On the Scalability of Simple Genetic Algorithms

Dirk Thierens

Department of Computer Science
Utrecht University
P.O. Box 80089, 3508 TB Utrecht
The Netherlands

Abstract

Scalable evolutionary computation has become an intensively studied research topic in recent years. The issue of scalability is predominant in any field of algorithmic design, but it became particularly relevant for the design of competent genetic algorithms once the scalability problems of simple genetic algorithms were understood. Here we present some of the work that has aided in getting a clear insight in the scalability problems of simple genetic algorithms. Particularly, we discuss the important issue of building block mixing and show how the need for mixing places a boundary in the GA parameter space that together with the boundary from the schema theorem delimits the region where the GA converges reliably to the optimum of problems of bounded difficulty. This region - or sweet spot as it has been called - shrinks unfortunately very rapidly with increasing problem size unless the building blocks are tightly linked in the problem-coding structure. In addition we look how straightforward extensions of the simple genetic algorithm - namely elitism, niching, and restricted mating - are not significantly improving the scalability problems.

1 Introduction

Simple genetic algorithms (sGA) have been around for decades now. Holland pioneered their development during the sixties and seventies (Holland, 1975), and ever since the mid eighties genetic algorithms have enjoyed an ever increasing popularity. sGA did lead to a number of successful applications, and users are commonly attracted to them by their ease of use, general applicability, and promise of robustness. Like any other search algorithm however genetic algorithms make assumptions - although implicitly - about the structure of the search space, and if these assumptions match the problem and the way it is represented to the GA well enough, then successful results can be achieved. Understanding these assumptions is not only important for properly applying the genetic algorithm, but it is also a prerequisite to be able to design more competent extensions.

In the next section we will review the search assumptions or inductive bias of the simple genetic algorithm, and look at the possible strategies to deal with it. Section 3 discusses the mixing issue and its relation with the selection process. Section 4 analyses the effect of the interaction between selection and mixing on the scalability properties of the simple genetic algorithm. In section 5 we consider whether some straightforward extension to the simple GA might improve the scaling problems. Finally we discuss the ramifications of our findings. Part of the work reported here has been published in (Thierens & Goldberg, 1993; Thierens, 1995).

2 Inductive Bias of Genetic Algorithms

Genetic algorithms are search procedures and by definition this implies that they have to make some assumptions about the underlying structure of the search space which guides their decision making in order to be more efficient than enumerative search would be on the same problem. To put it another way, search procedures perform induction on the solutions yet encountered to generate new, potentially better solutions. Suppose they would use no such inductive information then there would be no correlation of whatever kind between successive steps in the search space, and therefore this would be equivalent to a random or enumerative process. At the other hand, if the inductive mechanism leads one away from better solutions then the search algorithm is actually worse off than a random or enumerative algorithm. These observations have been known since long - see for instance (Watanabe, 1969) and (Mitchell, 1982) or in a somewhat different formulation (Wolpert & Macready, 1996). In the context of black box optimisation and specifically in relation to genetic algorithms an insightful analysis is made in (Kargupta & Goldberg, 1996).

In general the genetic algorithm's search mechanism can be viewed upon as an adaptive sampling and recombining of particular similarity subsets - often called schemata. Reliable information processing is only achieved for those schemata that receive enough samples or equivalently, are tried in many different combinations with other promising schemata. The particular set of schemata that is well processed depends on the problem-coding and the genetic operators, particularly crossover. It is here that the genetic algorithm's main inductive bias can be found: those schemata that are well sampled and can be juxtaposed by crossover are guiding the search trajectory, and whether this leads to optimal solutions depends on the match between problem space and inductive bias. It is clear that the building block hypothesis (Goldberg, 1989) which states that highly fit, short, low-order schemata combine to form better solutions is basically a matter of correspondence between the inductive bias of a specific genetic algorithm implementation (particularly the recombination operators), and the problem-coding.

There are a number of different strategies one can undertake to deal with the inductive bias.

1. *Ignore bias assumptions*

   Although every search algorithm has its bias to explore a given search space, this does not necessarily mean that a user has to take the bias into account in order to use the algorithm successfully. Many search algorithms have an implicit bias so the kinds of problems on which they perform either well or poorly is not known to the inexperienced user. Ignoring bias only becomes problematic when the problem's structure and the algorithm's inductive assumptions do not match. Ignoring the mismatch will result in disappointing results. When the algorithm's bias is however quite robust - this is, when many problems most likely tried to be solved do indeed match the bias to some degree - the simple strategy to ignore the bias assumptions pays off. Simply taking "of the shelf" standard implementations will lead to quick and satisfying results. The many successful applications of standard GAs reported in the literature are a testimony of the GAs robustness, so ignoring its bias does not necessarily lead to poor performance. Clearly however there is a limit to this approach: a lot of problems do not fit into this category and high performance can only be obtained when the GA is no longer used as an "of the shelf" recipe ready to use as it is.

2. *Experimental design of representation plus genetic operators*

   For many problems the inductive assumptions or bias of the standard GA do not match the problem's structure well enough to give an efficient search process. By far the most commonly applied strategy to fix the mismatch problem is to design an appropriate genotype representation and accompanying genetic operators.

   One class of problems where the strategy to adapt the GA's bias by experimental design of genotype representation an operators is the so called ordering problem class, for instance travelling salesman problem where a large number of representations and genetic operators have been proposed.

3. *Prior domain knowledge*

   Some problems have such an obvious structure that the linkage between interacting decision variables is known beforehand, and it would be foolish not to exploit this knowledge. An example of such a problem is the map labelling problem (figure 1) where the names of geographical entities (for instance cities) have to be layed out on a map following a number of constraints. Due to the local topological interaction between the decision variables it is immediately clear where the building blocks are situated and a crossover operator respecting this structure can be readily designed (van Dijk, Thierens, & de Berg, 1998).

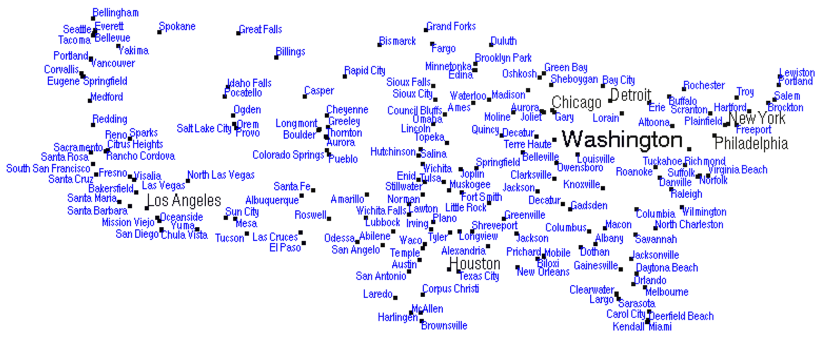

Figure 1: Map label placement by a GA.

4. *Linkage learning techniques*

   Ever since the first research efforts in genetic algorithms the idea of learning the right linkage has inspired people to come up with computational methods to achieve this. Holland proposed to use the inversion operator (Holland, 1975), although earlier research indicated it was not a sufficiently efficient operator for this task (Bagley, 1967; Frantz, 1972).

