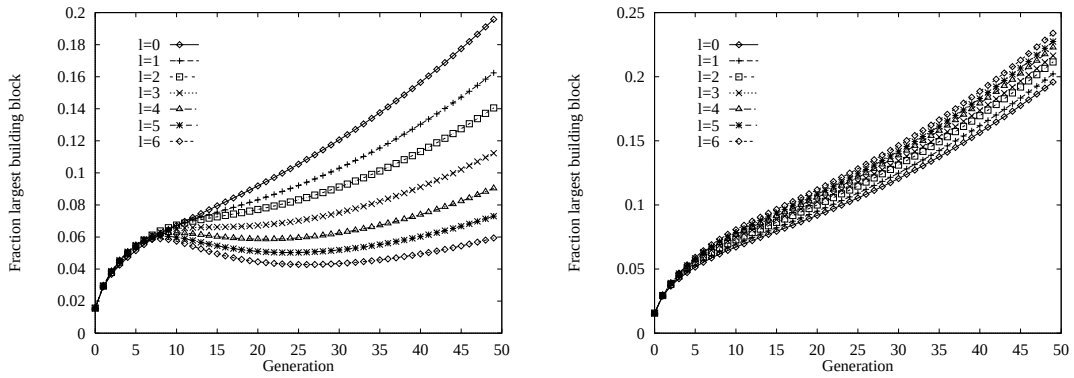

Figure 8: Fraction of strings containing building block of order d for elitist recombination when $d = 6$, $l = 0, 1, 2, 3, 4, 5,$ or 6, and $\alpha = 1.1$ (left) or $\alpha = 2.7$ (right).

The elitist recombination algorithm always converges to the optimum due to the elitism. The left graph of Figure 8 shows the results for a problem with $d = 6$, $l = 0 - 6$, and $\alpha = 1.1$. The right graph of Figure 8 corresponds to $\alpha = 2.7$. For $\alpha = 1.1$ the convergence significantly slows down as l increases. Using this low value of α the second best solution will not contain the building block. When $\alpha = 2.7$ all individuals matching the optimal schema for the deceptive part are relatively fit. In that case the convergence velocities do not depend strongly on l anymore. When comparing the performance of elitist recombination and fitness proportional selection we see that elitist recombination converges slower. But elitist recombination performs reliably over a wider range of both l and α. Elitist recombination is less sensitive to the value of d than 2-tournament selection.

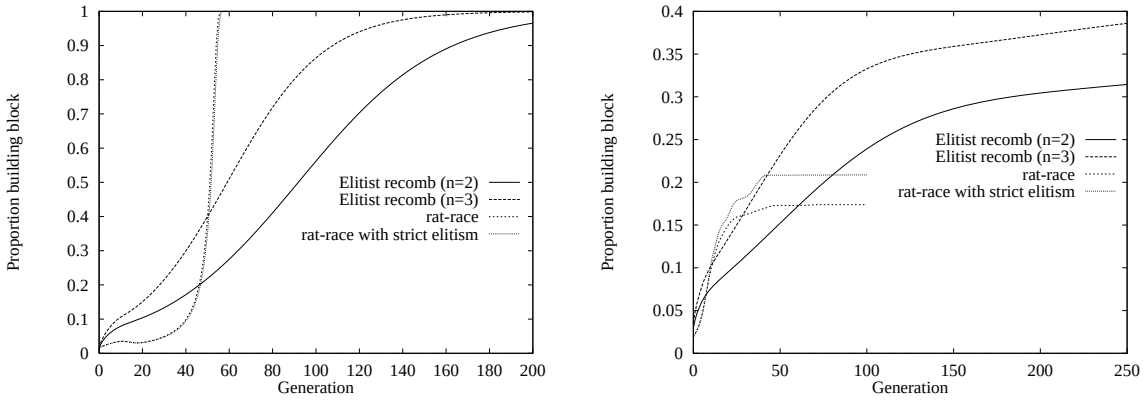

Figure 9: Comparison of the different algorithms for population size $N = \infty$ (left) and $N = 20$ (right)

A comparison is made between elitist recombination, elitist recombination with three offspring per couple, triple-competition, and triple-competition with strict elitism. The models have been evaluated for a problem instance with $\alpha = 1.5$, $d = 6$, and $o = 6$. For the finite population model a threshold $\delta = 0.001$ is applied when tracing the first three models. Only in case of triple-competition selection with strict elitism a value $\delta = 0.01$ is used to limit the amount of computation.

