Centrum voor Wiskunde en Informatica
REPORT*RAPPORT*

Cluster evolution strategies -- enhancing the sampling density function using representatives --

C.H.M. van Kemenade

Computer Science/Department of Software Technology

CS-R9648 1996

Report CS-R9648
ISSN 0169-118X

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

Cluster Evolution Strategies
– enhancing the sampling density function using representatives –

C.H.M. van Kemenade

CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
kemenade@cwi.nl

Abstract

Most randomized search methods can be regarded as random sampling methods with a (non-uniform) sampling density function. Differences between the methods are reflected in different shapes of the sampling density function and in different adaptation mechanisms that update this density function based on the observed samples. We claim that this observation helps in getting a better understanding of evolutionary optimizers. An evolutionary algorithm is proposed, that uses an enhanced selection mechanism which uses not only fitness values but also considers the distribution of samples in the search-space. After a fitness based selection, the individuals are clustered, and a representative is selected for each cluster. The next generation is created using only these representatives. The set of representatives is usually small and the efficient incorporation of local search techniques is possible.

AMS Subject Classification (1991): 68T20
CR Subject Classification (1991): G.1.7, I.2.8
Keywords & Phrases: evolutionary algorithms, optimization
Note: This paper was presented at the IEEE conference on Evolutionary Computation, Nagoya, Japan 1996

1. Introduction

Evolutionary algorithms belong to the large class of randomized search methods, containing amongst others, Monte-Carlo methods, simulated annealing [1, 6], clustering methods with random sampling [11], evolution programming [2, 4], and evolution strategies [2, 9, 10]. All these methods have in common that they try to improve upon pure random search by focusing the search process on a subset of the search space. The robustness of such algorithms is strongly related to the probability that the global optimum is located in this subset. If it is not, then one can only hope to find a sub-optimal solution of reasonable quality. There are at least two ways to circumvent this problem. The first method is to avoid the exclusion of parts of the search space. Simulated annealing takes this approach. Instead of restricting the search to a subset, the algorithm modifies the sampling density function. As the search proceeds this density function gets increasingly peaked around the current best solution. Even though it is never impossible to find the location of the global optimum during the next iteration of the simulated annealing algorithm, the probability that this will happen can become infinitely small. The second method is to control the subset selection process by requiring a certain amount of evidence before a restriction of the search-space is allowed. This second mechanism is implicitly used in many population based evolutionary algorithms. In this case the distribution from the current population in through the search-space combined with the selection mechanism and the evolutionary operators defines a sampling density function and hence the subset of the search space which is reacheable.

The main difference between these two approaches is that the first approach does a sampling of the search space according to a density function that is non-zero everywhere, while the second approach does allow the density function to become zero at certain locations. For this reason it is often argued

that the first approach is superior to the second one, as convergence to the optimum with probability one can often be proved when allowing an infinite amount of computation. On the other hand, under these conditions most methods using the second approach can also be proven to find the optimum with probability one, when restarts are allowed. It is interesting to compare these two approaches when only a limited amount of computation is allowed. But this kind of questions seems to be impossible to answer, without having a more specific definition of the optimization problem to be handled. The evolutionary algorithm proposed in this paper applies the second approach.

2. Cluster Evolution Strategy

Many evolutionary algorithms are known to have problems with general function optimization tasks. A possible reason for these difficulties is that most evolutionary algorithm have a strong bias towards large regions with a high overall fitness [8]. When the global optimum is present within this region this behavior does not cause a problem, but if the region of attraction of the global optimum does not coincide with this region then the evolutionary algorithm is likely to converge to a suboptimal solution. At first sight this might seem to be a problem that is inherent to evolutionary algorithms. The fast convergence to regions having a high average fitness is one of the important reasons why an evolutionary algorithm is able to outperform a pure random search in many cases. But as we shall try to show, a careful design of an evolutionary algorithm can increase the reliability of the algorithm. In a given evolutionary algorithm the sampling density function is mainly determined by the distribution of the population over the search-space and the definition of the genetic operators. Large regions having a high average fitness will be discovered early, and will contain many individuals. As a result a kind of avalanche effect can occur, because the large number of individuals in this region will result in a high probability of introducing new individuals in the same region. A simple way to lower the probability of such oversampling is to assign a limited lifetime to individuals, for example by using a generational approach. Such a limited lifetime can prevent an individual from producing an agglomerate of almost identical individuals in its neighborhood. A different way to prevent such agglomerates is the application of more disruptive operators. A disruptive operator is less likely to produce nearly identical copies of the parent, and therefore is less susceptible to premature convergence even if individuals have a long lifetime [12]. Limiting lifetime of individuals has to be done with care. The population size should be chosen large enough to allow well-performing individuals to propagate their information before its lifetime expires. The position information of individuals residing in a region having a low density, for example a region corresponding to a small peak, are more likely to get lost than positional information regarding individuals in a broad peak, which probably has many neighbors.