   Especially during recent years some effort was spent in trying to design more efficient linkage learning, but it is not always clear how well the proposed methods scale with increasing problem complexity (Paredis, 1995; Smith & Fogarty, 1996, Harik, 1997).

   The aforementioned work all tried to get the linkage right adaptively, that is during the normal runtime of the genetic algorithm. A different approach to linkage learning is taken in the messy GA and its successors. Here the building blocks are first identified and this information is subsequently used to code tight building blocks. Research in the messy GA is still continuing but has already achieved impressive results (Goldberg, Korb, & Deb, 1989; Goldberg, Deb, & Korb, 1990; Goldberg, Deb, Kargupta, & Harik, 1993; Kargupta, 1996; Kargupta, 1998).

3 Selection versus Mixing

Holland identified building-blocks as the fundamental unit of GA processing primarily by examining the schema theorem (Holland, 1975). A somewhat generalised version of the theorem may be written as

$$m(h,t+1) > m(h,t)\phi(h,t)[1-\epsilon(h,t)]$$

with $\phi(h,t)$ the reproduction ratio, $\epsilon(h,t)$ the disruption factor, and $m(h,t)$ the number of strings within the schema h at generation t. This inequality is basically saying that we can grow building blocks if the number of instances created by selection is larger than the number of instances that are destroyed by the genetic operators such as recombination, or

$$\phi[1-\epsilon] > 1 \qquad (1)$$

where ϕ is the growth rate under selection only and ϵ is a conservative estimate of the probability of a schema being disrupted by the genetic operators. The schema theorem is very important to understand GA performance, but that it is not the only factor may be established with the following reasoning. Since the schema theorem contains the product of two terms we may establish the necessary condition either by (1) increasing selection pressure ϕ to counterbalance some bounded loss ϵ or by (2) decreasing ϵ through parametric control of the disruption loss. Pushing harder does not necessarily mean running the risk of premature convergence because now we can also be more disruptive. Increasing the selection pressure ensures that we are able to grow the building blocks (BBs) without any need of linkage information, so why would we worry about tight codings as in the traditional GA approach, where one only looks at the disruption factor ϵ. Whenever GA difficult or deceptive problems are faced, the emphasis is put on the use of low disruptive crossover operators and tight linkage so ϵ is kept small.

Increasing the selection pressure ϕ or decreasing the disruption factor ϵ however does have a cost. This can easily be seen if we take either of these approaches to their logical conclusions and end up with an absurdity. If a selection pressure is chosen to counterbalance a very high loss, all population diversity will be lost almost immediately resulting in no exchange and an extreme form of premature convergence. If disruption is made very small, exchange will never take place and despite meeting the schema theorem criterion, the growing building blocks will never exchange. This points out an often ignored fact of GA practise: growing building blocks is one thing, mixing them is another. The schema theorem does not talk about the crucial issue of building block exchange: having the building blocks in large proportions in the population is not by itself a guarantee to find a good solution. Good building blocks of one string also have to be combined with good building blocks of another string to form a new string with a larger number of effective building blocks. There are two contradicting requirements on the recombination operator: on the one hand we wish to minimise the disruptive effect of crossover on the building blocks. On the other hand we want to maximise the building block exchange or mixing capability. Making copies of (or selecting) good building blocks and recombining building blocks are somewhat opposing processes.

To understand how genetic algorithms do their job it is instructive to break down their complex search dynamics into meaningful subprocesses. In the initial population we start with one or very few building blocks juxtaposed on certain individuals. Selection increases their proportion and by using the recombination operator we hope that eventually all BBs will come together in one optimal solution. In (Goldberg, Deb & Clark, 1992) the six pieces of the GA puzzle were identified as (1) knowing what GAs process - building blocks - (2) ensuring an adequate initial supply, (3) guaranteeing that BBs grow, (4) making BB decisions well, (5) solving problems that are not too BB-difficult, and (6) ensuring that BBs exchange to form better solutions.

Here we will specifically be concerned with the last point, namely the BBs exchange or mixing, and study its interaction with selection. In particular we look at simple GAs without linkage information, so no assumptions about tight BB linkage can be made, and therefor no positional biased crossover operator can be used. Instead we have to use uniform crossover and compensate for its destructive properties by increasing the selection pressure. First, we discuss the notion and probability of mixing events. Next, we quantify the mixing of two building blocks, and finally this is generalised to m BBs by developing the *mixing ladder climbing* model. Mixing is interrelated with selection to obtain a dimensional GA model (Ipsen,1960). All analytical predictions are verified with computational experiments on fully deceptive trap functions using uniform crossover.

4 Scalability Issues of Exchanging Building Blocks

There are basically two different ways for the crossover operator to increase the number of BBs on a particular string:

1. One possibility is that building blocks are *created* or *emerge* at a certain position. When both parents have for instance one half of the desired bit values of a particular building block, it is possible that crossover will combine them to create the BB. When the fitness function and coding is deceptive, most bit value combination that are not building blocks will differ only a few bits from the deceptive local optimum because they have a higher fitness contribution. The longer the deception length the more likely it is that less than half of the BB bit values are present, and therefore the less likely it becomes that building blocks can be created.
2. The alternative is that building blocks get *mixed*. The crossover operator now transfers complete building blocks from both parents to form an offspring that has a higher number of building blocks than either one of its parents. In fact this is really what a recombination operator is supposed to do: recombine basic entities in order to build up a better solution. Fortunately the proportion of these building blocks steadily increases by the work of the selection mechanism. Selection and building block mixing are the two fundamental mechanisms of Genetic Algorithms. The creation of BBs is just good luck: since all schemata of lower order than the BB-length k lead to the deceptive attractor there is no information available to the GA that enables it to create BBs in a systematic way.

To reach the global optimum we have to steadily increase the number of mixed building blocks until all of them are juxtaposed on one single individual. A particular crossover operation is successful if one of the offspring has more BBs than each of the parents: in the remainder we will call such a successful recombination a mixing event. Let us call p_{mix} the probability that this happens - p_{mix} is thus a measure of the crossover efficiency. To calculate p_{mix} we introduce the following notations: b_1 (resp. b_2) is the total number of BBs of the first (resp. second) parent, $b = |b_1 - b_2|$ is the different amount of BBs between the parents, d is the number of BBs that occur only in one of the parents (i.e. the unmatched BBs), and finally d_m is the number of matched BBs. It is easy to see that of the d unmatched BBs one parent has $\frac{d-b}{2}$ BBs while the other has $\frac{d+b}{2}$. Therefore a mixing event takes place when one of the children has at least $1 + \frac{d+b}{2}$ BBs.

A conservative view of mixing is obtained when we consider all non-BBs alleles to consist of all zeros versus the all ones BBs. So each individual string consists only of BBs and deceptive attractors. For uniform crossover with swapping probability $p_x = 0.5$ the probability that a BB is transferred to the other string is $\left(\frac{1}{2}\right)^k$ as is the probability for the BB be kept intact at the original string. The building block survival probability is therefore $p_{surv} = 2\left(\frac{1}{2}\right)^k$.

