On the effectiveness of genetic search in combinatorial optimization*

Kihong Park†
park@cs.bu.edu

Bob Carter
carter@cs.bu.edu

BU-CS-94-010
November 10, 1994

Computer Science Department
Boston University
Boston, MA 02215

Abstract

In this paper, we study the efficacy of genetic algorithms in the context of combinatorial optimization. In particular, we isolate the effects of cross-over, treated as the central component of genetic search. We show that for problems of nontrivial size and difficulty, the contribution of cross-over search is marginal, both synergistically when run in conjunction with mutation and selection, or when run with selection alone, the reference point being the search procedure consisting of just mutation and selection. The latter can be viewed as another manifestation of the Metropolis process. Considering the high computational cost of maintaining a population to facilitate cross-over search, its marginal benefit renders genetic search inferior to its singleton-population counterpart, the Metropolis process, and by extension, simulated annealing. This is further compounded by the fact that many problems arising in practice may inherently require a large number of state transitions for a near-optimal solution to be found, making genetic search infeasible given the high cost of computing a single iteration in the enlarged state-space.

*A short version will appear in *Proc. 10th ACM Symposium on Applied Computing, Genetic Algorithms and Optimization Track*, February, 1995.

†Supported in part by NSF grant CCR-9204284

1 Introduction

Genetic algorithms [8], viewed as general-purpose optimization procedures, are increasingly being applied to a diverse spectrum of problem areas, ranging from protein folding to crew scheduling in the airline industry, to name a few [3, 5, 6, 9, 11, 12, 15]. Although research abounds, the jury is still out with respect to the utility of genetic search as a pure optimization technique. In part, this is due to the nonuniformity of problem instances which makes comparing results across different domains difficult. In other cases, the problem instances themselves are selected without specific regard to their "hardness," the problem sizes may be too small to render the reported results typical, or the merit of using genetic algorithms relative to other techniques is not explicated. To facilitate the productive application of genetic search to real-world problems, it is imperative that its power be delineated, providing an evaluation of potential benefits and costs, a sense of expected performance, and an overall ranking compared to other optimization techniques.

In this paper, we study the performance of genetic search and its components in the context of MAX-CLIQUE, the problem of finding the size of a maximum clique in a graph. MAX-CLIQUE is an NP-complete problem whose approximation problem has recently been proven to be NP-hard [1]. That is, in general, even finding approximate solutions may be inherently difficult. With this nontrivial problem domain as the backdrop, we proceed to isolate the contribution of cross-over in the search process, evaluating its effects on both "easy" and "difficult" problem instances. This is done by comparing the performance of the full search procedure (cross-over, mutation, selection), denoted $\mathcal{CMS}$, and a partial version (cross-over, selection), $\mathcal{CS}$, against $\mathcal{MS}$, the version that uses only mutation and selection. The latter turns out to be equivalent, in a qualitative sense, to the Metropolis process (simulated annealing at a fixed temperature) because both are exactly describable as time-homogenous Markov chains, high-fitness states are preferred but with stochastic backtracking enabled, and the next state is gotten by local perturbation. $\mathcal{CMS}$, the full genetic search, is also describable as a time-homogenous Markov chain, but with the notable difference that its next state is achieved via nonlocal means through cross-over.

This paper is an extension of previous work [4], where it is shown that for nontrivial problem instances above a certain size, the performance of genetic search degrades ungracefully as the problem size is increased. Two main causes are cited, one being the high computational cost which puts a strain on the number of generations of a GA that can be run, and the other being

the limited applicability of the building-block hypothesis to a range of problem instances, the feature that genetic search is conjectured to exploit. The *building-block hypothesis* [7] attempts to characterize a class of problems which have the property that if two good, disjoint subsolutions are suitably combined, then with nonnegligible probability an even better solution is obtained. There is rigorous evidence to indicate that genetic search is effective for problems where the building-block hypothesis (BBH) is readily applicable [13, 14], but it must be pointed out that problems satisfying this "superposition principle" are, by definition, only superficially nonlinear, and hence should be amenable to more efficient means of attack.

When dealing with problems stemming from real-world applications, there looms the perennial question of how difficult a class they actually represent. The answer may be highly problem dependent, and a satisfactory characterization difficult to come by. This paper will show evidence indicating that with respect to the utility of genetic search, one is faced with a no-win situation, independent of the problem at hand. That is, for "easy" problems where BBH holds true, finding a near-optimal solution is an inherently easy task, and other more efficient algorithms are preferable to the compute-intensive GA. For more "difficult" problems where BBH is not readily applicable, cross-over yields only a negligibly small probability of success, while turning into a burden by restricting the number of states that can be visited in the search space within reasonable resource bounds. Our focus on identifying and evaluating the contribution of the cross-over component in genetic search augments the first-approximation conclusions of [4]. Furthermore, the component-based treatment allows us to compare genetic search with simulated annealing (SA), leading to the following ordering relationships:

Quality of solution:

$$\text{Genetic Search}\,(\mathcal{CMS}) \quad \simeq \quad \mathcal{MS} \quad \prec \quad \text{Metropolis Process} \quad \preceq \quad \text{Simulated Annealing}$$

Time complexity:

$$\text{Genetic Search}\,(\mathcal{CMS}) \quad \prec \quad \mathcal{MS} \quad \prec \quad \text{Metropolis Process} \quad \simeq \quad \text{Simulated Annealing}$$

The quality of solution is based on a resource bound of several hours of workstation CPU time (as opposed to days) where "$\simeq$" denotes "on average equal" and "$\prec$" stands for "consistently better." Admittedly, the ordering relations are imprecise, but for the qualitative conclusions to be drawn, they will suffice. With regard to time-complexity, a rigorous definition can be given without an increase in unnecessary formalism. Time-complexity is measured as a function of the problem size

n and population size m, for computing a single iteration (or generation). Assuming m is subsumed in n via some functional relationship, $A \prec B \iff Time_B(n) = O(Time_A(n))$, where $Time_A$ and $Time_B$ are the time-complexity functions of A and B, respectively. That is, B is "faster" than A. Combining the previous orderings, we get

$$\text{Genetic Search} \quad \prec \quad \text{Simulated Annealing}$$

both quality-of-solution, and time-complexity wise. For the cross-over and selection only strategy, $\mathcal{CS}$, it turns out that with respect to time-complexity, $\mathcal{CMS} \prec \mathcal{CS} \prec \mathcal{MS}$, and with respect to the quality of solution, $\mathcal{CS} \prec \mathcal{CMS}$ and $\mathcal{CS} \prec \mathcal{MS}$. The time-complexity relationships are straightforward to establish, and the remainder of the paper will be concerned with establishing the quality of solution orderings[1].

