Benefits of a Population: Five Mechanisms that Advantage Population-Based Algorithms

Adam Prügel-Bennett
School of Electronics and Computer Science, University of Southampton, SO17 1BJ, UK.

***Abstract*—This paper identifies five distinct mechanisms by which a population-based algorithm might have an advantage over a solo-search algorithm in classical optimisation. These mechanisms are illustrated through a number of toy problems. Simulations are presented comparing different search algorithms on these problems. The plausibility of these mechanisms occurring in classical optimisation problems is discussed.**

The first mechanism we consider relies on putting together building blocks from different solutions. This is extended to include problems containing critical variables. The second mechanism is the result of focusing of the search caused by crossover. Also discussed in this context is strong focusing produced by averaging many solutions. The next mechanism to be examined is the ability of a population to act as a low-pass filter of the landscape, ignoring local distractions. The fourth mechanism is a population's ability to search different parts of the fitness landscape, thus hedging against bad luck in the initial position or the decisions it makes. The final mechanism is the opportunity of learning useful parameter values to balance exploration against exploitation.

***Index Terms*—Populations, genetic algorithms, crossover, building blocks, critical variables, focusing, low-pass filtering, hedging, parameter tuning.**

I. Introduction

EMPIRICALLY population-based algorithms have been found to perform well on many real world problems. This has led to an effort by theorists to understand and explain this behaviour. This paper attempts to categorise the different mechanisms whereby a population has an advantage over a solo-search algorithm in the belief that a better understanding of the benefits of populations might lead to improved algorithms.

This paper focuses on mechanisms that provide a benefit to population-based algorithms which are "generic" to a reasonably large set of real world problems. There is considerable subjectivity in making this judgement as the set of potential real world optimisation problems is unknown. This paper has a clear bias towards mechanism studied by the author. The paper excludes carefully manufactured problem even though some of these show that population-based algorithms have provably superior performance to solo-search algorithms (e.g. [1], [2]). The problems presented in the cited papers, and many similar papers, highlight the fact that there are a huge number of mechanisms by which a population can be beneficial (or sometimes detrimental). However, many of these mechanisms depend on specific properties of the landscape structure. The hope of this paper is that each of the five mechanisms we have identified is applicable to a large 'genus' of problems in the hierarchy of all problems. We know from the *no free lunch* theorem [3] that, when considering a sufficiently large set of problems, populations offer no benefit on average over other search algorithms. This does not contradict the premise of this paper as, at least in the author's view, the problems of interest in the real world are only a small subset of all possible problems.

Although we are interested in mechanisms that should be relevant to real world problems we illustrate many of them using toy problems, as these make the mechanism more easily understood. An attempt has been made to choose "natural toy problems" which do not require much fine tuning. Again there is a considerable subjective bias in deciding what constitutes a natural toy problem. Many of the toy problems described in this paper have been proposed elsewhere, although a few of the problems appear here for the first time. By collecting them together it is hoped they will provide a more rounded picture of the possible benefits of populations. This paper is informal in that it provides no proof of time complexity, instead we give plausibility arguments for the efficiency of different algorithms. In the case where concrete problems have been proposed, we often show simulation results. These are intended to illustrate the behaviour of different algorithms on the problems of interest and are not exhaustive tests. For those with a rigorous bent, this paper can be viewed as a challenge to prove time complexity results for these, or related, problems.

As an additional caveat, it should be stressed that this paper focuses on classical optimisation of, primarily, combinatorial problems. There are other application areas where the use of populations carry benefits. For example, in multi-objective optimisation, populations provide a natural way to approximate the Pareto front. As another example, in statistical sampling using Monte Carlo methods, populations provide a greater independence. This paper does not attempt to survey these important applications of population-based algorithms, primarily dues to the lack of expertise in these areas of the author. Populations have also proved very successful in solving problems where evaluating the fitness function is subject to sampling noise or in dynamically changing environments where the fitness function is no longer static. Although important, these applications fall outside of the remit of this paper. Finally, by focusing on classical optimisation, no attempt has been made to describing mechanisms that are plausible from a biological perspective.

The paper is organised as follows. We start by outlining the cost of using a population, since the benefit provided by using a population must out-weigh this cost. The next five sections take each of the mechanisms in turn and describe how they work. We start with building block problems and those involving critical variables in section III. In section IV, we discuss focusing produced by crossover and strong focusing produced by averaging. We then consider the ability of populations to ignore distractions in the landscape in section V. In section VI, we consider the benefits of exploring different parts of the fitness landscape. The last mechanism we discuss is learning useful parameters, discussed in section VII. Finally, we draw conclusions in section VIII.

II. Cost of Populations

Using a population as opposed to a solo-searcher comes with a potential cost that every member of the population must be moved. Given a population of size P this leads to a potential increase in the required number of function evaluations by a factor P. Whether this price is paid will depend on the problem. For example, this would be the (approximate) loss in performance if we tried to solve the *Onesmax problem* using a population of hill-climbers, as the typical time to solve this problem varies little between runs. Although Onesmax is very well-known, variants of it will reappear many times in this paper and so we take the trouble of defining it formally. The problem consists of a binary string $\boldsymbol{X} = (X_1, X_2, \ldots, X_n)$, where $X_i \in \{0, 1\}$. The fitness, which is to be maximised, is equal to the number of 1's in the string

$$F_{\mathit{Onesmax}}(\boldsymbol{X}) = \sum_{i=1}^{n} X_i.$$

Starting from a random string, a solo hill-climber has to change all its 0 variables to 1's. Since almost all members of the initial population will have approximately equal numbers of 1's and 0's, there is very little variation in the run time of different hill-climbers.

In problems where the solution could appear anywhere in the search space, the linear cost of having to move every member of the population is often offset by the fact that there are P searchers. For example, in the *Needle-in-a-haystack problem*, where one point in the search space is optimal and all other points have the same fitness, a population would search the space as effectively (or perhaps we should say ineffectively) as a solo-searcher.

Problems of interest tend to fall somewhere between the Onesmax problem and the Needle-in-a-haystack problem—that is, Onesmax is of little interest as it is too easy with very strong heuristics about the location of the optima, while the Needle-in-a-haystack is of little interest as it is too hard have no heuristic information about the location of the global optimum. Problems lying in the middle of this spectrum give more scope for populations to provide a significant advantage compared to a solo-searcher.

III. Building-Blocks and Critical Variables

A. Concatenated-$\mathcal{V}$ Problem

One of the oldest explanations for the benefits of a population is that it allows different parts of the solution to be discovered in different individuals and then these 'building-blocks' are put together through crossover. This mechanism, clearly, relies on combining solutions using crossover. Despite the attractions of this idea it is non-trivial to construct a toy problem that demonstrated how the building-block hypothesis would work (we will, however, do this below). Perhaps the best known toy problem, that was originally proposed to show this mechanism at work, is the *Royal Road function* [4]. The problem is described by a binary string divided into a number of blocks. The fitness is proportional to the number of blocks consisting of all 1's. The idea is that the blocks could be solved in different individuals and then combined together by single-point crossover. This problem, however, failed to show any benefit from using a population, with a simple hill-climber out-performing a genetic algorithm. The reason for this is that there is no advantage to discovering the blocks in different members of the population as opposed to a single member. We shall see below that a modification of this block problem can lead to a problem where a genetic algorithm substantially out-performs more traditional search algorithms. Other problems have also been proposed which attempt to show that populations using crossover can put together building blocks, such as the H-IFF problem [5], [6]. These need to be carefully tuned in order to work and are not discussed here.

An example, of a problem that demonstrates a building-block mechanism is the *Concatenated-$\mathcal{V}$ problem*. Again we consider a binary string where the variables are grouped into blocks. For simplicity we assume the blocks consist of k variables and there are m blocks, where the fitness is equal to

$$F_{\mathit{CV}}(\boldsymbol{X}) = \sum_{i=0}^{m-1} F_i(\boldsymbol{X}), \quad F_i(\boldsymbol{X}) = \left| \frac{k-1}{2} - \sum_{j=1}^{k} X_{k\,i+j} \right|.$$

An example of the block fitness function, $F_i(\boldsymbol{X})$, for a block with 7 variables is shown in figure 1. The total fitness is equal to the sum of the fitnesses of each block, and each block is maximised when all the variables in it are equal to 1. However, each block also has a local maximum when all the variables are equal to 0. To illuminate the properties of this problem

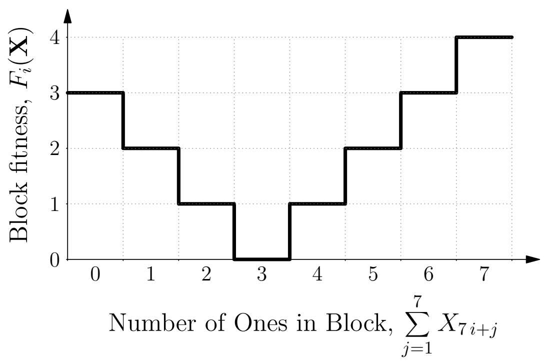

Fig. 1. Example of the block fitness function used in the Concatenated-$\mathcal{V}$ problem for a block of 7 bits.

we can draw the 'barrier tree' of the landscape [7], [8]. Barrier trees provide a pictorial representation of the fitness landscape which highlights the local maxima, and the fitness barriers needed to move between them. They are constructed by lumping together configurations of the same fitness which are accessible through a path that does not exceed the fitness of the current configurations. Figure 2 shows a barrier tree for the block problem with $m = 4$ blocks of $k = 7$ variables each. Each point represents a set of configurations which are placed vertically according to their fitness (following the usual convention for barrier trees, higher fitness configurations are plotted lower on the tree). The nodes in the tree are connected if there are neighbouring configurations in the two sets of configurations and there are no other configurations with a smaller fitness gap. To save space we have truncated the tree

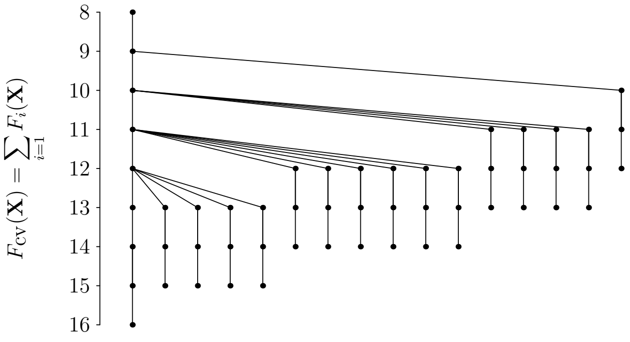
$F_{\mathrm{CV}}(\mathbf{X}) = \sum_{i=1}^{m} F_i(\mathbf{X})$
8
9
10
11
12
13
14
15
16

Fig. 2. Barrier tree of the Concatenated-$\mathcal{V}$ problem with $m = 4$ blocks each consisting of $k = 7$ variables. A Hamming neighbourhood is used in constructing the tree. Only high cost configurations are shown.

at a fitness of 8, lower fitness solutions would lie on a single main trunk which goes to a fitness of zero. Each leaf in the barrier tree represents a local optimum state. The tree shows the lowest barrier that needs to be crossed to move between local maxima through single variable flips. Thus to move from any of the 4 local maxima at cost 15 to the global maximum at cost 16 requires going through a configuration of cost 12 in the best case (since we have to move from a configuration in which one of the blocks is all 0's to a configuration where 3 of the variables are 1's, at which stage a hill-climber could reach the global maxima). Note, that the number of leaves in the tree is 2^m and thus grows exponentially with the number of blocks, m, while the heights of the barriers grow linearly with the number of variables in each block, k.

A local hill-climber (flipping one bit at a time) will drive each block to a state of all 1's or all 0's depending on the initial number of 1's in the block. There is a small bias towards the all 1's state, so that for a block of size $k = 7$ a hill-climber will end up in the all 1's state with a probability of $p_1 = 21/32 = 0.656$ (this is easily computed by enumerating all possible start states and computing the probability of them finishing in a particular end state). This probability slowly decreases towards 0.5 as k increases. The probability of a hill-climber performing single-site mutation finishing with l blocks in the all 1's state and $n - l$ blocks in the all 0's state is simply given by a binomial distribution with probability p_1. Thus, for $m = 4$ and $k = 7$ (the problem illustrated in figures 1 and 2) the expected fitness reached by a hill-climber is 14.625 ± 1 while the probability of ending up in the all 1's state with fitness 16 is 0.185; for $m = 10$ and $k = 7$ the expected fitness is 36.6 ± 1.5 with the probability of ending up in the all 1's state with fitness 40 being 0.015; while for $m = 100$ and $k = 7$ the expected final fitness for a hill-climber is 366 ± 5 with the probability of ending up in the all 1's state with fitness 400 is 5×10^{-19}. We note that as the number of states grow it becomes exponentially unlikely for a hill-climber to find the optimum state.