The probability to juxtapose i specific BBs and to disrupt the remaining $d - i$ BBs is given by $(\frac{1}{2^k})^i(1 - \frac{1}{2^k})^{d-i}$. Computing all possible BB combinations that give one child a higher number of BBs than its parents, and realizing that a mixing success can happen at either one of the two children (which brings in a factor 2), we can calculate the probability p_{mix} that a mixing event between two parents will take place by:

$$p_{mix} = 2 \sum_{i=1+\frac{d+b}{2}}^{d} \binom{d}{i} (\frac{1}{2^k})^i(1 - \frac{1}{2^k})^{d-i} \qquad (2)$$

The chance for a mixing event is thus very low and has a pronounced maximum value when the two parents have the same number of BBs ($b = 0$) and only two unmatched BBs ($d = 2$).

4.1 Mixing Two Building Blocks

4.1.1 Dimensional Model

Before considering the exchange of building blocks in its general form, we first look at the most elementary mixing problem, namely constructing the optimal solution in a problem with only two building blocks. In this case a single mixing event will juxtapose the two building blocks and create the global optimum. If the selection pressure is high enough, we expect that enough copies of the optimum will be made so that some of them survive the recombination phase and can start to take over the population. In the conservative view the recombining strings consist of a building block and a deceptive attractor (e.g. if "11111" and "00000" represent resp. the building block and deceptive attractor, the mates are for instance "11111 00000" and "00000 11111"). With uniform crossover the probability that an offspring will inherit the optimal allele is $\frac{1}{2}$. The mixing probability p_{mix} for juxtaposing 2 BBs of length k is thus $p_{mix} = 2(\frac{1}{2})^{2k}$, where the factor two comes in because the mixing event can happen at either one of the offspring. This is only a lower bound and in a less conservative view we might hope that the parents have a few optimal alleles in common so we can write the mixing probability as:

$$p_{mix} = \frac{2}{2^{\mu k}} \tag{3}$$

with $\mu \leq 2$.

We now calculate the mixing time t_x as the expected number of generations to obtain one mixing event; assuming a population size n, a crossover probability p_c, a building block length k, and 2 BBs, there are $\frac{n}{2}.p_c$ recombinations in one generation so the mixing time t_x is:

$$t_x = \frac{1}{\frac{n}{2} p_c p_{mix}} = \frac{2^{\mu k}}{n p_c}. \tag{4}$$

To interrelate recombination with selection we can use the takeover-time model for the selection time (Goldberg & Deb, 1991):

$$t_s = \frac{\ln n}{\ln s}. \tag{5}$$

When we increase the selection pressure s too much, or when our population size n is too small, selection will be too fast for the BBs to get exchanged well enough. So when the selection time t_s is smaller than the mixing time t_x we expect premature convergence to occur. On the other hand when the selection time t_s is larger than the mixing time t_x, we expect that the mixing process will have time enough to juxtapose all BBs in one single individual - the global optimum. Note that the selection and time equations are dimensional models and express exact relationships up to a constant factor c. Therefore a necessary condition for converging to the global optimum in the 2 BB case is thus:

$$t_s > c.t_x$$

or

$$n \ln n > c \frac{2^{\mu k} \ln s}{p_c}. \tag{6}$$

Recall that these dimensional models only say something about the functional interrelations between the GA parameters: for instance we can conclude from Equation 6 that the boundary between sufficient building block exchange and premature convergence will decrease linearly with the crossover probability p_c, and will grow logarithmically with the selection pressure and exponentially with the $BB-$length.

In Figures 3 and 4 we have plotted the predicted functional relations between the population size n and the crossover probability p_c, and between the selection pressure s and the crossover probability p_c. The enclosed region - or GA sweet spot (Goldberg, 1998) - between the mixing and selection boundary is the region where the GA will reliably converge to the global optimum.

In the next paragraph we will check experimentally the dimensional relation.

4.1.2 Empirical verification

All experiments in this section and the next are performed with fully deceptive trap functions with signal value equal to $\frac{1}{3}$ (Deb & Goldberg, 1993). For a single parameter combination 50 runs are tried and called successful when at least 49 of them converge to the global optimum. We use uniform crossover with allele-wise swapping probability $p_x = 0.5$, no mutation and block selection (i.e. the $\frac{n}{s}$ best strings in the current population get s copies in the next generation). For the two building block experiments we have to be careful with the initial population: for uniform crossover with fully deceptive functions to work well in a simple GA we need very large population sizes n. Since we are only interested in the mixing behaviour of the two BB, all global optimal strings - those that already have the two BBs juxtaposed - are removed from the initial population.

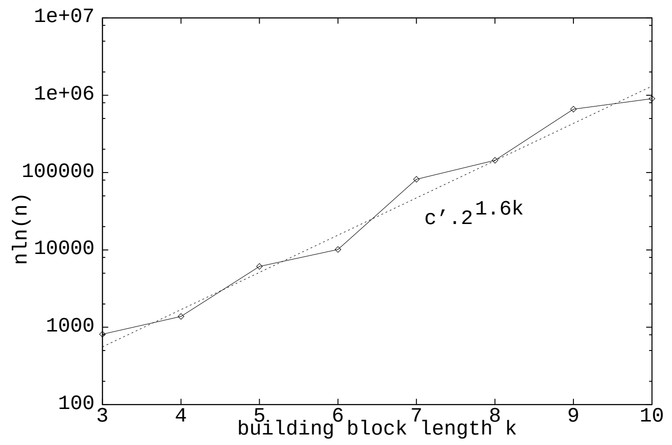

Figure 2: *Dimensional relation between BB-length k and population size n when mixing two building blocks with selection pressure $s = 4$ and crossover probability $p_c = 0.5$. The solid curve represents $n \ln n$ with n values shown in table 1. The dotted line $c'2^{1.6k}$ ($c' = 20$) indicates the linear relationship between $n \ln n$ and $2^{\mu k}$ for mixing 2 building blocks.*

In a first set of experiments we look at the relation between the population size n and the building block length (kFigure 2). Results are shown for selection pressure $s = 4$, crossover probability $p_c = 0.5$, number of building blocks $m = 2$ and building block length k going from 3 to 10. The experiments confirm the exponential growth relation between $n \ln n$ and k. The μ (≈ 1.6) coefficient is a little less than 2 as expected.

Figure 3 gives the relation between the crossover probability p_c and the population size n for $s = 4$, $k = 4$ and $m = 2$. The bottom part of the curve represents the mixing boundary and has the functional form as expected by Equation 6. Below this curve the building blocks initially grow but do not get exchanged before the population converges. The top part represents the schema theorem (Equation 1): above this curve BBs do not grow. For any parameter combination falling in the enclosed area the GA will reliably converge to the global optimum.

