1993-11

How Good are Genetic Algorithms at Finding Large Cliques: An Experimental Study

How good are genetic algorithms at finding large cliques: an experimental study*

Bob Carter
carter@cs.bu.edu

Kihong Park†
park@cs.bu.edu

BU-CS-93-015
October 10, 1993

Boston University
Computer Science Department
Boston, MA 02215
Phone: 617 353-8919
Fax: 617 353-6457

Abstract

This paper investigates the power of genetic algorithms at solving the MAX-CLIQUE problem. We measure the performance of a standard genetic algorithm on an elementary set of problem instances consisting of embedded cliques in random graphs. We indicate the need for improvement, and introduce a new genetic algorithm, the *multi-phase annealed GA*, which exhibits superior performance on the same problem set.

As we scale up the problem size and test on "hard" benchmark instances, we notice a degraded performance in the algorithm caused by premature convergence to local minima. To alleviate this problem, a sequence of modifications are implemented ranging from changes in input representation to systematic local search. The most recent version, called *union GA*, incorporates the features of union cross-over, greedy replacement, and diversity enhancement. It shows a marked speed-up in the number of iterations required to find a given solution, as well as some improvement in the clique size found.

We discuss issues related to the SIMD implementation of the genetic algorithms on a Thinking Machines CM-5, which was necessitated by the intrinsically high time complexity ($O(n^3)$) of the serial algorithm for computing one iteration.

Our preliminary conclusions are: (1) a genetic algorithm needs to be heavily customized to work "well" for the clique problem; (2) a GA is computationally very expensive, and its use is only recommended if it is known to find larger cliques than other algorithms; (3) although our customization effort is bringing forth continued improvements, there is no clear evidence, at this time, that a GA will have better success in circumventing local minima.

*Part of this research was presented at the 2nd DIMACS Implementation Challenge on Combinatorial Optimization, DIMACS, October, 1993.

†Supported in part by NSF grant CCR-9204284

1 Introduction

This paper investigates the power of genetic algorithms at solving the MAX-CLIQUE problem. Genetic algorithms [7, 6] are general-purpose optimization methods that have gained wide popularity in recent years. They are an algorithmic formalization of the twin notions of "survival of the fittest" and "fit parents are likely to produce even fitter off-spring." Surprisingly, this combination has lead to a randomized algorithm with unique features which seems to assure its place among other well-known heuristics such as simulated annealing and gradient descent. As with simulated annealing, it is an open problem as to how powerful genetic algorithms are at solving combinatorial optimization problems, and few rigorous characterizations exist. Recent work in this regard includes [11, 12, 10]. On the empirical side, a large body of work exists in applying genetic algorithms to different application areas, as well as experimental evaluation on hard combinatorial optimization problems, notably the traveling salesman problem [9, 4, 13, 8]. Although the jury is still out, there is general agreement that genetic algorithms need to be tailored to the particular application at hand, and in the case of the traveling salesman problem, genetic algorithms have been exhibited which approach, and in some cases exceed, the performance of other well-known heuristics on certain problem instances. In contrast, not much work has been done on the maximum clique problem, and this paper is an effort in this direction.

Our test ground is MAX-CLIQUE, an NP-complete problem. The need to find large cliques arises in numerous pratical areas [2], and many constraint satisfaction problems can be naturally cast as MAX-CLIQUE. Furthermore, recent work in interactive proof systems has shown that unless P = NP, finding a good approximation to the maximum clique size is as hard as solving the exact problem [5, 1]. This, coupled with the special year on Combinatorial Optimization at DIMACS and its Implementation Challenge, has motivated us to approach the MAX-CLIQUE problem from the genetic algorithms perspective. We proceed in three stages. First, we test a standard genetic algorithm on a test bed of random graphs with controlled embedded cliques and show the need for improvement. We introduce a new genetic algorithm, *multi-phase annealed GA*, which is a two-way extension of the standard algorithm, and demonstrate its superior performance. The algorithm is obtained by incorporating an *annealed fitness function*[1] which assigns variable fitness values to

[1] "fitness function" is a term commonly used in the GA literature to refer to an *objective function*.

σ có vai trò để kiểm soát "phạt" cho sub graph ko phải cliques; T nhỏ phạt ít, T to phạt nhiều

subgraphs that are not cliques, parameterized by a control variable σ. This variable determines the degree to which subgraphs are penalized for not being cliques, and leads to an annealing schedule on σ whereby the penalty is increased as time progresses. Another feature of the algorithm is based on *population-driven search* where the statistical properties of the initial population are controlled to produce efficient search.

Next, the algorithm is tested on benchmark problems contributed to the DIMACS Challenge as well as other instances such as 1024-node random graphs. The conclusion is that the multi-phase annealed GA is getting stuck in local minima. As the problems get "harder" and the problem size increases, the algorithm does not scale satisfactorily resulting in suboptimal solutions. Several modifications are tested, including systematic local search and changing of the input representation via clustered encoding. In the most recent version (*union GA*), three features, *union cross-over*, *greedy replacement*, and *diversity enhancement* are combined resulting in an improved performance. This is most pronounced time-wise in that the number of iterations needed to find a given clique size is substantially reduced, accompanied by small improvements in the maximum clique size found. In union cross-over, two subgraphs are merged to form another graph by taking the union of the two vertex sets, then setting the new graph to be the vertex-induced subgraph. Nodes which were not in the intersection of the parent subgraphs are then pruned with some probability. In greedy replacement, each element is replaced with a new element if the new element represents an improvement over the former. Diversity enhancement of a population is achieved by probabilistically reallocating elements belonging to a saturated subpopulation to one that is not.

genotype

C → C'

In the final section, we discuss some issues related to the parallel implementation of the genetic algorithm on a Thinking Machines CM-5. Due to the high time-complexity of the algorithm ($O(n^3)$ for one iteration), which is intrinsic to GA's, we were forced to perform all our testing on the CM-5. The program was written SIMD-style, whereby every element of the population was treated as a virtual processor, subject to the same code. The main drawback of this approach lies in not being able to explicitly control how virtual processors are mapped to actual processors on a partition. Assuming this mapping is handled poorly by the operating system, a potential improvement in speed is attainable by switching to full-fledged MIMD programming whereby making use of a priori information regarding the communication pattern within a population, an improved mapping could be imposed at the extra cost of having to compute it. Otherwise, various hardware features of the

Độ ưu phân cứng ...