This paper is organized as follows. In the next section, a description of the algorithmic and experimental set-up is given. This is followed by the main section which provides evidence for "genetic search $\prec$ simulated annealing." Augmenting that are sections describing the effects of population size on the search process and the long-term behavior of the different algorithms. We conclude with a discussion of the strengths and weaknesses of the present methodology, and its implication on the scope of applicability of our results.

2 Set-up

A genetic algorithhm consists of three operators *cross-over* ($\mathcal{C}$), *mutation* ($\mathcal{M}$), and *selection* ($\mathcal{S}$), each taking a multi-set of some fixed size m, *i.e.*, *population* $\mathcal{H}$, and producing a new population $\mathcal{H}'$ at the next time step. If we let $T = \mathcal{M}\,\mathcal{C}\,\mathcal{S}$ denote the composition, then one *generation* (or iteration) of genetic search is defined as

$$\mathcal{H}' = T(\mathcal{H}).$$

The dynamics of $(T^t(\mathcal{H}))_{t=0}^{\infty}$ depends on numerous factors such as the choice of parameters, the problem encoding scheme, and various auxiliary mechanisms including the use of penalty functions. The following is a brief description of the algorithmic set-up employed in the experiments. A more detailed description can be found in [3].

[1] In the quality of solution ordering, we will establish $\mathcal{MS} \prec \mathcal{SA}$ directly, without going through the Metropolis process. The strict ordering holds true even when the CPU time-bound allotted to $\mathcal{MS}$ is several factors greater than that of $\mathcal{SA}$.

- *Problem encoding.* An element of the population, representing a graph, was encoded in the straightforward way as a binary string x of length n, where $x_i = 1$ if and only if the ith node was present in the graph. A preprocessing step, based on the idea of permuting the vertex labels so as to group related nodes together was applied (*relatedness* was a function of the graph's connectivity), with little effect. We believe the positive results reported in [2] are mostly applicable to specially engineered graphs, having little bearing on enhancing cross-over in general.

- *Cross-over.* Several cross-over schemes were tried, ranging from two-point to uniform to more structured schemes. For MAX-CLIQUE, if two elements x and y encoding cliques are crossed over, say, via the two-point scheme, the probability that the resulting element remains a clique is very slim. We have approached this issue in two ways. One, by allowing noncliques to remain as members of the population and assigning them a fitness value based on a penalty function (see below), or two, disallowing noncliques altogether by employing structured cross-over schemes that preserve the cliqueness property. Our experiments have shown that the latter approach yields slightly improved solutions, and current results are based on this method.

- *Penalty functions.* It was observed that when admitting noncliques as population elements, the use of a penalty function which assigns a fitness value weighted by a measure of the degree of noncliqueness was beneficial to the search process. In this paper, the penalty function degenerates to returning the clique size as the fitness value.

- *Mutation rates.* Several mutation rates were tried, and except at extreme values, marked differences were not observed.

- *Diversification schemes.* To balance the detrimental effect of premature convergence, a host of diversification schemes were implemented, all based on the idea that even though the probability distribution in the selection step may dictate so, the production of exact duplicates was discouraged. One method was *probability distribution damping*, whereby the distance of the computed distribution from the uniform distribution was controlled by a parameter. This is easily achieved with very little overhead, and the closer to the uniform distribution, the less bias is exerted toward high fitness elements. More explicit schemes were

based on the idea of *bucket maintainance* where for a range of clique sizes up to the current maximum, intervention is exerted to preserve and produce elements representing smaller cliques. Otherwise, due to the low fitness value of small cliques, selection eventually weeds them out, leaving only large cliques. This in turn diminishes the cross-over success rate. In the parallel implementation (see below), a *niching* scheme was tried for explicit diversity maintainance. The population was evenly divided among 32 nodes of a CM-5 partition, with interaction among niches controlled by a coupling parameter. Niching, in our range, did not exhibit a noticeable improvement in the solution found.

- *Parallelization.* Parallel implementations were carried out on a Thinking Machines CM-5, using a 32-node batch partition, and two 16-node time-shared partitions. Two classes of parallelizations were implemented, one based on the C^* data parallel programming language resulting in a SIMD approach, and the other using the CM message passing library yielding a full-fledged MIMD approach. Both implementations yield similar solutions although the MIMD implementation is faster than the SIMD version allowing for more iterations. For population sizes below 1000, the speed-up achieved over a Sparc 1+ workstation was not pronounced, mainly due to the communication overhead and its associated synchronization penalty. To keep the parallelization issues separate, and because of the small speed-up achieved in the < 1000 population range, experiments performed on a set of six dedicated Sparc 1+ workstations are reported here.

- *Initial population make-up.* Judicial choice of the initial population affected the search process by speeding it up and, in some cases, resulting in better solutions. Although obvious in its potential, we were unable to find a rule that remained effective over different problem instances and problem sizes.

The various degrees-of-freedom that define a particular genetic algorithm and their effects were clearly observable for small problem instances (graphs with < 200 nodes), be they "easy" or "difficult" [3, 4]. It is for "large" and "difficult" instances where a marked degradation in performance relative to other algorithms is observed. For this class of instances, our previously successful methods of attack became ineffective. The following sections give an explanation of why we believe this to be an inherent problem, and not just a lack of ingenuity on the authors' part.