The application of recombination can easily amplify unbalanced distribution of the parent population among different regions of attraction, due to the fact that a recombination operator always requires more than one parent. Figure 1 shows a density function for a one-dimensional numerical optimization problem. The solid line represents density function of parents in the search-space. Apparantly the evolutionary algorithm has discovered two regions of interest, centered around the values 2 and 7. The first region is sampled more densily than the second region. The broken line represents the density function of offspring when using uniform selection of parents and an intermediate crossover operator. We observe that the offspring is more strongly peaked than the density function of the parents. Furthermore it can be observed that the application of this 2-parent recombination operator results in an amplification of the existing differences in density of samples within the two regions.

In order to prevent these effects one can use a more sophisticated selection mechanism. Such a mechanism should not only be biased towards the fittest individuals, but also take the distribution of the population over the search-space into consideration. Examples of more advanced selection mechanisms are the Genetic Immune Recruitment Mechanism [3] and S.T.E.P. [7]. We have defined a two-stage selection process. During the first stage a subset of the complete population, containing

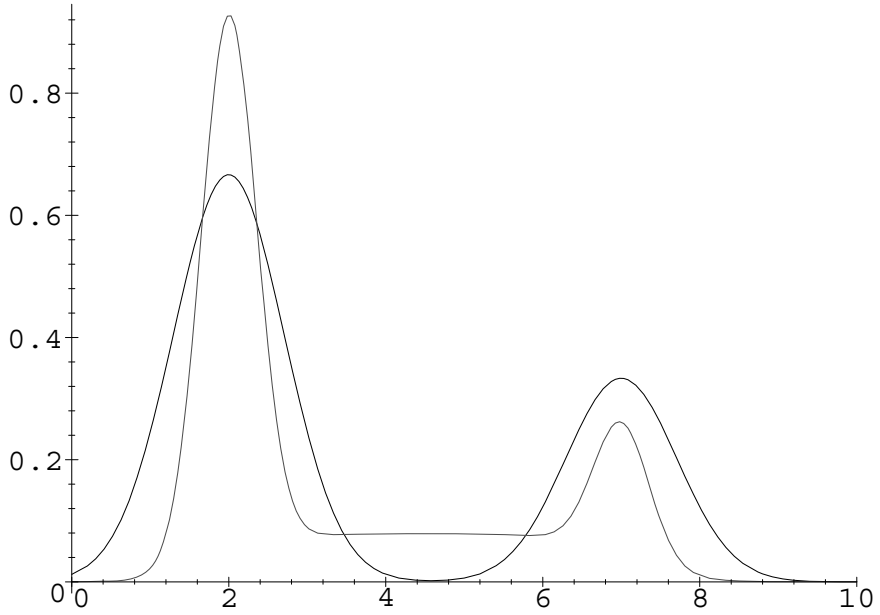

Figure 1: Density of parents and offspring when applying intermediate crossover

the best individuals, is selected. This step results in the required selective pressure, which guides the search towards regions of high average fitness. Next a clustering process is applied to the remaining individuals, and the best individual of each cluster is selected as a representative of that cluster. A new population is created by applying the evolutionary operators to these representatives only. Using only representatives as parents helps in preventing premature convergence. Nearly identical individuals are likely to belong to the same cluster, and thus only one of these individuals will be present in the set of representatives. As the presented two-stage selection schedule is not very susceptible to premature convergence it is relatively safe to preserve the selected individuals. This results in a population-elitism, which helps in preserving information and propagating obtained information to subsequent generations.

