Centrum voor Wiskunde en Informatica
REPORT*RAPPORT*

The Mixing Evolutionary Algorithm -- independent selection and allocation of trials --

C.H.M. van Kemenade

Software Engineering (SEN)

SEN-R9726 December 31, 1997

Report SEN-R9726
ISSN 1386-369X

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

The Mixing Evolutionary Algorithm
– independent selection and allocation of trials –

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

ABSTRACT

When using an evolutionary algorithm to solve a problem involving building blocks we have to grow the building blocks and then mix these building blocks to obtain the (optimal) solution. Finding a good balance between the growing and the mixing process is a prerequisite to get a reliable evolutionary algorithm. Different building blocks can have different probabilities of being mixed. Such differences can easily lead to a loss of the building blocks that are difficult to mix and as a result to premature convergence. By allocating relatively many trials to individuals that contain building blocks with a low mixing probability we can prevent such effects. We developed the mixing evolutionary algorithm (mixEA) in which the allocation of trials is a more explicit procedure than in the standard evolutionary algorithms. Experiments indicate that the mixEA is a reliable optimizer on a set of building block problems that are difficult to handle with more traditional genetic algorithms. In the case that the global optimum is not found, the mixEA creates a small population containing a high concentration of building blocks.

1991 Mathematics Subject Classification: 68T05, 68T20
1991 Computing Reviews Classification System: G.1.6, I.2.8
Keywords and Phrases: genetic algorithms, selection schemes
Note: Work carried out under theme SEN4 "Evolutionary Computation". This paper was presented at the IEEE conference on Evolutionary Algorithms 1997, Indianapolis, USA.

1. Introduction

Many problems involve a search-space which is too large to search it completely. In order to find solutions we have to make assumptions about the structure of the search-space. In many optimization problems we can use the building block hypothesis. This hypothesis, when applicable, states that a solution can be decomposed in a number of building blocks, which can be searched for independently and afterwards be mixed in order to obtain a good or even optimal solution. An interesting class of such problems is composed of the fully deceptive trap functions [DG93]. Evolutionary algorithms are interesting optimizers to find solutions to such problems. The search for the optimum usually involves two steps: growing the building blocks in the population and mixing these building blocks in order to obtain the global optimal solution. The mixing of building blocks can be difficult for a GA [GDT93, TG93]. One approach to increase the capabilities of GA's to handle such mixing difficulties is to measure correlations between bit-values. Two methods that take this approach are the GEMGA [Kar96] and the building block filtering [vK96]. Both methods estimate fitness contributions of bits. The GEMGA basically uses this information to guide the evolution process. The building block filtering procedure uses the information to make an actual decomposition of a bit-string in order to extract the actual bits that belong to the same building block and their optimal values. Mixing of building blocks can be considered as the problem of selecting and preserving the appropriate individuals. In this paper a different selection scheme is proposed that converges reliably on a set of problems that are difficult for a more traditional genetic algorithm.

The selection mechanism of an evolutionary algorithm is used to choose a set of parents from the current population. An evolutionary operator is applied to these parents in order to produce offspring

and a reduction mechanism prevents the set of individuals from growing unbounded. We consider a selection scheme to be a combination of a selection and a reduction mechanism. For example in a generational genetic algorithm a selection mechanism can involve fitness proportional selection while the reduction mechanism is implicit and involves discarding individuals on a generational basis. In a steady-state genetic algorithm a variety of reduction mechanisms can be applied, for example worst fitness deletion.