3 Effects of cross-over search

3.1 Simple problem

The simple problem instance was obtained by embedding a 52-node clique in a 700-node graph, with two mutually disjoint 21-node cliques partially overlapping the larger clique. A couple of high-degree nodes were added, and the remaining non-edges turned on with 0.1 probability to create further distractions. Finally, the vertex labels were randomly permuted to destroy adjacency correlations. The performance of $\mathcal{CMS}$ (genetic search), $\mathcal{MS}$, $\mathcal{CS}$, and simulated annealing (SA) are plotted as

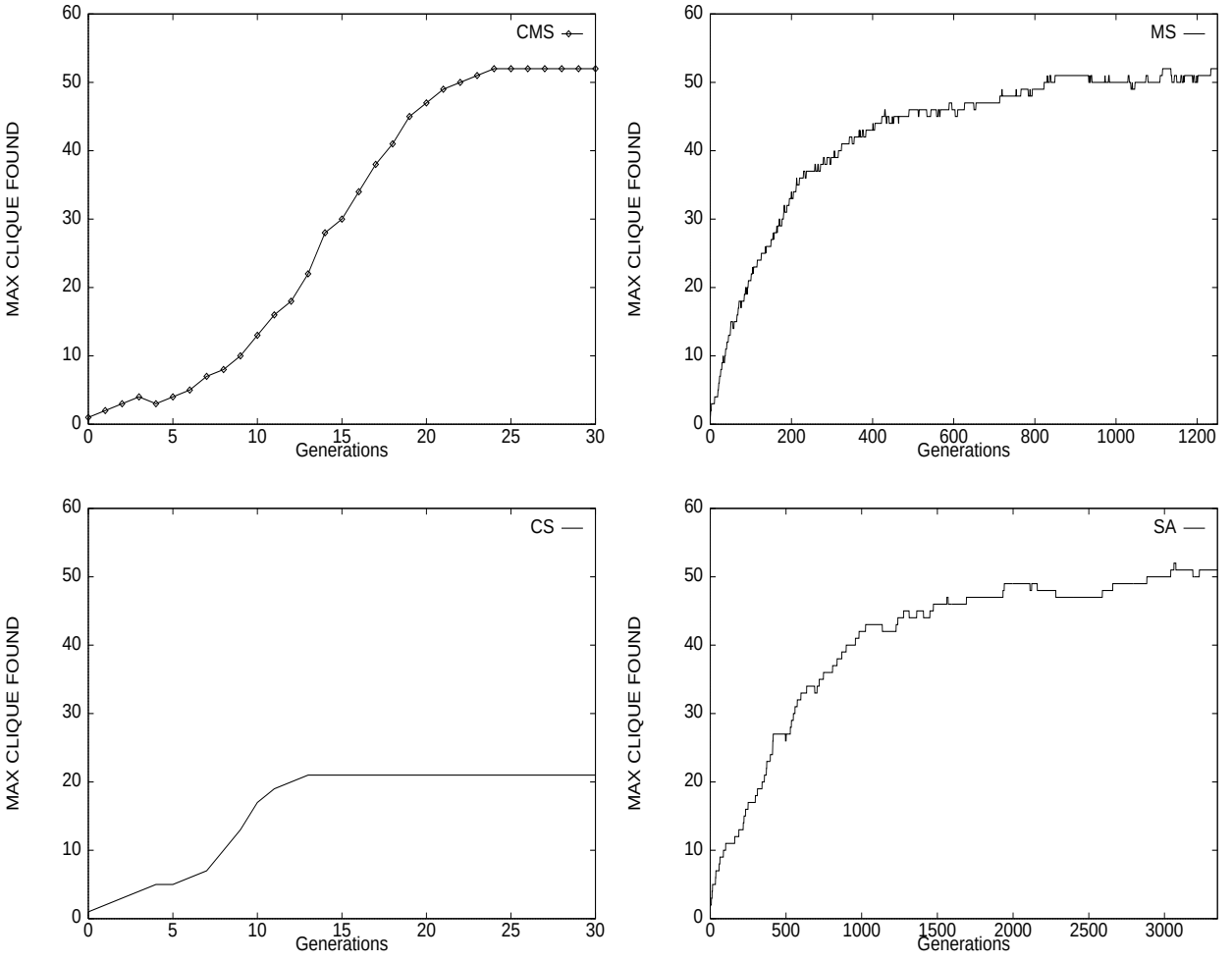

Figure 1: Performance diagrams for the simple graph. See section 1 for definitions of $\mathcal{CMS}$, $\mathcal{MS}$, $\mathcal{CS}$, and SA.

functions of time and shown in figure 1. The population size was fixed at 300. Since the building-

block hypothesis is well applicable to this problem instance, cross-over is able to speed up the search process by facilitating large-step improvements, finding the maximum clique at generation count 24. $\mathcal{MS}$, being limited to small-perturbation search, needs more generations to converge to its plateau, finding the optimal solution at generation 1115. A garden-variety simulated annealing algorithm with a linear annealing schedule also exhibits an exponential convergence to its plateau, reaching the maximum clique at iteration 3063. Due to the different time-complexities associated with $\mathcal{CMS}$, $\mathcal{MS}$, and simulated annealing for evaluating a single iteration[2], the iteration count needs to be adjusted by its time-complexity function. The actual CPU-time expended to find the optimal solution for each of the three algorithms is shown in table 1. Both genetic search and

Graph	$\mathcal{CMS}$	$\mathcal{MS}$	SA
Simple	1 min (52)	11 min (52)	1 min (52)

Table 1: CPU-time in minutes until optimal solution is found

simulated annealing require 1 minute to find the optimal solution whereas $\mathcal{MS}$ requires 11 minutes of CPU time. The point to note here is not so much the difference in absolute time (they are already smallish, including $\mathcal{MS}$), but the qualitative behavior of the search process which in all three cases is characterized by an exponential convergence to a plateau at which the optimal solution is quickly found. That is, for easy problems, convergence to the optimal solution occurs early on in the plateau, and cross-over only serves to further narrow the transient stage, already characterized by geometric convergence, making its length inconsequential. Cross-over with selection alone, $\mathcal{CS}$, is not a fruitful search procedure, as seen in figure 1. $\mathcal{CS}$ gets stuck at clique size 21, and does not improve over 10000 generations.

It should be stressed that for "simple" problem instances where the building-block hypothesis is well applicable, genetic search ($\mathcal{CMS}$) is effective even for large problem instances. Figure 1 clearly shows the way in which cross-over contributes to the search process, namely, by merging a pair of "good" solutions to yield an even better one. It is for "large," "difficult" problem instances that nonlinearity kicks in as a dominating factor[3], and the cost of cross-over will far outweigh its marginal benefit.

