A Deeper look into the 3SAT Problem with Genetic Algorithms

Ali Abbas, Ahsan Sanaullah, Fernando Trevino Ramirez
University of Central Florida
Orlando, FL 32816

Abstract

In this paper we will try to build an efficient algorithm for finding satisfying assignments of 3SAT problems. This will be done using a genetic algorithm (GA). The 3SAT problem is a classic NP-Complete problem and building an efficient algorithm for it using GAs would indicate that the GA could be a powerful tool for attacking other NP-complete problems. Source code is available at https://github.com/trevinofernando/3SAT_Solver_GA.

Introduction

The Three Boolean Satisfiability Problem (3SAT) is a classic NP-Complete problem. This means that all problems in NP reduce to it in polynomial time. No polynomial time algorithm has been discovered for NP-Complete problems yet. In this paper, we develop a GA to solve hard 3SAT instances. This demonstrates that GAs could be a powerful tool for solving other problems in NP.

The 3SAT problem is, given a 3CNF formula, does there exist a satisfying assignment for the variables? a 3CNF formula is a conjunctive normal form formula that only has clauses of size three refer to Eq. 1 for an example.

$$(x_1 \vee x_2 \vee x_3) \wedge (\overline{x_2} \vee x_3 \vee x_2) \wedge (x_4 \vee \overline{x_1} \vee \overline{x_3}) \quad (1)$$

Equation 1 is a 3SAT instance with 3 clauses and 4 variable. One satisfying assignment for Equation 1 is $x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 1$. We use 1 to represent true and 0 to represent false.

Representation

Each individual in the population of the GA represents one assignment of values to the variables. They were stored as a binary string, the value of the first character is x_1, the second character is x_2, and so on. The previous solution to Eq. 1 would be stored as "1011".

Two fitness functions were used. For the first one, the fitness of an individual was the number of clauses that it satisfied. An optimal individual is one that satisfies every clause and therefore has fitness equal to the number of clauses. The second fitness function was similar, however the fitness contributed per clause was scaled based on the number of individuals that satisfied it.

Motivation

The goal of this project is to develop an effective GA for solving 3SAT instances. NP-Complete problems are problems that have no discovered polynomial time deterministic algorithms. Unfortunately, many practical and common problems are NP-Complete, computer scientists are always looking for ways to solve or find good enough solutions for them. With this project, we hope to show that genetic algorithms are a good tool for tackling NP-Complete problems. Genetic algorithms find good enough solutions very quickly. We will show that genetic algorithms are up to the task of solving NP-Complete problems by showing that even their weakest aspect, finding best solutions, performs well on hard 3SAT instances.

Approaches

Three approaches were taken to building a good genetic algorithm for the 3SAT problem: a smarter fitness function, dynamic mutation, and deterministic crowding.

Adaptive Fitness

The adaptive fitness approach attempts to reward individuals that satisfy clauses that are rarely satisfied much more than those that satisfy common ones. Instead of awarding one fitness per clause satisfied, the value of a clause is the reciprocal of the number of individuals that satisfy it. In other words, the fitness of an individual x is

$$f(x) = \sum_i \begin{cases} \frac{1}{n_i} & \text{if clause } i \text{ is satisfied} \\ 0 & \text{otherwise.} \end{cases} \quad (2)$$

Where i is an index of the clauses of the 3SAT instance. This means that if an individual is the only one to satisfy a clause, it will gain one fitness for that clause while the fitness gained for clauses that most individuals solve is approximately $1/\text{population size}$. This will cause the beneficial schema of these individuals to proliferate in the population.

The parameters that this method was test with were uniform crossover, crossover rate $= 0.8$, rank selection, and bit flip mutation with a mutation rate of $0.5/$number of variables.