Within an evolutionary algorithm the selection scheme is the driving force of the evolution. By choosing the appropriate individuals during selection and reduction one hopes to drive the population towards more promising parts of the search-space and thereby to find increasingly better solutions. In our view the reliable selection scheme for evolutionary optimization should discriminate between the two tasks it has to perform. The first task is selection of the individuals that are allowed to reproduce themselves. This selection is used to drive the population towards regions of relatively high fitness. The second task is the allocation of trials in such a way that we get an efficient exploitation of the building blocks present in a certain parent. By allocation of trials we mean the decision about the number times one is going to try to mix an individual by means of recombination. Relatively many recombination trials should be allocated to those individuals that contain building blocks with a low value of P_{mix} that is defined as the probability of mixing a building block successfully. In traditional GA's one does not discriminate between the selection and allocation of trials, but we are going to argue that in some cases it does make sense to do so. We introduce a novel selection scheme, called the mixEA, that makes the separation between these two tasks more explicit. We give empirical evidence that the mixEA performs well on a set of optimization problems that have a building block decomposition.

This paper is organized as follows. In section 2 we discus the "double" role of the selection mechanism. Section 3 introduces the mixEA which separates these two tasks. The results of some empirical studies are shown in section 4. This is followed by the conclusions in section 5.

2. Double role of selection

The selection mechanism has two tasks to perform in an evolutionary algorithm. The most explicit task is to determine the individuals that are allowed to reproduce themselves. Furthermore the selection mechanism has a second task being the allocation of trials to individuals, where more trials should be allocated to individuals showing better prospects. In traditional genetic algorithms this allocation of trials an indirect consequence of performing the parent selection. Individuals that are selected more often for reproduction can spread more duplicates of their genes in the next generation, which results in an (indirect) allocation of trials.

In most GA's the fitness is taken to be equal to the objective value of the function to be optimized. So the selection of parents is usually biased towards the individuals having a relatively high objective value. As the number of trials (recombination operations) assigned to a selected parent is fixed there will be a direct relation between the number of times an individual is selected and the assigned number of trials. Fitness undoubtly is one of the important factors to be considered when determining allocation of trials. Another factor is the ease of recombination. Assume that an individual contains building block that is part of the optimal solution. Then it is the task of the recombination operator to mix this building block with other (complementary) building blocks present within the population. The lower the probability of transferring this building block to another individual (= P_{mix}), the more trials we should allocate to the individual that contains this building block. This scheme is closely connected to constructional selection [Alt94], which is defined as proliferation of pieces of code in genetic programming not accounted for by the Schema Theorem.

Genetic algorithms were originally developed as adaptive systems that should have a good on-line performance in a dynamically changing environment (context). Allocation of trials based on fitness seems to be appropriate. The P_{mix} factor is not so important for such on-line systems as one is not very interested in building blocks that are difficult to mix, as these will not allow for a good on-line performance.

Nowadays evolutionary computation methods are often used for optimization problems where the problem definition (context) does not change and in which we are only interested in the (off-line) best possible solution. In that case compensating for differences in the value P_{mix} of different building blocks is more important as a bias towards building blocks that are easy to recombine is undesirable.

The factor P_{mix} usually is not considered explicitly as a steering factor for the allocation of trials in GA's, as one does not discriminate between the two tasks of the selection scheme. Instead it is considered implicitly when tuning the selective pressure. This indirect steering is far from optimal. First the tuning of the selective pressure is done off-line, which means that several independent runs are needed to find a good setting for the selective pressure. A second problem is that selective pressure is a global setting and therefore we do not consider the fact that different building blocks can have a different P_{mix}. Furthermore we often see that GA's tend to converge to broad peaks in the search landscape as these are discovered early during the search process. In the next section we describe the selection scheme of the mixEA. This scheme makes the separation between the two tasks of the selection scheme explicit and tries to adapt to the individual values of P_{mix} for the different building blocks. A related approach is the reproductive evaluation [Whi87] were additional copies are assigned to an individual that has created relatively many good offspring. We think reproductive evaluation indeed might help to obtain rapid convergence, but when one is interested in actual optimization such a system is likely to fail as it magnifies the preference of the GA for building blocks with a large value of P_{mix}.

3. Mixing evolutionary algorithm

The mixEA is designed to handle problems where the optimal solution can be decomposed in a number of building blocks, which can be located independently of each other. The primary goal is to locate the optimum, but if this goal is too difficult to attain, the algorithm is meant to locate a (small) set of relatively good solutions that do contain as many building blocks as possible.

