A Study of Genetic Algorithms to Find Approximate Solutions to Hard 3CNF Problems

Jeremy Frank

frank@cs.ucdavis.edu

(916) 758-5925

Division of Computer Science
University of California at Davis
Davis, CA. 95616

June 3, 1994

Abstract

Genetic algorithms have been used to solve hard optimization problems ranging from the Travelling Salesman problem to the Quadratic Assignment problem. We show that the Simple Genetic Algorithm can be used to solve an optimization problem derived from the 3-Conjunctive Normal Form problem. By separating the populations into small sub-populations, parallel genetic algorithms exploits the inherent parallelism in genetic algorithms and prevents premature convergence. Genetic algorithms using hill-climbing conduct genetic search in the space of local optima, and hill-climbing can be less computationally expensive than genetic search. We examine the effectiveness of these techniques in improving the quality of solutions of 3CNF problems.

1 Introduction

The Genetic Algorithm proposed by Holland [Ho] has become the basis for a new approach to optimization problems. In a genetic algorithm, sets of solutions called populations are succeeded by new solutions which are closer to the desired optima. Successive generations are created by selecting solutions from the previous generation using a fitness function related to the objective function of the optimization problem. The new population is then subject to genetic operators such as crossover and mutation to extend the search to other parts of the search space [Go]. Genetic algorithms (hereafter called GAs) have been tested on a wide variety of problems, including the combinatorially difficult problems in the class NP. A typical NP-Hard problem is the Boolean Satisfaction Problem, where we desire an assignment of boolean variables which satisfies a boolean formula. One version of the Boolean Satisfaction problem is called the 3-Conjunctive Normal Form problem (3CNF) in which conjunctions of 3 distinct variables are disjoined in a formula. The optimization problem for 3CNF requires maximizing the number of true terms (conjunctions) in the formula, and is NP-complete.

$$(\neg X \vee \neg Y \vee \neg Z) \wedge (X \vee Y \vee \neg V) \wedge (X \vee \neg Y \vee \neg Z) \wedge (U \vee V \vee W)$$

Figure 1: A 3CNF problem with 4 terms and 6 variables.

Figure 1 shows a sample problem of this type.

Backtracking procedures are used to find exact solutions to 3CNF problems. Recent research has shown that, for randomly generated formulas, if the ratio of terms to variables is close to 4.3 then these problems are the most difficult to solve [MiSe]. Greedy local techniques, used to solve the optimization version of 3CNF, show improvement over even the best backtracking methods [Se], [SeKa]. However, there are problems which can fool greedy algorithms into finding local optima unless the algorithms are specially modified [SeKa]. This brings to mind the so-called "deceptive" functions which are shown to fool GAs into finding suboptimal solutions [Mu].

To improve the performance of GAs on deceptive problems researchers have introduced parallel genetic algorithms, (referred to generally as PGAs). PGAs are designed to enhance genetic diversity by separating sub-populations. In some PGAs individuals are separated and only allowed to mate with certain neighboring individuals [Mu2], while in other algorithms sub-populations are allowed to exchange individuals after a certain time period has elapsed [Ta]. To improve some parallel algorithms, hill-climbing is introduced. Performing crossover on local optima gives better results than crossover on other points, and in cases hill-climbing improves individuals more quickly than the genetic operators [MuSc].

In this paper we will construct several GAs to solve the optimization version of the 3CNF problem. We shall analyze these GAs and investigate the settings of parameters such as crossover probability, mutation probability, migration rate, migration interval and population parameters. We shall use a variety of performance measures to analyze the results of our algorithms. For the test suite, we propose to use randomly generated "hard" 3CNF problems of different sizes.