Within many randomized clustering methods local optimization is applied for two reasons [11]. First, if the search space around an individual is relatively smooth, there is no reason to use a slow global optimization method. Second, by applying local optimization in an early phase the potential of different samples can be compared better. For the same reasons, we have incorporated local search. Local optimization is applied to the representatives only, as these representatives are assumed to be typical for the current population. Applying local optimization to the (small) set of representatives reduces the amount of computation required and decreases the probability of locating the same local optimum multiple times. As high quality representatives will survive for several generations due to the population-elitism, the local optimization can be done in stages. During each generation only a limited number of local optimization steps is spend on each representative, so the number of local optimization steps spend on an individual is proportional to the number of generations it is able to survive. This multi-stage local optimization prevents that a lot of local optimization steps are spend on an individual which is discarded immediately afterwards.

Figure 2 presents the algorithm in pseudocode, where the scope is given by the level of indentation.

In Figure 2, P_i denotes the population created during the i^{th} generation, N is the maximal cardinality of the set P_i, N_{repres} is the maximal number of representatives, λ is the minimal number of offspring per parent, $\#R$ is the cardinality of set R, and $R_i \subseteq P_i$ is the actual set of representatives.

```
i ← 0
N ← λ · N_repres
P_0 ← random population
repeat
  i ← i + 1
  /* two-stage selection */
  P_i ← select best N_repres individuals from P_{i-1}
  cluster individuals in P_i
  R_i ← select representatives from P_i
  if ¬ ready then
    /* local optimization of representatives */
    ∀x ∈ R_i local optimize x
    /* production of new offspring */
    while #P_i < N do
      if #R_i > 1 then
        select p_1 and p_2 ∈ R_i, (p_1 ≠ p_2)
        P_i ← P_i ∪ recombine(p_1, p_2)
      else
        P_i ← P_i ∪ mutate(p), p ∈ R_i
until ready
```

Figure 2: Pseudo-code of the Cluster Evolution Strategy

The ready predicate becomes true when the desired objective value is obtained, when the total allowed number of function evaluations $\max_{eval}$ is reached, or when all clusters have a size below a certain minimal threshold. An implementation of this algorithm can be obtained by contacting the author. In the next subsections we give a more detailed description of the different parts of the algorithm.

2.1 Clustering method

The clustering problem can be stated as follows: Given a space S and a set X of samples $\vec{x}_i \in S$ try to discover a number of subsets $A_i \subseteq S$ under the restrictions $\cup_i A_i = S$ and $A_i \cap A_j = \emptyset$ if $(i \neq j)$, that minimizes a certain measure $F(A)$. The exact definition of $F(A)$, and the (maximal) number of clusters to be discovered differs per application. Clustering problems tend to be difficult, and especially if the number of clusters is not known in advance, there are no good general purpose clustering algorithms available [5].

For the problems we are handling $S = \mathbb{R}^n$. It is inherent to these problems that the density of samples is very low. When allowing a high density of samples that covers the complete search-space there would be no need for learning approach such as an evolutionary algorithm. The low density of samples within the space S makes it difficult to use the existing clustering methods. In order to get a good balance between speed and quality of the clustering a specialized clustering heuristic was developed.

In order to avoid the curse of dimensionality, and to increase the density of samples a one-dimensional mapping of S is created by projecting all samples on a single axis, which corresponds to the mapping

$$g : \mathbb{R}^n \rightarrow \mathbb{R} = \vec{x} \cdot \vec{e}_i.$$

The set S is sorted on the value $g(\vec{x})$ and $D(\vec{x})$ is defined as the distance between $\vec{x}$ and its predecessor in this ordered set. The expected value of $D(\vec{x})$,

$$E[D(S)] = (\max_x\{g(\vec{x})\} - \min_x\{g(\vec{x})\})/N_{repres},$$

is independent of the actual distribution of S. A cluster boundary is assumed in front of each sample $\vec{x}$ satisfying the condition,

$$D(\vec{x}) \geq \tau \cdot \min\{E[D(S)], r_{min}\},$$

where r_{min} is a lower bound on the resolution. For each cluster a representative is chosen, being the individual with the highest fitness within this cluster, and the size of the cluster is set to

$$C = \min\{g(\vec{x}_{last}) - g(\vec{x}_{first}), r_{min}\}.$$

By repeating this process for each dimension a complete set of representatives is obtained. Note that an element of the set R can be the representative of more than one cluster, when $n > 1$.