CM-5 were exploited to yield a faster code.

We conclude by discussing where we are now in the evaluation process, the tentative conclusion being that we have as yet found no firm evidence to suggest that a genetic algorithm may outperform other well-known heuristics such as simulated annealing for the MAX-CLIQUE problem. On the other hand, our experience so far leads us to believe that the current performance can be further improved to levels matching other heuristics. If this is so, the benefit of using a GA needs to be weighed against the fact that its time-complexity is very high.

2 A simple genetic algorithm and the need for customization

2.1 Simple genetic algorithm

Let $G = (V, E)$ be an undirected graph where V is the vertex set and E is the set of edges. Let $n = |V|$. A *clique* of size k is a complete subgraph of G with k vertices. For any graph G of n vertices, we will use $X = \{0,1\}^n$ to encode its subgraphs. That is, $x = x_1 x_2 \ldots x_n$ corresponds to the unique subgraph $G' = (V', E')$, where $x_i = 1$ if and only if $i \in V'$ for all $i = 1, 2, \ldots, n$. A *fitness function* $f : X \to \mathcal{R}_+$ assigns to each subgraph a nonnegative number, its *fitness value*. A simple way of defining a fitness function for the MAX-CLIQUE problem is given by,

$$\forall x \in X, \qquad simple(x) = \begin{cases} k & \text{if } x \text{ is a clique of size } k \\ 0 & \text{otherwise.} \end{cases}$$

A *population* is a multi-set over X. Let $\mathcal{H}(t)$ be the population at time t, and let $N = |\mathcal{H}(t)|$ for all $t \geq 0$. A genetic algorithm using a fitness function f is characterized by three operations, *reproduction* (T_R), *cross-over* (T_C), and *mutation* (T_M). The operations are described as follows:

- *Reproduction.* To compute $T_R(\mathcal{H})$, generate N random samples from the distribution $P_r\{h = x\} = f(x)/\sum_{y \in \mathcal{H}} f(y)$, where h is a random variable over sample space $\mathcal{H}$.
- *Cross-over.* To compute $T_C(\mathcal{H})$, repeat N times: select $x = x_1 x_2 \ldots x_n$, $y = y_1 y_2 \ldots y_n \in \mathcal{H}$ randomly using the uniform distribution. Select a random index i, $1 \leq i \leq n+1$. Form $z = x_1 \ldots x_{i-1} y_i \ldots y_n$. Include z in $T_C(\mathcal{H})$.
- *Mutation.* To compute $T_M(\mathcal{H})$, for all $x \in \mathcal{H}$, with probability p_m, do the following: select a random index i, $1 \leq i \leq n$. Flip the i'th bit x_i.

Let T be the composite map $T = T_M T_C T_R$. If $\mathcal{H}(t)$ is the population at time t, then $\mathcal{H}(t+1) = T(\mathcal{H}(t))$. One such step is called a *generation*. We call T the *simple genetic algorithm* with mutation. In theoretical considerations, one is interested in understanding the behavior of $(\mathcal{H}(t))_{t=0}^{\infty}$.

The *basic test data set* is described by a triple $n/k/p$ where n is the number of nodes in the test graph G, k is the size of the clique embedded in the graph, and p is the probability by which edges are added to the already existing edge set induced by the imbedded clique. For p small, the maximum clique size, $\omega(G)$, will equal k.

2.2 Multi-phase annealed GA

2.2.1 Relaxing the fitness function

In an attempt to improve search, we introduce an *annealed* fitness function that has the following form. Let $\mathcal{N}(i) = \{j \in V : \{i,j\} \in E\} \cup \{i\}$ be the neighborhood of node i, and let $I(x) = \{i : x_i = 1,\ i = 1,2,\ldots,n\}$. Define

$$\rho(x) = \frac{\sum_{i \in I(x)} |\mathcal{N}(i) \cap I(x)|}{|I(x)|^2}.$$

Clearly, $0 \le \rho(x) \le 1$, and $\rho(x) = 1$ if and only if x encodes a clique. The closer $\rho(x)$ is to 0, the more "deficient" x is from being a clique. Biasing the fitness function towards large subgraphs and combining with $\rho(x)$, we get the *annealed fitness function*

$$\kappa_\sigma(x) = \rho(x)^\sigma \, |I(x)|$$

where $\sigma \ge 1$ is a parameter that can be used to control the degree to which x is penalized for not being a clique. It is easily seen that

$$\lim_{\sigma \to \infty} \kappa_\sigma(x) \;=\; \lim_{\sigma \to \infty} \rho(x)^\sigma |I(x)| \;=\; simple(x).$$

Note, for $\sigma < \infty$, it is possible that a large, densely interconnected subgraph is assigned a higher fitness value than $\omega(G)$, the maximal clique size. The penalty parameter σ is similar to temperature in simulated annealing in that a controlled fuzziness factor is injected to enhance search. This suggests imposing an annealing schedule on $\sigma(t)$. In the current implementation, we have used the following rule for the annealing schedule *Anneal*:

$$\sigma := \sigma \left(1 + c_a \frac{|\text{max\_all} - \text{max\_clique}|}{\text{max\_all}}\right)$$

where max_all $= \max\{\kappa_\sigma(x) : x \in \mathcal{H}\}$, and max_clique $= \max\{\kappa_\sigma(x) : x \in \mathcal{H} \wedge x \text{ is clique}\}$. $c_a > 0$ is a parameter that controls the rate of change of σ. Let *diversity* be a function that measures the degree to which a population is nonhomogeneous. We define it by the formula

$$div = \frac{1}{nN} \sum_{i=1}^{n} \max\{\sum_{x \in \mathcal{H}} x_i\,,\ N - \sum_{x \in \mathcal{H}} x_i\}.$$

After the coordinate change $div' = 2(1 - div)$, $0 \leq div \leq 1$, and $div = 0$ if and only if $\mathcal{H}$ consists