2 The Genetic Algorithm and 3CNF

To build a simple genetic algorithm (SGA) to solve the 3CNF problem we first investigate the encoding. A natural encoding is a string of 0s and 1s indicating the truth assignment to each boolean variable indicated by the position of the bit in the string. The fitness function we use in this case is exactly the function to be optimized, the number of true terms for a particular assignment. Crossover and mutation naturally follow from such an encoding and objective function. For the problem in Figure 1 a sample chromosome for $UVWXYZ$ is 100110; all terms in the problem evaluate to true using this assignment.

2.1 The Hill-climbing Genetic Algorithm

In the Hill-climbing Genetic Algorithm (HGA), hill-climbing is done before crossover and mutation; thus the HGA can be considered to be a genetic algorithm operating in the space of local optima for a given problem. It should be pointed out that there may still be an exponential number of optima in the search space. In cases where it is important to find as many local optima as possible, this method guarantees finding only optima and attempts to find better optima using the genetic operations. It may also be less expensive to use hill-climbing to do some of the work of the GA, thereby reducing the population size and/or the number of generations required to converge [Mu2]. For our experiments we shall use steepest ascent hill-climbing; among the options to be explored later are tabu search [Gl], simulated annealing and greedy local search [Se].

2.2 The Parallel Genetic Algorithm

Genetic algorithms are just as susceptible to premature convergence as other search procedures. To address this problem, genetic algorithms induce diversification throughout the lifetime of the algorithm using parallel populations which exchange individuals for genetic crossover [Mu]. These algorithms are known as parallel genetic algorithms or PGAs. Wright argues that for PGAs to avoid local optima a PGA should be divided into isolated sub-populations, one of which will eventually take over the others [Mu]. We take this view in designing a PGA to solve 3CNF problems. Accordingly, the PGA is divided into sub-populations arranged in a ring [Mu]. Each subpopulation sends a small number of randomly picked individuals to it's neighbors after a number of generations [Ta]. These individuals randomly replace members of the populations they are sent to. By keeping the rate of migration small and the intervals of migration reasonably large, we maintain diversity in the sub-populations, but allow opportunities for improvement by exchanging individuals. Other schemes include elitism in choosing migrants and reverse elitism in choosing the individuals to replace in the new population. This algorithm can be modified by allowing hill-climbing.

3 Analysis

Our goal is to find the best GA configuration to solve 3CNF problems. We ran a variety of experiments varying each of the GA parameters available and measured the value and variance of the optima found on randomly generated problems. We tested problems of 25 variables. We tested the GAs on 100 randomly generated "hard" CNF problems [ChKa]. In order to analyze our results we measured the average number of terms solved by each algorithm. In many cases this average is skewed by problems which have satisfying assignments, so we also recorded the number of problems solved by each algorithm.

3.1 Population Size and Generations SGA

Goldberg [Go2] presents a method of sizing populations and determining the number of generations to convergence for serial and parallel GAs. We used this work to find the optimum number of generations for SGA using both the model of constant time convergence and logarithmic time convergence. For these experiments we fixed the probability of crossover at 20% and the probability of mutation at 2%. Figure 2 shows the results of this exper-

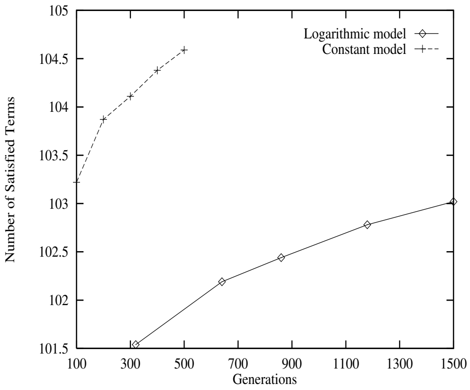

Figure 2: Performance of GAs under constant and log models.

iment. The logarithmic model requires only 4 members, and convergence time is logarithmic in the number of variables. In the constant model, the population size is 40, with the time to convergence constant. We see that the performance of the constant model is superior to that of the logarithmic model. For the remainder of the experiments we used 300 generations and 40 population size. It is interesting to note that the performance difference may be due to the population disparity, and should be the subject of future work.