2.2 Evolutionary operators

The evolutionary operators also influence the shape of the sampling density function. Making the proper choices when designing these operators can have a strong influence upon the reliability of the evolutionary optimizer. Our main operator is both a recombination and a mutation operator. For each dimension it chooses the value of one of its parents with equal probability and adds some Gaussian noise to it. Given two different parents $\vec{x}^{(p1)}$ and $\vec{x}^{(p2)}$, an offspring is created according to

$$x_i^{(o)} = (x_i^{(p1)} \textbf{ or } x_i^{(p2)}) + N(0, \sigma)$$

were $N(0, \sigma)$ is a normally distributed random variable with mean zero and variance $\sigma = |x_i^{(p1)} - x_i^{(p2)}|/3$. Figure 3 shows the density function for the one-dimensional case. The two dots indicate the location

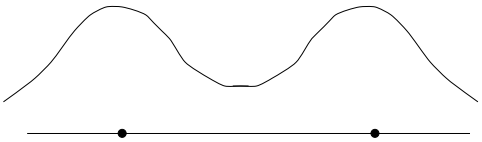

Figure 3: Density function used by main recombination operator

of the parent-samples. Using this operator the center of a cloud of individuals is not likely to be oversampled. By taking different alleles from different parents, which will correspond to different regions, this operator can easily create new combinations of existing alleles. This operator shows some resemblance to the fuzzy recombination operator defined by Voigt [13].

This operator does a good job at preventing that the sampling density function narrows too rapidly. But after some time, when all well-performing values for different dimensions have been collected, this operator might have problems in combining different alleles in order to find the optimal solution. As the Gaussian noise is proportional to the distance between the two parents along that dimension, values that are far apart can not be exchanged easily. In order to enlarge the probability of this kind of long-distance exchanges a discrete recombination operator is applied with a low probability $P_{discrete}$. The discrete recombination operator creates a value for the offspring using the formula

$$x_i^{(o)} = x_i^{(p1)} \textbf{ or } x_i^{(p2)}.$$

Figure 4: Density function used by mutation operator

When there is only a single representative remains it is not possible to apply recombination anymore. Only under these circumstances mutation is applied. Figure 4 shows the density function used by the mutation operator for a one-dimensional sample. The dot indicates the location of the parent, and C is the size of the cluster this parent belongs to. In order to prevent that the algorithm converges to a small neighborhood around this single representative we apply a uniform sampling over the complete cluster represented by this single parent. We also extended the uniform sampling by two slopes, which make it possible for the cluster to grow if it was too small along a certain dimension.

2.3 Local search strategy

In order to get a better picture of the relative quality of the different clusters, some kind of local search is needed. In order to keep the global algorithm simple and to avoid the necessity of complex decision mechanisms to determine whether the local optimizer should be applied, we apply a simple local optimizer, that only uses a limited number of function evaluations s_{loc}. Samples that perform well will be preserved due to the population-elitism. Such samples are passed to the local optimizer several times, so a single pass of the local optimizer just has to perform a partial optimization. We have implemented two different local optimizers that satisfy these requirements. The first optimizer is based on Lagrange polynomials, the second optimizer applies a $(1+1)$-Evolution Strategy.

A simple approach to create such a local optimizer is to draw two random samples in a neighborhood of the sample $\vec{x}$ to be optimized, build a one-dimensional quadratic model by means of elementary calculus and then use this model to estimate the location $\vec{x}'$ of the optimum. We have implemented this procedure as follows. Given the location of the original sample $\vec{x}$, we choose at random one of the base vectors $\vec{e}_i$ and a step size $h \in [0, C_i/2]$, where C_i is the size of the cluster $\vec{x}$ belongs to along direction $\vec{e}_i$. Next we determine the second-order Lagrange polynomial using the triple of fitness values at the samples $\vec{x}, \vec{x} - h \cdot \vec{e}_i$, and $\vec{x} + h \cdot \vec{e}_i$. The sample $\vec{x}'$ is set to the location of the maximum of this polynomial. If $\vec{x}'$ is unbounded it is replaced with $\vec{x} \pm 3 \cdot h \cdot \vec{e}_i$ where the sign is determined by which of the two samples $\vec{x} \pm h \cdot \vec{e}_i$ has the highest fitness. The best of the three evaluated samples will replace the original $\vec{x}$ if this results in an increased fitness. A selected base vector $\vec{e}_i$ will not be used again before all other base vectors $\vec{e}_j (j \neq i)$ also have been used.