Figure 9 shows the results for an infinite population (left) and for a population of size 20 (right). The horizontal axes represent the number of generations and the vertical axes denotes the proportion of individuals that contain the optimal schema for part D. The number of function evaluations per generation differs for the different algorithms. Elitist recombination uses $Nn/2$ evaluations per generation, while triple-competition uses $N/3$ evaluations per generation. In case of the infinite population model the triple-competition algorithm converges more rapid than the elitist recombination algorithm. When using $N = 20$ elitist recombination performs more reliably. Large differences can be observed between the variants of the different algorithms. In case of elitist recombination increasing the number of offspring per couple results in a faster convergence and a higher probability of finding the optimum. In case of the triple-competition selection the variant with strict elitism located the optimum with a higher probability.

We have performed both simulations and experiments for all the selection schemes discused in previous sections. The simulations are done by tracing the behavior of both the finite population model and the infinite population model. Plots are shown for population size $N = 10, 20, 40, 80, 160,$ and∞. We have also performed experiments using implementations of the different selection schemes. The experimental curves are obtained by computing the averages over 1,000 or 10,000 runs. The population size N refers to the number of parent individuals that are used to create the set of offspring. We have chosen this definition as the size of the set of parent individuals is a measure for the amount of genetic information that can be transferred to the next generation. Note that sometimes the population size is defined as the number of offspring that is produced during a single generation. In case of the real

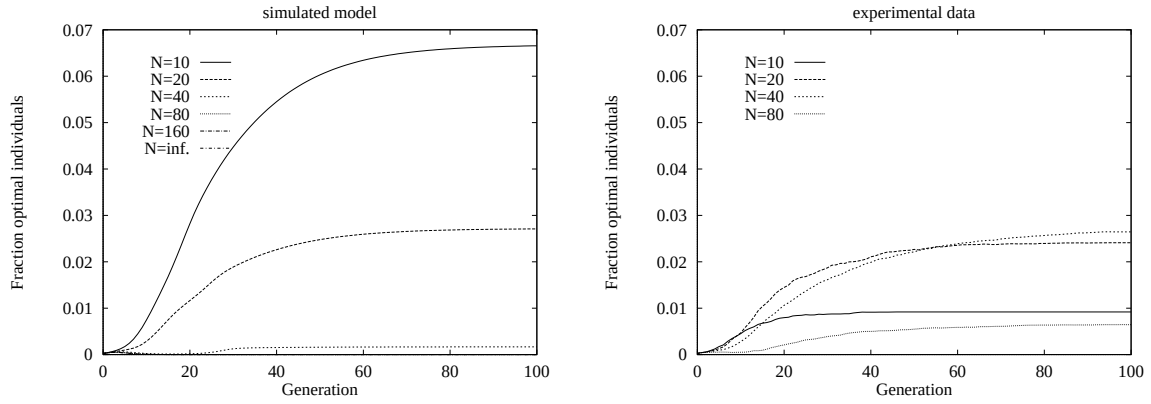

Figure 10: Fractions of optimal solutions for the canonical genetic algorithm both simulated (left) and experiments (right).

genetic algorithms the individuals are coded as bit-strings of length 12. It is important to note that the problem-instance used in the model and the problem-instance used during the experiments are not completely identical. The discrepancy arises due to the formulation in terms of equivalence classes used in case of the model. When grouping a set of loci in an equivalence class we using two assumptions. First of all we assume that all loci within this class have the same probability of containing a one-bit. Second we assume that there is no linkage disequilibrium between the alleles at different loci. Both assumptions will hold for large populations, but when using small populations these assumptions might be violated. When using a small population genetic drift can take place among the loci in the same equivalence class. The fitness of an individual is solely determined by the total number of one-bits in an equivalence class, so there is no fitness-benifit for having the one-bits at certain locations within the equivalence class. Due to drift the probabilities of finding a one-bit at different loci within the equivalence class might differ. Therefore the predicted curves by the model-simulation are likely to be too optimistic.