To illustrate the performance of different algorithms on this problem we run them on a problem with $m = 64$ blocks and with $k = 7$ variables per block. The maximum fitness for this problem is 256. Figure 3 shows the performance of different algorithms on this problem. Each algorithm was run 1000 times. The curves show the mean fitness as a function of the number of function evaluations. The empirical error bars are similar in size to the line width and are not shown. We

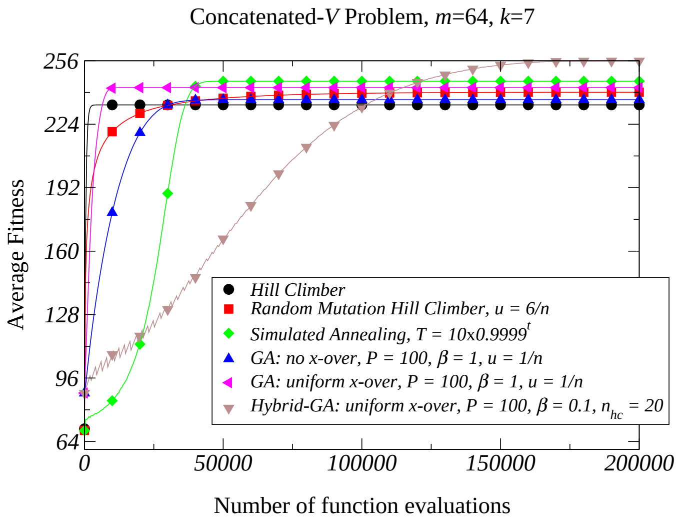

Fig. 3. The fitness averaged over 1000 runs for different algorithm on the Concatenated-$\mathcal{V}$ problem with m=64 and k=7 versus the number of function evaluations.

treat the hill-climber as a baseline algorithm. For this problem it reaches a mean fitness of 234 with an expected standard deviation of 3.8. The probability of a hill-climber reaching the global maximum is 2.7×10^{-13}. Thus trying to solve the problem by performing repeated runs of a hill-climber is a very slow method to solve this problem.

A simple elaboration of a hill-climber is the random mutation hill-climber where at each step, rather than mutate a single variable, we attempt to mutate each variable of the string with a probability u. If it reaches a state of equal or higher fitness than the current state, then it accepts the mutations, otherwise it stays in the current state. Compared with the hill-climber this allows the random mutation hill-climber to mutate a block from the local optimum state to the global optimum state, however, because these states are far apart this is very unlikely unless the mutation rate is very high. However, because there are many blocks a large mutation rate is likely to disrupt a good block so that, as the solution improves, it becomes increasing difficult to find an improving move. In consequence, a random mutation hill-climber does little better then a normal hill-climber. For small mutation rates it behaves very similarly to a normal hill-climber. For larger mutation rates it explores the the search space through a set of large macro-mutations. Although this slows down the search, given enough time larger mutation rates are found to give slightly better performance than smaller rates. In figure 3, we show a random mutation hill-climber with a mutation rate of $6/n$ so that on average about 6 of the 448 variables are mutated at each step. Through initial experimentation this seems to be close to the optimal fixed mutation rate (we may have done better by annealing the mutation rate, but this was not tried).

A stochastic hill-climber sometimes makes a move that decreases the fitness. In principle, this can allow it to escape from a state where a block is in the local optimum to the global optimum, but as with the random mutation hill-climber the chance of this is low because of the large Hamming distance between a block of all 0's and a block of all 1's. In figure 3, we show the result of simulated annealing where the stochastic hill-climber tries flipping a randomly chosen variable. If the new configuration has an equal or higher fitness than the current state then the searcher moves to the new state, otherwise it accepts the move with a probability $\exp(\Delta F/T)$ where ΔF is the (negative) change in fitness and T is the annealing temperature (this is just the famous Metropolis algorithm used in Monte Carlo simulations). The annealing temperature was chosen to be $T = 10 \times 0.9999^t$ where t is the number of steps—thus the annealing temperature, and consequently the probability of making a fitness decreasing move, drops slowly towards zero. Again these parameters were chosen after some experimentation. Finding the optimum annealing schedule is hard because there is a lot of flexibility in its choice. For small sized systems, it is possible to build an exact Markov Chain model which then allows the optimum annealing schedule to be found. We describe optimal annealing schedules computed for this problem with m=10 and k=7 in appendix A. For the problem discussed here with $m = 64$ and $k = 7$ it is not feasible to find the optimal annealing schedule, although the results in appendix A suggest that using the optimal annealing schedule is unlikely to substantially improve the performance of simulated annealing over a well chosen exponential schedule like the one shown. As can be seen in figure 3, simulated annealing was the best algorithm we tested after a hybrid-GA.

We also attempted to solve this problem with different variants of genetic algorithms. In each case, we used scaled Boltzmann selection where each member of the population is chosen with a probability proportional to $\exp(\beta\, F/\sigma)$, where F is the fitness of the member of the population, σ is the standard deviation in the fitness of the population, and β is selection parameter. This mechanism was used as it provides easy control of the selection strength in addition to having nice theoretical properties [9]. We also used stochastic universal sampling [10] (as opposed to roulette-wheel sampling) to reduce sampling fluctuations. In the simplest GA we tested, selection with a selection strength of $\beta = 1$ was combined with mutation with a mutation rate of $u = 1/n$. Crossover was not used. The GA without crossover performs little better than hill-climbing.

In addition, we considered a classic GA where, as well as selection and mutation, we also allowed crossover. In this case, we performed crossover on the entire population. Interestingly, uniform crossover was found to perform better than single-point crossover. This may appear surprising as the blocks are contiguous in this problem. Thus single-point crossover is much less disruptive than uniform crossover, however, uniform crossover provides much better mixing [11], [12], which, from the empirical evidence, appears to be more important than being less disruptive. A genetic algorithm using crossover has an advantage over solo-search methods as it has the potential of swapping different parts of the solution. In particular, if different solutions have all 1's blocks at different places, then crossover has the potential of combining these good blocks from solutions to get a child with even more all 1's blocks. However, the population in a GA tends to correlate due to selection. This leads to "fixation" where all the variables across the population, at a particular location, are in the same state. If the variables in a block are fixated in the all 0's state then crossover is unable to reach the all 1's state. For this problem the selection strength has to be relatively high for mutation to move the blocks to a high fitness state, but this increases the likelihood of fixation. Thus, although the GA with crossover has some advantage over hill-climbing on this problem we could not find a set of parameters which allowed it to out-perform simulated annealing.

The final algorithm we tried was a hybrid-GA where, instead of performing mutation, we carried out $n_{hc} = 20$ hill-climbing moves. Since hill-climbing drives the solutions to the local optimum, it is not necessary to use such a strong selection pressure—we used a selection pressure of $\beta = 0.1$. As a consequence there is a much higher diversity in the population. Crossover explores the different combinations of blocks. Again uniform crossover was found to be superior to single-point crossover, although this is, arguably, less surprising as hill-climbing can repair the disruption caused by crossover. The hybrid-GA was able to find the global optimum in almost all runs. Although, a small population size would converge on a good solution quicker than a larger population, it is vulnerable to fixation leading to a sub-optimal state being found. The gain in performance may not at first sight appear so dramatic in figure 3, however, it should be borne in mind that the problem gets increasingly difficult towards the global optimum. By comparison, simulated annealing has a fitness of 10 below the global optimum (i.e. on average it finds a solution with 10 of the 64 blocks in their local optimum state). It would probably take many orders of magnitude more computation time for a solo-search algorithm to achieve the same quality of solution as the hybrid-GA. Furthermore this gap in performance is magnified in larger problems.

A closely related problem to the Concatenated-$\mathcal{V}$ problem, again involving blocks of variables, has been proposed by Watson and analysed in reference [13]. In that problem, a simple GA is proved to substantially out-perform a hill-climber, although in that case preserving linkage through using single-point crossover is essential to achieving the result.

As a toy problem to illustrate the building-block mechanism, the Concatenated-$\mathcal{V}$ problem has the desirable feature of being very simple, however, it is arguably too simple in that it is just a sum of blocks where there is no interaction between the blocks. Any rational solver that spotted this structure would attempt to solve each block as a sub-problem and then solve the full problem by combining the solutions of the sub-problems. Using this strategy, this problem is easy to solve. Furthermore, it would be relatively easy to learn the separable structure of the problem. Even more critically, most optimisation problems that we try to solve in practice do not conveniently split into pieces, but contain long-range, non-linear correlations between all the variables (although, the correlations are not necessarily due to direct interactions, but are often produced through interactions with intermediaries). This raises the question of whether this problem is relevant to real world problems. In the following, we will argue that variants of this problem are possibly good analogues of some real world problems.

B. Critical-Variable Problems

A large class of optimisation problems have the property that there are a few variables that need to be correctly set to get very good results. After the variables associated with these *critical features* are set then a fairly basic algorithm such as a hill-climber can find very good solutions. It is easy to imagine (though harder to prove) that problems such as the travelling salesperson problem (TSP) or graph colouring might fit this picture. In TSP there might be some critical parts of the tour which need to be chosen correctly, thereafter the rest of the tour might be found easily by a greedy algorithm. Similarly, in graph-colouring there may be a few nodes which need to belong to the same colour class. Once these variables have been set then the rest of the colouring problem might become relatively simple. What makes these problems difficult is that we do not know what variables are critical variables.

A common observation that appears to lend support to this interpretation is that, when solving many large hard problems using heuristic algorithms, it can take many minutes or even hours of work to achieve a particular level of fitness. If this state is then perturbed by applying a large macro-mutation the fitness can jump considerably so that the fitness has the same value that was obtained in the first few seconds of optimising the problem. However, on applying a hill-climber we return in a fraction of a second to the same level of fitness that initially took minutes or hours to achieve. We illustrate this with a graph-colouring problem. We consider a random instance of $G(1000, 1/2)$, which are graphs with 1000 nodes, where the probability of each edge is one half. The task is to *minimise* the number of colour-conflicts (i.e. edges whose nodes have the same colour). Empirically, it is found that almost all instances of this class of graphs can be coloured with no colour-conflicts using 83 colours. However, these are typically extremely hard instances needing a lot of time and very powerful algorithms to solve them [14]. In figure 4, we show an iterative local search (consisting of a descent algorithm with occasional kick-starts) minimising the number of colour-conflicts with 90 colours. The best configuration found so far was remembered and the system returned to that state if a kick-start produced worse results. This algorithm was run for 950 seconds. We then perturb the system by changing 50 randomly chosen variable. The number of colour-conflicts jumps from 27 to 303, which is equal to the cost reached well within the first second of optimisation. On applying a descent algorithm the searcher return to 27 colour-conflicts in less than 0.002 seconds (this initially took over 300 seconds to find). One explanation of this

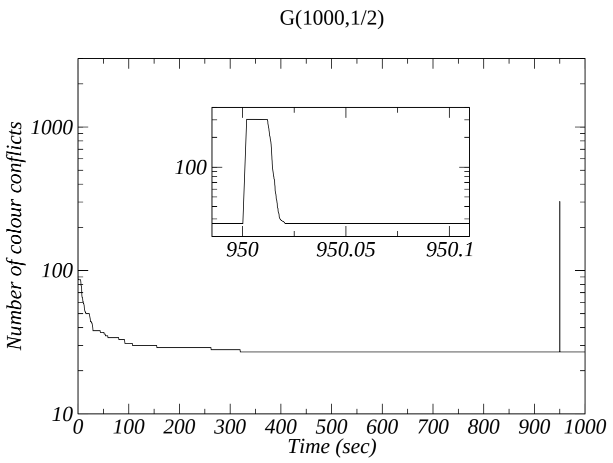

Fig. 4. Number of colour conflicts versus time for an instance of the graph colouring problem $G(1000, 1/2)$ with 90 colours. After 950 seconds a perturbation of 50 variable is made. The system returns to the cost it started with before the perturbation in a fraction of a second. The insert shows the perturbation with the time axis magnified (the plateau at 950 seconds is due to an initial book-keeping stage in the descent algorithm).

behaviour is that it takes a large amount of time to discover a good set of critical variables. The perturbation is unlikely to change the critical variables, as there are relatively few of them. Finally, a descent algorithm is able to find good values for the non-critical variables.