The second local optimizer is based on a simple two-membered (1+1)-Evolution Strategy developed by Rechenberg [9]. When applying this strategy a single parent is used. This parent consists of a vector $\vec{x}^{(p)}$ of d real numbers that encode a possible solution to the numerical optimization problem. A single offspring is created using the formula,

$$x_i^{(o)} = x_i^{(p)} + N(0, \sigma)$$

where $N(0, \sigma)$ is a normally distributed random variable with mean zero and variance σ. The offspring replaces the parent if it outperforms the parent on the function f to be optimized. The value of σ is adjusted by means of the 1/5-success rule [10]:

> *determine the ratio of the number of successful mutations to the total number of trials. If the ratio is greater than 1/5 the variance σ should be increased, if it is less than 1/5 than decrease the variance σ.*

Within our algorithm each sample $\vec{x}$ is associated with a cluster that has size $\vec{C}$. It is easy to incorporate this information into the optimization process by setting $\sigma = \sigma' \cdot C_i$, where σ' is a scalar parameter with initial value 0.5 that will be updated according to the 1/5-success rule.

The 1/5-success rule is implemented by means of a multiplication. After a failed optimization step the σ' is multiplied by the parameter $m_{fail} \leq 1$, and when a successful step occurred σ' is multiplied by m_{fail}^5.

Parameter	value
s_{loc}	9
λ	2
τ	1.5
r_{min}	10^{-4}
$P_{discrete}$	$\frac{1}{10}$
m_{fail}	0.95
$\max_{eval}$	50000/100000
N_{repres}	100

Table 1: Parameters of the algorithm

3. Experiments

For our experiments we used the test-problems and performance measures defined for the first International Contest on Evolutionary Optimization, taking place during the Third IEEE conference on Evolutionary Computation (Japan 1996). Performance is measure in terms of the Expected Number of Evaluations per Success (ENES), Best Value reached during 20 independent runs (BV), and Relative Time (RT) which measures the ratio between the amount of computation spend the evolutionary algorithm and computation spend on pure function evalutations.

During our experiments we used the parameter settings presented in Table 1, where s_{loc} represents the number of function evaluations during a single application of the local optimizer, λ is the minimal number of offspring per representative, τ is the decision boundary for the clustering algorithm, r_{min} is the minimal allowed resolution, $P_{discrete}$ is the probability the discrete recombination is applied, m_{fail}

Test problem	ENES	RT
Sphere 5D	14459/**1452**	31/75
Sphere 10D	35275/**3462**	27/32
Griewangk 5D	46212/**22039**	2.2/2.0
Griewangk 10D	65590/**19125**	1.8/1.1
Shekel 5D	**43067**/51845	1.17/0.96
Shekel 10D	**39151**/363685	0.88/0.53
Michalewicz 5D	15600/**10661**	4.7/3.9
Michalewicz 10D	104993/**41765**	3.8/2.6
Langerman 5D	12543/**11343**	2.7/1.8
Langerman 10D	75477/**61729**	1.8/0.8

Table 2: Performance measures ENES and RT for the cluster evolution strategy with a (1+1)-ES (left) and Lagrange (right) local optimizers.

is the multiplier used for updating the σ value of the $(1+1)$-ES local optimizer, $\max_{eval}$ the maximal number of function evaluations during a single run, which is set to 50000 for the 5-dimensional and to 100000 for the 10-dimensional test problems, and N_{repres} is the maximal number of representatives. All parameters have been set using only a few experiments, or their value is determined by means of an educated guess. Reason for doing so is that optimal value for parameters tend to be problem specific. One should therefore be interested in a fixed set of parameter-values that perform well over a wide range of problems.

We performed two sets of experiments. During the first set we used a (1+1)-ES as local optimization method. The second set of experiments uses a Lagrange interpolation during the local optimization step. A single Lagrange interpolation will require three function evaluations. The results are shown in table 2, where the number on the left side of the slash corresponds to the (1+1)-ES method and the right side of the slash to the Lagrange method.