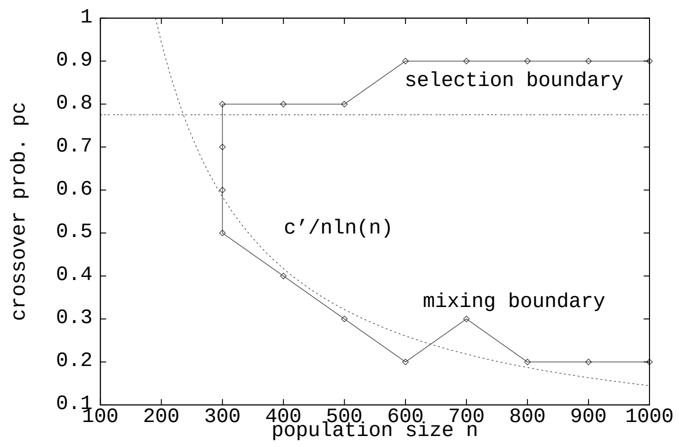

Figure 3: *Dimensional relation between population size n and crossover probability p_c when mixing two building blocks with selection pressure $s = 4$ and building block length $k = 4$. The region enclosed by the solid curve shows the n vs. p_c parameter combinations where at least 49 out of 50 runs converged to the global optimum. The lower part of this region is determined by the mixing boundary, which is well modelled by the dimensional relation $p_c > c'/n \ln n$ as indicated by the dotted line $c'/n \ln n$ ($c' = 1000$).*

Figure 4 shows p_c versus $\log s$ ($k = 4$, $n = 600$ and $m = 2$). Again the upper curve represents the schema theorem. The part of the lower curve with small $s-$values is the mixing boundary indicating a linear relationship between p_c and $\log s$. The vertical boundary at $s = 2^k$ is the cross-competitive boundary where selection is too high (Goldberg, Deb & Thierens, 1993).

4.2 Mixing Several Building Blocks

4.2.1 Dimensional Model

In the previous paragraph the simplest mixing process was considered - the exchange of two building blocks. To extend this analysis to the general m building block case, we have to look explicitly how the GA juxtaposes all the BBs. From Equation 2 we know that the mixing probability p_{mix} between two parents rapidly decreases when the number of unmatched BBs ($= d$) increases and the different amount of BBs between the parents ($= b$) for a given d-value increases. By far the highest mixing probability p_{mix} is obtained when the parents have the same number of BBs and they only have two unmatched BBs ($d = 2$, $b = 0$). In this case $p_{mix} = \frac{2}{2^{2k}}$ or when taking into account that non-BBs also have some optimal alleles $p_{mix} = \frac{2}{2^{\mu k}}$ with $\mu \leq 2$.

Let us call the number of building blocks present in an individual string its mixing level. The probability that a child will have two BBs more than one of its parents is negligible in comparison with the probability of having only 1 BB more. We can safely assume that after a mixing event has taken place one of the children will have one BB more juxtaposed and therefore will be at a mixing level that is only one higher than its parents: so the increase in mixing level goes step by step.

Another interesting consequence of the mixing probability calculation is that the level increase has to occur more or less simultaneously for all building blocks: to see this let us

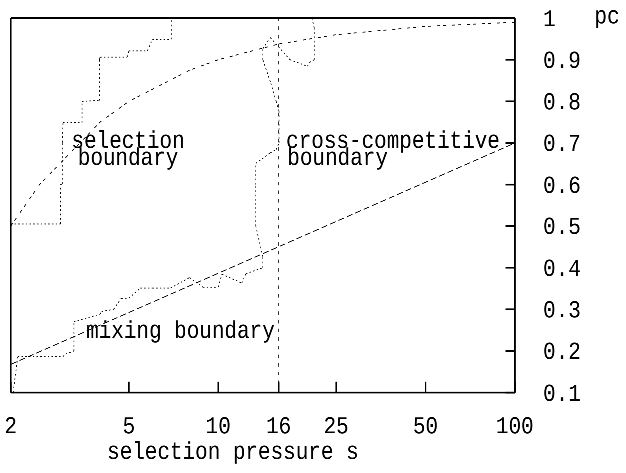

Figure 4: *Region of reliable convergence to the global optimum when mixing two building blocks with building block length* $k = 4$ *and population size* $n = 600$ *in function of the selection pressure* s *and the crossover probability* p_c. *The mixing boundary shows a linear relationship between* $\ln s$ *and* p_c *as predicted by the dimensional analysis.*

suppose that the mating parents are on a different mixing level. For instance the first parent might be at level two and have its 2 *BBs* at the first and second position ('bb***' with 'b' standing for the presence of a building block and '*' for a non-building block), while the other parent might be at level four ('*bbbb'). In addition let us assume that the first *BB* is covered until level 2, this means that there are no strings in the entire population that have the *BB* in the first position and are at the same time on a mixing level higher than 2. Now for the first *BB* to be covered at level four we need 2 mixing events: a first mixing event is needed to create a level three individual that has the first *BB* and two others juxtaposed. A second mixing event needs to mix the first *BB* with three other *BBs* to cover it at the fourth mixing level. However, while we are waiting on these two mixing events to occur, selection causes the strings at a higher mixing level to take over the population, hereby pushing the first *BB* to extinction. The greater the difference between the different *BB* coverage the more likely we will have premature convergence. To avoid this all *BBs* should more or less go simultaneously to the successive mixing levels and in a simple GA this can only be obtained by increasing the population size n.

To quantify this phenomenon we introduce the *mixing ladder climbing* model. We hypothesise that only the most probable mixing events actually determine the mixing dynamics: the mixing takes place between strings that are at the same mixing level and have only 2 unmatched *BBs* ($d = 2$, $b = 0$). Although the model takes a conservative view of GA mixing, it becomes sufficiently accurate with increasing building block length k. Any serious deviation from it (for instance when the population size n is too small) will result in bad *BB* exchange and premature convergence.

Suppose that in the current population all *BBs* are covered up to the mixing level m_l and all strings are at the same mixing level: each string has thus m_l building blocks juxtaposed. Since mixing is only likely to occur when the two mating parents have only 2 unmatched *BBs* between them, we can calculate the mixing probability p_{mix} by realizing that for each string with m_l *BBs* there are $m_l(m - m_l)$ different strings with which it has 2 unmatched *BBs* (e.g. for 'bbb****' we have '*bb***b', 'b*b**b*',...). The mixing probability is therefore:

$$p_{mix} = \frac{m_l(m-m_l)}{\begin{pmatrix} m \\ m_l \end{pmatrix}} \frac{2}{2^{\mu k}} \tag{7}$$

Now how do we reach mixing level m_l when we are at level $m_l - 1$? After n_x mixing events we have n_x individuals at level m_l, and the question to be answered is how many mixing events we need in order to have all BBs covered at this higher level. Since all of the n_x strings have m_l BBs we expect that $(1-(1-\frac{m_l}{m})^{n_x})m$ BBs are covered. By increasing the number of mixing events n_x the probability that all BBs are covered comes arbitrarily close to 1:

$$(1-(1-\frac{m_l}{m})^{n_x})m > (1-\alpha)m \tag{8}$$

with $\alpha < \frac{1}{m}$.
After simplifying, approximating $\ln(1-\frac{m_l}{m}) \approx -\frac{m_l}{m}$ and letting $\ln\alpha = -c$ we get:

$$n_x > c\frac{m}{m_l} \tag{9}$$

The number of mixing events needed to climb one step of the mixing ladder is thus proportional to the number of building blocks.