The selection scheme of the mixEA differentiates between the two tasks: selection of prosperous parents and allocation of trials. In the mixEA we start with a large population and the mixEA tries to condense all building blocks in a decreasing set of individuals. The ideal final situation corresponds to all building blocks being condensed in a single individual representing the optimal solution. In order to attain this goal the mixEA uses a (rapidly) decreasing population size. Simultaneously the number of trials being allocated to each remaining individual is increased. Optimization can benefit from an increasing allocation of trials to individuals as the creation of superior offspring usually gets more difficult aas the evolution proceeds. This is due to the fact that more information has to be preserved during recombination as more and larger building blocks tend to be present in the individual. In most GA's the allocation of (recombination) trials to an individual is directly correlated to the number of times the individual is selected for recombination. So the number of trials allocated is independent of the P_{mix} and individuals with a high value of P_{mix} will grow and mix rapidly. As a result such a duplication-oriented allocation of trials tends to be biased towards small building blocks that are easy to preserve and recombine. The mixEA does allocation of trials by actually assigning an increasing number of trials to a single individual as the evolution proceeds. Furthermore duplication is prevented in order to reduce the bias towards small building blocks. To prevent duplication each parent is only allowed to create only a single offspring, as creating more than one offspring already can result in a rapid replication of certain building blocks.

Given a certain generation, the next generation is created according to the following rules. Two parents are selected uniformly at random. The recombination operator is used to create a single offspring. Only if the offspring outperforms both parents with respect to fitness it will be accepted and be put in the pool of the next generation, and the parents will be discarded. Such a local selection process have been described and used in other GA's too [Alt94, TG94]. Duplication is minimized as each parent can participate in only one offspring. When an offspring outperforms both of its parents it assumed to be the result of a successful mixing event. If the offspring does not outperform both parents it will be discarded and the parents will stay within their pool. This process continues until

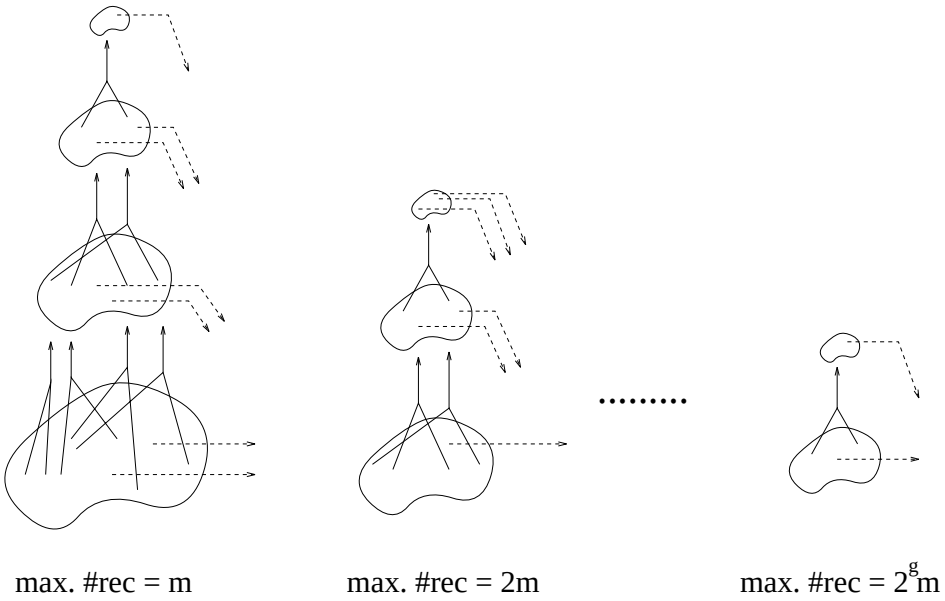

Figure 1: A schematic outline of the mixing evolutionary algorithm, where max. #rec denotes the maximal number of recombination per individual for the evolution process shown as a stack above this number.