When using the ENES-measure we see that all test-problems can be solved within a reasonable number of function evaluations. The best result is shown in a bold font. The Lagrange method outperforms the (1+1)-ES method on eight out of the ten test problems. Only in case of the Shekel function the (1+1)-ES method performs better. Probably the quadratic model used in the Lagrange method does not give a good local approximation in case of the Shekel test problems. When comparing the results for the corresponding 5-dimensional and 10-dimensional test problems we see that the increase in the ENES-measure is moderate in most cases. In case of the Shekel test problem the 10-dimensional test problem even seems to be easier to optimize than the 5-dimensional version, when using the (1+1)-ES. We expect that this effect is a result of a larger number of representatives being selected in the 10-dimensional case, resulting in a better exploration of the complete search-space.

The BV measure is of no interest in our case as the defined acceptance threshold is reached for all test problems. As a result this measure only reflects the location of the acceptance threshold.

The RT measure shows that the overhead of the algorithm is reasonable the on presented test problems. Only in case of the sphere function the overhead is large, but this mainly due to the fact that the sphere function is so easy to compute.

When comparing both cluster evolution strategies we see that the Lagrange method performs best in most cases. Nevertheless we prefer the (1+1)-ES methods. This method makes less assumptions regarding the actual problem being optimized, and is therefore expected to result in a more reliable evolutionary algorithm.

4. Conclusions

Viewing an evolution process as self-adaptive sampling density function helps in pin-pointing some of the problems that can occur during a function optimization task. If the applied selection mechanism and the recombination operator do not fit together this easily result in oversampling of certain parts of the search-space, resulting in a sub-optimal solution being obtained. A large region, having a high overall fitness acts like an attractor to the population. A two-phase selection schedule, using a cluster analysis, can help in enhancing the sampling density function, and thereby in preventing premature convergence.

A nice additional advantage of the clustering approach is that it allows the efficient use of local optimization techniques. By applying local optimization to representatives only, the probability of obtaining the same optimum several times becomes smaller. Elitism is needed within our algorithm because the local optimization is done incrementally. The longer the lifetime of a representative, the more local optimization is performed on the corresponding sample.

Further research will be devoted to finding a more rigid mathematical foundation for different parts of the algorithm, and to the application of a similar algorithm to constrained optimization problems.

Acknowledgement: I would like to thank Joost N. Kok for his useful comments.

REFERENCES

1. E. Aarts and J. Korst. *Simulated Annealing and Boltzmann machines.* Wiley, 1989.
2. Th. Bäck, H.-P. Schwefel, and R. Männer. An overview of evolutionary algorithms for parameter optimization. *Journal of Evolutionary Computation*, 1:1–23, 1993.
3. H. Bersini and G. Seront. In search of a good evolution-optimization crossover. In *Parallel Problem Solving from Nature II*, pages 479–488, 1992.
4. D.B. Fogel. *Evolving artificial intelligence.* Doctoral dissertation, University of California, 1994.
5. L. Kaufman and P.J. Rousseeuw. *Finding Groups in data, an introduction to cluster analysis.* John Wiley & Sons, Inc., 1990.
6. S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.
7. S. Langerman, G. Seront, and H. Bersini. S.T.E.P.: The easiest way to optimize a function. In *IEEE World congress on computational Intelligence*, 1994.
8. H. Mühlenbein and H.M. Voigt. Gene pool recombination in genetic algorithms. In I.H. Osman and J.P. Kelly, editors, *Metaheuristics international Conference, Norwell.* Kluwer Academic Publishers, 1995.
9. I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzboog, 1973.
10. H.-P. Schwefel. *Evolution and Optimum Seeking.* Sixth-Generation Computer Technology Series. Wiley, New York, 1995.
11. A. Törn and A. Žilinskas. *Global optimization.* Lecture Notes in Computer Science 350. Springer, 1989.
12. C.H.M. van Kemenade, J.N. Kok, and A.E. Eiben. Controlling the convergence of genetic algorithms by tuning the disruptiveness of recombination operators. In *Second IEEE conference on Evolutionary Computation*, pages 345–351. IEEE Service Center, 1995.
13. H.-M. Voigt, H. Mühlenbein, and D. Cvetković. Fuzzy recombination for the Breeder Genetic Algorithm. In *Sixth International Conference on Genetic Algorithms*, pages 104–111, 1995.