We now have all the necessary pieces to calculate the mixing time t_x - i.e. the number of generations needed for n_x mixing events to occur. This gives us:

$$t_x = c\frac{m}{m_l}\frac{1}{\frac{n}{2}p_c p_{mix}} \tag{10}$$

In order to allow the GA to climb each level of the mixing ladder step by step - and thus preventing premature convergence - the selection takeover time t_s has to be larger than the mixing time t_x at each level m_l. The worst case will take place when t_x is maximal and because of the combinatorial factor in Equation 7 this will happen at $m_l = \frac{m}{2}$. By using the Stirling approximation $m! \approx (\frac{m}{e})^m\sqrt{2\pi m}$, we see that:

$$\begin{aligned} \begin{pmatrix} m \\ m/2 \end{pmatrix} &= \frac{m!}{(m/2)!(m/2)!} \\ &= \sqrt{\frac{2}{\pi}}\frac{2^m}{\sqrt{m}} \end{aligned}$$

and thus

$$p_{mix} = \frac{\sqrt{2\pi}}{4}\frac{m^2\sqrt{m}}{2^m}\frac{1}{2^{\mu k}}$$

and Equation 10 can be simplified to:

$$t_x = c\frac{2^{\mu k}}{np_c}\frac{2^m}{m^{\frac{5}{2}}} \tag{11}$$

Interrelating the selection time t_s with the mixing time t_x we finally obtain:

$$n\ln n > c\frac{2^{\mu k}\ln s}{p_c}\frac{2^m}{m^{\frac{5}{2}}} \tag{12}$$

This dimensional relation shows the limitations of the basic genetic algorithm that has no linkage information and uses uniform crossover: by increasing the selection pressure fully deceptive building blocks can be grown but in order to mix them the population size has to increase exponentially with the number of building blocks !

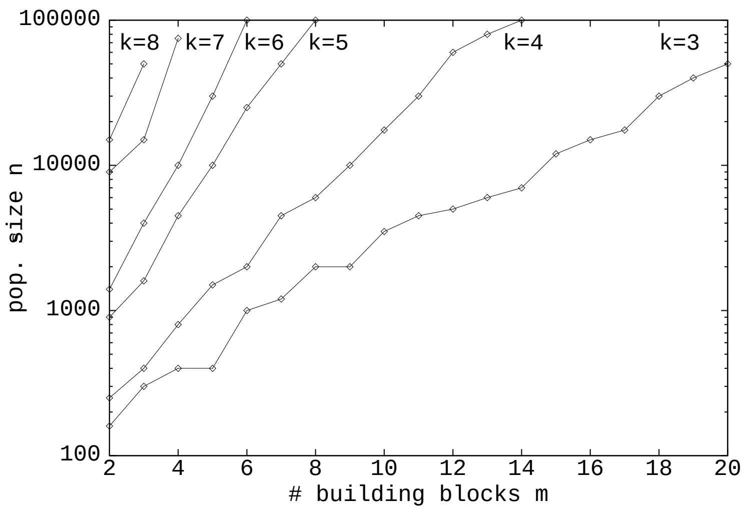

Figure 5: *Minimal population size n needed in function of the number of building blocks m in order to converge to the global optimum in at least 49 out of 50 runs. Results are shown for different building block lengths k, with selection pressure $s = 4$ and crossover probability $p_c = 0.5$.*

4.2.2 Empirical verification

In Figure 5 we have plotted the population size n versus the number of building blocks m for different values of $BB-$length. Selection pressure s and crossover probability p_c are resp. 4 and 0.5. Clearly the exponential growth of n with m is confirmed. For low building block length values ($k = 3$ and $k = 4$) the growth is a little slower: obviously the influence of emerging building blocks on the pure mixing is more pronounced in these cases.

4.3 Conclusion

We have analysed the exchange of building blocks - building block mixing - in simple GAs for GA-hard functions without the use of any prior linkage information. The traditional view on genetic optimisation considers crossover to be a source of building block disruption and therefore very disruptive operators such as uniform crossover are supposed to be of no use for problems where one needs to process building blocks as a whole. The most pronounced problem where this is the case are the fully deceptive functions, and general GA wisdom says that a basic GA can only solve these problems to global optimality if the building blocks are tightly linked and one uses one-point crossover, which has only a small disruptive effect. Assuming that the building blocks are tightly linked is however an extremely strong requirement which certainly limits the applicability of the basic genetic algorithm.

We have argued that there is a way to solve - at least in principle - fully deceptive functions by using uniform crossover. Since uniform crossover is linkage independent, this means that there is no need to have the building blocks tightly linked. The key factor lies in the selection pressure: by increasing the selection intensity, more copies are made of the building blocks and thus we are allowed to use a more disruptive crossover operator. The schema theorem can be used to see how much we have to increase the selection pressure. Experimental results show indeed that fully deceptive functions are solved to global optimality by using uniform crossover and a high selection pressure. So do simple genetic algorithms need tightly linked building blocks ? As far as the schema theorem is concerned they do not. There is however a caveat in all this, and it is here that the importance of the need for mixing analysis shows itself most profoundly. Having a high selective pressure ensures that the building blocks increase in number, but we have showed that in order to juxtapose or mix them we need an exponentially large population size.

Until now we have restricted our attention to a building block mixing analysis for fully deceptive functions with uniform crossover in simple GAs. By increasing the selection pressure we can increase the proportion of BBs but unfortunately the BB mixing becomes intractable. A number of extensions to the simple GA can be brought in to improve the mixing complexity. Since the mixing probability p_{mix} is highest when the mating parents have only two unmatched BBs, a mating scheme that restricts recombination to strings within a certain Hamming neighbourhood should decrease the mixing time t_x. Niching techniques might help us to climb the mixing ladder in a more smooth way without the need of excessive population sizes. It has been shown here that mixing events are rare, therefore we should be careful not to destroy individuals at the highest mixing level and elitism might help us to achieve this. These issues will be discussed in the next section.

5 Extending the Simple Genetic Algorithm

The previous section discussed where we could expect reliable convergence to the global optimum for the simple genetic algorithm, and how the failure boundaries scaled with the GA parameters (population size, string length, building block length, selection pressure, and recombination rate). Specific attention was given to what we have called the mixing failure, and an important conclusion was that due to this mixing failure, simple genetic algorithms are severely limited in their building block processing capability, unless these building blocks are tightly linked.

In this section we will go beyond the simple genetic algorithm and ask ourself whether some straightforward extensions to the simple GA might significantly improve the mixing problem. In particular we will add three mechanisms to the simple GA: *elitism*, *niching*, and *restricted mating*. A need for these three mechanisms can be reasoned upon by making the following observations:

1. First, in the simple GA the children produced by applying crossover are always replacing there parents. For hard problems good parent strings are difficult to construct, and it is a waste of computational effort to simply replace them by their offspring who have a distinctly high chance of being less fit. The philosophy in simple GAs behind this blind replacement strategy is that due to the ever increasing proportion of building blocks the chance of juxtaposing them becomes better all the time, so it does not matter if we throw good solutions away, they can easily be reassembled latter on. Again this philosophy is a testimony of how simple GAs are conceived to operate: problems are coded such that the substructures needed to compose (near)-optimal solutions have a low chance of being destroyed by the crossover operator and - although this is often more implicitly assumed - they have a high chance of being assembled (or mixed) together.