[2]Note, the time-complexity ordering of a single iteration is given by $\mathcal{CMS} \prec \mathcal{MS} \prec$ simulated annealing.

[3]In essence, BBH is just another way of capturing the difference between linear and nonlinear problems. The simple problem instance, even though it is called "simple," is not trivially so by construction.

3.2 Difficult problems

The first difficult problem instance is a 776-node Keller graph [10], belonging to the DIMACS combinatorial optimization benchmark set. Figure 2 shows the the time evolution of $\mathcal{CMS}$, $\mathcal{MS}$,

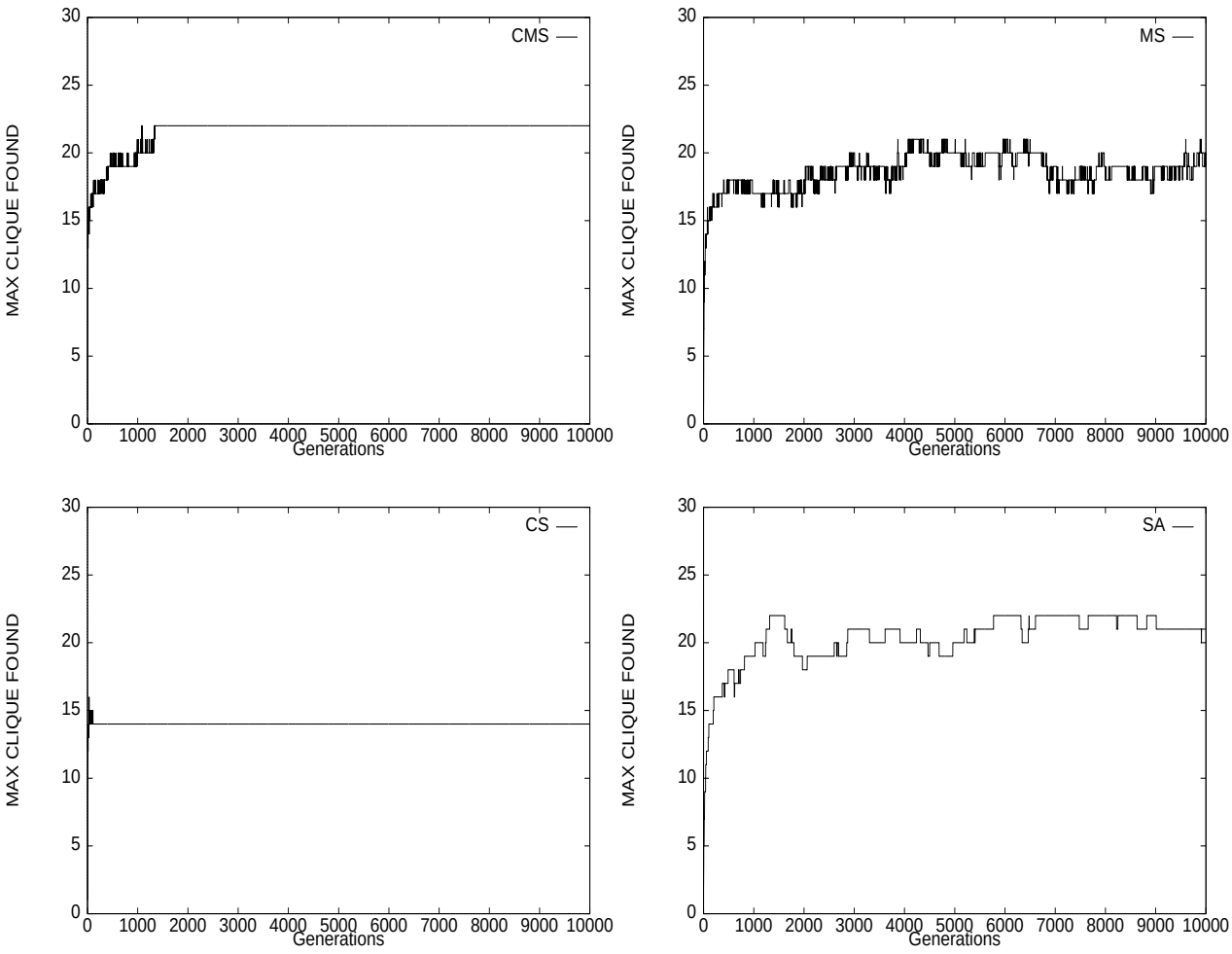

Figure 2: Performance diagrams for Keller graph.

$\mathcal{CS}$, and simulated annealing, respectively. None of the four algorithms finds the optimum clique size 27 within 10000 generations. As before, the search procedures are characterized by an exponential convergence to a plateau, followed by gradual changes. For this problem instance, BBH is not readily applicable, and the cross-over component in genetic search is of little value. That is, cross-over does not add any more power as we go from $\mathcal{MS}$ to $\mathcal{CMS}$. The performance of $\mathcal{CS}$ remains dismal, and will not be further illustrated in the paper.

The CPU-time to perform 10000 generations is shown in table 2. There is a factor 3.3 difference

between $\mathcal{CMS}$ and $\mathcal{MS}$, a factor 16.4 difference between $\mathcal{MS}$ and simulated annealing, and a factor 55.0 difference between $\mathcal{CMS}$ and simulated annealing. As we will see in the next section, it takes

Graph	$\mathcal{CMS}$	$\mathcal{MS}$	SA
Keller	385 min (22)	115 min (21)	7 min (22)
Random	495 min (12)	134 min (12)	4 min (12)
Sanchis	669 min (87)	105 min (87)	7 min (175)

Table 2: CPU-time in minutes for 10000 generations

at least several hundred thousand generations for the optimal solution to be found, and thus the costly nature of $\mathcal{CMS}$ places a severe burden on the number of generations that can be run with limited resources. For nontrivial problems, the cost-per-iteration (CPI) is an all-important measure, and we have

$$CPI_{simulated\ annealing} \quad \ll \quad CPI_{genetic\ search}.$$

It is this feature coupled with the ineffectiveness of the cross-over component for nontrivial problem instances which makes genetic search a questionable optimization technique.