Figure 10 shows the results when using a generational genetic algorithm with fitness proportional selection. A single generation requires N fitness evaluations. This genetic algorithm is not likely to find the global optimal solution. In order to get smooth curves we averaged the experiments over 10,000 independent runs. The match between simulations and experiments is not so good in this case, but if we disregard the results for the population of size 10 then we see that the experiments show a deterioration of performance when increasing the population size. The same behavior is observed in the simulation.

Figure 11 shows the results when applying tournament selection. A single generation requires N fitness evaluations. Tournament selection results in a very fast convergence. The optimal solution should be found before generation 17, otherwise it will nog be found at all. The fact that the curve for $N = 20$ goes down at approximately generation 5 and later starts to rise again is due to the fact that if an optimal solution is found and preserved for a number of generations then it will rapidly take over the complete population.

Figure 12 shows the results for a $(\mu + \lambda)$ selection where $\mu = \lambda = N$, so a single generation requires N fitness evaluations in this case. The simulations show the same qualitative behavior as the experiments.

Figure 13 shows the results for the (μ, λ) selection where $\mu = N$, and $\lambda = 7N$, so the production of generation G_t where $t > 0$ requires $7N$ fitness evaluations. Again the qualitative behavior of the simulated and experimental results match well.

Figure 14 shows the results for the triple competition. A single generation G_t for $t > 0$ requires

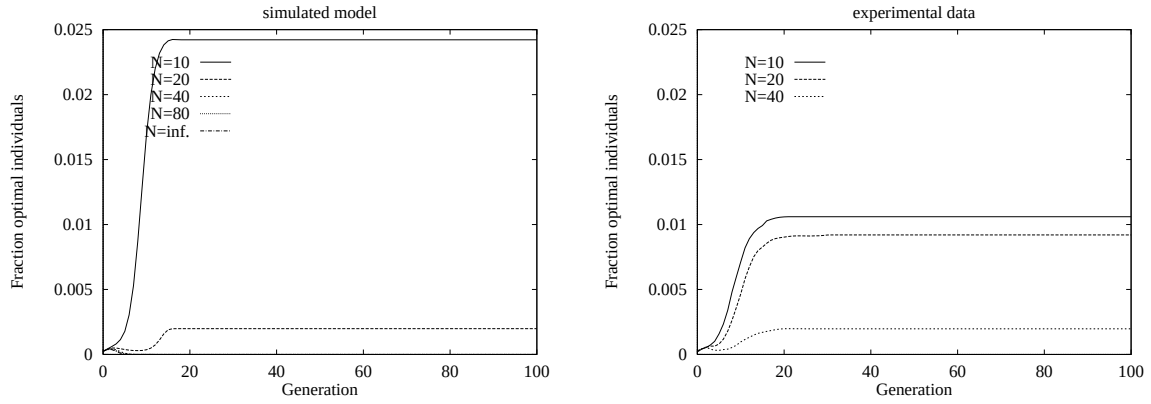

Figure 11: Fractions of optimal solutions for the generational genetic algorithm with 2-tournament selection both simulated (left) and experiments (right).