3.2 Crossover and Mutation for SGA and HGA

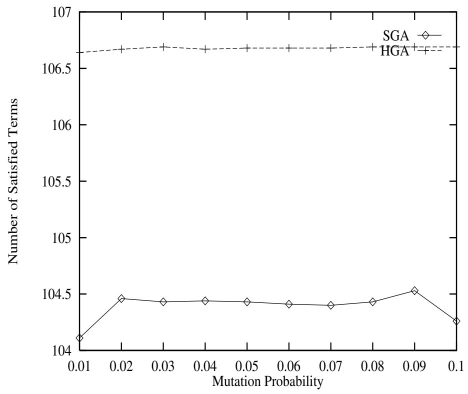

Figure 3: The Effect of Mutation on SGA and HGA.

We examined the effects of crossover and mutation on the performance of SGA and HGA. We varied the crossover probability from 10% to 100% by 10%, and varied the mutation probability from 1% to 10% by 1%. Figure 3 shows the effects of the mutation rate on the average number of terms satisfied for SGA and HGA. We see that there is little overall effect in altering the mutation rate for either algorithm, although the data suggests that high rates below 0.1 are best. The data also suggests that increased mutation rates improve the performance of HGA, however slightly.

Figure 4 shows the effects of crossover on SGA and HGA. Here we see that higher rates of crossover improve the performance of SGA and HGA. This is most interesting in the result for HGA, since it indicates that hill-climbing is solely responsible for the increase in performance of HGA over SGA.

Figures 3 and 4 indicate that HGA outperforms SGA by a significant margin. This is more pronounced when the number of problems solved is analyzed. SGA failed to solve more than 3 of the instances in either the crossover or the mutation experiments, while HGA typically solved 67-69 of the problems.

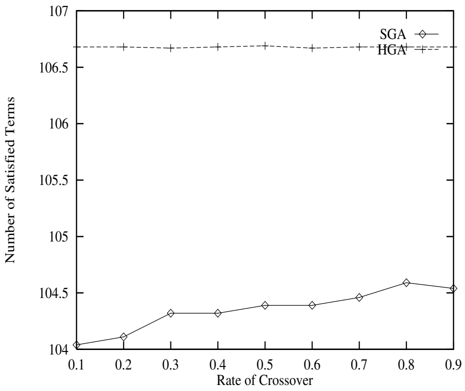

Figure 4: The Effect of Crossover on SGA and HGA.

3.3 Migration Rate and Interval for PGA and HPGA

Due to constraints imposed by the crossover mechanism, the size of any population must be both even and ≥ 4. In order to prevent larger population sizes from influencing the effects of parallelism on solving problems we divided the original population into different numbers of sub-populations. We required that the migration interval evenly divide the number of generations, and we also limited the number of migrations so that in all cases at least a few generations without migration would pass. Finally, while it is plausible to allow the rate of migration to be as high as 100% we bounded the number of migrants to 50%. Under these limitations we varied the following parameters: the number of subpopulations, the migration rate, and the migration interval. For these experiments we fixed the probability of crossover at 70% and the probability of mutation at 5%.

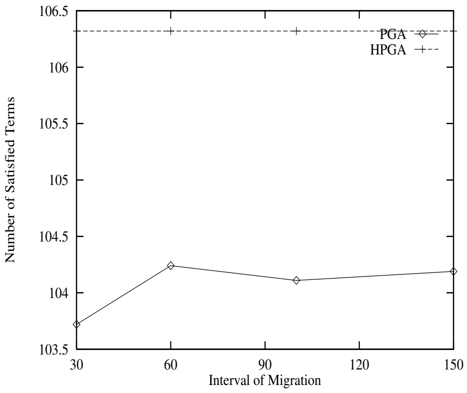

Figure 5: The Effect of Migration Interval on PGA and HPGA.