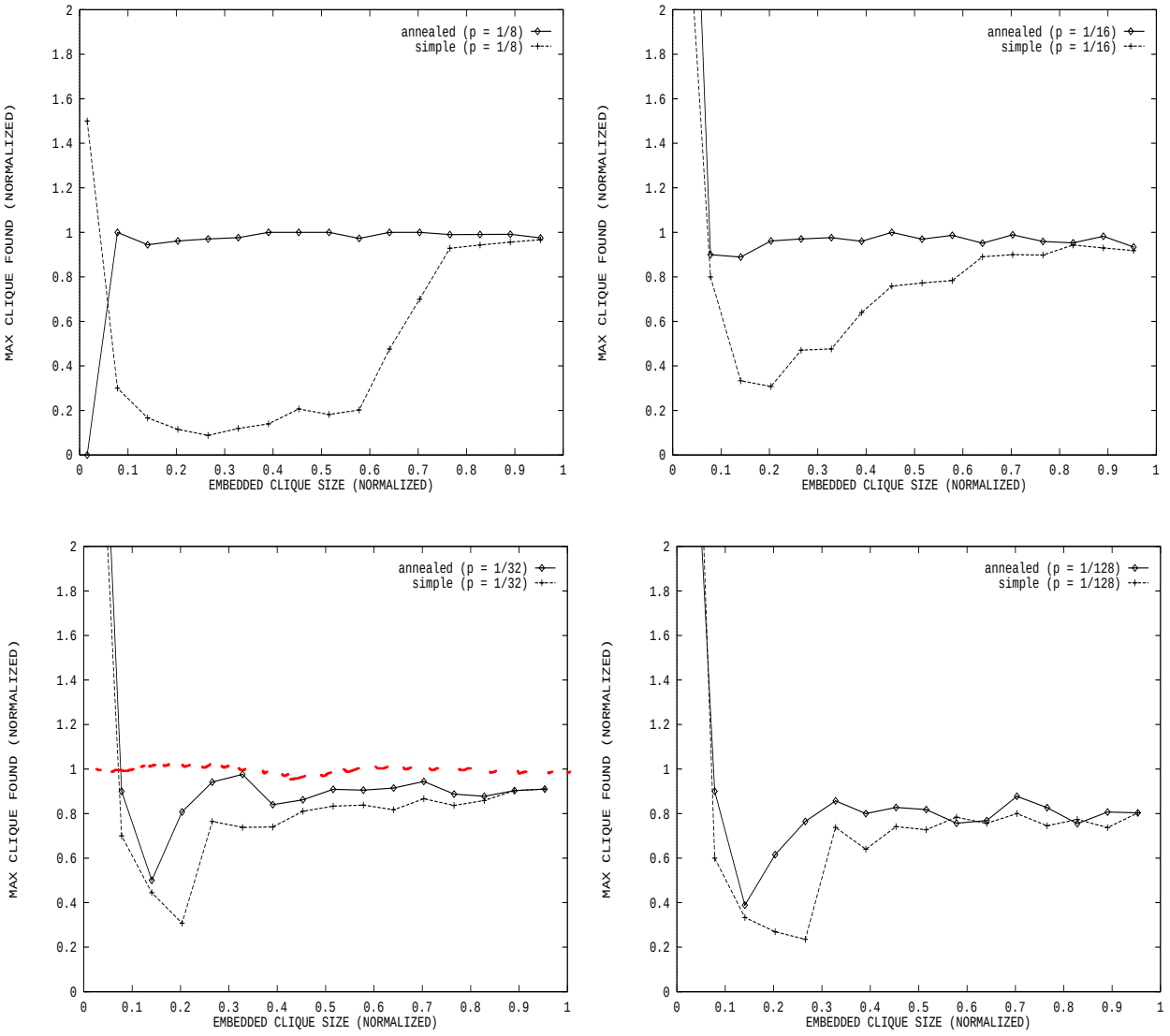


Figure 1: Effect of annealing. 128-node graph with $p = 1/8, 1/16, 1/32, 1/128$.

of N copies of a single element. The annealed genetic algorithm (A-GA) is given as follows:

A-GA$(\mathcal{H},\sigma)$

begin

  repeat

    $\mathcal{H} := T_R(\mathcal{H}, \kappa_\sigma)$.

    $\mathcal{H} := T_C(\mathcal{H})$.

    $div :=$ Diversity$(\mathcal{H})$.

    $\sigma :=$ Anneal$(\sigma, \mathcal{H})$.

  until $div < \theta$

end

Figure 1 captures the effect of annealing a fitness function. It compares performance of the annealed GA against the simple GA on the problem set $\{128/k/0.5 : k = 3, 10, 17, \ldots, 115\}$ for four initial populations generated with vertex probabilities $p = 1/8, 1/16, 1/32, 1/128$. The abscissa, *normalized embedded clique size*, represents k/n. The ordinate, *normalized max clique found*, refers to $\hat{\omega}/k$ where $\hat{\omega}$ is the size of the maximum clique found by the algorithm over its entire run. For all the runs reported in the paper, the population size was set at $N = 20n$. Figure 1 clearly shows the superior performance of the annealed fitness function in finding larger cliques. The effect is most pronounced when the initial population is generated with $p = 1/8$.

2.2.2 Population-driven search

Figure 2 shows the effect of changing the characteristics of the initial population. In the case where the simple fitness function is employed (left figure), a marked improvement is obtained when going from vertex probability $p = 1/8$ to $p = 1/32$ for normalized clique sizes below 0.7. As p is decreased further, a degradation in performance occurs as can be seen in the downward shift of the $p = 1/128$ curve. This suggests that the performance curve, as a function of p, is unimodal which introduces the problem of having to find its peak. Furthermore, the gain in overall performance is obtained at the expense of degraded performance at the higher end of the normalized clique size. The situation is different in the annealed fitness function case. Except for the failure to find a very small clique (normalized clique size 0.023), the $p = 1/8$ curve is highly flat, finding the optimal clique or one near to it, and clearly superior to $p = 1/32, 1/128$. Furthermore, the downward shift in the performance curve is monotonic as p is decreased. The only gain in making p small is the

compensatory effect obtained when the size of the maximum clique is very small. This suggests using a procedure which commences with an initial population characterized by a high value of p,