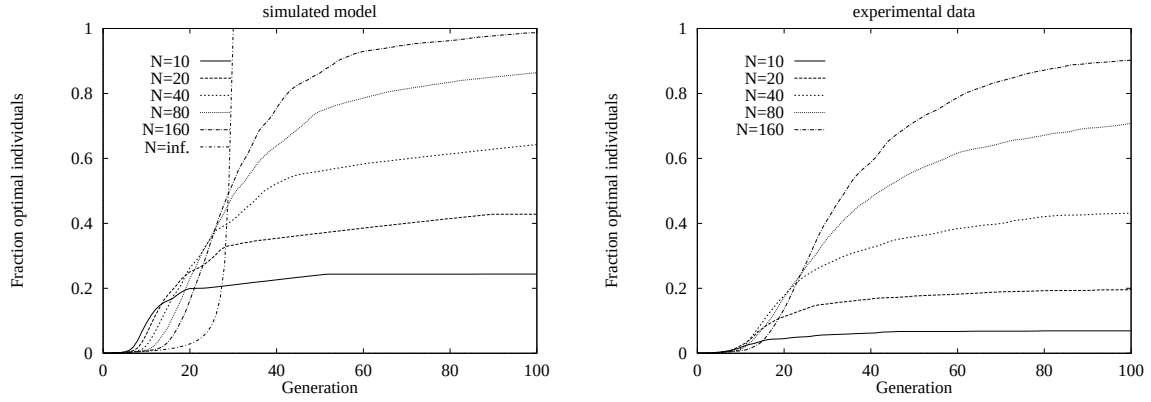

Figure 12: Fractions of optimal solutions for the $(\mu + \lambda)$ selection (CHC) both simulated (left) and experiments (right).

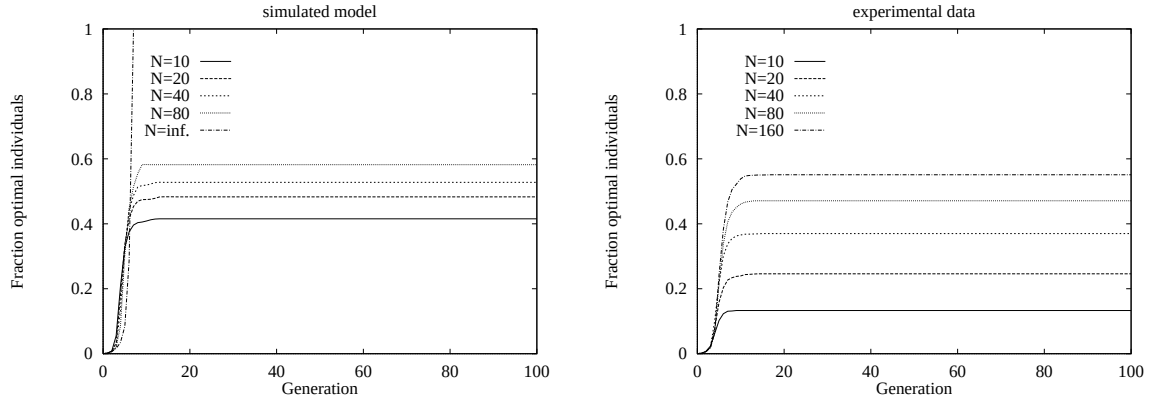

Figure 13: Fractions of optimal solutions for the (μ, λ) selection (BGA) both simulated (left) and experiments (right).

