Evolutionary Algorithms for MAXSAT

Martin Bernard, James Boyle, and Justin Wallace

Abstract

Genetic Algorithms (GA) and Population-Based Incremental Learning (PBIL) are used to search a solution space by evolving candidate solutions. In this paper, we implemented both algorithms for the purpose of solving MAXSAT problem and tested them with varied parameters in order to compare their performance and ease of implementation. After finding optimal parameter selections for our GA and PBIL implementations, we tested both on MAXSAT problems. While GA found more accurate solutions across the board, PBIL found its solutions at a much faster rate, and also required far less code. Further optimizations to our code as well as further testing where PBIL is allowed to run for more iterations would aid in making a more concrete recommendation as to which algorithm is best in which context.

1 Introduction

MAXSAT is a type of conjunctive normal form (CNF) satisfiability problem in which the goal is to find a variable assignment that satisfies as many clauses as possible. This can be difficult because of the large number of possible assignments in MAXSAT problems. Because of the effectiveness of Genetic Algorithms (GA) and Population-Based Incremental Learning (PBIL) for searching large solution spaces, these are two heuristics that are useful for solving MAXSAT problems and the ones employed herein.

GA works at a high level by randomly creating a population of candidate solutions within the solution space, selecting candidates for breeding (i.e., passing some candidates onto the next generation of candidate solutions) based upon a fitness score, breeding the candidate solutions selected for the breeding pool through a simulation of genetic crossover, and then mutating the resulting offspring with some small probability. In this way, more fit solutions are more likely to pass on their genetic material than less fit solutions.

PBIL is also an iterative process. The candidate solutions are binary bit strings, and each bit is determined by a single probability vector (where each element in the probability vector corresponds to the probability of a bit being positive). First, the algorithm creates N candidate solutions using the probability vector. The fitness of each candidate solution is evaluated. The candidate solutions with the highest fitness and the lowest fitness are then used to update the probability vector accordingly. At this point, the probability vector is exposed to the possibility of mutation before moving onto the next iteration.

In our experiments, we tested a number of possible settings for GA and PBIL on a variety MAXSAT problems to determine optimal settings for GA and PBIL. We then used these optimal settings for GA and PBIL and tested on many different problems with a number of trials to compare GA and PBIL and their efficacy for solving MAXSAT problems, as well as various tradeoffs between the two approaches.

In Section 2 we describe MAXSAT and what it accomplishes, in Section 3 we give background to GA, in Section 4 we described Population-Based Incremental Learning and how it works, in Section 5 we describe our experimental methodology, in Section 6 we discuss and analyze our results, in Section 7 we discuss further work, and in Section 8 we summarize and conclude our work.

2 MAXSAT

MAXSAT problems consist of a Boolean formula represented as a conjunction of clauses, where each clause is a disjunction of literals. Literals, in this case, are either variables or their negation. Below is an example of a MAXSAT problem with 4 variables and 4 clauses. Each variable is denoted as an integer. Each negation is represented as the negative value of the variables integer.

$$(1 \vee -2) \wedge (-3 \vee 4) \wedge (3) \wedge (-4)$$

A solution to a MAXSAT problem is a truth assignment that makes the maximum number of clauses true. In the example above, the highest number of clauses that can be true is 3. There does not exist a solution that allows for all 4 clauses to be true. There are, however, multiple truth assignments that lead to the maximum number of clauses being true.

$$\begin{aligned} A &= \{1, 2, -3, -4\} \\ B &= \{1, 2, 3, -4\} \end{aligned}$$

Due to the nature of MAXSAT problems, Genetic Algorithms and Population Based Incremental Learning are well suited to find high quality truth assignments (truth assignments that satisfy a relatively high number of clauses. Candidate Solutions take the form of a truth assignments and can be implemented as a vector, where each element indicates whether a variable or its negation should be considered true.

3 Genetic Algorithms

At a high-level, a GA works by (1) creating a random population of individuals, (2) scoring the fitness of each individual with a fitness function, (3) selecting individuals for the breeding pool based upon fitness score and mating individuals to produce offspring by applying crossover, and (4) mutating each offspring with some probability. The result is a new generation of offspring of the parent generation; steps 2 through 4 are repeated iteratively, and the resulting offspring from each iteration of the algorithm replace the original population with each iteration.

The first step of a GA is creating a completely random population of individuals. If the user has preconceptions about what the starting population should look like, they can bias the starting population to find a solution more quickly, but doing so can run the risk of premature convergence.