   A classical way of ensuring that the best solution is never lost is to copy the best individual to the next generation. However from a mixing point of view this does not help much. Instead of keeping the best (or a few best) solution, we would like to minimise the impact of mixing failures by keeping all parents that are more fit than their offspring. One way of achieving this is by using the Elitist Recombination algorithm (Thierens & Goldberg, 1994), which we will discuss in the following section.

2. Next, since in the simple GA the building blocks are involved in a selection versus mixing race, it might be beneficial to significantly decrease this race against time by using a niching technique. Niching is traditionally being used when optimising multimodal problems in order to let multiple optima coexist in the population. Since building blocks are typically member of different local optima, the use of niching can

keep them in the population and effectively buying them time to get mixed with other building blocks. In section 5.2 we will go into this in more detail.

3. Finally, since the mixing probability is dependent on the hamming distance between the parents, an obvious candidate to improve the mixing success is to restrict mating to parents which are neighbours in Hamming space. When the mating parents are close together in Hamming space, most building blocks will be the same in both parents and cannot be disrupted by any crossover operator, thus increasing the mixing probability. Section 5.3 discusses the mating restriction.

5.1 Elitist Recombination

In the basic genetic algorithm individual strings are copied during the selection phase and then recombined to create new strings. To reduce the schema disruption factor $\epsilon(h, t)$, the probability of applying crossover p_c is often less than one, so some of the string copies simply go to the next generation without further modifications. However the best solutions from the current generation will not necessarily be present in the next generation. To make sure that the best solution found so far is always present in the population we can simply copy the best string from the current population to the next generation. This approach - called elitism - has however a disadvantage: if the problem is very hard and recombination cannot create new, better strings in a small number of generations, then the amount of copies of the best solution will increase rapidly and will take over a large part of the population, driving out other good solutions and thus severely decreasing the search quality. An additional problem is that with this elitist method only the best is preserved while other strings that might have potentially good building blocks juxtaposed may be destroyed by recombination.

An interesting alternative is to use a population-wide elitism where the offspring has to compete with the entire parent population. This approach has been applied successfully in many implementations, for instance CHC, Genitor, $(\mu + \lambda)$-ES, and the Breeder GA. In population-wide elitism, offspring that is worse than their parents often replace other parents which have a low fitness. From a selection point of view this is what we want, however from a mixing point of view this might not be optimal because there is no direct correspondence between entering the next generation and successful mixing events. Recall that we have called a mixing event successful whenever one of the children was better than one of its parents. In population-wide elitism, offspring can enter the next generation even after unsuccessful mixing. In order to have a more direct "credit assignment" between successful mixing and entering the next generation we here apply the elitist recombination algorithm. In the basic genetic algorithm, selection and recombination are separated into two distinct phases. For every generation, selection first generates an intermediate population by making extra copies of the best individuals and by eliminating the worst performing individuals. Thereafter recombination is applied to generate the next generation. In order to promote building block growth one typically chooses a crossover probability value $p_c \leq 1.0$. There are however no guidelines - except from some empirical studies - on how to choose a value for p_c.

The elitist recombination GA is a much simpler implementation: no separate selection and recombination phases are needed and crossover is applied for every mating pair so there is no need to specify a value for p_c. Elitist recombination works entirely at the level of each family: for every mating pair two offspring are created and the best two of these four individuals go to the next generation. Parents are replaced by their offspring when these have a higher fitness value, but this elitism is only working at the family level not at the population level. There is also no separate selection phase where copies are made of good strings. As a result elitist recombination does not suffer from the danger of extreme premature convergence that often jeopardises other elitist schemes (of course premature convergence might still occur, for instance when the population size is much too small). It has also been shown that this family competitive scheme has the same selection intensity as tournament selection with tournament size 2, which is in general a fairly good selective pressure for most problems.

Elitist Recombination algorithm:

```
1. initialise population
2. for every generation
   (a) random shuffle population
   (b) for every mating pair:
       • generate two offspring
       • keep best two of the four individuals
```

Having ensured that hard found good solutions are not lost anymore, we now incorporate the use of niching to combat the race against time seen in the simple genetic algorithm.

5.2 Niching

In a simple genetic algorithm selection and mixing are actually involved in a race against time: in order for recombination to have enough time to juxtapose the optimal alleles or building blocks, the mixing time t_x has to be smaller than the selection time t_s. When the problem is hard from a mixing point of view, the population size needs to be increased considerably to allow good mixing before the population converges due to the selective pressure. An alternative to the population size increase is the use of niching techniques. Niching basically keeps multiple good solutions in the population and prevents the GA from converging to a single string. Strings that are composed of different building blocks can thus coexist in the population and crossover has all the time it needs to mix or juxtapose the different building blocks. Put it another way, niching increases the selection time dramatically while leaving the mixing time untouched, and the condition for good mixing to occur - $t_x < t_s$ - is now easily fulfilled.

A number of mechanisms to induce niches in a genetic algorithm population have been proposed in the literature. These niching mechanisms can be classified into two classes: *sharing* schemes and *crowding* schemes.

Here we will generalise the elitist recombination algorithm into a crowding algorithm called the generalised crowding algorithm. De Jong's crowding algorithm selects randomly a subset of the population and replaces the string that matches the child most closely whether or not the child's fitness value is better. Deterministic crowding and elitist recombination let the children compete only with their parents but they need to have a better fitness value to enter the population. Generalising the subset selection mechanism and the local elitism principle we obtain a generalised crowding algorithm that captures the characteristics of both schemes:

Generalised Crowding algorithm:

```
1. randomly shuffle the population
2. for each pair of strings: create two children
3. for each child:
   a) randomly select $w - 2$ strings from the population
   b) determine the closest string from these
      strings and the two parents
   c) let the child compete with this string
```

To understand how the generalised crowding algorithm maintains different solutions in the population it is instructive to compute how the growth of the highest valued string is limited and thus prevented from taking over a substantial part of the population and pushing other solutions out of the population. Assume the best solution has a proportion of $p_i(t)$ strings, the population has size n and the crowding window size w. The number of copies of the best solution can only increase whenever a new copy competes with an element that is not a copy of the best string. This can only happen if none of the w strings in the crowding window are from the best set. Since the crowding window is randomly selected without replacement we have:

$$P[p_i(t+1) > p_i(t)] \quad = \quad \frac{\binom{n(1-p_i(t))}{w}}{\binom{n}{w}}$$

$$
\begin{aligned}
&= (1 - \frac{np_i(t)}{n})(1 - \frac{np_i(t)}{n-1}) \dots (1 - \frac{np_i(t)}{n-w+1}) \\
&< (1 - p_i(t))^w \\
&< e^{-wp_i(t)}
\end{aligned}
$$

Both inequalities are close to being an equality when the window size w is much smaller than the population size n, and $p_i(t) << 1$. The probability that the number of copies of the best string increases is thus an exponentially decreasing function of the present proportion and the crowding window.