We can easily construct a problem which is closely related to the Concatenated-$\mathcal{V}$ problem which has critical variables. We call this the *Concatenated-critical-variable problem*. In its simplest form it consists of m blocks, each with k-variables, but one of the variables is the critical variable. We can write the fitness function slightly more transparently if we use ± 1 variables, $S_i \in \{-1, 1\}$. The fitness can written as

$$F_{CCV}(\boldsymbol{S}) = \sum_{i=0}^{m-1} F_i(\boldsymbol{S}), \quad F_i(\boldsymbol{S}) = S_{i\,k+1}\left(1 + \sum_{j=2}^{k} S_{i\,k+j}\right).$$

One method for representing problems whose fitness function decomposes is by using a factor graph. These are bipartite graphs with one set of nodes (traditionally drawn as circles) representing variables and the second set of nodes (traditionally drawn as squares) representing the interaction between variables. The factor graph for the Concatenated-critical-variable problem is shown in figure 5. In this problem, the fitness can be viewed as equal to a sum of terms coming from the interaction nodes, which, in turn, is equal to the product of variable nodes connected to the interaction nodes. The critical variables, $S_{i\,k+1}$, interact with all other variables

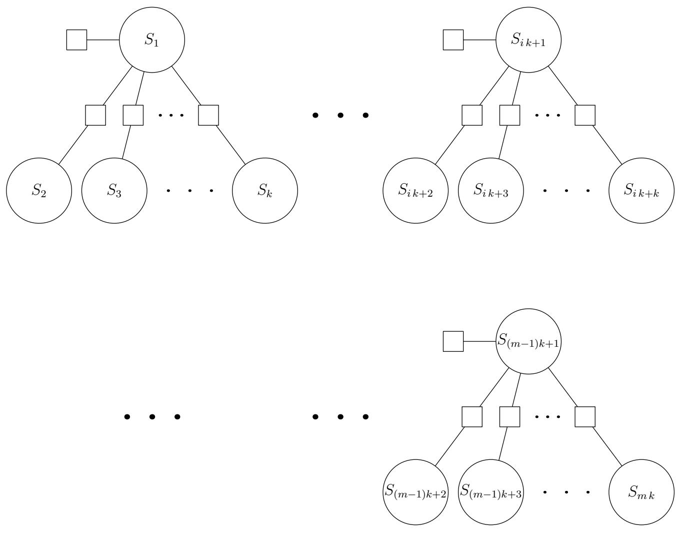

Fig. 5. Factor graph for the Concatenated-critical-variable problem

in the block. We also show an interaction involving only the critical variable term to represent the fitness contribution which is proportional to the value of the critical variable alone. The non-critical or normal variables are optimised when they take the same value as their corresponding critical variable. In a similar manner to the Concatenated-$\mathcal{V}$ function, each block has a global maximum when all the variables are 1's with a block fitness of $F_i = k$, and a local maximum when all the variables are -1's with a block fitness of $F_i = k - 2$. If we fix the critical variables then the normal variables can be optimised by a hill-climber. Heuristic algorithms behave almost identically on this problem as the Concatenated-$\mathcal{V}$ function (we do not show results for this problem, but we show results on a more complex version below). For this problem, we can view the hybrid algorithm as working at two levels. The hill-climber optimises the normal variables, so that the fitness reflects the quality of the critical variables. Selection and crossover effectively explore the space of critical variables, in doing so they disrupt the normal variables, but these are repaired by hill-climbing. This second level, is almost as if we were solving a Onesmax problem with the m critical-variables using a GA with selection and crossover only. Fixation of the variables associated with a particular block in the sub-optimal state can still occur, but this is unlikely in large populations if we use uniform crossover and weak selection. The reason for this is that uniform crossover destroys linkage much faster than weak selection can replicate good solutions, thus preventing hitchhiking.

This problem again suffers from the same criticism that it is completely separable into easier sub-problems, which is not typical of many real world problems—of course, it also illustrates that when a real world problem can be separated then it is almost always beneficial to do so. We can however easily extend the Concatenated-critical-variable problem so that the problem is no longer separable. One way to do this is to introduce some new normal variables that depend on more than one critical variable. In this vein, we propose the *Interacting-critical-variables problem* defined by

$$F_{ICV}(\boldsymbol{S}) = \sum_{i=0}^{m-1} F_i(\boldsymbol{S})$$

$$F_i(\boldsymbol{S}) = S_{i(k+l)+1}\left(1 + \left(\sum_{j=2}^{k} S_{i(k+l)+j}\right)\right.$$
$$\left. + S_{\pi_i(k+l)+1} \sum_{j=k+1}^{k+l} S_{i(k+l)+j}\right).$$

where $\boldsymbol{\pi} = (\pi_0, \pi_2, \ldots, \pi_{m-1})$ is a random permutation of the sequence 1, 2, …, m, such that $\pi_i \neq i$. We show the factor graph for this problem in figure 6. Notice that the only

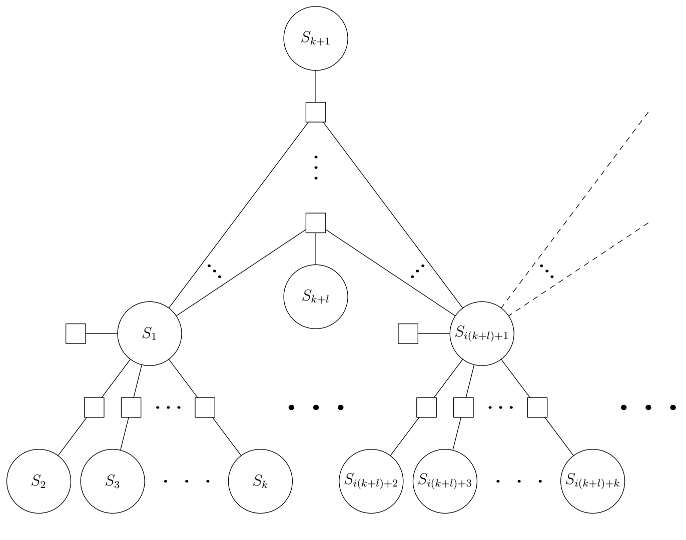

Fig. 6. Part of the factor graph for the Interacting-critical-variables problem. We assume $\pi_0 = i$.

difference between this problem and the Concatenated-critical-variable problem is the inclusion of l extra variables per block which are optimal when they have the same sign as the product of critical variables for the two blocks $S_{i(k+l)+1}\, S_{\pi_i(k+l)+1}$. This introduces a non-epistatic interaction between the different blocks.

In figure 7, we show the performance of different heuristic search algorithms on an instance of this problem with $m = 64$, $k = 7$ and $l = 3$. This is a considerably harder problem than the Concatenated-$\mathcal{V}$ problem we considered earlier, partly because there is an extra three variables per block that need to be optimised, but more crucially because these extra variables makes learning the critical variables more complicated as they are now epistatically linked. Thus for the hybrid-GA to solve this problem with high probability we needed to use a 5-times larger population running for twice as many generations. Once again we have tuned the parameters by hand to obtain good performance for the other algorithms. The ordering of the performance of all algorithms is similar to that for the Concatenated-$\mathcal{V}$ problem.

Another view of critical variables is in terms of the search space. We can think of the search space as a set of points connected by some adjacency graph. The adjacency graph

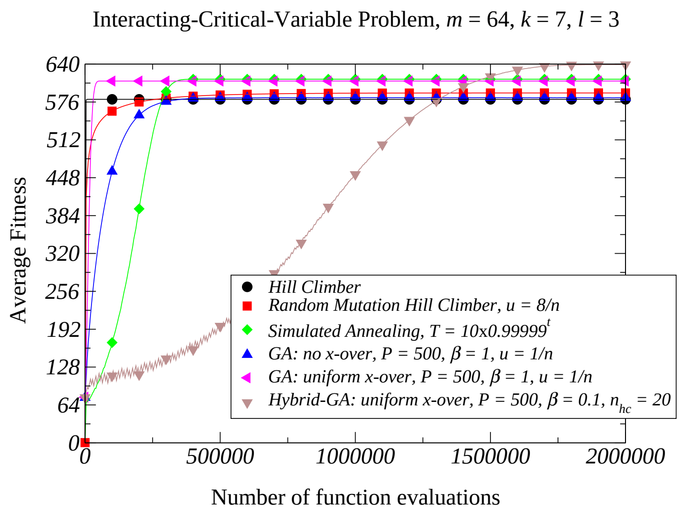

Fig. 7. Mean fitness versus the number of function evaluations for 1000 different runs on an instance of the Interacting-critical-variable problem with $m = 64$, $k = 7$ and $l = 3$.

depends on what configurations we treat as neighbours. For a binary variable problem, where configurations at a Hamming distance of 1 are neighbours the adjacency graph will form a hyper-cube. A large number of optimisation problems arise through imposing a series of constraints each of which divides the search space into those configurations that satisfy the constraint and those which do not. We show a caricature of the search space with one constraint in figure 8. In this picture we have shown the adjacency lattice as a 2-dimensional lattice; in most real problems of interest the search space is much higher dimensional (e.g. an n-dimensional hypercube). In this

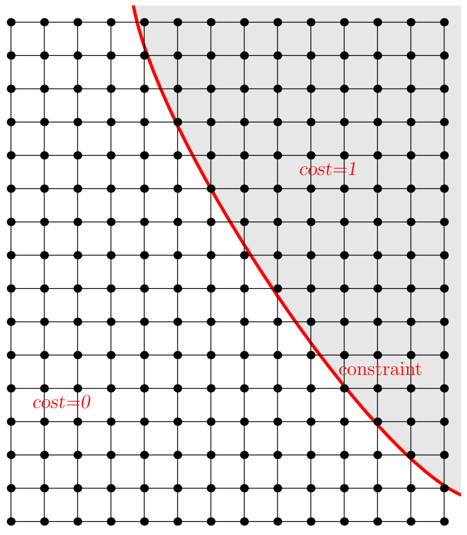

Fig. 8. Caricature of a search space for a problem with one constraint.

problem we assign a cost of 1 if we break the constraint and a cost of 0 otherwise. As we increase the number of constraints the problem, typically, becomes less easy to solve. As more and more constraints are added a stage is reached when there are no longer any points in the search space that satisfy all the constraints. At around this critical number of constraints, it is frequently found that the problem becomes much more difficult to solve because of the formation of a substantial number of local optima. These persist as we increase the number of constraints. As a consequence, many problems show a phase transition between being easy to solve and hard to solve, which happens close to the point where the set of constraints is no longer satisfiable (for decision problems as opposed to optimisation problems, such as deciding if a set of clauses is satisfiable, the instances often become easy to decide above this transition point). In figure 9, we show the cost landscape after adding more constraints—note that, as the cost counts the number of unsatisfied constraints, then to optimise this problem we need to *minimise* the cost. We observe that

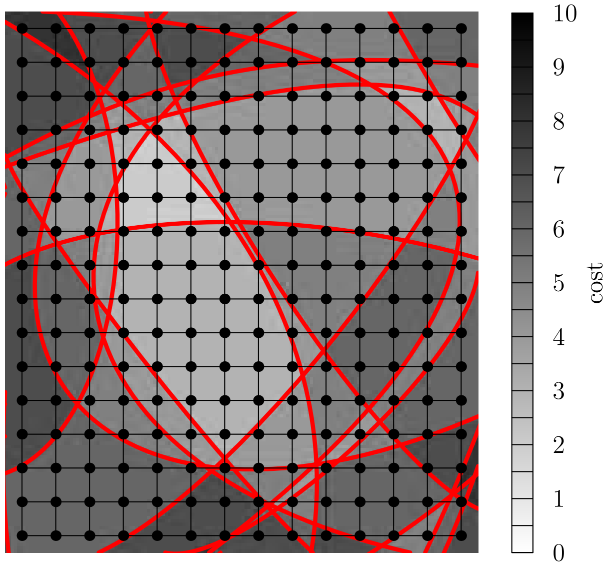

Fig. 9. Caricature of the hard phase after adding more constraints.

by adding more constraints we tend to increase the number of local-minima making the problem hard to solve. For a binary search problem each time we fix a variable we cut the search space in half. The idea of critical variables is that if we set a few well-chosen variables correctly then we might be able to restrict the search to a part of the search space where the global optimum has a large basin of attraction. This is shown schematically in figure 10. In complex problems there may be different sets of critical variables, which, once set, simplify the landscape sufficiently that the problem can be solved using a simple heuristic algorithm.

Some direct empirical evidence for critical variables comes from investigations of SAT, albeit on rather small problems. Small SAT instances were solved using a standard SAT solver. These problems were found to be sufficiently hard that it took a long time for the SAT solver to find a solution which satisfied all the clauses. It was found that, by setting a small number of variables, a SAT solver could solve the problem trivially. These sets of variables had to be found through exhaustive search so that it was only possible to look for these sets of variables in small SAT instances. These sets of variables have been called backdoors in the context of SAT [15], [16].