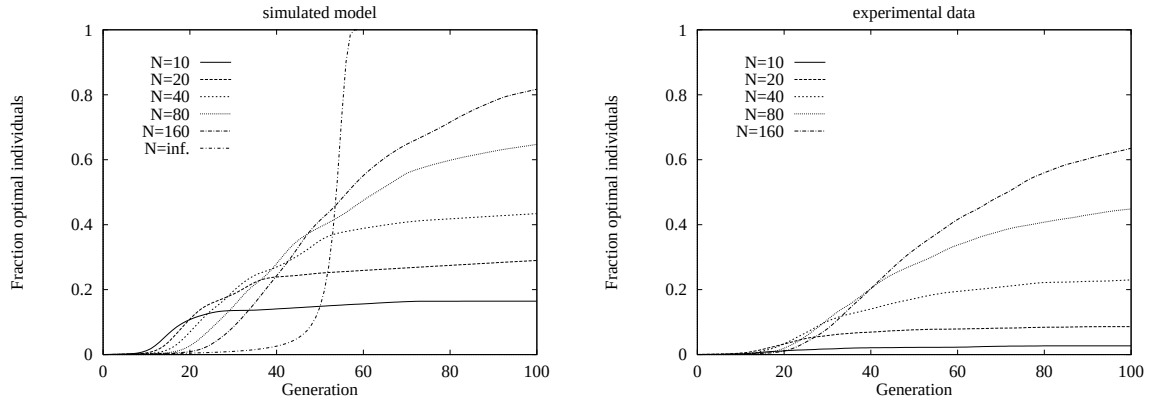

Figure 14: Fractions of optimal solutions for triple competition both simulated (left) and experiments (right).

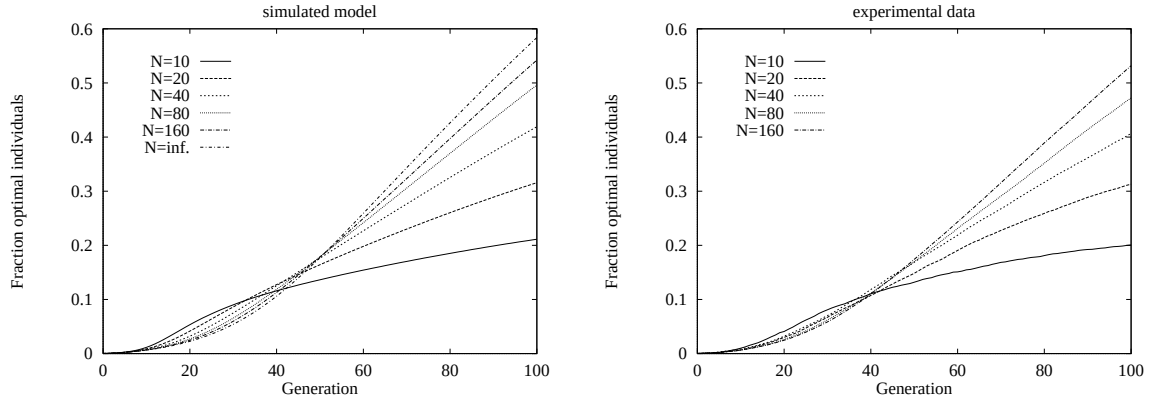

Figure 15: Fractions of optimal solutions for elitist recombination both simulated (left) and experiments (right).

only $N/3$ fitness evaluations.

FigureModelElRec shows the results for the elitist recombination algorithm. A single generation G_t for $t > 0$ requires only N fitness evaluations. Here we observe a very close match between simulations and experiments. We see this as an indication that elitist recombination is very robust against the negative influences of drift due to small population sizes.

9. Conclusions

The behavior of a genetic algorithm can depend strongly on the choice of the selection scheme. Elitism can be used to increase the performance of genetic algorithms. Here a comparison is made between a number of algorithms that involve elitism by evaluating their transmission function models.

Even though we are using a very simple model, where we only discriminate between two types of individuals, i.e. those containing a the optimal value of a certain high order building block and those that do not, we see a good prediction by the simulation of the qualitative behavior of the different genetic algorithms.

It is interesting to observe the quantitative differences between simulation and experiments. One

of the main reasons for these discrepancies is the fact that the simulation ignores drift of bits that are in the same equivalence class. We see the close match between the simulations and experiments for elitist recombination as an indication for the potential high robustness of such schemes involving family competition.

Acknowledgements I would like to thank Joost N. Kok, Dirk Thierens, Han La Poutré, and the anonymous referees for their useful comments.

References

[Alt94] L. Altenberg. The evolution of evolvability in genetic programming. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic programming*, chapter 3, pages 47–74. M.I.T. Press, 1994.