Dynamic Mutation

With this approach, we explored the idea of an adaptive mutation rate (Doerr et al. 2017) with some modifications to fit the 3SAT problem. Dynamic mutation was triggered when the standard deviation of a generation was relatively small this was done to introduce diversity into the population and avoid premature convergence. We experimentally found that the best threshold was 10%. This means that when the standard deviation is smaller than 10% of the range of possible solutions, dynamic mutation is triggered. If dynamic mutation is triggered, only individuals with fitness greater than the average were allowed to reproduce, of their children, those with fitness greater than the average of the parent population are labeled as elite. The mutation rate of non elite children is doubled. The mutation rated of an elite individual is equal to $100\frac{c-u}{c}$, where c is the total number of clauses and u is the average fitness of the parent generation.

This forces the population to explore more solutions until the standard deviation increases enough, indicating diversity in the population. The mutation rate of elite individuals is proportional to the number of missing clauses, this causes the mutation to be more aggressive towards the start of the run and more conservative towards the end. The conservativeness at the end is desired since the population doesn't need as many bit-flips to get to the optimal solution (Doerr et al. 2017). In the experiment we will show how this design choice ended up being good for early exploration but detrimental towards the end due to getting stuck in local optima.

This idea of having a mutation rate proportional to the expected number of bit-flips to get to the optimal answer came from Doerr *et. al* in their "Fast Genetic Algorithms" (Doerr et al. 2017). and while their results showed promise for a faster GA, we failed to find a reliable formula that would indicate the number of necessary bit-flips to get to the answer, or in other words, the number of variables that were incorrectly assigned. Instead using the percentage of missing clauses was proposed with the idea that closer to the end the mutation should be lower, and higher at the beginning.

Deterministic Crowding

In a standard GA all individuals compete with each other for the chance to reproduce and pass on their genes to the next generation. Individuals with higher fitness values have a higher chance of reproduction, this often causes them to dominate the population and drive others to extinction. This in turn leads to the problems of premature convergence and loss of diversity, which are especially problematic for fitness landscapes with many local optima as is the case for the 3SAT problem. In a standard GA, after performing genetic operators, we get children which replace their parents as the new population. The crowding technique changes the replacement stage, such that a subset of the population is selected for each child, and then the fitness of the most similar parent int he subset is compared to the fitness of the child. If the child has higher fitness, it replaces that parent in the new generation. The idea here is that similar individuals compete for a spot on the next generation which leads to less similar individuals and more diversity (Wong 2015).

There are some problems with the basic crowding technique, for example, the need to select a crowding factor parameter. If the value of this parameter is low, it can lead to the child competing with and replacaing dissimilar parents, this is called replacement error. In trying to solve some of these issues, the Deterministic Crowding technique was developed, which does not need a crowding factor parameter and tries to reduce the replacement error by making each child compete with its direct parents. In this algorithm, the selection method is to simply choose two parents randomly without replacement and produce two children via genetic operators. Then we compare each child to its most similar parent and it's only allowed to be copied to the next generation if it has a higher fitness value, otherwise the parent goes on to the next generation. The simlarity between individuals is calculated using Hamming distance. The following pseudocode is repeated until we have a new generation as large as the current generation (Das et al. 2011).

Algorithm 1: Offspring and replacement pseudocode

```
Select p1 and p2 from population without
  replacement;
Perform genetic operators to get children c1 and c2 if
 d(p1, c1) + d(p2, c2) ≤ d(p1, c2) + d(p2, c1) then
    if f(c1) ¿ f(p1) then
        c1 copied to next generation;
    else
        p1 copied to next generation;
    if f(x2) ¿ f(p2) then
        c2 copied to next generation;
    else
        p2 copied to next generation;
 else
    if f(c2) ¿ f(p1) then
        c2 copied to next generation;
    else
        p1 copied to next generation;
    if f(c1) ¿ f(p2) then
        c1 copied to next generation;
    else
        p2 copied to next generation;
```

Where d is the Hamming distance function and f is the fitness function. The fitness function in this case was the number of satisfied clauses. The genetic operators were uniform crossover and bit flip mutation. Crossover rate was 1 and mutation rate was 0.01.