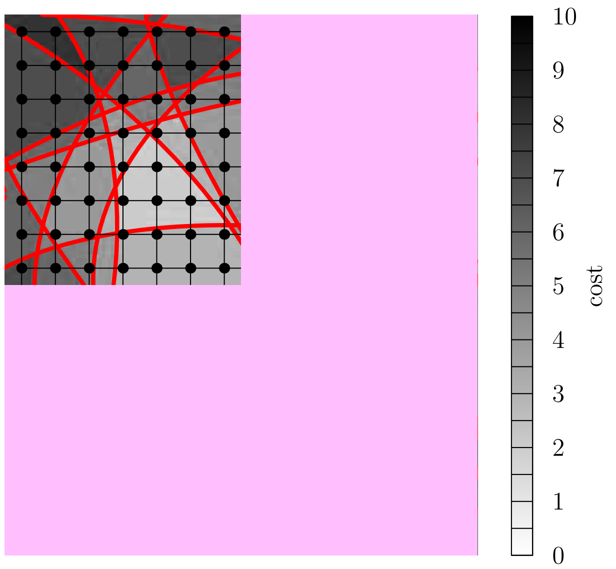

Fig. 10. Schematic illustration of reducing the search space by fixing two of the variable so that the remaining problem becomes easy to solve.

When working with large combinatorial optimisation problems it is much harder to be sure whether they possess critical variables. One candidate problem which might have this property is graph-colouring in dense graphs. This is a problem where the best-known solution method is a hybrid genetic algorithm due to Galinier and Hao [14]. Not only did they introduce a powerful crossover operator, they also introduced a very efficient tabu search as a local search operator. Nevertheless, the quality of the solutions they found was due to the crossover operator [17].

IV. Focusing Search

We now consider a second possible mechanism by which a population can be beneficial. Again this mechanism relies on crossover. Most crossover operators have the property that if both parents share the same value of a variable, then the child will also have the same value for that variable—this property of crossovers has been called *respect* by Radcliffe [18]. As a consequence crossover only explores the part of the search space where individuals disagree. In contrast, mutation explores the entire search space. This focusing of the search by crossover, can dramatically reduce the time needed to find a good solution.

This mechanism has been illustrated previously in a number of toy problems. For example, in the *Basin-with-a-barrier problem* [19], [20], [21] and the closely related *Jump function* [22] and, to some extend, in the *Hurdle problem* [23] (although, in this case, we can also interpret the success of GAs as, at least partly, due to it acting as a low pass filter). Here we consider a variant of the Basin-with-a-barrier problem, which we will call the *Iceberg problem*. We represent this problem by a binary string $\boldsymbol{X} = (X_1, X_2, \ldots, X_n)$ where $X_i \in \{0, 1\}$. The fitness is taken to be a function of the number of 1's in the string $F_{Iceberg}(\boldsymbol{X}) = g(\sum_{i=1}^{n} X_i)$ where

$$g(k) = \begin{cases} k & \text{if } 0 \geq k < \frac{8\,n}{10} \\ \frac{8\,n}{10} & \text{if } \frac{8\,n}{10} \leq k \leq \frac{9\,n}{10} \\ k - \frac{n}{10} & \text{if } \frac{9\,n}{10} \geq k \geq n \end{cases} .$$

This function is shown in figure 11. This diagram does not,

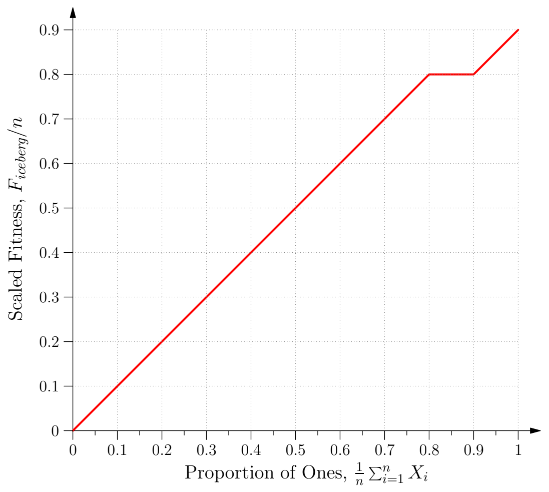

Fig. 11. Fitness function of the Iceberg problem as a function of the proportion of 1's in the string.

however, show the number of configurations with a particular number of 1's. This is given by the binomial function $\binom{n}{k}$ where k is the number of ones in the string. Figure 12 shows the logarithm of the number of states, $N(y)$, divided by n in the large n limit, where $y = k/n$ is the proportion of 1's in the string. For large n, almost all strings have an equal number

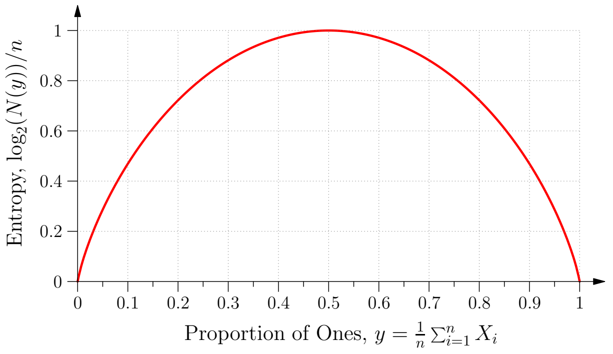

Fig. 12. The entropy (logarithm (to the base 2) of the number of configurations) per variable is shown as a function of the proportion of 1's, in the limit $n \longrightarrow \infty$.

of 1's and 0's. As we approach the global maximum there is a very strong fall-off in the number of states (or entropy). Because of the large decrease in entropy, the configurations with fitness above $0.9n$ look like a small Iceberg in a large ocean containing strings with 80–90% of the variables equal to 1.

A hill-climber will, with high probability, start in a state with approximately equal number of 1's and 0's. It will then rapidly reach the state where 80% of the variables are 1's. It could carry on and reach the state where 90% of variables are 1's, however, this is extremely unlikely because it would require the vast majority of hill-climbing moves to change 0's to 1's, however, there are many more 1's than 0's so most of the time a random move will change a 1 to a 0 and take the solution back to edge of the ocean. The first-passage time to reach the global optimum will consequently scale exponentially with the problem size, n. Figure 13 shows the logarithm of the expected first-passage time versus the system size. The expected first-passage time for a hill-climber was computed exactly by solving a Markov chain model using the techniques described in appendix B.

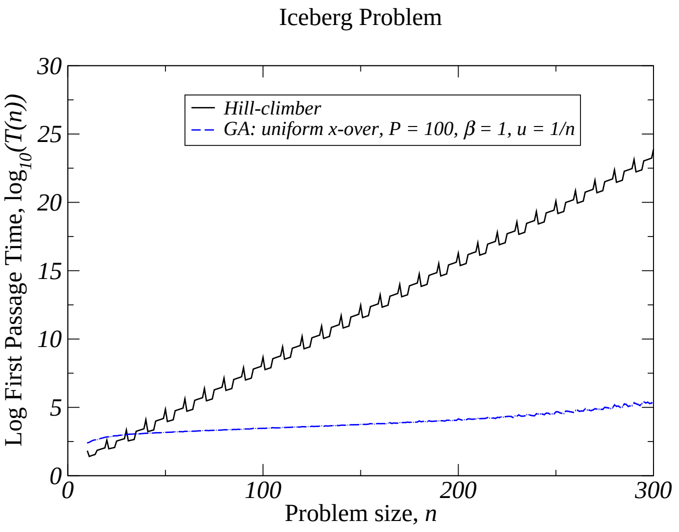

Fig. 13. Logarithm of the mean first-passage time versus size for the Iceberg problem. $T(n)$ counts the mean number of function evaluations needed to solve the problem. The results for the GA are averaged over 1000 runs, while those for the hill-climber are exact.

We also measured the first-passage time empirically for a standard genetic algorithm with a population of size 100 using Boltzmann selection ($\beta = 1$), uniform crossover, and a mutation rate of $1/n$—no attempt was made to fine tune these parameters. The results are also shown in figure 13. The first-passage time appears to grow exponentially which is what we would expect (since the ratio of the size of the iceberg to the ocean becomes exponentially small in the size of the problem), nevertheless, we see that a standard genetic algorithm allows us to solve reasonably large problem sizes where a hill-climber would take far too long. For this problem a random mutation hill-climber, simulated annealing or a GA without crossover all suffer from the same problem as hill-climbing. Namely, that almost all mutations take the searcher away from the all 1's state. The good performance of the GA arises because crossover focuses the search. An analysis of a closely related problem is given in [20], [21]. The reason why the GA is out-performed by the hill-climber for small problem sizes is simply because of the linear cost of carrying a population described in section II. Had we chosen a smaller population size we would be able to beat the hill-climber on smaller-sized problems as well.

This toy problem is open to the criticism that even the GA solves this problem in exponential time, which is often regarded as impractical. However, in the example shown the exponent is rather small so a GA provides a practical solution even for fairly large problem sizes. We could also modify the problem so that the width of the ocean is fixed rather than growing with n. This brings the problem closer to the Hurdle problem introduced in [23] and described in section V, where the run times for all algorithms are polynomial rather than exponential.

A. Strong Focusing

There is another approach to solving the Iceberg problem which is dramatically more efficient than even the GA. That is, to run a population of hill-climbers for a fixed number of steps and then average the variables at each site in the string. This is then rounded to zero or one. Given a reasonable sized population (say 100) then with overwhelming probability the resulting string will be the all 1's string. Averaging causes strong focusing of the search. It has not been used much, perhaps because it breaks the analogy with natural selection. It focuses the solutions so strongly that it is only useful to apply this operator once.

The success of averaging depends on a very specific geometry of the solution space. That is, the high-fitness regions of the search space has to lie directly in the centre of a larger region containing good quality solutions. This very particular landscape might appear slightly artificial and unlikely to occur in real world problems, however, in a recent paper this strategy was found to be very effective on one of the classic combinatorial optimisation problems, namely MAX-3-SAT [24].

The MAX-SAT problem consists of a set of m clauses constructed from n Boolean variable. Each clause is a disjunction of literals (a literal being either a variable or a negation of a variable). The optimisation problem is to maximise the number of satisfied clauses (i.e. clauses that are true). In MAX-3-SAT all clauses consist of three literals (this paper uses the convention that these literals involve different variables and all clauses are different). The paper studied the case of randomly generated MAX-3-SAT instances where the ratio of clauses to variables was 8. This is believed to be well within the hard phase.

For averaging to provide a performance advantage, the only modification made was to run a hill-climber on the solution found after performing the average. An even more successful strategy was to run multiple hill-climbers and then to cluster the resulting solutions based on their Hamming distances to each other. This was done using k-means clustering. The centroid of each cluster was clipped to the closest vector of 0/1 variables and used as a starting point for another round of hill-climbing. This was found to substantially out-perform hill-climbing on large randomly generated instances of MAX-3-SAT. We illustrate the performance of a hill-climber, hybrid-GA, and the k-means clustering algorithm in figure 14 with a graph taken from reference [24]. Note that we plot the number of unsatisfied clauses rather than the number of satisfied clauses, thus lower values are better. Ten hill-climbers were used in parallel and the best of them is plotted (this gives better performance than running a single hill-climber ten times longer). The hybrid-GA uses a population of size 10, the selection rate and number of hill-climbs between selection and crossover was carefully chosen. In the k-means clustering an initial population of 100 hill-climbers was run for 27 000 steps. At this stage k-means clustering with $k = 10$ was carried out and 10 hill-climbers were restarted from the feasible solutions closest to the 10 centroids. The jump in number of satisfied clauses at around 10 seconds shown in figure 14 is caused by the k-means clustering step. For larger-sized instances the performance of k-means clustering became even more pronounced. Using this algorithm the authors obtained superior results on large randomly chosen instances to all other local search algorithms they tested against. Further details of these results are given in reference [24].

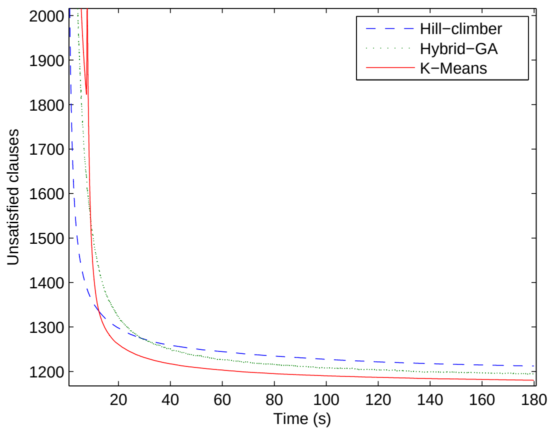

Fig. 14. Performance of a hill-climber, hybrid-GA and k-means clustering algorithm on random MAX-3-SAT instances with 6000 variables and 8 clauses per variables. The results were averaged over 100 instances.