Figure 3 depicts a more detailed, internal view of the $\mathcal{CMS}$ search process for the Keller graph. We noted in section 2 that bucket-based diversification schemes were used to combat premature convergence. Figure 3, top, shows the time-evolution of clique size distributions (left), and cross-over improvement distributions (right), when such a bucket scheme was active, corresponding to the $\mathcal{CMS}$ run of figure 2. In the clique histogram graph, we see the peak of the distribution concentrated near the current maximum clique size, the whole distribution shifting in time as larger cliques are found. After the transient period, the distribution levels off at a plateau and remains qualitatively invariant for subsequent generations. Due to the diversification scheme, smaller clique sizes remain part of the distribution, even after the plateau at clique size 22 has been reached. The busy activity in the cross-over improvement histogram shows that small step improvements are more likely than large ones, with some improvement steps occuring in the range of 8 and above. The cross-over improvement histogram also stays qualitatively invariant in the long run.

Contrast this to figure 3, bottom, which shows the same statistics for a $\mathcal{CMS}$ run employing only probability distribution damping. Although the clique histogram evolution shows a similar shift in the peak of the distribution toward large clique sizes leveling off at a plateau of 20, note the absence of small cliques after the transient period, which in turn manifests itself as a lack of activity in the clique improvement histogram. In either case, with or without explicit diversity

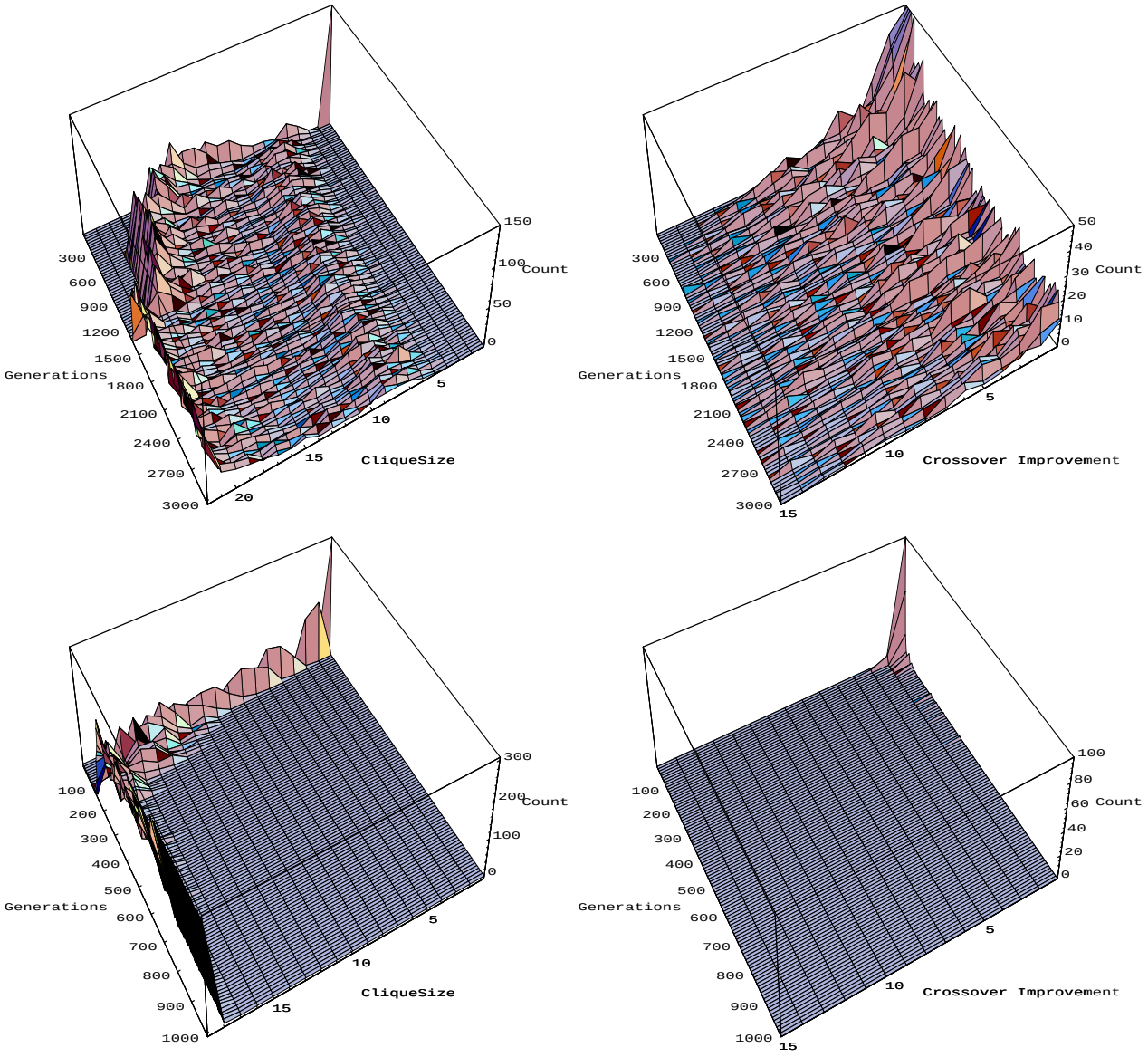

Figure 3: Clique size histogram and cross-over improvement histogram for the Keller graph. Top: with bucket diversity scheme. Bottom: without bucket diversity scheme.

intervention, cross-over is not able to produce improved solutions, indicating its ineffectiveness in the 776-node Keller graph and other nontrivial problem instances considered in the paper. Figure 4 shows a population diversity plot of the $\mathcal{CMS}$ and $\mathcal{MS}$ runs of figure 2. Both show the diversity