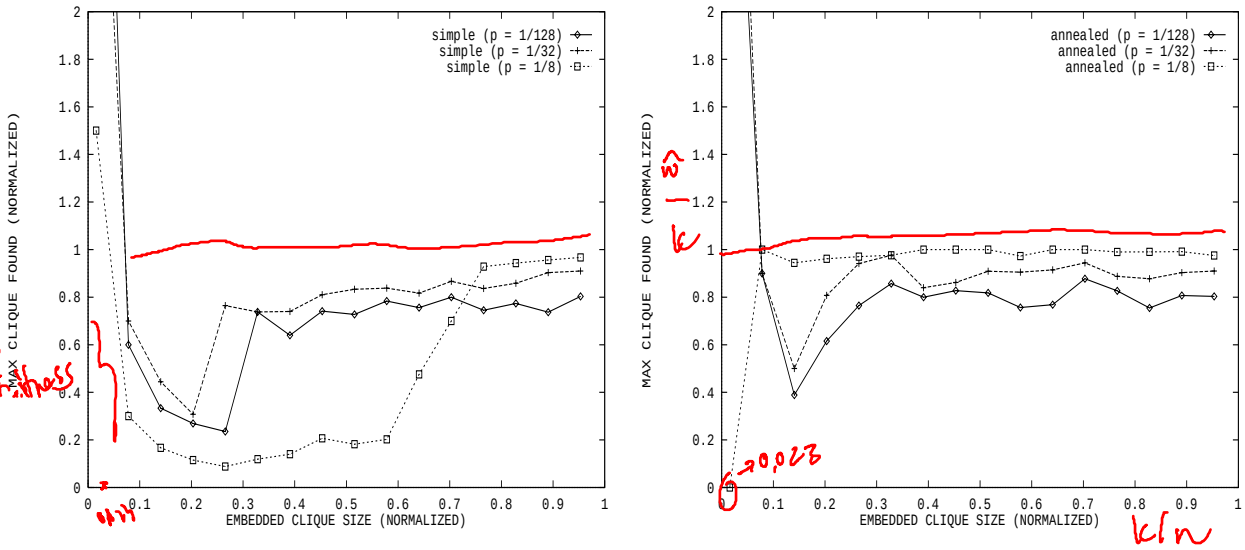


Figure 2: Effect of initial population. 128-node graph. (left) Simple fitness function. (right) Annealed fitness function.

then repeats the search over several phases, each with a smaller probability. This strategy is called *multi-phasing*, and a genetic algorithm which exploits this feature is described next.

MPA-GA:

begin

$p := 1/2$.

Initialize $\mathcal{H}$ using the distribution $P_r\{x_i = 1\} = p, \ \forall i$.

repeat

$\sigma := 1$.

A-GA$(\mathcal{H}, \sigma)$.

$p := p/2$.

Reset $\mathcal{H}$ using the distribution $P_r\{x_i = 1\} = p, \ \forall i$.

until $pn \leq \gamma$

end

The initial population is generated by setting each bit to 1 with probability 1/2. Next, the annealed GA is invoked, its termination concluding the first phase. In subsequent phases, p is exponentially decreased by a factor of 1/2 until the last phase is reached at $pn = \gamma$. In the current implementation, $\gamma = 1$.

For a typical test graph $G \in 64/k/0.5$ (not shown here), disregarding the embedded k-clique and looking only at the 64-node 0.5-random graph, one may estimate (very roughly) the maximum clique size to be around 8, using the threshold function $2\log_{\frac{1}{p}} n - 2\log_{\frac{1}{p}} \log_{\frac{1}{p}} n + 2\log_{\frac{1}{p}} e/2 + 1$ [3]. The actual size of the embedded clique corresponding to normalized clique size 0.2 is 13. The height at 0.2 for the multi-phase simple case is 0.6 which corresponds to a clique size of 8. Hence, quite plausibly, the algorithm is getting stuck in local minima created by the underlying p-random graph. To test this hypothesis, we ran the algorithm on several test sets $128/k/p$ for different p's. The outcomes of $128/k/0.3$, $128/k/0.7$, and $128/k/0.9$ are shown in figure 3. The shift of the dip to the right as edge probability p is increased is clearly visible, confirming our suspicion.

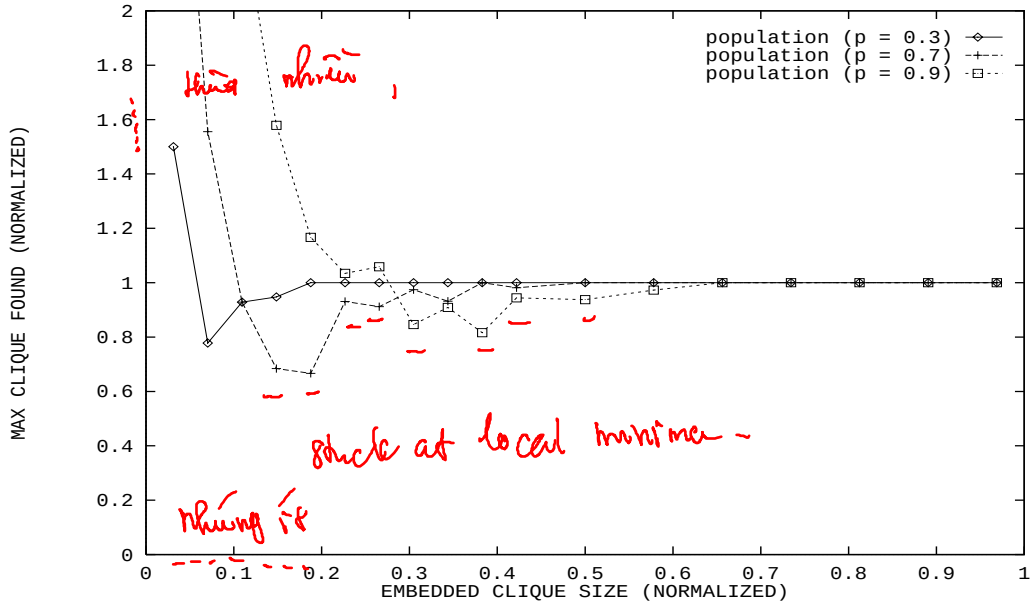
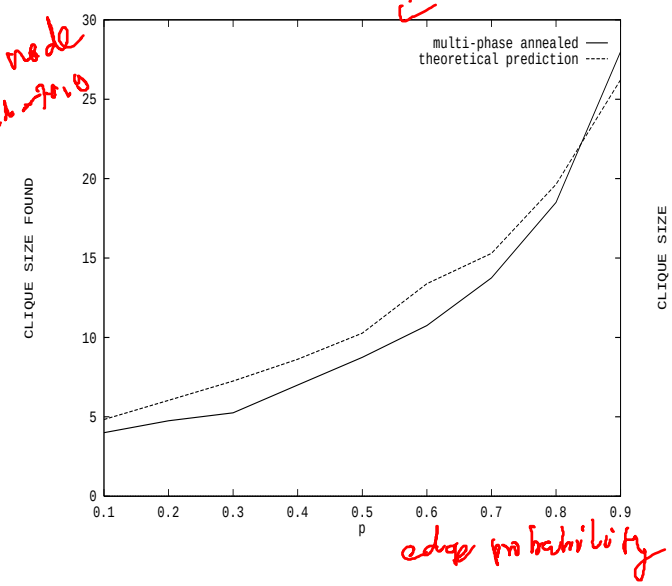


Figure 3: Local minima problem and the shift in the dip as a function of edge probability p.