The argument put forward in reference [24], which was supported by a number of empirical observations, was that there exists a small number of global optima some of which were well separated (e.g. having a Hamming distance of 40% of the variables). Around each global optima were good quality local optima. A caricature of the fitness landscape is to think of the search space as a world, where the height of each point represents its fitness. A cartoon of this search space is shown in figure 15. Good quality solutions lie in mountain ranges which are weakly correlated, so they do not exist in completely random places in the world, but tend to lie in one hemisphere. The number of foothills out-number the mountains so that a hill-climber will find a foothill with a high probability. Figure 16 illustrates schematically how the k-means algorithm might work. Clustering picks out those solutions that lie in foothills surrounding a mountain range. The centroid of these solutions would then lie in the mountain range. As the landscape is rugged the centroid might not be a high quality solution, however, by hill-climbing we quickly find a good quality solution. In this way the population learns about the large-scale features of the fitness landscape.

These ideas can be generalised to other problems. We can

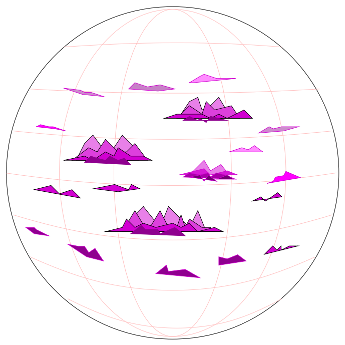

Fig. 15. Cartoon depiction of the fitness landscape of MAX-3-SAT.

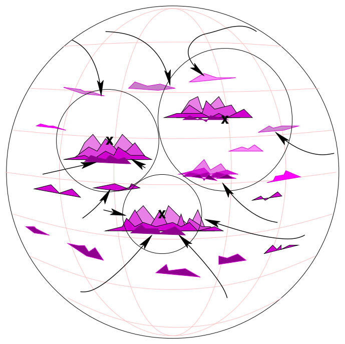

Fig. 16. Schematic illustration of the k-means clustering algorithm. Independent hill-climbers find solutions in the foothills of mountainous (i.e. high fitness) regions. The k-means clustering algorithm groups solutions which predominantly lie in the foothills of the mountain range, so that the centroids lie close to high quality solutions. Performing hill-climbing starting from the nearest feasible solution to the centroids allows good quality solutions to be found rapidly.

define the average or centroid of a set of configurations to be the configuration with the smallest average distance to all the members in the set. There is then some flexibility in how we choose to measure distance. In reference [24] it was shown that moving to the centroid of a set of solutions obtained through hill-climbing also appears to be useful in other NP-hard problems such as graph-colouring and the Ising perceptron.

V. Low-Pass Filtering

A third mechanism by which a population can be beneficial is by ignoring distractions in the landscape. That is, population-based algorithms can behave as if they ignore short-length scale features in the landscape. Populations acting as a low-pass filter has been proposed previously based on a mechanical model of a genetic algorithm [25]. This robustness to distractions comes from the averaging effect of having a large population and the fact that at each generation the individuals tend to move so that over a few generations sub-populations respond to their average fitness. As an illustration of this mechanism in operation we consider the *Noisy-Onesmax problem* where we add a *static* random noise to the fitness so that

$$F_{NO}(\boldsymbol{X}) = \text{round}\left(F_{Onesmax}(\boldsymbol{X}) + \sqrt{n}\,\text{hash}(\boldsymbol{X})\right),$$

where ‘$\mathrm{round}(y)$’ returns the integer closest to y, $F_{Onesmax}(\boldsymbol{X})$ is the fitness function for Onesmax and ‘$\mathrm{hash}(\boldsymbol{X})$’ is a function that returns an independently chosen Gaussian random deviate for each $\boldsymbol{X}$ with zero mean and unit variance. Note, that this is a static fitness function, in that it returns the same fitness value if it is called with the same binary string.

The landscape is a Onesmax landscape with a random function added on top. We note that the contributions of these terms differ considerably over different (Hamming) distance scales. The contribution to the fitness from the Onesmax function changes by 1 for Hamming neighbours, while the contribution from the random noise term produces changes of order $\sqrt{n}$. By contrast a change in the number of 1’s of order n causes a change in the Onesmax function of order n, while the random noise term still only causes a change of order $\sqrt{n}$. Thus the noise function can be viewed as producing short range variation while the Onesmax function produces long range variations in the Hamming space. We argue below that a genetic algorithm averages out the short range noise, but picks up the long range trends. In this sense, we view this as a kind of low-pass filtering. Technically, however, assuming a Hamming neighbourhood the search space has the topology of a hyper-cube and there is no high-frequency component in any direction. Thus, the idea of filtering should not be taken too literally. A more accurate description is that a population can averages out short range noise while still being sensitive to long range trends.

As with the previous problems we have empirically measured the performance of a number of algorithms attempting to solve this problem. The results are shown in figure 17. All

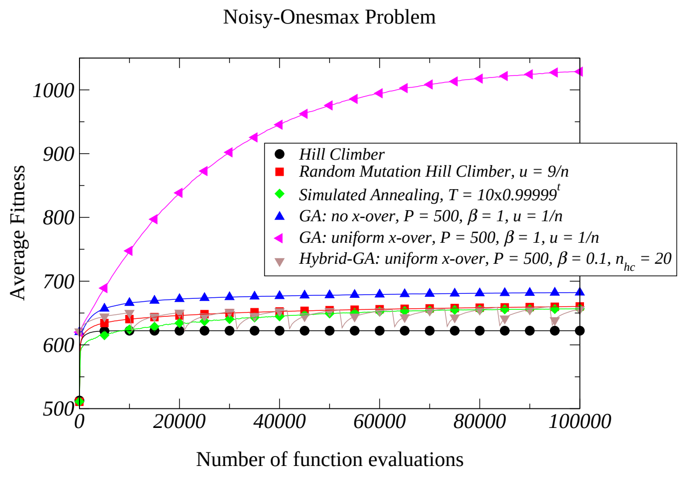

Fig. 17. Empirical measured performance of algorithms on the noisy-Onesmax problem with $n = 1024$. The results are averaged over 1000 runs.

solo-search algorithm quickly find a nearby local optimum where they become effectively trapped. A hybrid-GA performs no better than a random mutation hill-climber or simulated annealing. Using an averaging operator as discussed in the previous section does not help much either since there is not a significant bias in the number of 1’s of the solution found by the hill-climbers (results are not shown in figure 17). The best algorithms for solving this problem are the traditional genetic algorithms and this is particularly true for a GA using crossover. The reason for this substantial improvement in performance is that the GA does not locally optimise the fitness function because mutation and crossover are always changing the configuration. As a consequence the GA is learning the average fitness as a function of the Hamming distance, and therefore it is effectively ignoring the static noise. It is almost as if it is solving a Onesmax problem. In fact, the GA would perform almost identically if the noise being added was not static, but changed at each function evaluation.

The significant difference in performance between the GA with crossover compared to the GA without crossover is probably due to the focusing effect of crossover. That is, crossover is much more likely to create a child with an increased number of 1’s compared with mutation. This illustrates an important point that more than one of the mechanisms described in this paper might be in operation on a particular problem. Another problem which is solved effectively by a combination of focusing and ignoring distractions in the *Hurdle problem* proposed in reference [23]. As with the Iceberg problem, the fitness function is taken to be a function of the number of 1’s in a binary string $F_{Hurdle}(\boldsymbol{X}) = h(\sum_{i=1}^{n} X_i)$ where

$$h(k) = k - 2 \left[\!\left[n - k \text{ is odd} \right]\!\right] ,$$

and $[\![\textit{predicate}]\!]$ is Iverson’s notation for an indicator function, which is equal to 1 if the *predicate* is true and 0 otherwise. The Hurdle function is shown in figure 18. Again this is a problem where a GA does extremely well, particularly when using crossover. It was shown in reference [23] that the run time performance of a GA with crossover on the Hurdle problem was very similar to that of the same algorithm working on the Onesmax problem with added (non-static) noise.

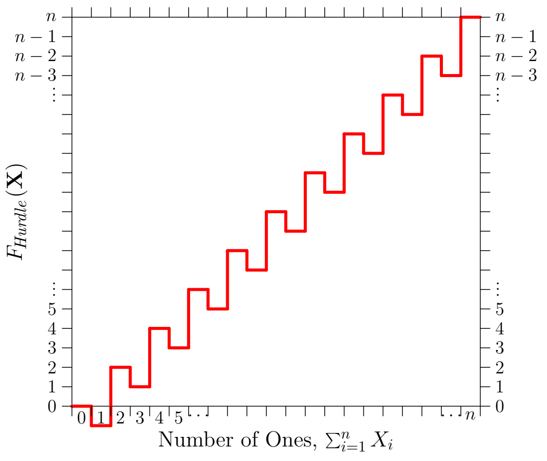

Fig. 18. The fitness function of the Hurdle problem as a function of the number of ones in the string.

VI. Hedging

In the last two decades many algorithms have been proposed based on parallelising a state-of-the-art solo-algorithm using a population [26], [27], [28], [29], [30]. These “Hybrid-GAs” have often been found to out-perform the solo-algorithm, with crossover often playing an unimportant role. One plausible explanation for the improvement is that by running multiple times they hedge against bad luck in their starting position or the choices they make. This provides the basis of our fourth mechanism which exploits the variation in the performance of an algorithm depending on the part of the landscape it is currently searching. In its simplest form we can imagine a problem with many local optima of different quality. A hill-climber will get stuck in a particular local optimum so given enough time there is clearly an advantage to running multiple hill-climbers and choosing the best. For many algorithms such as simulated annealing and random mutation hill-climbers that can escape from local optima, it can still be advantageous to run these algorithms in parallel as the success of the algorithm may depend strongly on the starting position. Thus, a population might provide an advantage by hedging against being unlucky. This is a mechanism that does not depend on crossover. There is clearly an exploration-exploitation balance between the cost of using a population as described in section II and the benefit it might provide by having a member of the population in a better part of the search space.

This hedging mechanism might appear rather trivial and unimportant, however, it can take on rather subtle forms. It can also potentially lead to a dramatic speed-up in performance. We illustrate this in the following example. For many optimisation problems, a search algorithm often needs to be able to make large steps from a local optimum to another optimum with a higher fitness value. A fitness landscape with a global optimum and a number of local optima is shown schematically in figure 19 (we will assume in the following that we are solving a discrete optimisation problem where the fitness takes integer values only). We consider a hill-climbing algorithm which escapes from a local maximum by systematically searching all neighbours of increasing distance from itself until it finds a fitter solution. Suppose this algorithm has reached one of the local maxima with fitness $F_{max}-1$. We will call such a local maximum a sub-global maximum. The algorithm now has to move from the sub-global maximum it is in to the global optimum. By definition, there are no configurations of higher fitness that lie between the sub-global and global maxima, thus there is no heuristic information that the algorithm can easily exploit. As a consequence, the time it will take to make this step will scale with the number of configurations in a ball around the sub-global maximum whose radius is the distance to the global optimum. In a problem with n binary variables the number of configuration within a Hamming ball of radius h is

$$N(h) = \sum_{i=0}^{h} \binom{n}{i}.$$

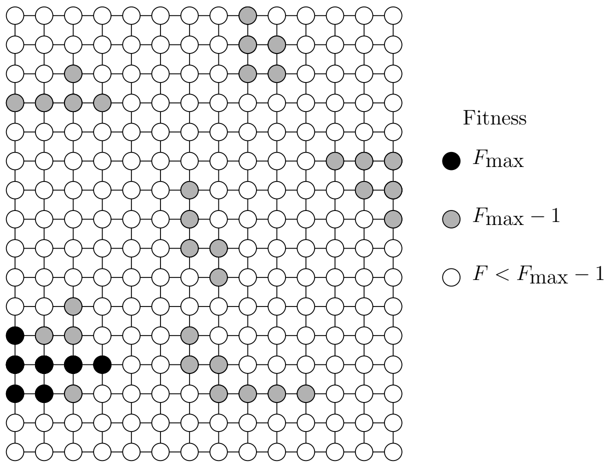

Fig. 19. Cartoon of the fitness landscape showing the global maximum configurations and the configuration with the next highest fitness. Configurations with any other fitness are shown as white circles. In this example there is a single global maximum and five sub-global maxima.