After generating a population, it is necessary to score each individual in the population in accordance with a fitness function; the fitness function is given an individual and determines a score for that individual that is later used to determine whether or not that individual will be selected for the breeding pool. The number output by the fitness function is not the probability for selection, as there are many breeding pool selection strategies; rather, it is simply a number that is important relative to other individuals' fitness scores. A larger fitness score for an individual will give that individual a probable advantage over individuals with lower fitness scores. We employ a fitness function that measures the fitness of a candidate solution to a MAXSAT problem as the total number of clauses that the candidate solution satisfies.

Following fitness scoring, the GA then probabilistically chooses individuals in the population, biased toward more fit ones, to add to the breeding pool. This step of the process is important for providing greediness and exploitation, which allows the GA to bias its decisions to favor more fit individuals. There are a number of strategies for selecting individuals for the breeding pool, and a number of options that affect how the breeding pool is constructed (elitism, allowing individuals to be selected multiple times, etc.).

In this paper, we test three different selection types: tournament selection, rank selection, and Boltzmann selection. Tournament selection works by randomly selecting M candidates and taking the best $k < M$ candidates for the breeding population. In this paper, $M = 2$ and $k = 1$ were used.

Rank selection ranks candidates from 1 to N, with N as the population size, such that the candidate solution with the best fitness has rank N and that the candidate solution with the worst fitness has rank 1 and selects an individual at rank $i \in [1, N]$ with probability

$$P(cs_i \text{ getting selected}) = \frac{i}{\sum_{i=1}^{N} i}.$$

The third rank method we tested is Boltzmann selection, which is similar to rank selection, but exponentially increases the probability of selection for good candidates. Boltzmann selection therefore creates a larger discrepancy in selection probability for better and worse candidate solutions. These probabilities are calculated as follows:

$$P(cs_i \text{ getting selected}) = \frac{e^{f_i}}{\sum_{j=1}^{N} e^{f_i}}.$$

The final steps of a genetic algorithm are the recombination and mutation steps, which provide randomness and exploration of the search space by combining the traits of two individuals to form an offspring, and then by randomly mutating the traits of the recombined offspring, respectively. We implemented both 1-point crossover and uniform crossover for our genetic algorithm. 1-point crossover takes two parent solutions and a random crossover point, then swaps the parents genes at that crossover point. Each pair of parents results in two children whose genes are the inverse of one another (each has the parent genes that the other does not.) Uniform crossover evenly distributes both parents' genes throughout the children solutions. In our implementation we choose each child's genes randomly from either the mother or father solution. In our algorithm uniform crossover also produces 2 children from 2 parents, but the children are independent of each other (not inverse like in 1 point crossover.) After recombination, mutation operations are applied by randomly mutating each point in the symbol string of the offspring with some small probability.

4 Population-Based Incremental Learning

Population based Incremental learning is a variant of the basic genetic algorithm structure proposed by Shumeet Baluja in 1994. In PBIL, the population of candidate solutions is regenerated with every iteration using the probability vector. The probability vector is iteratively updated with the goal of improving the vector each time. Because the population is iteratively regenerated using a probability vector, PBIL is best suited for problems which have solutions that can be represented as bit strings.

The algorithm begins with this probability vector, with each value initialized to 0.5, and each iteration generates a new population of candidate solutions. These solutions are then tested for fitness, which in the context of MAXSAT is the number of clauses that the solution satisfies. We then select the most fit individual and modify the probability vector in favor of that individual; that specific individual's traits will become more likely to be generated in the next iteration because the probabilities in the probability vector will have been increased (or decreased) accordingly. The probability vector is also modified away from the traits of the least fit solution before being exposed to the possibility of mutation. The purpose of mutation is to increase randomness in order to consider a larger solution space. The vector is updated using the following method:

```
for $i = 1$ to $LENGTH$:
    $P(i) = P(i) \cdot (1.0 - LR) + bestVector(i) \cdot (LR)$

for $i = 1$ to $LENGTH$:
    if($bestVector(i) \neq worstVector(i)$):
        $P(i) = P(i) \cdot (1.0 - NLR) + bestVector(i) \cdot (NLR)$

for $i = 1$ to $LENGTH$:
    if($U(0,1) < mutateProbability$)
        if($U(0,1) \leq 0.5$) then $mutateDir = 1$
        else $mutateDir = 0$
        $P(i) = P(i) \cdot (1.0 - mutAmount) + mutateDir \cdot (mutAmount)$
```

The mutation adds some differentiation between the iterations. We repeat these steps for a set number of iterations, updating the probability vector each time.