Figure 5 shows the effects of varying the migration interval on PGA and HPGA. To do this we kept the same maximum number of generations (300) and allowed migration after different length intervals. We held the number of subpopulations to 2 and the migration rate at 10%. If we recall Figure 2 we see that performance increases as the maximum number of generations increases, so we expect to see the same result here. Indeed, the figure shows that longer migration intervals improve the performance.

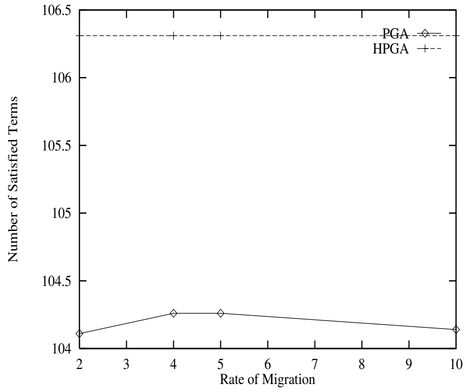

Figure 6: The Effect of Migration Rate on PGA and HPGA.

Figure 6 shows the effects of migration rate on PGA and HPGA. Here we held the migration interval at 100 and the number of sub-populations at 2 and tested migration rates of 10%, 20%, 25% and 50%. We see that migration rates of 20% and 25% performed slightly better than the others. It is our opinion that using an elitist scheme would improve this result. Since we select migrants and replace at random, we do not guarantee introducing better material into subpopulations, merely new material.

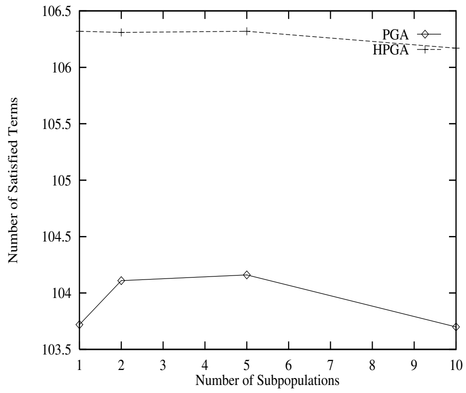

Figure 7: The Effect of Migration Rate on PGA and HPGA.

Figure 7 shows the effects of varying the number of subpopulations on performance. When the number of migrants, and the interval are held constant, this has the effect of varying the rate of migration and the rate at which information propagates from one sub-population to another. For this experiment we held the number of migrants to 2 and the interval of migration at 100. The figure indicates that the optimal number of subpopulations is 5, which corresponds to a migration rate of 25%. This result matches the previous result on the effects of migration rate.

Algorithm	Maximum Problems Solved
SGA	3
HGA	69
PGA	1
HPGA	32

Figure 8: Maximum Problems Solved.

Figures 5, 6, and 7 indicate that hill-climbing improves the effectiveness of PGA. Again, this is most apparent if we consider the number of problems solved by each algorithm. PGA never solved more than 1 problem, which HPGA solved as many as 32 problems. We can therefore conclude that adding hill-climbing can significantly enhance the power of GAs. However, more work must be done to analyze the cost of adding hill-climbing to GAs.

3.4 Does Parallelism Improve GA Performance?

It is interesting to note that parallelism as implemented in this work does not enhance the performance of GAs in solving CNF problems. The average number of terms solved by SGA reached a maximum of 104.5 as shown in Figure 3, while for PGA the average was only 104.25 as shown in Figure 5. Again, the story is best told by examining the number of problems solved. We recall from the previous sections that HGA solved as many as 69 problems, while HPGA never solved more than 32.

4 Analysis

The data clearly shows that the addition of hill-climbing enhances the performance of GAs to solve 3CNF problems. When the GA operates in the space of local optima it automatically improves the quality of solutions since any local optima is clearly a better solution than any of it's neighbors. Where there is an inexpensive algorithm to find local optima, this technique can enhance the performance of GAs at little cost.