This is one aspect where the cartoon landscape depicted in figure 19 differs dramatically from the landscape of high-dimensional problems, in that it misrepresents the number of configurations within a small Hamming distance of a particular configuration. Figure 20 shows the number of configurations within different Hamming distances for a binary problem of size 100. Notice that there is a dramatic increase in the number of configurations as a function of the radius. As a consequence the exhaustive hill-climber described above will take dramatically different times to find the global optimum depending on the Hamming distance between the sub-global and global optimum. If the probability of our algorithm reaching any sub-global maxima is the same for them all, then the median number of function evaluations need to make the last step and reach the global optimum will be $N(\bar{h})$ where $\bar{h}$ is the median distance between the sub-global and global maxima, and $N(h)$ is the number of configurations in a Hamming-ball of size h. If we have a population of P searchers using the same algorithm, but at different sub-global maxima, then the number of function evaluations until the global optimum is reached will be $P\,N(h_{min})$ where h_{min} is the smallest Hamming distance between a member of the population and the global optimum. Thus, if there is some variation in this Hamming distance then the expected time for a population to find the global optimum will be much smaller than that for a solo-searcher.

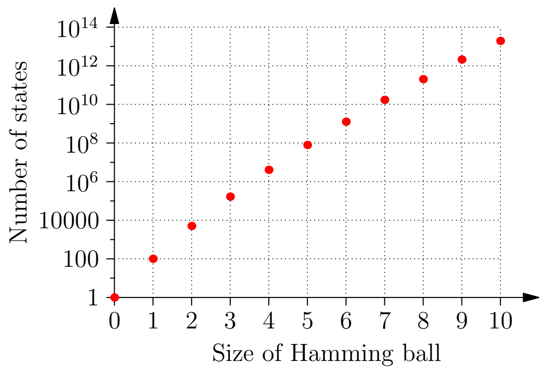

Fig. 20. Shows the number of configurations within a Hamming-ball of different radii for a binary problem with 100 variables. Note that we are using a logarithmic scale to measure the number of states.

To make the argument above concrete we concentrated on an exhaustive hill-climber that systematically expanded its area of search. However, the same argument could equally well be made for a random mutation hill-climber or for a stochastic hill-climber as the expected time to find a better solution is likely to grow, at least, as much as the number of configurations in a Hamming ball of size h, where h is the Hamming distance to a better solution. In most real optimisation problems it is likely that an algorithm will have to make multiple jumps between local optima of increasing fitness. In the above scenario the gain in performance came from the fact that it is necessary only to wait for the first member of the population to make an improvement. However, if the whole population moved to the best solution then diversity would be lost and a population would no longer appear to have an advantage. There may still be an advantage for a population as, in many problems, the number of good quality solutions tends to decrease rapidly as the fitness increases towards the maximum fitness. Consequently it will always be easier to find solutions of the same fitness than a fitter solution. Thus a population will tend to diversify as new local maxima of the same fitness are discovered. In figure 19, we can imagine it would be easier for a population starting at a sub-optimal maximum, say in the top right-hand corner, to diversify to other sub-optimal maxima rather than immediately find the global maximum.

This mechanism depends on a particular structure of the landscape, although it seems plausible that some problems might have these properties, an important question is whether these properties are common? As a partial answer to this question we have studied the structure of MAX-3-SAT empirically for problems up to 100 variables. To do this we have run a modified hill-climbing algorithm adapted to exhaustively search all neighbouring configuration and their neighbours at the current cost, until either a better solution is found or all neighbouring configurations are searched. This guaranteed that a local fitness maximum is reached. For each MAX-3-SAT instance we performed 10 000 such hill-climbs. For the sizes of problems we have investigated we believe that we are able to find the global-maximum solutions. The reason for this belief is that we found all maximum-cost solutions multiple times—on average 250 times and in a sample of 100 instances with a worst case of 23 times. Given that the basins of attractions tend to increase with higher fitness it seems unlikely that we would have missed a optimum solution with higher fitness. Even the second highest solution we found on average around 125 times. In fact, as an empirical observation it appears that on average the basin of attraction of the local optima approximately halves each time the fitness (i.e. the number of satisfied clauses) decreases by one. The reason why solving MAX-3-SAT becomes difficult is that, as the fitness decreases from the maximum, the number of local maxima increases more quickly than the decease in the size of the basins of attraction. Thus for random MAX-3-SAT instances with $n = 100$ variables and $m = 8\,n$ clauses, the mean number of global optima is around 3, the mean number of local optima with fitness of 1 less than the global optima is approximately 11, while for fitness of 2 less than the global optima there are approximately 90 local optima on average, etc. These mean numbers vary considerably between instances. On average each local optima consists of around 20 configurations of the same cost that are accessible by changing a single variable at a time.

Figure 21 shows a histogram of the minimum Hamming distance between every configuration in the sub-global maxima and a configuration of at least the same fitness in the basin of attraction of the global maxima. This is computed for a single randomly generated instance of MAX-3-SAT so as not to confound the variation due to different sub-global maxima with variations between instances. This instance had 2 global maxima and 5 sub-global maxima, with each optima having a large number of configurations associated with it. As can be seen in this example there is a very large spread of minimum Hamming distances suggesting that a population may have a considerable advantage compared with a solo-searcher. Although there is a lot of variation between different instances, nevertheless a big variation in the minimum Hamming distance between sub-global and global optima is typical. The purpose

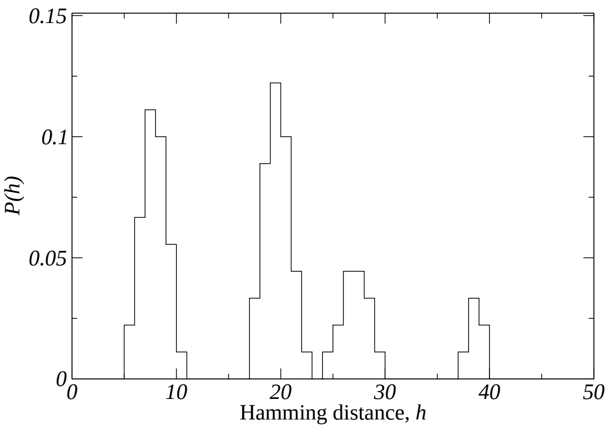

Fig. 21. Histogram of minimum Hamming distance between configurations in the local and global maxima for a randomly generated instance of MAX-3-SAT with 100 variables and 800 clauses.

of figure 21 is to illustrate that, at least in random MAX-3-SAT problems, there is a very significant variation in the distance between the sub-global and global maxima. Thus for any algorithm which attempts to find a global maximum from a sub-global maximum there will be a huge difference in the expected run time depending on which sub-global maximum the algorithm starts from. If by using a population there is more chance of, at least, one individual being at a sub-global maximum closer to a global maximum then the gain in expected run time can easily compensate for the cost of using a population as described in section II.

Although this mechanism provides a plausible explanation of why running an algorithm as a population might be advantageous, it is less obvious whether there is a significant advantage of using a population as opposed to running the algorithms multiple times sequentially. For the population to be advantageous it must explore a larger proportion of the search space than a solo-search algorithm would. A population of independent searchers with no interaction would achieve the greatest diversity, however, such a population would gain no benefit from hedging during the run. By introducing some selection it is possible to concentrate the search on parts of the search space that appear to be more productive, however in doing so it correlates the population. In consequence, this mechanism relies on a fine balance between exploiting the best solution found in the population and maintaining diversity to increase the likelihood of "being lucky" (i.e. being at a location close to an improving solution).

VII. Parameter Tuning

The final mechanism by which a population might afford an advantage over a solo-searcher is by using the population to learn about good parameters of the algorithm. This is rarely done explicitly although there are some notable exceptions.

Choosing good parameters is well studied within simulated annealing where the performance of the algorithm is often found to depend critically on the annealing schedule—that is, the functional behaviour of the "temperature" versus step number (see appendix A for more details on optimal annealing schedules). The temperature governs the probability of making a move that reduces the fitness; it thus controls the degree of exploration versus exploitation. Usually, the temperature is started quite high allowing a lot of exploration and reduced slowly over time to ensure at the end that the searcher finishes close to a (local) optimum. In many problems it is found that the dynamics of the Monte Carlo algorithm used in simulated annealing undergoes a phase transition as the temperature is reduced beyond a critical "freezing temperature". Above the freezing temperature there is a high probability of the Monte Carlo algorithm accepting a move, while below the freezing temperature the searcher gets trapped in a local optimum with an exponentially small probability of escaping. It is found that a good annealing schedule for many problems involves setting the temperature to just above the freezing temperature [31], [32]. Many heuristics have been developed within the simulated annealing community to choose good annealing schedules based on the performance of the searcher. A population might afford a more reliable means of determining the annealing temperature and more generally of balancing exploration versus exploitation, however, I am not aware of this being done explicitly.

The most notable situation where the parameters of a population-based algorithm are learnt is in continuous optimisation. This is an application where it is essential for any competitive algorithm to learn an appropriate step length to search the landscape. A number of evolutionary strategies use the population to learn this step size. A transparent example of this is the Covariance Matrix Adaptation Evolution Strategy (CMA) [33] which uses a population to learn an approximation to the Hessian describing the local curvature of the fitness landscape.

In applications of genetic algorithms to combinatorial optimisation problems, the most prominent attempts to explicitly learn parameter values from the population is through evolving the parameters of the search at the same time as evolving the solutions. Although there is considerable work in this area it is an idea that has not taken off, presumably because in general the benefits of doing this do not sufficiently compensate for the extra complication in the implementation. This lack of success may suggest that using the statistics of the population to balance exploitation and exploration is difficult.

There is, however, one way in which a genetic algorithm using crossover implicitly provides a slow shift from exploration to exploitation. This is caused by the diversity of the population regulating the step size produced by crossover. This arises because the distance between a child and its parents in crossover reduces as the parents become more alike. Selection causes an inevitable correlation as members of the population are replicated. As a consequence of this reduction in diversity, crossover becomes less exploratory. Thus, loss of diversity causes a reduction in the search area, which can be interpreted as an annealing of the search operators. This loss of diversity is often beneficial.

This may seem counter-intuitive—in crossover based evolutionary algorithms diversity is often regarded as purely beneficial. After all, if there was no diversity then crossover would just replicate members already in the population. However, this view is an over simplification. A completely diverse population would contain random strings, and crossover would just produce new random strings. Furthermore in many problems there exist very different arrangements of the variables that correspond to good solutions (sometimes this is due to symmetries in the problem, although, it can also arise from a spontaneous shattering of the fitness landscape as depicted in figure 9). In such situations a loss of diversity might be essential to confine the population to a region of the search space with one dominant optimum. This has been discussed at length in reference [34]. Thus, regulating the diversity may have a significant impact on the performance of a search algorithm using crossover. For example, in Onesmax, starting from random strings, it makes sense to make reasonable large steps at the beginning of the search; crossover will naturally do this as the members of the population are far apart in Hamming distance. Later on, when the strings have a high proportion of 1's, large steps are much more likely to be disruptive. However, the population is likely to have converged so that crossover will produce smaller jumps.

It is even conceivable that, on some problems, the diversity of the population is benignly related to the roughness of the landscape. If the population is so diverse that it covers a region with very different fitness values, then crossover will lead to an offspring population with a large variation in fitness. Selection is then likely to strongly reduce diversity. On the other hand, if the population strongly converges on a region with little variation in fitness (e.g. a plateau region) then there would be little differential selection and the diversity of the population is likely to grow through mutation. As a consequence knowledge about the roughness of the landscape encoded in the population might be beneficially determining the amount of exploration produced by crossover.

To illustrate that this may be happening we consider a final toy problem; we call the Multi-Step Iceberg Problem. The fitness is again a function of the number of ones, but now this function consists of a set of steps. This is illustrated in figure 22. As we can see this consists of a series of

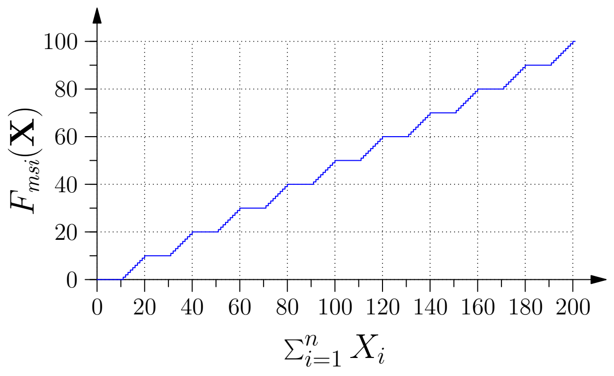

Fig. 22. Fitness versus the number of ones for the Multi-Step Iceberg Problem.

plateaux. Crossing the plateaux become harder towards the extremes because of the entropy gradient (see figure 12). In figure 23, we show empirical results for a GA using crossover, mutation (with mutation rate $1/n$) and Boltzmann selection. For comparison purposes we show results for a simple hill-climber (these are exact results obtained using a Markov Chain analysis). The first-passage time for the hill-climber on this problem is 1.705×10^{12} function evaluation.