the current population is empty, or all individuals in this population have had the maximal number of recombination operations being allowed per individual. As a result of this scheme each population is at least two times as small as the previous population. An overview of the mixEA is presented in Figure 1. The evolution process starts by creating a random initial population, which is depicted by the blob in the left-bottom corner of the picture. Furthermore a maximal number of recombinations per individual m is set. Evolution is applied and results in a series of populations that decrease in size (moving upwards in the picture). This process terminates when a population has less than two individuals. *All individuals* that did not produce offspring are gathered. These individuals are copied to the first population of a second evolution process. Duplicates are removed from this population and the maximal number of recombination per individual is doubled, thus allowing more trials for the individuals in this evolution process. The same process is applied to this new population and a sequence of evolution processes is obtained. The reason for using such a sequence is that all building blocks that have a large value of P_{mix} will be accumulated in a relatively small number of individuals. So the second evolution process receives either individuals which either have a few building blocks that are difficult to recombine, or individuals that contain a large number of small building blocks. During the subsequent evolution processes the maximal number of recombination per individual is increased in order to make it possible to mix building blocks having a low P_{mix} or individuals containing many small building blocks.

The mixEA requires only three parameters, i.e. the size of the initial population, the maximal number of recombinations for the first evolution process, and the maximal number of subsequent evolution processes. The algorithm is not very sensitive with respect to the actual values of these parameters, so parameter tuning is easy. Furthermore the algorithm terminates automatically when no more mating pairs are available and we do not need complex stopping criteria.

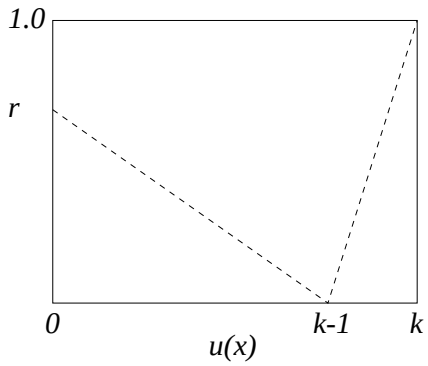

Figure 2: Fitness calculation for a single building block. The function $u(x)$ counts the number of 1-bits int string x.

size	num.	mixEA			GGA			SSGA		
bb	bb	fbb B	fbb P	% succ.	fbb B	fbb P	% succ.	fbb B	fbb P	% succ.
3	13	1.0	1.0	100	0.83	1.0	10	1.0	1.0	100
4	10	1.0	1.0	100	0.33	0.26	0	0.71	0.97	0
5	8	0.86	1.0	10	0.18	0.03	0	0.19	0.48	0
6	7	0.76	1.0	10	0.12	0	0	0.12	0.3	0
7	6	0.68	1.0	0	0.07	0	0	0.08	0.1	0
8	5	0.6	1.0	0	0.02	0	0	0.04	0.03	0
10	4	0.48	0.99	0	0	0	0	0	0	0

Table 1: Results for different sizes of the building blocks (first set of problems), where fbb B/P represents the fraction of building blocks present within the Best solution/final Population, and % succ. shows the percentage of successful runs.

4. Experiments

During our experiments we used the fully deceptive trap function [DG93]. The fitness contribution for a set of k bits that together can compose a building block is calculated according to the following formula,

$$f(x) = \begin{cases} 1 & \text{if } u(x) = k \\ r\frac{k-1-u(x)}{k-1} & \text{otherwise} \end{cases}$$

where $u(x)$ counts the number of 1-bits in string x, and $r < 1$ denotes the fitness ratio between the optimal and the sub-optimal solution. The function $u(x)$ is shown in Figure 2. This type of building blocks is difficult to detect as all lower order schemata direct the search towards the local optimum containing just 0-bits. We used a fitness ratio of $r = 0.7$ and made no assumptions regarding the linkage of bits. The fitness of an individual is taken to be the sum of the partial fitnesses of the different sub-functions.