[BHmS91] T. Bäck, F. Hoffmeister, and H.-P. Schwefel. A survey of evolution strategies. In *proceedings of the Fourth International Conference on Genetic Algorithms*, pages 2–9, 1991.

[Boo92] L.B. Booker. Recombination distributions for genetic algorithms. In *Foundations of Genetic Algorithms - 2*, pages 29–44, 1992.

[Bra90] R.N. Brandon. *Adaption and Environment*. Princeton University Press, 1990.

[ECS89] L.J. Eshelman, R.A. Caruana, and J.D. Schaffer. Biases in the crossover landscape. In *proceedings of the Third International Conference on Genetic Algorithms*, pages 10–19, 1989.

[Esh91] L.J. Eshelman. The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In G. Rawlins, editor, *Foundations of Genetic Algorithms - 1*, pages 265–283. Morgan Kaufmann, 1991.

[GDT93] D.E. Goldberg, K. Deb, and D. Thierens. Towards a better understanding of mixing in genetic algorithms. *Journal of the Society for Instrumentation and Control Engineers, SICE*, 32(1):10–16, 1993.

[Gol89] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[Hol75] J.H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* The university of Michigan Press/Longman Canada, 1975.

[KF95] J.N. Kok and P. Floréen. Tracing the behavior of genetic algorithms using expected values of bit and walsh products. In *proceedings of the Sixth International Conference on Genetic Algorithms*, pages 201–208, 1995.

[MSV94] H. Mühlenbein and D. Schlierkamp-Voosen. Predictive models for the breeder genetic algorithm. *Journal of Evolutionary Computation*, 1(1):25–49, 1994.

[Rec94] I. Rechenberg. *Evolutionsstrategie '94*. Frommann-Holzboog, 1994.

[Rud94] G. Rudolph. Convergence analysis of canonical genetic algorithms. *IEEE transactions on*

Neural Networks, 5(1):96–101, 1994.

[Rud96] G. Rudolph. Convergence of evolutionary algorithms in general search spaces. In *proceedings of the Third IEEE conference on Evolutionary Computation*, pages 50–54. IEEE press, 1996.

[Sal71] W.C. Salmon. *Statistical Explanation and Statistical Relevance*. University of Pittsburgh Press, 1971.

[Sch95] H.-P. Schwefel. *Evolution and Optimum Seeking*. Sixth-Generation Computer Technology Series. Wiley, New York, 1995.

[SEO91] J.D. Schaffer, L.J. Eshelman, and D. Offutt. Spurious correlations and premature convergence in genetic algorithms. In G. Rawlins, editor, *Foundations of Genetic Algorithms - 1*, pages 102–112. Morgan Kaufmann, 1991.

[Sla70] M. Slatkin. Selection and polygenic characters. In *Proceedings of the National Academy of Science U.S.A. 66*, pages 87–93, 1970.

[sP93] M. srinivas and L.M. Patnaik. Binomially distributed populations for modelling gas. In *proceedings of the Fifth International Conference on Genetic Algorithms*, pages 138–145, 1993.

[TG93] D. Thierens and D.E. Goldberg. Mixing in genetic algorithms. In S. Forrest, editor, *proceedings of the Fifth International Conference on Genetic Algorithms*, pages 38–45. Morgan Kaufmann, 1993.

[TG94] D. Thierens and D.E. Goldberg. Elitist recombination: an integrated selection recombination GA. In *proceedings of the First IEEE conference on Evolutionary Computation*, pages 508–512. IEEE Press, 1994.

[vK97a] C.H.M. van Kemenade. Cross-competition between building blocks, propagating information to subsequent generations. In *proceedings of the Seventh International Conference on Genetic Algorithms*, pages 1–8, 1997.

[vK97b] C.H.M. van Kemenade. Modeling elitist genetic algorithms with a finite population. In *Proceedings of the third Nordic workshop on Genetic Algorithms*, pages 1–10, 1997.

[Whi92] D. Whitley. An executable model of a simple genetic algorithm. In *Foundations of Genetic Algorithms - 2*, pages 45–62, 1992.