Figure 4 (left) shows the performance of the multi-phase annealed GA and multi-phase simple GA on a set of p-random graphs with 128 vertices where p was sampled at $p = 0.1, 0.2, \ldots, 0.9$. Overall, MPA-GA outperforms the simple algorithm, but not by very much. Figure 4 (right) shows



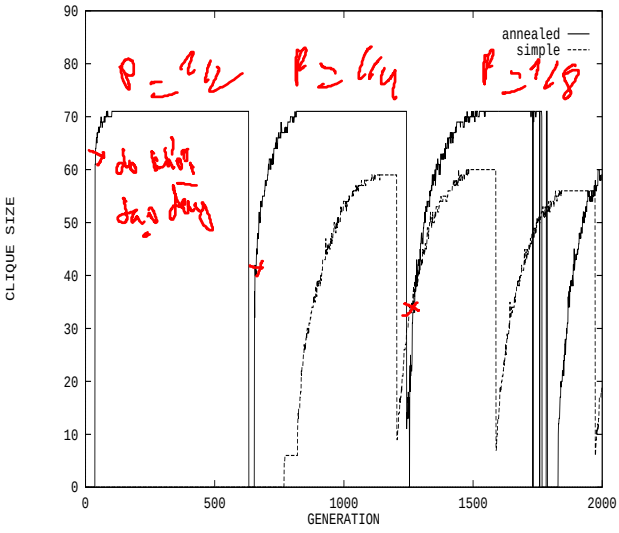


Figure 4: (left) p-random graphs. Normalized clique size found against edge probability p. (right) 128-node overlap graph. Normalized clique size found against normalized running time.

a similar performance comparison on a 128-node graph where one large clique (64 nodes), and three smaller ones (57, 47, and 47 nodes) were embedded with mutual overlap. There was a 31 node overlap between the 64-node clique and each of the two 47-node cliques. There was further overlap among the other combinations to a smaller extent. Over 2000 generations, the multi-phase annealed GA found a clique of size 71 (in fact, the maximal clique) whereas the multi-phase simple GA found one of size 60. The three long plateaus corresponding to the annealed algorithm are in part caused by a stringent termination threshold $\theta = 0.0001$. The three little humps corresponding to the simple algorithm have shorter width due to a bound on the phase length, which for this run was set at $3n$ generations. This was instantiated so as to give the multi-phase simple GA as many phases as possible without compromising the search over each phase by too much. Without a time-limit on the phase length, we noticed that the simple GA would often reach a plateau, and spend an inordinate amount of time converging without improving its position. Due to constraints on the total number of generations the algorithm is allowed to run, the simple GA would end up exploring only a few phases before reaching its termination. In the annealed GA, due to the annealing schedule on σ, convergence was not a big problem. Adopting a less stringent termination threshold can further expedite search.

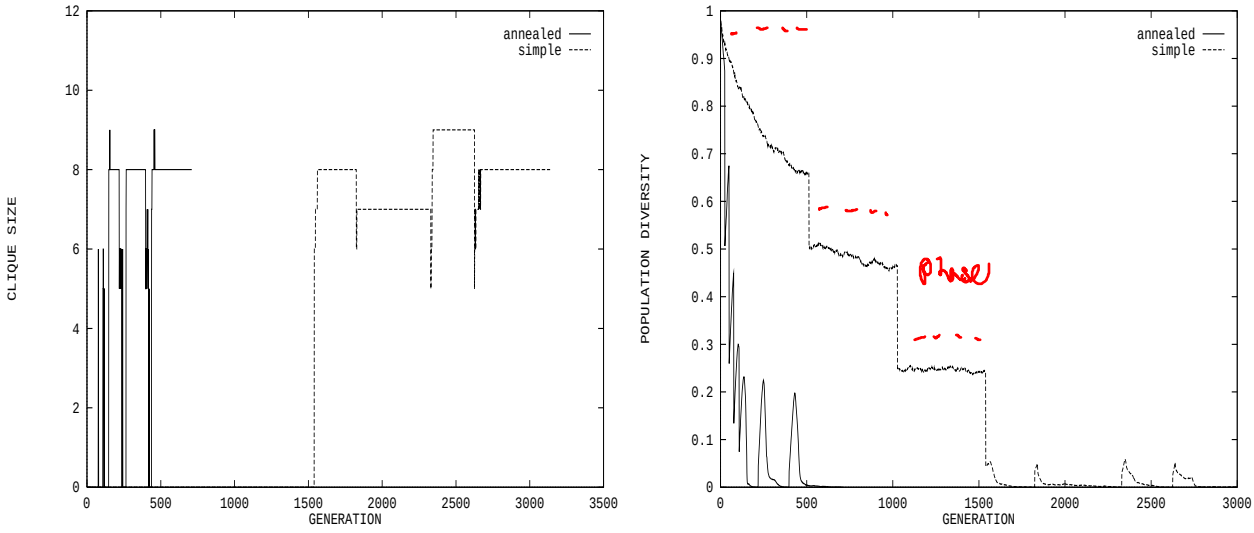


Figure 5: 171-node Keller graph. (a) Clique size found against number of generations. (b) Diversity against number of generations.

Figure 5 (left) shows the performance of the annealed and simple algorithms on a 171-node Keller graph with a known maximum clique size of 11. Both the annealed and simple genetic algorithms find a clique of size 9, but do not succeed in finding the optimal one. Figure 5 (right) shows the population diversity, div, plotted against generation count. As is evident, the annealed algorithm goes through all its phases within 500 generations. For this particular run, an additional termination condition was tested which prompted the premature termination of phases, without properly reaching the normal termination threshold $\theta = 0.0001$. The diversity plot of the simple GA reveals a typical convergence pattern observed in other runs, where the first three phases (until generation $\approx$ 1500) are fruitless and get terminated by the $3n$-rule. The four little convergence bumps correspond to the four "high-rise blocks" in figure 5 (left) where cliques are being found. Since during the period leading up to generation 1500 all elements have fitness value 0, reproduction is effectively turned off, and cross-over induced mixing is the only process at play. The first three "terraces" of the diversity graph show this phenomenon. The slope differences are due to variance and finite sample effects.