5 Experimental Methodology

We ran two kinds of experiments, preliminary GA and PBIL runs on three MAXSAT problems to determine our optimal parameters for GA and PBIL, and final experiments, in which we ran GA and PBIL with the predetermined optimal settings on a larger number of MAXSAT problems and with a greater number of iterations.

5.1 Preliminary Experiments

For the preliminary experiments, we ran GA and PBIL with three different files, varying the parameters, and averaging the fitness for a given set of parameters over 10 runs. The files we tested on are listed below along with their number of variables, clauses, and optimal number of clauses in Table 1. The optimal settings for GA and PBIL found from testing different settings on these files are listed below in Table 2 and Table 3, respectively. We performed preliminary testing on 3 MAXSAT problems that were not particularly large (around 400 clauses at most) and with known optimal solutions so that we could find results in a reasonable amount of time and compare our results to the known optimal results.

Table 1: Three MAXSAT problems used to optimize parameters

MAXSAT Problem	# Variables	# Clauses	Known Optimum
t3pm3-5555.spn.cnf	27	162	145
p_hat1000-1.clq.cnf	40	352	300
johnson8-2-4.clq.cnf	28	420	345

To determine which set of paramenters would be best suited for our comparison of GA and PBIL, we pooled the results of several parameter combinations and used multiple linear regression analysis to determine our parameters. We found it necessary to take this approach because GA and PBIL do not necessarily have a best set of parameters since every MAXSAT problem is different.

For GA, we created the pool by iterating through 144 parameter combinations and finding the best fitness for each parameter combination 10 times. These fitness values of each parameter combination were then averaged. This process was repeated for two additional MAXSAT problems to create a final pool of 144 unique parameter combinations tested on three different MAXSAT problems to create 432 observations. These were pooled together and then used to calculate ordinary least square linear regression coefficients using STATA. The regression excludes the rs, 1c, and w162clauses variables to maintain linear independence.

Table 2: Regression results for GA parameters

Variable	Coefficient	$P > \|t\|$
numInd	.0034	.052
bs	2.2486	.000
ts	-.0208	0.919
uc	-.2000	0.233
cProb	-.02037	0.808
mProb	-27.06791	.000
numGen	.0013	.010
w420clauses	200.0208	.000
w352clauses	152.784	.000
constant	143.5688	0.000

While the not all coefficient values are statistically significant, this regression analysis revealed to us which parameters are most important to our implementation of GA. Boltzmann selection is the obvious choice when it comes to selection method, 1-point crossover seems more effective than uniform crossover (but not by much and not to a significant degree). Of the variants we tested, the ones with a smaller crossover probability (0.7) performed better. The same can be said for the mutation probability (0.01). Number of individuals and number of generations both behave intuitively (more of either will usually produce more accurate results), but we found the significance to be imprecise and the coefficients to be small in magnitude. We reason that at some point, increasing either number of indivuals or number of generations will fail to produce a significant difference in accuracy. We instead opt to use 100 individuals for both the GA and PBIL testing to make comparison of the two algorithms more straightforward.

A smilar process was implemented to determine our choice parameters for PBIL. We tested 216 parameter combinations 10 times each to get an average fitness value for each parameter combination. This was done for 3 different MAXSAT problems[1]. The results were pooled together and then used to calculate OLS linear regression coefficients. This regression also accounts for fixed effects from each MAXSAT problem. The only omitted variable is w162clauses to prevent linear dependence.

[1]For the MAXSAT problem that contained 420 clauses, only 190 of the 216 parameter combinations were tested. This was due to a time constraint, but the results show that this missing data did not impact the robustness of this regression.

Table 3: Regression results for PBIL parameters

Variable	Coefficient	$P > \|t\|$
numInd	.0072	.000
plr	-20.9836	.000
nlr	.7833	.146
mProb	30.6864	.000
mAmount	4.1629	.002
numIt	.0005	.000
w420clauses	199.8832	.000
w352clauses	148.9157	.000
constant	138.569	0.000

These results indicate that, of the parameter combinations we tested, the ones with a lower positive learning rate and a nonzero negative learning rate performed best. We found that our strongest parameter combinations were those that had a mutation probability equal to 0.02 and a mutation amount of 0.1. As was true with GA, PBIL performs more accurately with higher counts of individuals and iterations. For the purpose of comparison, we chose 100 individuals for PBIL to match the chosen GA parameter.

5.2 Final Experiments

After determining optimal parameters for GA and PBIL, we selected 10 MAXSAT problems to run, allowing us to compare GA and PBIL using their optimal settings. We selected these by picking problems between 600 and 900 clauses for ease of testing. We also selected problems with known optimums so that we could evaluate our algorithms not just in relation to one another but also objectively. Table 4 shows the 10 MAXSAT problems we picked by filename, including the number of variables, number of clauses, size of problem, and optimum for the problem.