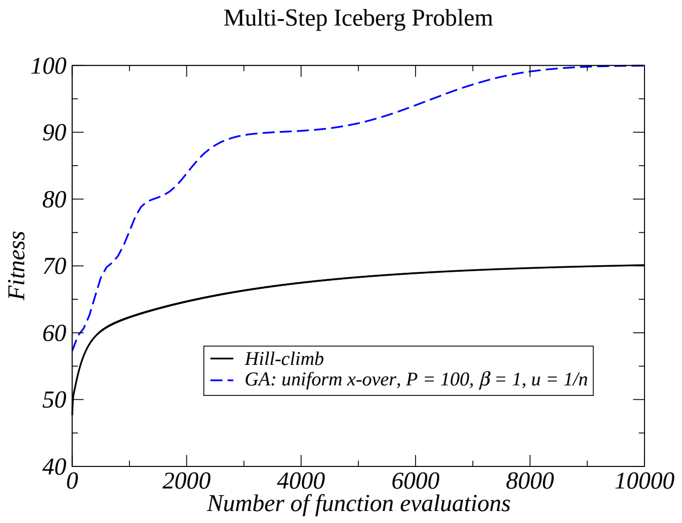

Fig. 23. The expected fitness as a function of the number of fitness evaluations is shown for a GA and a hill-climber on the multi-step iceberg problem. The GA curve is estimated from 1000 independent runs of the algorithm.

In figure 24, we show the correlation between members of the population as a function of the number of 1's for the GA. A correlation of 1 means that all strings in the population are identical, while a correlation of 0 means that the strings differ at 50% of the sites. This data is taken from the same set of simulations shown in figure 23. We also show the expected correlation between random strings with a fixed number of 1's. We notice, that when the population reaches a plateau its correlation starts to decrease, indicating a diversification of the population. This is particularly true for the last plateau which is by far the hardest to cross. In this case the correlation almost reaches that expected of random strings with the same mean number of 1's. This diversification allows crossover to make much larger jumps. As we argued in section IV on focusing, the crossover operator is much more effective at crossing this plateau than mutation alone. In contrast, for the non-plateau sections of the search mutation is effective and there is no particular advantage of having a large diversity. Thus, at least in this problem, the population seems to be benignly controlling the diversity.

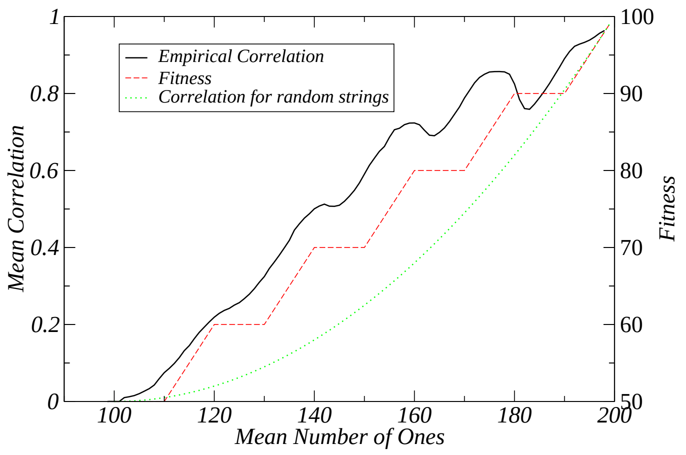

Fig. 24. The correlation between members of the population averaged over 1000 independent runs is shown as a function of the number of 1's. Also shown is the fitness value, and the expected correlation between random strings with a fixed number of 1's.

This final benefit of a population is rather speculative. It relies on a careful balance between selection, mutation and crossover which might be difficult to achieve in more realistic problems. Furthermore, in the argument above we considered crossover as a macro-mutation whose mutation rate is coupled with the diversity of the population, however, crossover is different from a macro-mutation as the focusing mechanism discussed in section IV makes clear. It might be hard empirically to differentiate the benefits due to focusing from those caused by changing the effective step size. Thus, although a plausible mechanism, there is little evidence, to the author's knowledge, showing this mechanism in operation.

VIII. Conclusions

To clarify the contribution of this paper I make a bold (and possibly fallacious) hypothesis with a few qualification. Namely, that *in any real world problem where a generic population-based algorithm substantially out-performs all solo-search algorithms, the advantaged enjoyed by the population-based algorithm is attributable to one of the five mechanisms outlined above*. Clearly this contention can be refuted by coming up with another *natural* toy problem, where a population-based algorithm beats a conventional algorithm, that does not fit any of the five mechanisms. I accept there are already toy problems with carefully designed fitness landscapes that demonstrably show an advantage for populations (e.g. [1], [2]), however, I contend these are not natural (e.g. they have to be carefully tuned and often have unrealistic features such as fitness values that vary in size exponentially in the size of the system). My belief is that problems with these features occur so rarely that they can be neglected—this, of course, is a second conjecture based entirely on personal prejudice. I would therefore reject these models as unnatural or not generic, although I must concede that this is a subjective judgement. I would also accept that there are likely to be problems with very well defined structure where a very specific population-based algorithm may be beneficial and where the benefit comes from a different mechanism to that described here. However, such cases would only refute my hypothesis if the population-based algorithm was generic (i.e. not using very specialised operators). I would argue, in the toy problems given in this paper, the population-based algorithms used natural operators and were not highly tuned. My final proviso is that the problem is a classic combinatorial optimisation problem and not, for example, a multi-objective optimisation problem where there may be additional benefits to populations in describing the Pareto front, etc.

In the majority of cases the benefit comes from the use of crossover, although that is not the case in hedging. It is also possible that a population could perform some kind of low-pass filtering or parameter tuning without crossover, although the benefits seem to be magnified when crossover is used. It should also be noted that problems such as the Iceberg problem, or Hurdle problem are rather artificial in that crossover tends to have little cost on average. In more realistic problems, crossover is often so disruptive that it comes at a considerable cost. To compensate for the cost of crossover it is often necessary to use a hybrid-GA where the disruption caused by crossover can be quickly repaired.

These mechanisms provide a somewhat different view of how population-based algorithms work than is often presented. For example, strong-focusing produced by averaging is seldom used. If these mechanisms are important, then it opens two research directions. The first is to develop algorithms that better exploit the mechanisms. The second is to investigate what class of problems benefit from these different mechanisms. In the authors view, both of these research agendas have significant potential to furthering the success of population-based algorithms.

Acknowledgements

The author would like to thank his collaborators who have contributed much to the understanding of these mechanisms. In particular, Jonathan Shapiro, Alex Rogers, Celia Glass, Will Benfold, Jonathan Hallam and Mohamed Qasem have all made direct contributions to studying the models discussed in this paper.

Appendix

A. *Optimal Annealing Schedules*

Some of the toy models discussed in this paper have a sufficient degree of symmetry so that, for moderate sized problems, many of their properties can be studied using Markov chains. In this appendix, we concentrate on the Concatenated-$\mathcal{V}$ problem, while in appendix B we consider the Iceberg problem.

In the Concatenated-$\mathcal{V}$ problem with m blocks of size k, it is sufficient for many algorithms to describe a solution in terms of the number of blocks with l of the variables in the 1's state where $l \in \{0, 1, \ldots, k\}$. This can be viewed as an exact coarse-graining of the model from the initial search space of size $2^{m\,k}$ to a state space of size $\binom{m+k}{k}$. Each state (usually) correspond to many possible configurations all with the same fitness. It is straightforward to compute the probability of changing from one state to another.

To set up a Markov chain model describing the dynamics of a solo-search algorithm such as hill-climbing or simulated annealing we denote the probability of being in state i at iteration t, by $p_i(t)$. We can describe the probability of being in all possible states by a vector $\boldsymbol{p}(t) = \big(p_1(t), p_2(t), \ldots, p_s(t)\big)^{\mathsf{T}}$ where $s = \binom{m+k}{k}$. We denote the transition probabilities between states by a matrix $\mathbf{M}(t)$, with elements $M_{ij}(t)$ giving the probability of making a transition from state j to state i at iteration step t. The dynamics of the system is described by the matrix equation

$$\boldsymbol{p}(t+1) = \mathbf{M}(t)\,\boldsymbol{p}(t)$$

where $\boldsymbol{p}(0)$ describes the probability distribution for the initial population. In the following analysis, we assume that the population consists of random strings so that $p_i(0)$ is proportional to the number of configurations in state i.

For simulated annealing (and, for that matter, hill-climbing) the matrix $\mathbf{M}(t)$ is very sparse. We therefore never explicit write the matrix $\mathbf{M}(t)$, but rather consider only the neighbours of each state. This allows the exact computation of the dynamics for surprisingly large systems (e.g. $m = 30$ and $k = 7$), although as our aim is to find optimal annealing schedules (which requires a lot of additional computation) we consider a much smaller problem instance.

In simulated annealing there is flexibility in choosing the annealing temperature which determines the probability of making a move which decreases the fitness. The set of temperatures is known as the annealing schedule. To optimise the annealing schedule we have to choose some criteria we wish to optimise. In this appendix we choose the expected fitness at time T. If $\boldsymbol{c} = (c_1, c_2, \ldots, c_s)^{\mathsf{T}}$ where c_i is the cost of state i, then the objective will be to optimise $\boldsymbol{c}^{\mathsf{T}}\boldsymbol{p}(T)$. Other optimisation criteria could also been considered such as optimising the expected best solution found throughout the entire run—this however, requires even more computation so is not considered here. If we parametrise the annealing schedule by a set of parameters $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots)$, then our task is to find an assignment of $\boldsymbol{\theta}$ which maximises $\boldsymbol{c}^{\mathsf{T}}\boldsymbol{p}(T)$. This is a standard multi-dimensional continuous optimisation problem which can be solved by standard methods. To speed up the search we can compute the gradients

$$\frac{\partial \boldsymbol{c}^{\mathsf{T}}\boldsymbol{p}(T)}{\partial \theta} = \boldsymbol{c}^{\mathsf{T}}\frac{\partial \boldsymbol{p}(T)}{\partial \theta}$$

where we can use the recursion relation

$$\frac{\partial \boldsymbol{p}(t+1)}{\partial \theta} = \frac{\partial \mathbf{M}(t)}{\partial \theta} \boldsymbol{p}(t) + \mathbf{M}(t) \frac{\partial \boldsymbol{p}(t)}{\partial \theta}$$

to compute the gradient of $\boldsymbol{p}(T)$. More details of this approach and various extensions are given in reference [35].

In this paper, we considered 3 parametrisations of the annealing schedule. In the first case, we consider the annealing temperature at step t to be $T_0\, a^t$ where T_0 and a are to be determined. As a richer parametrisation of the annealing schedule we chose the probability of making a step from one state to a neighbouring state with a fitness lower by 1 to be

$$\mathrm{e}^{1/\beta(t)} = \frac{1}{1 + \mathrm{e}^{\sum_{i=0}^{15} c_i \, \mathrm{Cheb}_i(2t/T-1)}}$$

where $\mathrm{Cheb}_i(x)$ are the Chebyshev polynomials defined by

$$\mathrm{Cheb}_{i+1}(x) = 2\, x\, \mathrm{Cheb}_i(x) + \mathrm{Cheb}_{i-1}(x)$$

and the c_i's are a set of parameters to be tuned. Although this looks rather complicated, the purpose was to provide a flexible parametrisation of smooth functions between 0 and T. Details of Chebyshev polynomials can be found in reference [36]. The logistic function was used to ensure the probability was constrained between 0 and 1. The final parametrisation was to choose the annealing temperature independently at each step. In this last case, it was found that the annealing temperatures either became zero or infinity, indicating that a better strategy for this problem than simulated annealing was to perform hill-climbing with occasional random walks. The strategy found by gradient descent optimisation was found to be locally optimal only. To find better strategies we optimised a schedule of hill-climbs and random walks using a hill-climber. That is, we defined a schedule for a hill-climber/walk algorithm by a vector of 1's an 0's where 1 signified a hill-climbing step and 0 a random walk step. We then optimised this binary vector using a hill-climbing algorithm.