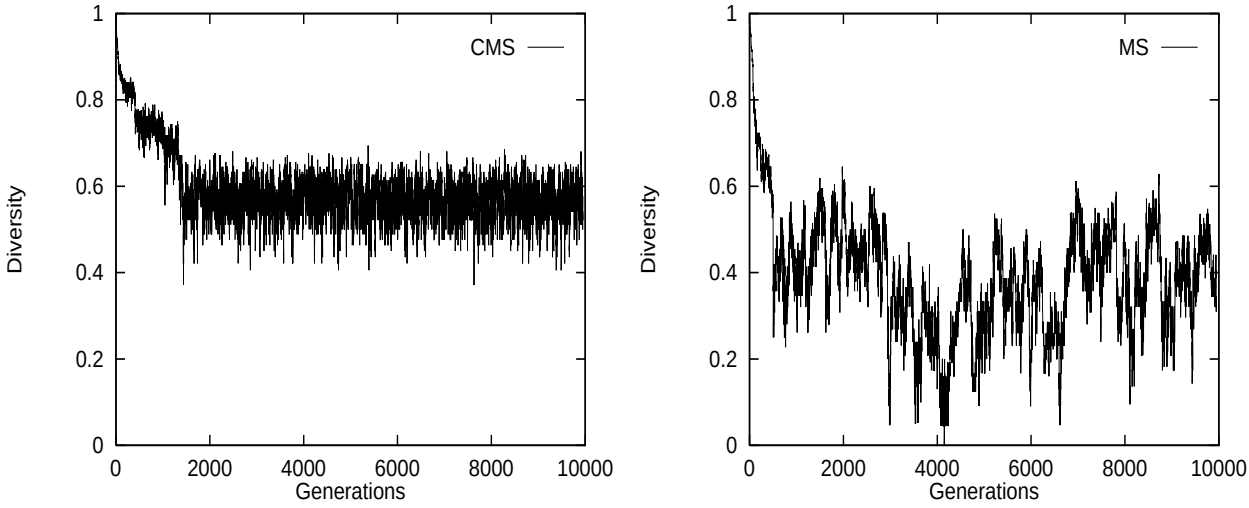

Figure 4: Population diversity diagrams for the Keller graph.

being maintained in the long run, $\mathcal{CMS}$ at a higher level than $\mathcal{MS}$, as expected.

Figure 5 shows the performance runs of $\mathcal{CMS}$, $\mathcal{MS}$, and simulated annealing on a 1024-node random graph (left column), and a 700-node Sanchis graph (right column), also part of the DIMACS benchmark set. The random graph has a maximum clique size of ≥ 14, and the performance plots show that all three algorithms find a clique size 12 within 10000 generations. In the long-term behavior section, we will see a more pronounced difference over many more iterations. In the case of the Sanchis graph which has maximum clique size 175, both $\mathcal{CMS}$ and $\mathcal{MS}$ get stuck at a local minimum near 87 when run for 10000 generations, whereas simulated annealing is able to find the optimum clique size. It is not clear why simulated annealing does better from the outset, this being the behavior observed over several runs. The CPU-time needed to run the three algorithms for the previous two graphs is shown in table 2. As with the Keller graph, there is a big difference in the associated time-complexities, making it infeasible to run genetic search for a large number of generations. This, in conjunction with the ineffectiveness of the cross-over component to cut down the generation count needed to find a near-optimal solution, implies "genetic search $\prec$ simulated annealing."

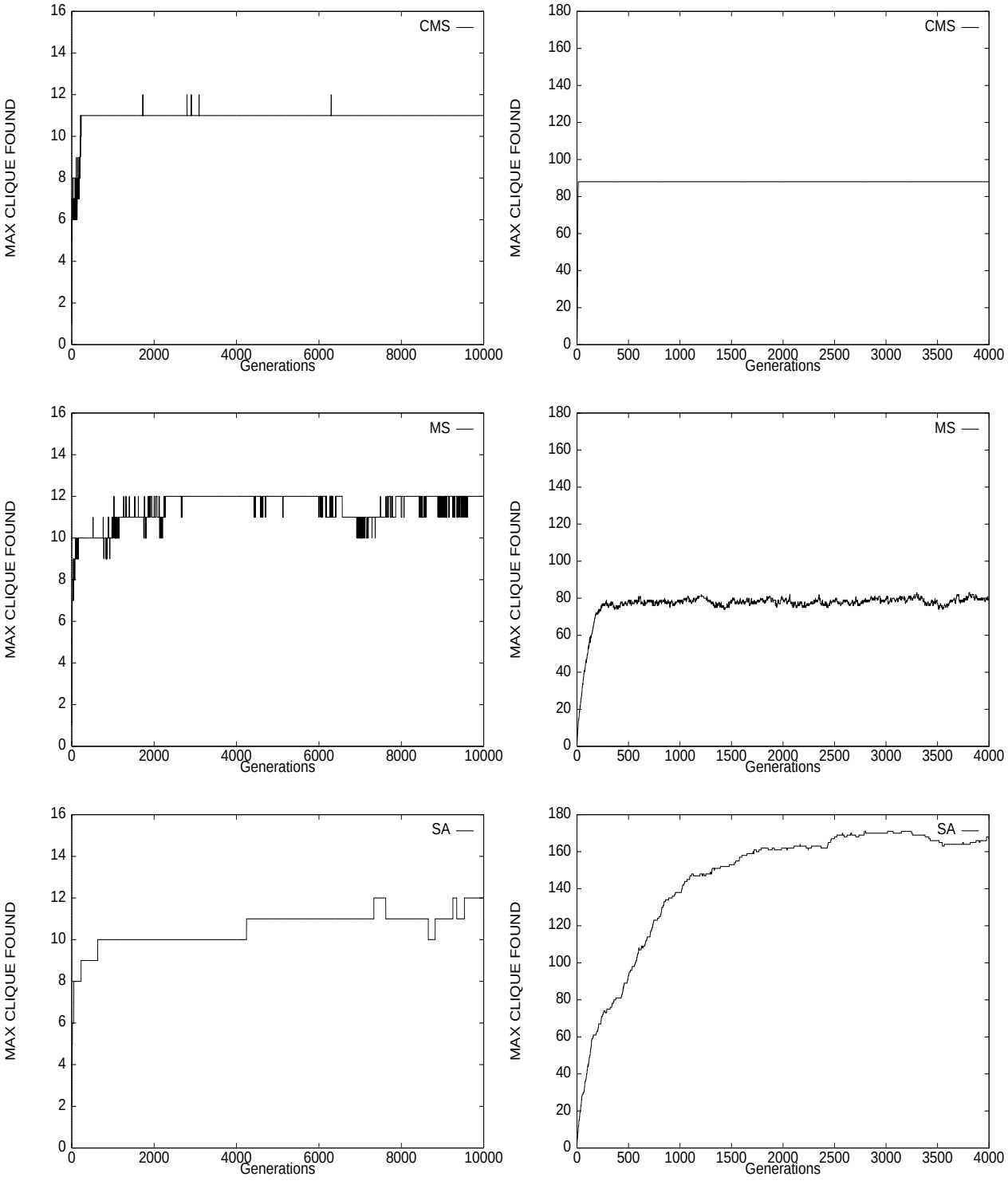

Figure 5: Performance diagrams. Left column: random graph. Right column: Sanchis graph.