5.3 Restricted Mating

Extending the genetic algorithm with a local elitist mechanism and with a niching technique helps the basic GA in overcoming some aspects of the mixing problem. Elitism makes sure that good solutions do not get lost unless they can be replaced by better ones. Niching eliminates the race against time that is going on in a basic GA between selection and recombination. Both mechanisms however do not change the mixing probability p_{mix}. The mixing ladder climbing analysis from the previous chapter showed that the mixing probability for bounded deceptive problems is extremely low:

$$p_{mix} = \frac{m_l(m - m_l)}{\binom{m}{m_l}} \frac{2}{2^{\mu k}}$$

where m is the total number of building blocks, m_l the number of building blocks that are already juxtaposed and k the deception length (μ is just a factor between 1 and 2).

In order to reduce the mixing complexity we have to increase the mixing probability p_{mix}. Recall that for a given deception length k the mixing probability p_{mix} was minimal when strings had half of the building blocks juxtaposed or $m_l = m/2$. The mixing probability was in this case :

$$p_{mix} = \frac{\sqrt{2\pi}}{4} \frac{m^2\sqrt{m}}{2^m} \frac{1}{2^{\mu k}}$$

The exponential term in the denominator causes the mixing to become intractable with increasing number of building blocks for a basic GA without a priori linkage information.

The mixing probability p_{mix} decreases exponentially with both the deception length k and with the number of building blocks m. The exponential dependence on the deception length however should not concern us: the use of bounded but fully deceptive problems was just intended to push the GA to its limits and each deceptive subfunction is basically a needle in a haystack so there is no information in the search space that might help any search algorithm of finding it. The mixing probability for the basic GA without a priori linkage information, is also exponentially dependent on the number of building blocks. From the previous section (Section 3) we recall that the factor $\frac{m_l(m-m_l)}{\binom{m}{m_l}}$ expresses the probability that two randomly chosen strings with m_l building blocks are neighbouring strings in Hamming space. This means that the two strings each have only one building block that is not matched by the other string, e.g.:

$$
\begin{aligned}
p1 &: 111\ 000\ 000\ \underline{111}\ 111\ 000\ 000\ 111 \\
p2 &: 111\ 000\ \underline{111}\ 000\ 111\ 000\ 000\ 111
\end{aligned}
$$

An apparently obvious way to increase the mixing probability is to replace the random selection of mating strings by a restricted mating mechanism. For every mating string we scan the population until we find a neighbouring string, so the factor $\frac{m_l(m-m_l)}{\binom{m}{m_l}}$ disappears and the mixing probability becomes $p_{mix} = 2^{1-\mu k}$. This assumes that the population is large enough so we can find a neighbouring string. Suppose - as we did in the building block mixing analysis - that all strings have m_l building blocks juxtaposed. The probability that a string has a neighbouring string in a population of size n is then:

$$P[neighbor] = 1 - \left(1 - \frac{m_l(m-m_l)}{\binom{m}{m_l}}\right)^n$$

This probability is lowest when the fraction $\frac{m_l(m-m_l)}{\binom{m}{m_l}}$ is smallest, or when $m_l = m/2$. Taking this value for m_l we can compute the necessary population size n to let the probability $P[neighbor]$ come arbitrary close to 1 as:

$$1 - \left(1 - \frac{\sqrt{2\pi}}{8}\frac{m^2\sqrt{m}}{2^m}\right)^n > 1 - \epsilon$$

or

$$n > \ln \epsilon^{-1}\, 4\sqrt{\frac{2}{\pi}}\,\frac{2^m}{m^2\sqrt{m}}$$

Of course it is not necessary for every string in the population to have a neighbouring string but even if we relax $P[neighbor]$ considerably the exponential increase in the number of building blocks remains overwhelming.

5.3.1 Experimental verification

The above discussion indicates how family elitism, niching and restricted mating help to improve the search properties of the simple genetic algorithm. Unfortunately the building block mixing probability p_{mix} is still very low due to two reasons:

1. The probability that two parents are neighbouring strings in Hamming space is significantly higher when using restricted mating but it is still substantially low for problems with a large number of building blocks m.
2. The mixing probability p_{mix} is proportional to $2^{-\mu k}$ and thus the mixing of building blocks becomes very unlikely with increasing deception length k and with increasing μ-factor. Recall that when the strings in the population consist only of building blocks and deceptive attractors, the factor μ is equal to two.

To illustrate the remaining problem we have integrated the mechanisms into the following extended genetic algorithm:

Extended genetic algorithm

```
A. create random population
B. evaluate population
C. for t: 1 → max_generation
  1. shuffle population
  2. select mates:
       for i: 1 → pop_size
         for j: (i+1) → (i+1+mating_window)
           mates(i) = closest_string(j)
  3. create offspring:
       for i: 1 → pop_size
         crossover(pop(i),mates(i))
         evaluate(offspring)
  4. insert offspring:
       for i: 1 → pop_size
         for j: i → (i+crowding_window)
           competitor(i) = closest_string(j)
           if fitness(competitor(i)) < fitness(child(i))
             replace competitor(i) with child(i)
```

As an example we have run this extended GA on the fully deceptive trap function with 10 building blocks of deception length $k = 5$, a population size $n = 500$, mating window size $w_m = 50$ and crowding window size $w_c = 25$. We assume no a priori linkage information and use uniform crossover. Some typical strings after 25 generations were:

p_1 : 00000000000000011111000000000000000000000000000000
p_2 : 00000000000000000000000000000011111000000000000000
p_3 : 00000111110000000000000001000000000000000000011111
p_4 : 11111000001111100000000000000000000111111111100000
p_5 : 11111000000000000000000000000000000000100000000000

Due to the elitism and niching the entire population consists almost solely of building blocks and deceptive attractors. The $\mu-$factor in the mixing probability is thus approximately equal to 2: $p_{mix} = \frac{2}{2^{2k}}$. During the first generations the GA quickly identifies the locally optimal bit combinations but then its performance is severely hampered by the destructiveness of uniform crossover. Uniform crossover is a blind recombination operator and the probability of exchanging bit values is equal at all bit positions. Due to the extensions though, all the building blocks remain in the population and if we would only change the allele swapping distribution in function of the current population, then the mixing would become much faster. This leads us to our overall conclusion: in order to be efficient and scale up well with increasing problem size, genetic algorithms need to be given information about the building blocks. This information can already be in the problem-coding - when building blocks are tightly linked - or mechanisms are needed that identify the building blocks such that they can be efficiently mixed together.

6 Discussion

We have discussed the mixing requirements of simple genetic algorithms and showed how it forms a part of the boundary of the region in GA parameter space where reliable and efficient search takes place. The need for proper building block mixing limits the simple genetic algorithms performance on hard problems. Without any knowledge of the building block linkage structure these problems can be solved by increasing the selection pressure and applying uniform crossover, but unfortunately this does not scale up with increasing problem size. From a mixing point of view algorithmic extensions such as family elitism, niching, and restricted mating all have a beneficial effect on the building block mixing process, but they do not result in scalable genetic algorithms. To obtain competent, scalable genetic algorithms one need to ensure that the building block mixing is explicitly taken care off.

One approach might be to adapt the crossover recombination distribution at run time. Traditional crossover operators do not change their recombination pattern during the search process: one point crossover for instance always selects a cut point at random with equal probability for all possible positions in the string. Uniform crossover exchanges each allele with a fixed swapping probability. Building block exchange would be much more efficient when the crossover operator is made adaptive such that its recombination probability distribution reflects the assumed position of the building blocks in the current population. Identifying building blocks during the GA search does however holds the potential risk of being misled early in the search and therefore missing some important information.