Methods

All experiments consist of 10 batches of 100 runs of each of the 4 approaches. Each run had a maximum of 500 generations with a population size of 500. We ran 4 differ-

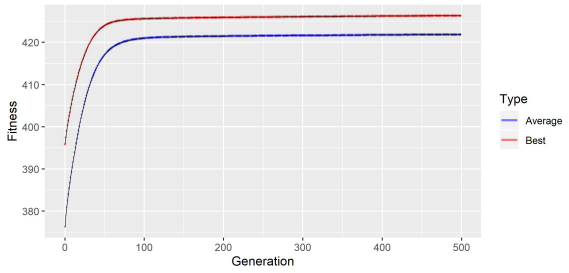

Figure 1: 3SAT Standard GA. Number of Variables = 100. Average and best fitness with 95% confidence intervals averaged over 1000 runs.

ent GA versions: “Adaptive Fitness”, “Deterministic Crowding”, “Dynamic Mutation”, and “Standard GA,” which is the general genetic algorithm provided by Dr. Wu. The standard GA was only modified by changing the fitness function to count the number of satisfied clauses.

Results

In terms of average and best fitness over generations performance, all approaches performed similarly.

Figure 1 shows the standard GA performance for a 100 variables 3SAT problem. Its performance is representative of all approaches. We can see that the average and best are close for the most part, and that the best quickly gets very close to the optimum. This suggest that there are a lot of local optima that can easily trap the est individual. We can conclude that the standard average and best fitness over generation graph is not a good metric for comparing GAs for the 3SAT problem since many times the GA will get trapped in local optima that have fitness values close to optimal fitness.

The metrics that we use for measuring performance are:

1. Average success rate: success rate is the ratio of the number of runs that find the optimum out of 100 runs. For example, if 60 runs out of 100 found the optimum, then the success rate is 0.6. We perform these 100 runs 10 times. This will give us 10 success rates, from these we calculate the average and confidence intervals.
2. Average and minimum of earliest generation to find the optimum over 10×100 runs.

The success rate of Standard GA is decreasing linearly with respect to the increasing number of variables. Average and best follow each other closely. Earliest generation to find optimum for Standard GA seems to be increasing exponentially.

The Adaptive Fitness GA vastly outperformed the other GAs in terms of the difficulty of the hardest problem it solved. It solved 125 and 175 variables, which no other GA solved, unfortunately, it’s success rate was very low with these problems.

We can see that although the success rate of DC decreases as the number of variables increases, it does so much more slowly than the Standard GA. Also DC is the only approach here that has 100% success rate for the 50 variable problem.

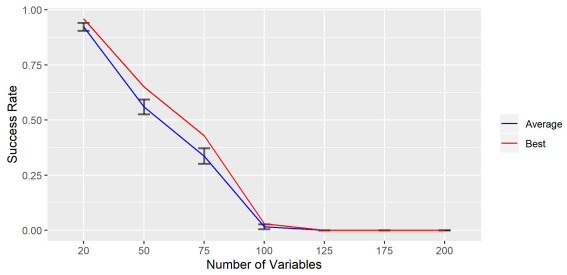

Figure 2: Success Rate of 3SAT Standard GA vs. Number of Variables. Averaged rate with 95% confidence intervals. Averaged over 10 batches of 100 runs.

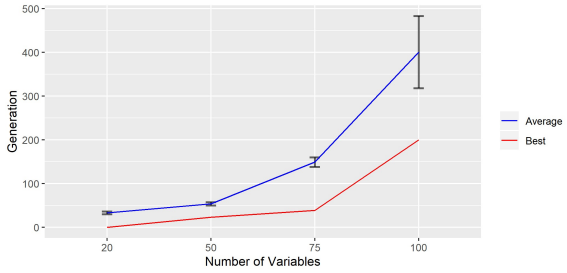

Figure 3: Earliest Generation to find optimum solution of 3SAT using Standard GA vs. Number of Variables. Averaged with 95% confidence intervals. Averaged over 10 batches of 100 runs.