Table 4: 10 MAXSAT problems used to evaluate GA and PBIL

MAXSAT Problem	# Variables	# Clauses	Optimum
p_hat1000-2.clq	40	742	600
p_hat700-2.clq	42	822	668
p_hat500-2.clq	42	910	734
p_hat300-2.clq	43	736	601
c-fat200-5.clq	40	646	530
s3v80c800-8.cnf	80	800	775
s3v80c800-10.cnf	80	800	773
s3v80c900-5.cnf	80	900	868
s3v80c800-9.cnf	80	800	774
San1000.clq	40	744	605

6 Results

Results indicate that while GA is more accurate than PBIL, at least in the implementation we used, but that PBIL is much faster than GA. For the case of MAXSAT problems, since MAXSAT problems are only solved when an optimum is found, it is more difficult to solve a MAXSAT problem using PBIL versus GA when using the same number of generations and individuals.

Table 5 shows the fitnesses, times, and differences in fitnesses and times for each MAXSAT problem run. For every MAXSAT problem tested, GA performed better in terms of the best fitness score found; on the other hand, PBIL always performed better in terms of the speed of problem-solving, being 1.91 times as fast as GA on average, but missing 3% more of the total clauses than GA on average. There are therefore obvious tradeoffs to these implementations of GA and PBIL.

Table 5: Completion results for GA and PBIL on 10 MAXSAT problems

MAXSAT Problem	# Clauses	GA Time	PBIL Time	GA Fitness	PBIL Fitness
p_hat1000-2.clq	742	171	90	597.2	585.9
p_hat700-2.clq	822	190	99	666.6	650.4
p_hat500-2.clq	910	210	111	730.8	716.1
p_hat300-2.clq	736	169	90	597.3	584.7
c-fat200-5.clq	646	149	79	529.8	512.9
s3v80c800-8.cnf	800	199	111	770.4	739.7
s3v80c800-10.cnf	800	230	127	769.4	737
s3v80c900-5.cnf	900	256	151	864.3	827.5
s3v80c800-9.cnf	800	229	134	769.6	738.2
San1000.clq	744	171	90	604.3	587.8
average		197.4	108.2	689.97	668.02

Not only is GA slower than PBIL, it is also requires more implementation as it has about 4 times as many lines of code as PBIL. This means that while it is not necessarily more difficult to implement GA, because of the greater variability in factors like selection strategy, crossover type, etc., a complete GA might take more time to implement than a complete PBIL, in writing the code and especially in debugging. All things considered, PBIL looks like a better solver of MAXSAT problems than GA. In order to confirm this, it would be necessary to perform further testing that determines how many extra generations and/or individuals PBIL needs compared to GA to make up for the 3

7 Further Work and Conclusions

The most important next steps in determining if GA or PBIL is preferable in solving MAXSAT problems would, again, be quantifying the tradeoff between speed as gained in PBIL versus completeness of the solution. PBIL needs to make up for a 3% difference in the number of satisfied clauses, on average, compared to GA, and the question of how–or if–it can do this in the minimal amount of time is one worth pursuing. It could well be that PBIL need only be run for a small number of individuals or generations beyond GA to make up for its lower accuracy.

Other work that should be done is optimizing and speeding up the MAXSAT problem solver codes–both PBIL and GA. Because both sets of code share a getFitness, and both seem to perform many orders of magnitude slower than other groups MAXSAT solvers, this suggests that the getFitness function employed by this MAXSAT solver is not optimal. There are also other lower-level, non-algorithmic adjustments that could be made in the code, especially regarding caching data and avoiding recalculations.

Contingent upon optimizing the code, it would be reasonable to perform more experiments with some larger MAXSAT problems (e.g. hundreds of thousands of clauses), and also to see what impact if any the number of variables can have on accuracy or speed of the solver; in selecting our MAXSAT problems, number of variables is not something we considered, while we tried to use problems with a similar number of clauses.

In conclusion, PBIL seems to be significantly faster than GA (1.91 times) and only 3% less accurate. It is worth performing further testing to determine at what number of generations and individuals a PBIL problem solver will be as accurate on average as a GA solver for a given MAXSAT problem type. Performing this analysis would allow us to conclude with more certainty about PBIL compared to MAXSAT and would allow us to quantify the tradeoff between speed and accuracy in choosing PBIL or GA, respectively. It is necessary to perform further optimizations and fixes in the code for these algorithms to facilitate a wide variety of testing to refine our understanding of the differences in these algorithms.