An alternative is to identify the building blocks before the actual GA search. This method has been the basic principle of the messy genetic algorithm family (Goldberg, Korb & Deb, 1989; Goldberg, Deb & Korb, 1990; Goldberg, Deb, Kargupta & Harik, 1993; Kargupta, 1996; Kargupta, 1998).

Also, a number of recent genetic algorithms try to learn the building block information by explicitly estimating the distribution of the promosing subregions in the search space (De Bonet,Isbell, & Viola, 1997; Baluja & Davies, 1997; Mühlenbein & PaaB, 1996; Mühlenbein, Mahnig, & Rodriguez, 1998; Pelikan, Goldberg, & Cantu-Paz, 1998). Here too, one first tries to acquire the necessary information such that the problem becomes "mixing-easy", and fast, reliable, and scalable genetic algorithms are subsequently obtained.

7 Conclusions

We have analysed the building block mixing capability of simple genetic algorithms when given no linkage information. It was shown that with a sufficiently high selection pressure and uniform crossover the simple genetic algorithm can solve GA-hard problems of bounded difficulty without any knowledge (explicitly or implicitly) of the building blocks. Unfortunately the process does not scale up with increasing problem size. Straightforward extensions as family elitism, niching, and restricted mating help to combat the selection-mixing time race, but are by themselves not enough to resolve the scalability problems. Competent, scalable genetic algorithm performance can only be obtained when building block linkage information is already in the problem-coding, or is identified before or during the genetic algorithm's search.

References

Bagley,J.D. (1967). The behavior of adaptive systems which employ genetic and correlation algorithms. *Dissertation Abstracts International*, 28(12), 510B.

Baluja,S. & Davies,S. (1997). Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. *Proceedings of the 11th International Conference on Machine Learning.* pp.30-38. Morgan Kaufmann.

De Bonet,J.S., Isbell,C.L., & Viola,P. (1997). MIMIC: Finding optima by estimating probability densities. *Advances in Neural Information Processing Systems.* Vol.9, pp.424. MIT Press.

Deb K., & Goldberg D.E. (1993). Analyzing deception in trap functions. *Proceedings of Foundations of Genetic Algorithms FOGA-II.* ed. D. Whitley. Morgan Kaufmann.

Frantz,D.R. (1972). Non-linearities in genetic adaptive search. *Dissertation Abstracts International*, 3(11), 5240B.

Goldberg D.E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley.

Goldberg,D.E. (1998). *The race, the hurdle, and the sweet spot: lessons from genetic algorithms for the automation of design innovation and creativity.* IlliGAL Report No. 98007. University of Illinois at Urbana-Champaign, Illinois Genetic Algorithm Laboratory.

Goldberg D.E., & Deb K.(1991). A comparative analysis of selection schemes used in genetic algorithms. *Proceedings of Foundations of Genetic Algorithms FOGA-I.* ed. G. Rawlings. pp.69-93. Morgan Kaufmann.

Goldberg D.E., Deb K., & Clark J.H.(1992). Genetic algorithms, noise, and the sizing of populations. *Complex Systems, Vol.6,* pp.333-362.

Goldberg D.E., Deb K., Kargupta H., & Harik G. (1993). Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. *Proceedings of the Fifth International Conference on Genetic Algorithms ICGA-93.* ed. S. Forrest. pp.56-64. Morgan Kaufmann.

Goldberg D.E., Deb K., & Korb B. (1990). Messy genetic algorithms revisited: studies in mixed size and scale. *Complex Systems, Vol.4,* 415-444.

Goldberg D.E., Deb K., & Thierens D. (1993). Toward a better understanding of mixing in genetic algorithms. *Journal of the Society for Instrumentation and Control Engineers, SICE Vol.32,* No.1 pp.10-16.

Goldberg D.E., Korb B., & Deb K. (1989). Messy genetic algorithms revisited: motivation,analysis, and first results. *Complex Systems, Vol.3,* pp.493-530.

Harik,G. (1997). *Learning linkage to efficiently solve problems of bounded difficulty using genetic algorithms.* Doctoral dissertation, Dept. Computer Science, University of Michigan, Ann Arbor.

Holland J.H. (1975). *Adaptation in natural and artificial systems.* Ann Arbor: University of Michigan Press.

Ipsen D.C.(1960). *Units, dimensions, and dimensionless numbers.* New York: McGraw-Hill.

Kargupta, H. (1996). The gene expression messy genetic algorithm. *Proceedings of the IEEE International Conference on Evolutionary Computation,* 814-819, IEEE Press.

Kargupta, H. (1998). Gene expression and large scale evolutionary optimization. *Computational Aerosciences in the 21st Century.* Kluwer Academic Publishers.

Kargupta,H. & Goldberg,D.E. (1996). SEARCH, blackbox optimization, and sample complexity. *Foundations of Genetic Algorithms,* 291-324, Morgan Kaufmann.

Mitchell, T.M. (1982). Generalization as search. *Artificial Intelligence,* 18(2), 203-226.

Mühlenbein,H. & PaaB,G. (1996). From recombination of genes to the estimation of distributions. I. Binary parameters. *PPSN IV*, pp.178-187, Springer-Verlag.

Mühlenbein,h, Mahnig,T., & Rodriguez,A.O. (1998). Schemata, distributions, and graphical models in evolutionary optimization. (Tech. Report GMD). *Submitted for publication.*

Paredis,J. (1995). The symbiotic evolution of solutions and their representations. *Proceedings of the 6th International Conference on Genetic Algorithms* pp.359-365, Morgan Kaufmann.

Pelikan,M., Goldberg,D.E., & Cantu-Paz,E. (1998). BOA: the bayesian optimization algorithm. *IlliGAL Report No. 98013.*

Smith,J. & Fogarty,T. (1996). Recombination strategy adaptation via evolution of gene linkage. *Proceedings of the IEEE International Conference on Evolutionary Computation* pp.826-831, IEEE Press.

Thierens D. (1995). *Analysis and Design of Genetic Algorithms.* Unpublished PhD thesis. Katholieke Universiteit Leuven, Belgium.

Thierens D., & Goldberg D.E. (1993). Mixing in Genetic Algorithms. *Proceedings of the Fifth International Conference on Genetic Algorithms ICGA-93.* ed. S. Forrest. pp.38-45. Morgan Kaufmann.

Thierens D., & Goldberg D.E. (1994). Elitist Recombination: an integrated selection recombination GA. *Proceedings of the IEEE World Congress on Computational Intelligence,* Orlando (VS). Vol.1 pp.508-512.

van Dijk,S., Thierens,D., & de Berg,M. (1998). Robust genetic algorithms for high quality map labeling. *Technical Report TR-1998-41,* Utrecht University.

Watanabe,S. (1969). *Knowing and guessing - A formal and quantitative study.* Wiley & Sons.

Wolpert,D.H. & Macready,W.G. (1996). No Free Lunch Theorems for Search. *SFI-TR-95-02-010.*