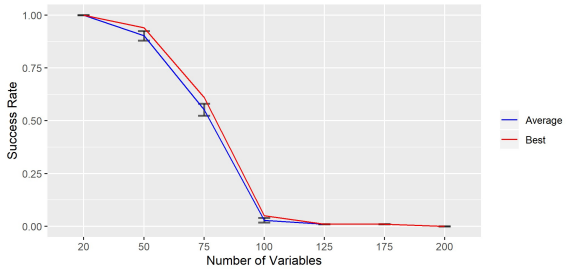

Figure 4: Success Rate of 3SAT Adaptive Fitness GA vs. Number of Variables. Averaged rate with 95% confidence intervals. Averaged over 10 batches of 100 runs.

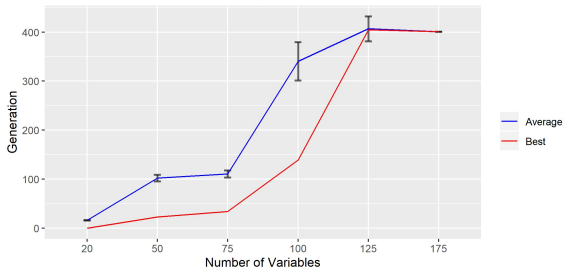

Figure 5: Earliest Generation to find optimum solution of 3SAT using Adpative Fitness GA vs. Number of Variables. Averaged with 95% confidence intervals. Averaged over 10 batches of 100 runs.

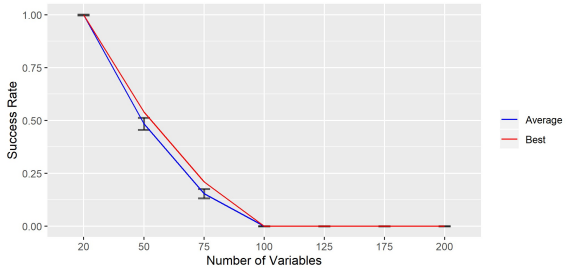

Figure 6: Success Rate of 3SAT Dynamic Mutation GA vs. Number of Variables. Averaged rate with 95% confidence intervals. Averaged over 10 batches of 100 runs.

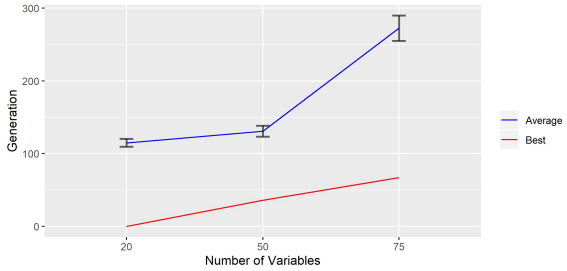

Figure 7: Earliest Generation to find optimum solution of 3SAT using Standard GA vs. Number of Variables. Averaged with 95% confidence intervals. Averaged over 10 batches of 100 runs.

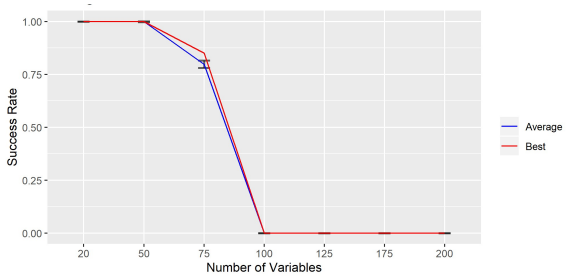

Figure 8: Success Rate of 3SAT Deterministic Crowding GA vs. Number of Variables. Averaged rate with 95% confidence intervals. Averaged over 10 batches of 100 runs.