During our experiments we compared the mixEA to two other evolutionary algorithms. The first is a generational genetic algorithm (GGA) using tournament selection (tournament size = 2), in which crossover is applied with probability 0.7 and the mutation rate is set to $1/l$, where l is the length of the bit-string. The second is a steady-state genetic algorithm (SSGA) applying uniform selection and truncation reduction, crossover is always applied and mutation rate $1/l$. Both GA's terminate when the optimal solution is obtained, or when the variance in fitness is zero, or when the number of function evaluations exceeds 500,000. The mixEA allows 32 recombination-trials per individual

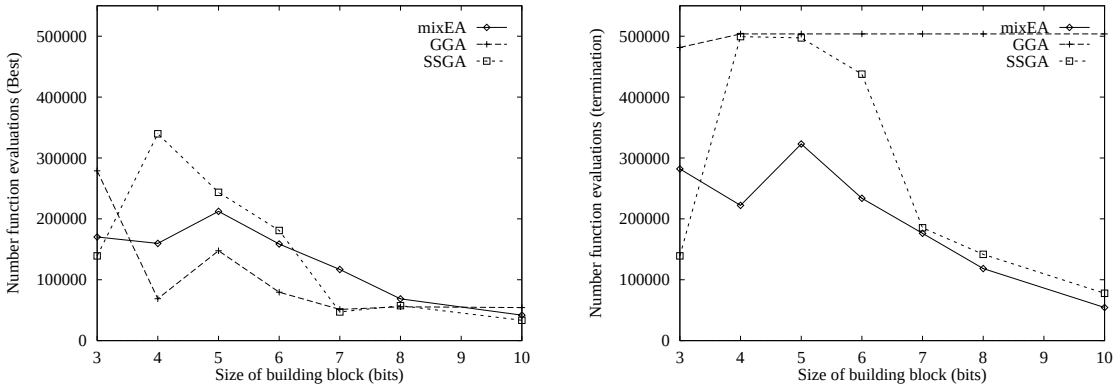

Figure 3: Number of function evaluations before obtaining the best solution (left) and before termination of the algorithms (right) for the first set of problems.

during its first evolution process. An upper bound of 16,384 is used for this number of recombinations per individual, which results in at most 10 subsequent evolution processes to take place. No mutation is applied. All three selection schemes have an (initial) population containing 4096 individuals and use uniform crossover. All results are obtained by taking an average over 20 independent runs.

Two sets of experiments are performed. First we are going to investigate how the different selection schemes behave when all building blocks have the same P_{mix}. Next we are going to investigate what happens when different building blocks have different values of P_{mix}. The first set of experiments is used to study the behavior of the different selection schemes in relation to the size of the building blocks k. The number of building blocks is adjusted such that the length of the bit-string is approximately 40 bits. The results are shown in Table 1. For selection scheme the first column in this table (fbb B) contains the fraction of all building blocks present in the best obtained solution. The mixEA performs significantly better than the other two selection schemes when using this measure. The second column (fbb P) denotes the fraction of all building blocks that are present in the final population. For the mixEA this fraction is close to 1.0 during this experiment, which means that almost all building blocks are still present within the final population. Usually this population consists of approximately ten individuals for the mixEA. In case of the GGA the fraction of building blocks drops rapidly when the size of the building blocks increase. An interesting observation is that this fraction is 0 for all problems having building blocks containing more than 5 bit positions even when the best solution does contain a small fraction of the building blocks. This means that the best solution is not present in the final population, and apparently the GGA lost track of the most important parts of the search space. The SSGA performs better, but it also is not able to preserve large building blocks in its population. The third column shows the percentage of successful runs (i.e. runs that located the global optimum). Figure 3 shows the number of function evaluations performed before the best solution is found (left) and the total number of evaluations performed before termination (right). Comparing the two graphs for the mixEA we see that this selection scheme terminates relatively fast after it has found its best solution, this in contrast to the GGA that almost always uses the maximal allowed number of function evaluations before it terminates.