3 Benchmarking and further customization

3.1 Evaluation of the multi-phase annealed GA

Table 6 shows the performance of the multi-phase annealed GA on a set of benchmark problems contributed to the DIMACS Challenge. *Size* refers to the maximum clique size found, and *time* is the generation count until a termination condition (mostly "plateau" detection) is met. As is evident, oftentimes the algorithm ends up finding a solution that is less than 60% of the known optimal level. The numbers are a little bit misleading because, for instance, in the case of the Keller graph *keller5.clq*, a solution of up to 21 can be found by disabling the plateau detection mechanism. By incorporating a local search procedure, this can be facilitated much faster, but we felt it was not important since it corresponds to climbing the final stretch to a local minimum which is always doable using hill-climbing.

DIMACS CLIQUE BENCHMARKS							
File	Nodes	Edges	Clique Size	Annealed GA		Union GA	
				Size	Time	Size	Time
keller4.clq	171	9435	11	9	957	9	43
keller5.clq	776	225990	27	16	173	18	92
hamming8-2.clq	256	31616	≥ 79	94	261	80	119
hamming8-4.clq	256	20864	16	13	123	16	47
san400_0.5_1.clq	400	39900	13	7	53	7	3
san400_0.7_1.clq	400	55860	40	16	165	20	57
san400_0.7_2.clq	400	55860	30	12	163	15	17
san400_0.7_3.clq	400	55860	22	12	1800	12	18

Figure 6: DIMACS contributed benchmark table

The performance on three 1024-node random graphs can be seen in table 7. Again, the numbers fall below what is known to be accomplishable (roughly $10, 15, 25$ for the $0.3, 0.5, 0.7$ random graphs), and the algorithm is getting stuck in local minima. The numbers are from the most recent single data collection run. Averages are available for smaller graph sizes, and are currently being collected for the 1024 size. Due to the high time-complexity of the algorithm, data collection is a very time consuming process, even on a CM-5, requiring hours of CPU time. A large bottleneck is the time required to compute the fitness function itself which is $O(n^2)$. For exact computations, this cannot be avoided since a genetic algorithm is driven by the fitness value as the guiding bias in extracting structure from the problem instance. Even with bit-packing this turns out to be

1024 NODE RANDOM GRAPHS						
File	Nodes	Edges	Annealed GA		Union GA	
			Size	Time	Size	Time
0.3 random	1024	24866	6		7	28
0.5 random	1024	43269	12	341	11	51
0.7 random	1024	61550	19	314	18	84

Figure 7: Random graph benchmark table

the single most costly operation, and we are currently looking at using lower-cost approximate evaluations.

3.2 Union GA

The union GA is the most recent modification on the multi-phase annealed GA in an effort improve the algorithm's performance. It is characterized by three features, union cross-over, greedy replacement, and diversity enhancement which are described as follows.

- *Union cross-over.* Let $x, y \in \{0,1\}^n$. Let $I_x = \{i : x_i = 1\}$, and similarly for I_y. z is the union cross-over of x and y if $\forall\, i \in I_x \cap I_y$, $z_i = 1$, and $\forall\, i \in I_x \triangle I_y$, $z_i = 1$ with some probability.

- *Greedy replacement.* Elements are classified as republicans, peronistas, and democrats. x is a republican if it is a clique of sufficient size. x is a peronista if it is a near-clique of sufficient size. Otherwise, x is a democrat. The rules of replacement are: (a) A republican only replaces itself with a bigger republican. In so doing, it posts its old state onto a bulletin board. (b) A peronista replaces itself if the candidate is a republican. Otherwise, it attempts to prune itself into a republican. (c) A democrat always replaces itself unless the candidate is a democrat. If so, it checks the bulletin board to replace itself by a republican. If unsuccessful, it replaces itself if the candidate democrat has a higher fitness value.

- *Diversity enhancement.* Let x be a clique of size k. Let $\mathcal{H}_k$ be the subset of the population consisting of cliques of size k. If the diversity of $\mathcal{H}_k$ is low, then elements in $\mathcal{H}_k$ probabilistically replace themselves with the outcome of a random union cross-over operation.

Union cross-over is different from single-point cross-over described in section 2.1 in that there is a monotonicity property associated with it which may allow two cliques that are part of a larger clique

to merge more easily to form a larger clique. The single most distinguishing feature of a genetic algorithm lies in the cross-over operator by which two elements may combine to produce elements that are far apart in Hamming distance. The most direct way this operation can enhance search is if two subsequences of the parent strings ("building-blocks"), say nonoverlapping, are merged into a newly combined string, and ends up being a superior string with respect to the objective function. This process is called the *building-block hypothesis*, and Holland's schema theorem [7, 6] is a formalization thereof. For *monotonic* objective functions, it is possible to prove optimality of a genetic algorithm capturing a form of "biased mixing" [10]. In our experiments, we have noticed that even with encodings that try to assign labels to the vertices that preserve the neighborhood structure of the given graph, most cliques turn out to have labels that are spread out which is intuitively clear for random graphs. Therefore, two such cliques, if they are part of a larger clique, are more easily merged using the union cross-over operation.