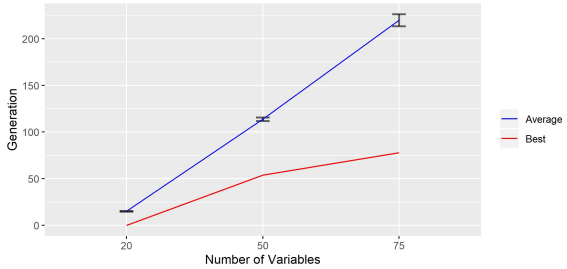

Figure 9: Earliest Generation to find optimum solution of 3SAT using Deterministic Crowding GA vs. Number of Variables. Averaged with 95% confidence intervals. Averaged over 10 batches of 100 runs.

When DC has a success rate, it's the same or better than others. In general when DC can find a solution, it does it more consistently than other approaches here, but the downside is that it wasn't able to solve harder problems than 75.

One notable difference between DC and other approaches is that its earliest generation to find optimum seems to be linear while for others is almost exponential. Although we should take these observations with a grain of salt, since we only have 3 samples for DC. Regarding dynamic mutation, we can see how this approach was outperformed by all others, including the standard GA. This may have been caused by a poor choice of the mutation rate formula or by losing diversity when forcing only elite individuals to mate. The only instance where dynamic mutation performed better than the standard GA was when solving uf20 where it solved it more consistently with a success rate of 100%, but this relatively small advantage was proved to not hold in the bigger problems and ultimately not worth using with the current implementation.

Discussions

We tried 3 different approaches for solving the NP-complete problem of 3-SAT. The Adaptive Fitness (AF) approach explored the idea of changing the fitness landscape to escape from local optima, while the Dynamic Mutation (DM), and the Deterministic Crowding (DC), explored the idea of maintaining diversity in the population. DM tried to maintain diversity by increasing the mutation rate when diversity is low, while DC made similar individuals to compete against each other such that different individuals can coexist in the population without direct competition which leads to a more diverse population.

We had various degrees of success with the approaches we tried. By far the most successful in terms of finding solutions for difficult problems up to 175 variables was AF, although it had a lower average success rate for easier problems of 20,50, and 75 compared to DC. We used the Standard GA (SGA) as a baseline, and found that both AF and DC had higher average success rates than SGA, although DC was unable to solve the 100 problem while SGA was able to solve it. DM performed worse than SGA, except for the 20 problem where it had 100% average success rate.

The adaptive fitness algorithm found a satisfying assignment of a 175 variable 3SAT instance in around 2 hours. This is no small feat. There are approximately 10^{53} possible variable assignments for such an instance. It would take an extremely powerful computer approximately 10^{35} years (many orders of magnitude above the age of the universe) to iterate through all of those assignments. This shows that GAs are very powerful tools for solving hard problems.

Further work can be done in increasing the performance of GAs on 3SAT. This can be attempted by incorporating number of clauses in the adaptive fitness scaling or combining the methods presented here. Furthermore, we have shown that GAs can be effective at tackling very hard problems that would take exponential time with a deterministic algorithm. Genetic algorithms should be applied to more NP-Complete problems to determine what more they are capable of.

Conclusion

We showed that changing the fitness landscape and maintaining population diversity can have positive effects on solving the 3-SAT problem. It is clear that not all modifications to a standard GA will improve its results, however, small changes like finding a different fitness function (like the adaptive fitness) can drastically change the fitness landscape and consequently increase or decrease its performance and ability to find the optimal solution. We have shown that genetic algorithms are an effective tool for solving NP-Complete and traditionally hard problems in general.

References

Das, S.; Maity, S.; Qu, B.-Y.; and Suganthan, P. N. 2011. Real-parameter evolutionary multimodal optimization—a survey of the state-of-the-art. *Swarm and Evolutionary Computation* 1(2):71–88.

Doerr, B.; Le, H. P.; Makhmara, R.; and Nguyen, T. D. 2017. Fast genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 777–784.

Wong, K.-C. 2015. Evolutionary multimodal optimization: A short survey. *arXiv preprint arXiv:1508.00457*.