Parallelism resulted in a degradation of GA performance, as clearly shown by Figure 8 and as seen in the graphs. It is interesting to note that the average number of terms satisfied did not fall significantly when parallelism was added to HGA. However, the number of problems solved did fall by at least half. This implies that HPGA almost finds satisfying assignments for many solveable problems. Our explanation is that, by cutting the population sizes in the PGA, we reduce the quality of local optima that are found in each sub-population, thereby degrading overall performance. If this is true, then migration using random selection and replacement is not effective enough at recovering the ground lost by using smaller population sizes. Elitism may help to solve this problem, but it is possible that the only solution is to increase the population sizes so that each subpopulation of the PGA has enough members to guarantee that good local optima are found.

5 Future Work

A great deal of future work remains to be done in this area. We should test other hill-climbing methods such as tabu search and GSAT in HGA, and analyze the CPU performance of GAs compared with GAS employing hill-climbing. In PGA we currently select migrants and replace elements of the new populations at random. A wide variety of selection schemes should be tested and compared, including elitism in choosing migrants and reverse elitism in replacing members of the host populations. In addition, it would be useful to consider either annealing schedules or adaptive schemes for migration. Such schemes would either send out migrants or request neighboring populations to send migrants when the subpopulation converges. We should also consider new neighborhood topologies such as grids or hypercubes. Finally, the ultimate goal of a parallel algorithm is to save time as well as improve performance. We should investigate distributed or massively parallel implementations of GAs and analyze real time performance gains.

References

[ChKa] P. Cheeseman, Bob Kanefsky, William Taylor. "Where the *Really* Hard Problems Are." *Proceedings of the* 12^{th} *IJCAI,* 1991.

[MiSe] D. Mitchell, B. Selman, H. Levesque. "Hard and Easy distribution of SAT Problems." *Proceedings, AAAI,* 1992, pp. 459-465

[SeKa] B. Selman, H. Kautz. "An Empirical Study of Greedy Local Search for Satisfiability Testing." *Proceedings, AAAI Spring Symposium on Artificial Intelligence and NP-Hard Problems,* 1993, pp. 149-155.

[Gl] F. Glover. "Tabu Search for Nonlinear and Parametric Optimizations", *Discrete Applied Mathematics*, 1991

[Go] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Menlo Park, Wiley and Sons, 1989.

[Go2] D. Goldberg. "Sizing Populations for Serial and Parallel Genetic Algorithms." *Proceedings, 3d International Conference on Genetic Algorithms*, 1989.

[Ho] J. Holland. *Adaptation in Natural and Artificial Systems.* Univ. of Michigan Press, 1975.

[Se] B. Selman. "A New Method for Solving Hard Satisfiability Problems." *Proceedings, AAAI,* 1992, pp. 440-446.

[MuSc] H. Mülenbien, M. Schomisch, J. Born. "The Parallel Genetic Algorithm as a Function Optimizer." *Proceedings, 4th International Conference on Genetic Algorithms,* 1991, pp. 271-278.

[Mu] H. Mülenbien. "Evolution in Time and Space: The Parallel Genetic Algorithm." In *Foundations of Genetic Algorithms*, G. Rawlins, ed., Morgan Kaufman, 1991.

[Mu2] H. Mülenbien. "Asynchronous Parallel Search by the Parallel Genetic Algorithm." *Proceedings, 3d IEEE Symposium on Parallel and Distributed Systems,* 1992, pp. 526-533.

[LiBa] G. Liepens, S. Baluja. "apGA: An Adaptive Parallel Genetic Algorithm." In *Computer Science and Operations Research, New Developments in Their Interfaces,* Balci, Sharda, Zeinos, ed., Pergamon Press, 1992.

[Ta] R. Tanese. "Distributed Genetic Algorithms." *Proceedings, 3d International Conference on Genetic Algorithms,* 1989.