Figure 25 shows the performance of simulated annealing with the best exponential schedule, the best parametrised Chebyshev schedule and the best found hill-climber/walk schedule. Although the hill-climber/walk algorithm can be viewed as a simulated annealing schedule with temperatures of zero and infinity, it differs considerably from a classical annealing schedule. Interestingly, the profiles for both the hill-climbing/walk algorithm and the parametrised-Chebyshev schedule are extremely close. In figure 26, we show the annealing schedules used to obtain the result shown in figure 25. It is hard to show the hill-climbing/walk schedule as this regularly jumps between 0 and 1. Instead we have computed the negative of the reciprocal of the log-probability of performing a walk averaged over a window of size 10—for this problem, this quantity plays a similar role as the temperature in determining the probability of making a random step. The hill-climber/walk schedule is extremely close to the optimal Chebyshev schedule, illustrating that the two strategies explore the search space in a very similar way. In fact, the difference is that hill-climber/walk strategy is deterministic in choosing to move in the wrong direction rather than stochastic.

A few comments are in order. Firstly, the difference between the optimum exponential schedule and the global optimum

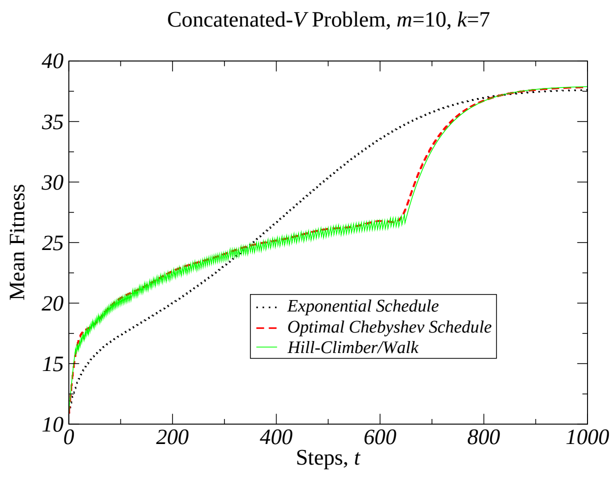

Fig. 25. Expected fitness versus iteration number for the Concatenated-$\mathcal{V}$ problem with $m = 10$ and $k = 7$. The expected fitness of the hill-climbing/walk algorithm exhibits high frequency fluctuations as a result of the rapid switch between hill-climbing and random walks.

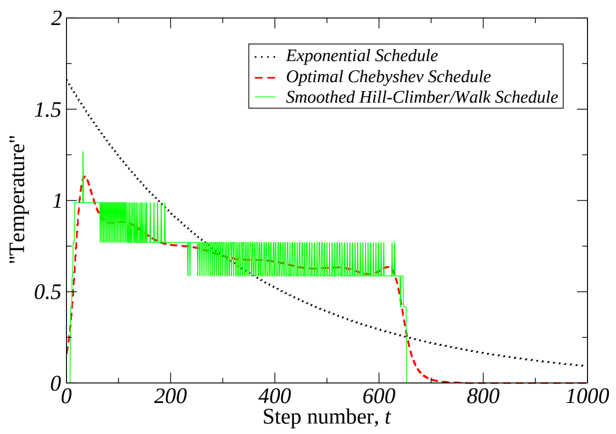

Fig. 26. Annealing schedules used in obtaining the results shown in figure 25 is shown for the optimal exponential schedule and the optimal parametrised-Chebyshev schedule. For the hill-climbing/walk algorithm the negative reciprocal log-probability of performing a walk averaged over a window of size 10 is shown.

annealing schedule is small. This provides support for the assertion made in the main text that we have not unduly penalised simulated annealing, by choosing a poor annealing schedule (although, the exponential annealing schedules used elsewhere this paper were optimised by hand after a small amount of preliminary tuning rather than through systematic optimisation as described in the appendix). The second observation is that there seems to be an annealing temperature where search spends most of its time (around 0.75 for this problem). This is commonly observed in many problems. It says that for a large part of the search time the optimal exploration-exploitation balance remains approximately fixed. This can be seen as a consequence of the exponentially large search space—most of the time the landscape being explored is statistical similar. Finally, the approach taken here comes with the caveat that the results are correct only up to numerical accuracy of the computer. Empirically, the Markov chain analysis appears to be numerically reliable. The optimisation is carried out to a certain accuracy, so the true optimal annealing schedule may differ slightly from that shown. Finally, this approach can only find locally optimal annealing schedules and provides no guarantees that they are globally optimal. Nevertheless, we have some confidence that they may well be global as we found the same solution from different starting points.

B. Exact First-Passage Times

In this appendix, we consider the problem of computing the expected number of iterations before reaching a global optimum. This is often referred to as the first-passage or first-hitting time. We can compute this exactly for a hill-climber using a Markov chain approach. We illustrate this procedure for the Iceberg problem. This problem can again be coarse-grained. In this case, configurations with the same number of 1's are lumped together in a state. Thus for a string of size n there are $n+1$ states which we can label by the number of 1's. The global optimum state is the state n. Assuming single-bit mutations the probability of a transition between states is again very easy to compute. The expected first-passage time is equal to the average time it takes to reach the global optimum

$$\sum_{t=0}^{\infty} t\left(p_n(t) - p_n(t-1)\right).$$

A standard result is that the expected first-passage time is given by

$$\mathbf{1}^{\mathsf{T}} \left(\mathbf{I} - \hat{\mathbf{M}}\right)^{-1} \boldsymbol{p}(0) - 1$$

where $\mathbf{1}$ is a vector of all 1's, $\mathbf{I}$ is the identity matrix and $\hat{\mathbf{M}}$ is the transition matrix modified so that the transition probability from the optimal state to all other states (including itself) is zero. A derivation of this formula is given in reference [23].

For the Iceberg problem the (modified) transition matrix is tri-diagonal so that inverting $\mathbf{I} - \hat{\mathbf{M}}$ can be computed in order n operations. Thus it is fast to find the expected first-passage time. However, as with all matrix inversions, the results can be numerically unstable as the matrix becomes large. Even using quadratic precision the program fails for n around 270. However, for the Iceberg problem the initial probabilities and the transition probabilities can all be expressed exactly as rational numbers. Thus to compute the first-passage time exactly an arbitrary length rational data structure was used. The result of this are shown in figure 13. (It should be noted that this Markov Chain analysis is only feasible because of the simplicity of the problem and the search algorithm.)

References

[1] T. Jansen and I. Wegener, "Real royal road functions: where crossover provably is essential," *Discrete Appl. Math.*, vol. 149, no. 1-3, pp. 111–125, 2005.

[2] C. Witt, "Population size versus runtime of a simple evolutionary algorithm," *Theoretical Computer Science*, 2008.

[3] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[4] M. Mitchell, J. Holland, and S. Forrest, "When will a genetic algorithm outperform hill climbing?" in *Advances in Neural Information Processing Systems*, J. Cowan, G. Tesauro, and J. Alspector, Eds. San Francisco, CA.: Morgan Kauffman, 1994, pp. 51–58.

[5] R. A. Watson, "Analysis of recombinative algorithms on a non-separable building-block problem," in *Foundations of Genetic Algroithms (FOGA-6)*, W. N. Martin and W. M. Spears, Eds. San Francisco: Morgan Kaufmann, 2001, pp. 69–89.

[6] M. Dietzfelbinger, B. Naudts, C. Van Hoyweghen, and I. Wegener, "The analysis of a recombinative hill-climber on h-iff," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 5, pp. 417–423, 2003.

[7] O. M. Becker and M. Karplus, "The topology of multidimensional potential energy surfaces: Theory and application to peptide structure and kinetics," *The Journal of Chemical Physics*, vol. 106, 1997. [Online]. Available: http://link.aip.org/link/?JCP/106/1495/1

[8] J. Hallam and A. Prügel-Bennett, "Large barrier trees for studying search," *IEEE Transaction on Evolutionary Computation*, vol. 9, no. 4, pp. 385–397, 2005.

[9] A. Prügel-Bennett, "Finite population effects for ranking and tournament selection," *Complex Systems*, vol. 12, no. 2, pp. 183–205, 2000.

[10] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm," in *Proceedings of the Second International Conference on Genetic Algorithms*. Lawrence Erlbaum Associates (Hillsdale), 1987.

[11] A. Prügel-Bennett, "The mixing rate of different crossover operators," in *Foundations of Genetic Algorithms 6*, W. N. Martin and W. M. Spears, Eds. San Francisco: Morgan Kaufmann, 2001, pp. 261–274.

[12] ——, "Modelling crossover induced linkage in genetic algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 4, pp. 376–387, 2001.

[13] R. A. Watson and T. Jansen, "A building-block royal road where crossover is provably essential," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2007)*, 2007, pp. 1452–1459.

[14] P. Galinier and J. K. Hao, "Hybrid evolutionary algorithms for graph coloring," *Journal of Combinatorial Optimization*, vol. 3, no. 4, pp. 379–397, 1999.

[15] R. Williams, C. Gomes, and B. Selman, "Backdoors to typical case complexity," in *Proc. of the 18th IJCAI*, 2003.

[16] P. Kilby, J. Slaney, and T. W. S. Thiebaux, "Backbones and backdoors in satisfiability," in *Proc. of the 20th National conference on artificial intelligence and the 17th innovative appllications of artificial intelligence conference*, 2005, Ed. Menlo park, CA: AAAI/MIT Press, pp. 1368–1373.

[17] C. A. Glass and A. Prügel-Bennett, "Genetic algorithms for graph colouring: Exploration of Galinier and Hao's algorithm," *Journal of Combinatorial Optimization*, vol. 7, pp. 229–236, 2003.

[18] N. J. Radcliffe, "Forma analysis and random respectful recombination," in *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann (San Mateo), 1991, pp. 222–229.

[19] J. L. Shapiro and A. Prügel-Bennett, "Genetic algorithms dynamics in two-well potentials with basins and barriers," in *Foundations of Genetic Algorithms 4*, R. K. Belew and M. D. Vose, Eds. San Francisco: Morgan Kaufmann, 1997, pp. 101–116.

[20] A. Rogers and A. Prügel-Bennett, "The dynamics of a genetic algorithm on a model hard optimization problem," *Complex Systems*, vol. 11, no. 6, pp. 437–464, 2000.

[21] ——, "A solvable model of a hard optimization problem," in *Theoretical Aspects of Evolutionary Computing*, ser. Natural Computing, L. Kallel, B. Naudts, and A. Rogers, Eds. Berlin: Springer, 2001, pp. 207–221.

[22] T. Jansen and I. Wegener, "On the analysis of evolutionary algorithms — a proof that crossover can really help," in *Proceedings of the 7th Annual European Symposium on Algorithms (ESA'99)*, J. Nešetřil, Ed. Berlin: Springer, 1999, pp. 184–193.

[23] A. Prügel-Bennett, "When a genetic algorithm outperforms hill-climbing," *Theoretical Computer Science*, vol. 320, no. 1, pp. 135–153, 2004.

[24] M. Qasem and A. Prügel-Bennett, "Learning the large-scale structure of the max-sat landscape using populations," *IEEE Transactions on Evolutionary Computation*, 2008, submitted.

[25] R. Poli, W. A., M. N., and L. W., "Emergent behaviour, population-based search and low-pass filtering," in *IEEE Congress on Evolutionary Computation*, p. 2006.

[26] H. Mühlenbein, M. Gorges-Schleuter, and . Kraemer, "Evolution algorithm in combinatorial optimization," *Parallel Computing*, vol. 7, pp. 65–88, 1988.

[27] D. Costa, A. Hertz, and O. Dubuis, "Embedding a sequential procdure within an evolutionary algorithm for coloring problems," *Journal of Heuristics*, vol. 1, pp. 105–128, 1995.
[28] E. Falkanauer, "A hybrid grouping genetic algorithm for bin packing," *Journal of Heuristics*, vol. 2, no. 1, pp. 5–30, 1996.
[29] B. Freisleben and P. Merz, "New genetic local search operators for the travelling salesman problem," in *Lecture Notes in Computer Science 114'*. Springer, 1996, pp. 890–899.
[30] P. Merz and B. Freisleben, "A genetic local search approach to the quadratic assignment problem," in *Proc. of the 7th International Conference on Genetic Algorithms*. Morgan Kauffman, 1997, pp. 465–472.
[31] H. Horner, "Dynamics of learning for the binary perceptron problem," *Zeitschrift für Physik: B*, vol. 86, pp. 291–308, 1992.
[32] K. Patel, "Computational complexity, learning rules and storage capacities: Monte Carlo study for the binary perceptron," *Zeitschrift für Physik: B*, vol. 91, pp. 257–266, 1993.
[33] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.
[34] A. Prügel-Bennett, "Symmetry breaking in population-based optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 1, pp. 63–79, 2004.
[35] W. Benfold, J. Hallam, and A. Prügel-Bennett, "Optimal parameters for search using a barrier tree Markov model," *Theoretical Computer Science*, vol. 386, 2007.
[36] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computation*, 3rd ed. Cambridge, UK.: Cambridge University Press, 2007.