4 Long-term behavior and population effects

4.1 Long-term behavior

Figure 6 shows the long-term behavior of $\mathcal{CMS}$, $\mathcal{MS}$, and simulated annealing over a large number of iterations. The left column shows the runs for the Keller graph over 150000 generations. Simulated annealing is able to find the maximum clique size 27 at iteration 143368 taking 1.6 hours of CPU-time to do so. On the other hand, $\mathcal{MS}$ finds clique size 25 at generation 118306, but requiring 21.2 hours to do so. Even when run for 300000 generations, at the expense of 53.7 hours of CPU-time, 25 remains the maximum clique size found. $\mathcal{CMS}$, due to its high CPI, could be run for only 150000 generations, finding a maximum clique size of 24 at 5.1 hours, and expending a total of 96.2 hours of CPU-time over its allowed maximum iteration limit.

The right column shows the runs for the 1024-node random graph, over 80000 generations. Simulated annealing finds clique size 14 at iteration 25591 taking 0.5 hours of CPU-time to do so. $\mathcal{MS}$ finds clique size 13 at generation 12522 requiring 2.7 hours of CPU-time. Even over 300000 generations, at the cost of 64.5 hours of CPU-time, 13 is the maximum clique size $\mathcal{MS}$ finds. $\mathcal{CMS}$ finds clique size 13 at generation 14026 using 16.9 hours of CPU-time. When run for the full 80000 generations, it still finds only size 13 at a cost of 96.7 hours of CPU-time.

4.2 Population size effects

Figure 7 shows the effect of using different population sizes on the quality of solution found for $\mathcal{CMS}$ and $\mathcal{MS}$ over 10000 generations in the case of the Keller graph. We believe that increasing the population size has the beneficial effect of allowing for a potentially more diverse population pool, which in turn may increase the success probability of cross-over search. In the ≤ 1000 population size range, no pronounced correlation is observed to indicate that cross-over empowers $\mathcal{CMS}$ over $\mathcal{MS}$. Much larger population sizes up to 131072 were run on the CM-5 parallel implementation for $\mathcal{CMS}$ alone, showing a small improvement in the quality of solution found, but at a huge increase in computation cost [4]. Our experience leads us to believe that the cross-over success rate, and hence its ultimate utility, is a slowly increasing function of population size (assuming diversity is properly maintained), and thus within reasonable resource bounds, significant benefits may not be achieved.

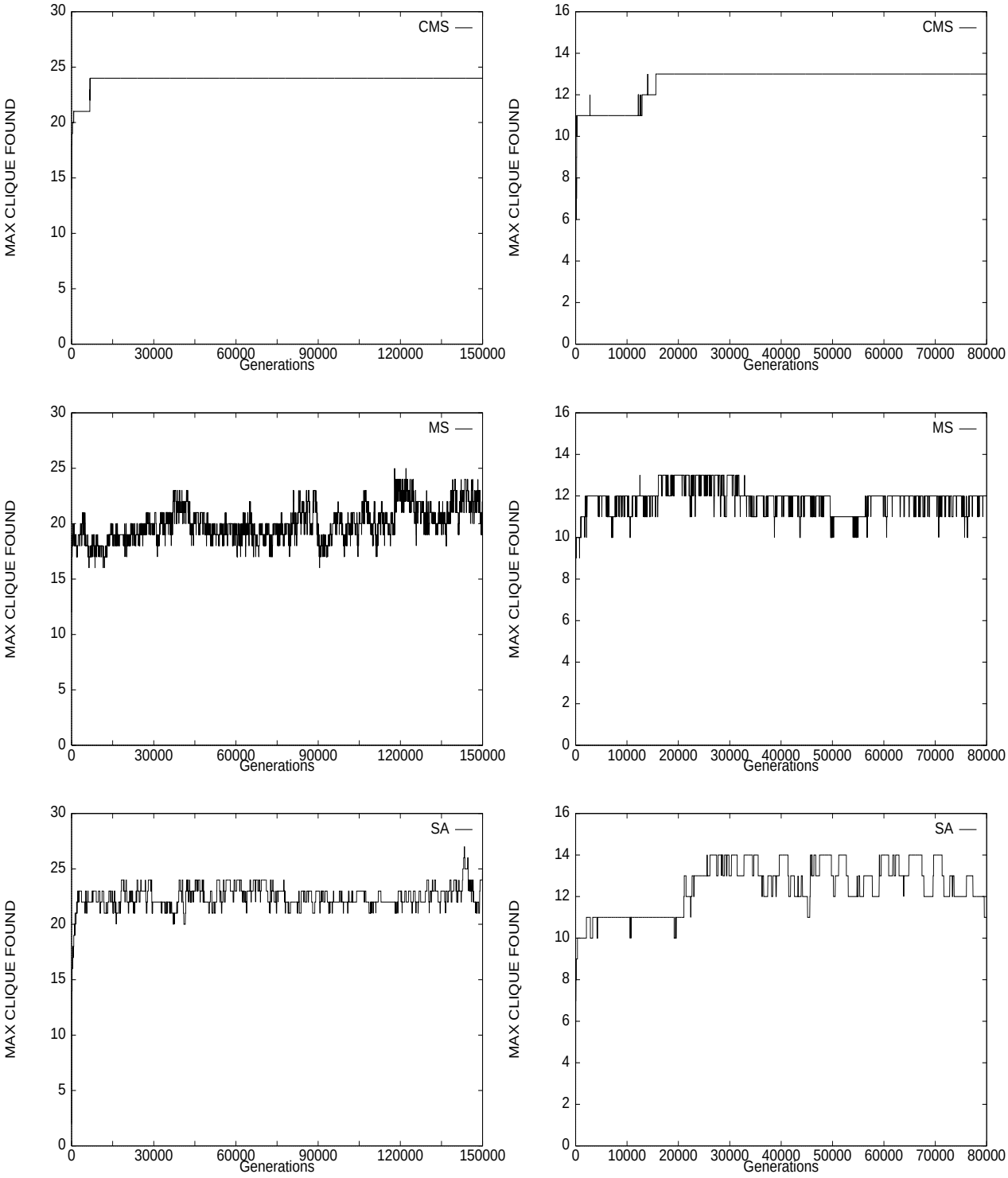

Figure 6: Long-term performance diagrams. Left column: Keller graph. Right column: random graph.