3.3 Evaluation of the union GA

The union GA tested here is the multi-phase annealed GA incorporating the three new features minus the multi-phase mechanism. Instead of multi-phasing, which would not have allowed us to gather some benchmark data on time, the search is started from an initial population where the bits in the strings are set with very small probability. The result of this testing on the DIMACS Challenge subset is shown in table 6 of section 3.1. There is a small improvement in the maximum clique size found, for example, finding the optimal solution of 16 for the Hamming graph *hamming8-4.clq*. The main advantage lies in the speed-up in the number of generations needed to find the best solution. The present algorithm, which has not been tuned with respect to its parameters, is still getting stuck in local minima as can be seen from the table. Figure 8 shows the population dynamics of the union GA when solving the 776-node Keller graph. The plot shows the behavior of the algorithm over 500 generations where *count* refers to the histogram count of the number of cliques of a certain size at a given time instance. The algorithm was run without a plateau counter for a population size of 250 (very small compared to our usual $20n$ rule). It is seen that the algorithm has converged to a distribution whose maximum clique size is 18. The process by which a genetic algorithm approaches the MAX-CLIQUE problem can be visualized by the movement of the distibution peak to the right (higher clique size), initially starting at the left end. The

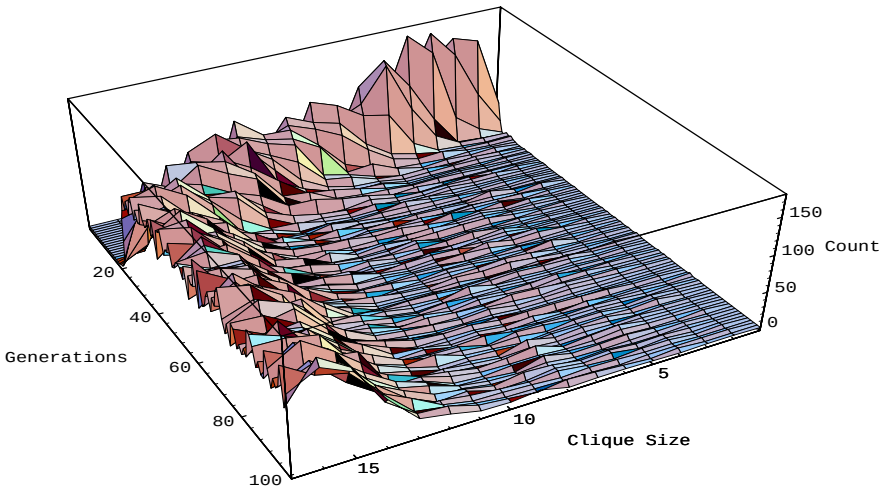


Figure 8: Population dynamics: time evolution of clique size histogram plot. 776-node Keller graph.

movement of the peak is fueled by the cross-over operator by which larger cliques are generated which in turn combine to generate even larger cliques. Ideally, this shift in the mass should continue until a maximum clique is found. As larger cliques are more rare, it is reasonable to expect for the height of the distribution to become dampened, and the rate of movement of the peak to slow down. The evolution shown in figure 8 gets stuck in a stationary distribution from which it has only a small probability of escaping. The conditional probability of finding an optimum element x given population $\mathcal{H}(t)$, $P_r\{x|\mathcal{H}(t)\}$, has also become stationary at a very small value, indicated by the fact that no improvements are seen over 400 generations. Figure 9 shows the performance of the multi-phase GA (top) against that of the union GA. The union GA finds the optimal clique 16 for this 256-node Hamming graph.

4 Parallel implementation issues

All results reported in this paper were produced using a Thinking Machines CM-5 using C*, a SIMD programming language. Our CM-5 consists of 128 processors divided into one 32-node, and two 16-node partions. A partition is the smallest available grouping of processors. Each partition is accessed through a partition manager which supervises the running of processes on the partition. C* allows one to program a very large number of virtual processors which are then emulated by the physical processors available on the CM-5.

The most straightforward way to implement a GA in SIMD style is to write from the point of view of an individual population element. In C*, an n-dimensional array of processors (called a *shape*) is defined with one conceptual data point at each element of the array. For a GA it makes the most sense to put the strings in a one-dimensional shape with one bit string at each processor along with whatever local data is needed to implement the GA.

There are several issues that must be addressed in a parallel implementation. A major issue with C* is the mapping of virtual processors to physical processors. We are relying on the system software to give an efficient mapping of populations of bit strings onto the available processors. Since general communication is required for the roulette wheel parent selection and there is no recurring pattern of communication among the bit strings, there is no advantage to be gained by using C*'s ability to specify the communication structure of a higher-dimensional shape without significant redesign of our implementation.

A more specific issue is the fast evaluation of the bit strings. Due to the $O(n^2)$ complexity of the fitness function evaluation, we decided that a more efficient implementation would pack the bit strings into 32-bit words. Since the adjacency matrix is also an array of bits packed into words, we can use the bitwise logical operators available from C* to compute the evaluation function more efficiently. Taking the view of each string, every row of the matrix is ANDed with the string and the number of bits in the resulting string is summed over all rows which are present in the string thereby computing the neighborhood size of a string. This is then normalized by the number of nodes in the string. The adjacency matrix is stored on the Partition Manager and is broadcast to all the processors as needed. The alternative of replicating the matrix in each processor would probably be faster but is only feasible if the matrix is small.

One important additional feature is the ability to take a checkpoint of the current state of the