From the first set of experiments it is clear that the mixEA can handle problems involving relatively large building blocks when all building blocks have the same length (same P_{mix}). A second series of experiments is performed to study the behavior of the selection schemes when applied to a problem instance involving a set of building blocks of different sizes. The actual problem instance used contains 8 building blocks having sizes in the range [3,10]. The results are shown in Table 2. Each row in this table summarizes the result for a building block of a certain size. For each selection scheme the

size	mixEA		GGA		SSGA	
bb	$P_{k,B}$	$P_{k,P}$	$P_{k,B}$	$P_{k,P}$	$P_{k,B}$	$P_{k,P}$
3	0.95	1.0	1.0	1.0	1.0	1.0
4	0.95	1.0	0.25	0	1.0	1.0
5	0.95	1.0	0	0	0	0.1
6	0.85	1.0	0	0	0	0
7	0.75	0.9	0	0	0	0
8	0.30	0.9	0	0	0	0
9	0.25	0.85	0	0	0	0
10	0	0.55	0	0	0	0

Table 2: Results for a fully deceptive problem involving building blocks of different sizes (second problem), where $P_{k,B}/P_{k,P}$ denote the probabilities the building block of size k is present in the Best solution/final Population respectively.

first column ($P_{k,B}$) shows the probability that the building block of size k is present within the best obtained solution. All three selection schemes easily find the building block of size 3, but the building block of size 4 already is difficult to find for the GGA and the SSGA fails on the block of size 5. The mixEA is able to find and mix the larger building blocks with a reasonable probability. When studying the probability that a building block is present in the final population we see roughly the same results. Again the mixEA significantly outperforms the other two methods. It is also interesting to note that the probability of the largest building block to be present in the final population is still 55%, which means that the final population of the mixEA still is very diverse and is likely to contain all building blocks. These building blocks are concentrated in a small population of approximately ten individuals.

5. Conclusions

A selection scheme basically has two tasks: the selection of parents to drive the population toward more promising parts of the search space and the allocation of recombination trials in order to get an effective mixing of building blocks. Most traditional GA's do not differentiate between these two tasks and both are steered by fitness only. As a result the search is usually biased towards finding short building blocks. The mixEA is a simple selection scheme results in a sequence of evolution processes. In the mixEA selection of parents is also steered by fitness, but the second task, allocation of trials, is coupled to rank of the evolution process. Subsequent evolution processes use an increasing number of trials as more complex building blocks are likely to be present.

During our experiments the mixEA significantly outperforms the two traditional GA selection schemes, especially when large building blocks need to be processed. Also when the problem instance contains many building blocks having different values of P_{mix}, the mixEA performs well. When the optimum is not obtained the mixEA usually terminates with a small final population that still contains most of the building blocks. Analysis of this population can reveal more information regarding the actual structure of the search space. A possible method to do such an analysis is the building block filtering method [vK96] which can be used to extract the actual building blocks.

Further research should include more detailed analysis of the cross-competition effects described in this paper.

Acknowledgement: I would like to thank Joost N. Kok, Dirk Thierens, 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.

[DG93] K. Deb and D.E. Goldberg. Analyzing deception in trap functions. In L.D. Whitley, editor, *Foundations of Genetic Algorithms - 2*, pages 93–108. Morgan Kaufmann, 1993.

[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.

[Kar96] H. Kargupta. Gene expression messy genetic algorithm. In *proceedings of the Third IEEE conference on Evolutionary Computation*, pages 814–819, 1996.

[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.

[vK96] C.H.M. van Kemenade. Explicit filtering of building blocks for genetic algorithms. In *Parallel Problem Solving from Nature IV*, pages 494–503, 1996.

[Whi87] D. Whitley. Using reproductive evaluation to improve genetic search and heuristic discovery. In *proceedings of the Second International Conference on Genetic Algorithms*, pages 108–115, 1987.