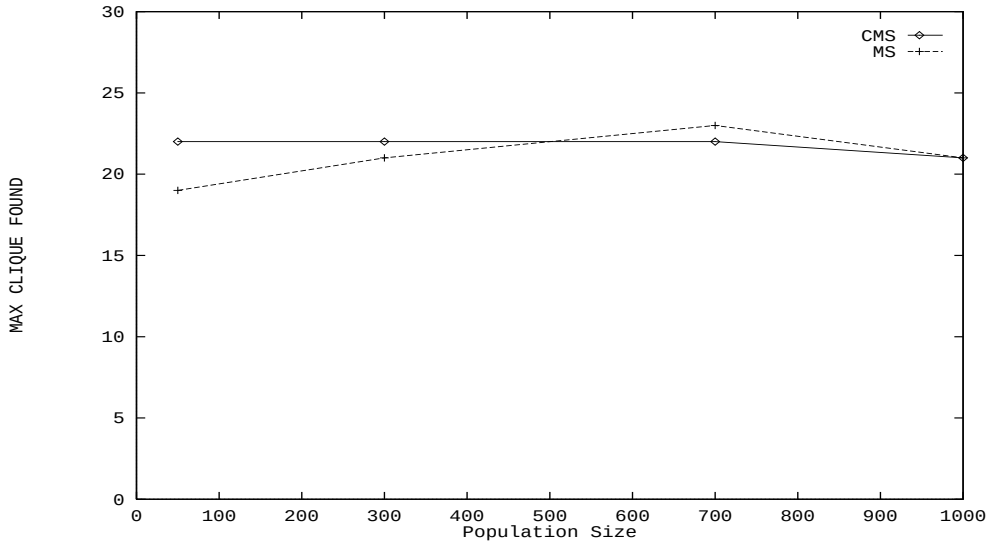

Figure 7: Effect of population size on performance for Keller graph.

5 Discussion

We have presented an investigation of the utility of genetic search as it pertains to the synergistic and singular effects of cross-over as the distinguishing search component. We have shown evidence to indicate that cross-over contributes only marginally to the search process, in effect, becoming a severe computational burden for nontrivial problem instances where the building-block hypothesis may not hold.

It is unclear how much of the reported prowess of genetic algorithms is due to cross-over being an indispensable component, or if the same results are achievable in the absence of cross-over, running mutation and selection as an enlarged Metropolis process. If the latter is true, then the ordering relationship established between genetic search and simulated annealing, $\mathcal{CMS} \prec \mathcal{SA}$, implies that in the context of strict combinatorial optimzation, genetic search is inferior to simulated annealing in a well-defined sense. If P $\neq$ NP, and assuming this to have some ramification on the inherent difficulty of real-world problems[4], then no matter what algorithm is being employed, difficult problems may require large amount of resources to be solved. If one is faced with such a situation and there are no short-cuts known, then the order relationship indicates that simulated

[4]To be precise, an assumption on probabilistic algorithms and their complexity hierarchy needs to be made, but the spirit of the statement should be clear.

annealing may still be preferable to genetic search for seeking a good approximate solution.

It is inherently more difficult to demonstrate negative than positive results experimentally since the former is a statement over the whole space of candidate algorithms, whereas the latter only involves exhibiting the existence of one. The present work is far from exhaustive to extrapolate our conclusions in general. In fact, this is the next order of business. In the mean time, what this paper does give, beyond the evidence and point-of-view on the utility of genetic algorithms as an optimization technique, is a diagnostic tool that may be of practical use in applications. This is so since it is easy to decouple the cross-over component and compare $\mathcal{CMS}$ against $\mathcal{MS}$ on a given problem domain. If their performance are comparable, then this suggests that using simulated annealing may be more effective.

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] T. Bui and B. Moon. Hyperplane synthesis for genetic algorithms. In *Proc. 5th International Conf. on Genetic Algorithms*, pp. 102–109, 1993.

[3] B. Carter and K. Park. How good are genetic algorithms at finding large cliques: an experimental study. Technical Report BU-CS-93-015, Computer Science Department, Boston University, 1993.

[4] B. Carter and K. Park. Scalability problems of genetic search. In *Proc. IEEE International Conference on Systems, Man and Cybernetics*, pp. 1591–1596, October, 1994.

[5] L. Davis (ed.). *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.

[6] K. De Jong and W. Spears. Using Genetic Algorithms to Solve NP-Complete Problems. In *Proc. 3rd International Conf. on Genetic Algorithms*, pp. 124–132, 1993.

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

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

[9] 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.

[10] J. Lagarias and P. Shor. Keller's cube-tiling conjecture is false in high dimensions. *Bulletin of the American Mathematical Society*, 27(2):279–283, 1992.

[11] D. Levine. A genetic algorithm for the set partitioning problem. In *Proc. 5th International Conf. on Genetic Algorithms*, pp. 481–487, 1993.

[12] H. Mühlenbein, M. Gorges-Schleuter and O. Krämer. Evolution algorithms in combinatorial optimization. *Parallel Computing*, 7:65–85, 1988.

[13] K. Park. A lower-bound result on the power of genetic algorithms. In *Proc. 5th International Conf. on Genetic Algorithms*, pp. 651, 1993.

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

[15] R. Unger and J. Moult. Genetic algorithms for 3D protein folding simulations. In *Proc. 5th International Conf. on Genetic Algorithms*, pp. 581–588, 1993.