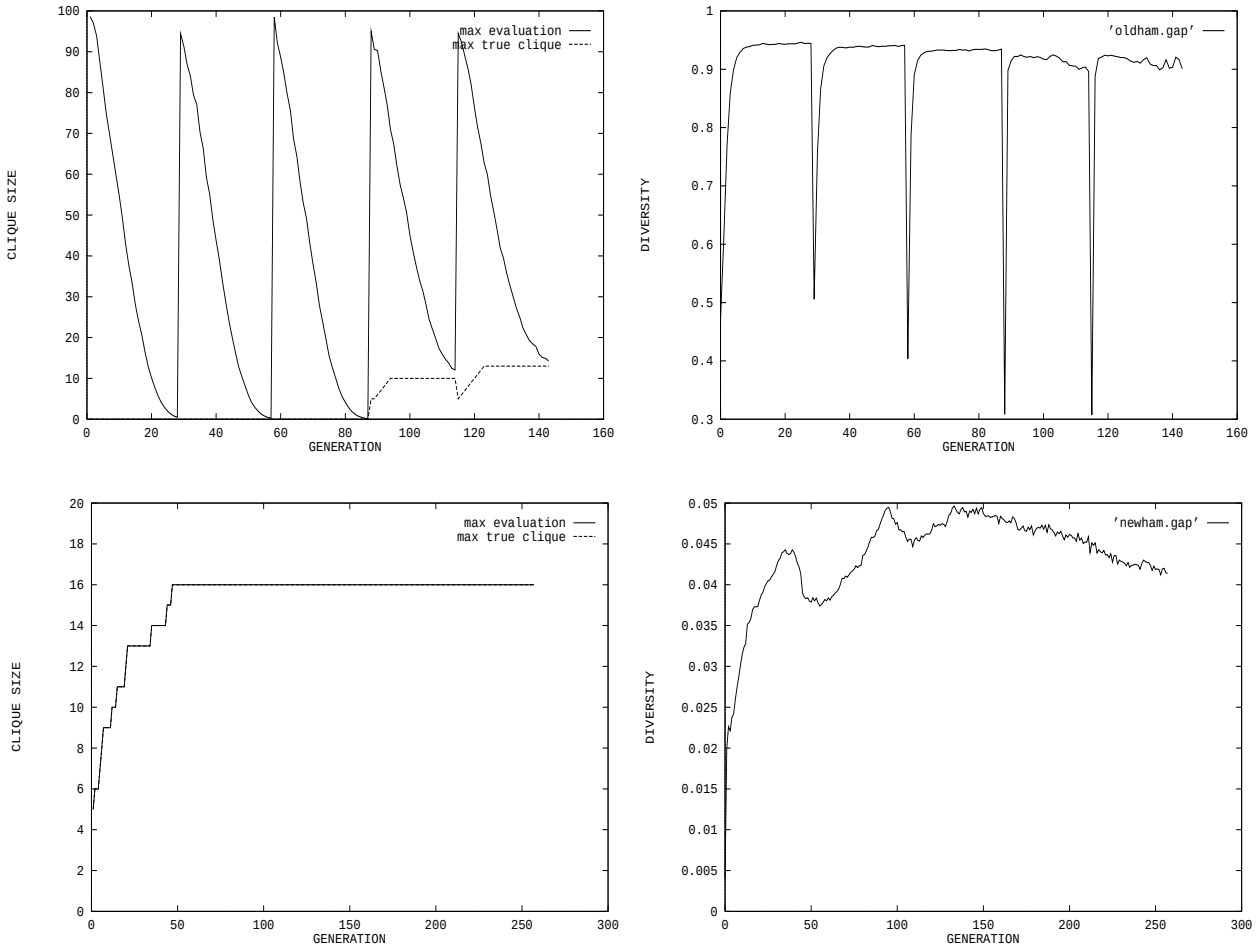


Figure 9: 256-node Hamming graph. Top: multi-phase annealed GA. Bottom: union GA.

GA. This enables long runs to be done in pieces and provides a degree of fault-tolerance. We take advantage of the Scalable Disk Array allowing parallel I/O for the storage of parallel variables.

Niching, which is a method for grouping a population into weakly coupled subpopulations, is easily done in C* using a segment bit to divide the set of processors into subsets during calculation of the running total and assigning niche boundaries at each processor. The number of niches can be varied over the duration of the run. For instance, "telescoping" from 20 down to 10 by merging adjacent niches into a niche of twice the original size. This process could continue until all strings belong to one fully-merged population

An alternative parallel implementation strategy in MIMD style could use CMMD, the message-passing library available on the CM-5. This might give increased performance and flexibility. For

instance, 32 subpopulations could be evolved independently with some schedule of interactions between subpopulations. Or perhaps a controller processor could feed tasks to many worker processors and thereby handle intermixing of the subpopulations.

Another simple enhancement is to allow "local" mating. One way to implement this is to let each chromosome conduct a pair of random walks within some radius to find its two parents. This is applicable to either the SIMD or MIMD style with the additional issue of whether inter-niche mixing should be allowed.

5 Conclusion

We have presented a preliminary study of the empirical power of genetic algorithms at solving the MAX-CLIQUE problem. Two new genetic algorithms, multi-phase annealed GA and union GA, were developed during the course of increasing performance, and they were tested against benchmark problems provided by the DIMACS Challenge as well as 1024-node random graphs.

Based upon our experience so far, we draw the following tentative conclusions. First, a GA needs to be heavily customized to have a chance to perform well. Second, a GA is computationally very expensive, and its usage seems justified only if it can be shown to outperform other general-purpose heuristics on particular clique problems. Third, our experience so far does not lead us to believe that genetic algorithms will outperform other algorithms, even though we are optimistic that some of them (e.g. simulated annealing) can be matched.

For "difficult" problems, we think that the building-block hypothesis may no longer apply, hence the potential speed-up obtainable by cross-over search may be nullified. Currently, we are in the process of completing tests for the union GA, extended to multi-phase, for a range of different population sizes. We hope to convey a more complete picture in the near future.

References

[1] S. Arora and S. Safra. Probabilistic checking of proofs: a new characterization of NP. In *Proc. 33rd IEEE Symp. on Foundations of Computer Science*, pp. 2-13, 1992.

[2] E. Balas and C. Yu. Finding a maximum clique in an arbitrary graph. *SIAM Journal on Computing*, 15(4):1054-1068, 1986.

[3] B. Bollobás and P. Erdös. Cliques in Random Graphs. *Mathematical Proceedings of the Cambridge Philosophical Society*, 80:419-427, 1976.

[4] K. De Jong and W. Spears. Using genetic algorithms to solve NP-complete Problems. In *Proc. 4th International Conf. on Genetic Algorithms*, pp. 124-132, 1991.

[5] U. Feige, S. Goldwasser, L. Lovasz, S. Safra, and M. Szegedy. Approximating clique is almost NP-complete. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pp. 2-12, 1991.

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

[7] J. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 1992.

[8] P. Jog, J. Suh, and D. Gucht. Parallel Genetic Algorithms Applied to the Traveling Salesman Problem. *SIAM Journal on Optimization*, 1(4):515-529, 1991.

[9] B. Manderick, M. de Weger and P. Spiessens. The genetic algorithm and the structure of the fitness landscape. *Proc. 4th International Conf. on Genetic Algorithms*, pp. 143-150, 1991.

[10] K. Park. A lower-bound result on the power of genetic algorithms. In *Proc. 5th International Conf. on Genetic Algorithms*, pp. 200-201, 1993.

[11] Y. Rabinovich and A. Wigderson. Analysis of a simple genetic algorithm. In *Proc. 4th International Conf. on Genetic Algorithms*, pp. 215-221, 1991.

[12] Y. Rabinovich, A. Sinclair and A. Wigderson. Quadratic dynamical systems. In *Proc. 33rd IEEE Symp. on Foundations of Computer Science*, pp. 304-313, 1992.

[13] D. Whitley, T. Starkweather, and D. Fuquay. Scheduling problems and traveling salesman: the genetic edge recombination operator. In *Proc. 3rd International Conf. on Genetic Algorithms*, pp. 133-140, 1989.