Evolutionary Algorithms for MAXSAT

A Comparison of Genetic Algorithms and Population Based Incremental Learning

September 30, 2018

Sean Cork, Kamaal Palmer, Luca Ostertag-Hill

Nature Inspired Computation
Project 1

***Abstract*—This project examined the effectiveness of Genetic and Population Based Incremental Learning algorithms on MAXSAT problems. The objective of MAXSAT problems is to maximize the number of satisfied clauses in a given set of expressions by assigning Boolean values to each given variable. After implementing these algorithms, we ran a series of tests varying the parameters of each algorithm to optimize performance in terms of the best solution found and the execution time. With the Genetic Algorithm, we varied both the crossover probability and mutation probability. We also implemented three distinct types of selection, including rank, tournament, and Boltzmann. With PBIL, we varied both the positive and negative learning rate. Our research shows that after optimizing the parameters of the Genetic Algorithm and PBIL listed above, the Genetic Algorithm outperformed PBIL in terms of its effectiveness in finding a more optimal solution, the length of iterations required to find that solution, and the execution time of the algorithm.**

I. Introduction

MAXIMUM satisfiability (MAXSAT) is a variation of Boolean satisfiability (SAT), the first problem proven to be NP-complete. The objective of SAT problems is to determine the assignment of values to Boolean variables $x_1, x_2, ..., x_n$ that makes a given set of Boolean expression true. These expressions, or clauses, are in conjunctive normal form (CNF). The purpose of the MAXSAT variation is to find the assignment of values that maximizes the number of true expressions.

A Genetic Algorithm (GA) is a model for solving optimization problems based on the concepts of natural-selection and evolution. GA operates by iteratively applying a form of selection, recombination, and mutation to an evolving population of individuals. Selection is accomplished through a fitness-based method, that provides individuals with a higher fitness an increased opportunity to breed and spread their characteristics to the next generation. Recombination and mutation are probabilistic and introduce randomness into the population, allowing for exploration of the solution space. Over the course of generations, this iterative process evolves the population toward an optimal solution.

Population Based Incremental Learning (PBIL) alternatively uses a probability vector, that stores probabilities for each gene, or assignment, that each individual has. These probabilities are used to generate a new population of individuals during each iteration. The probability vector is then updated based on the characteristics of the most and least fit individual found in the population. Unlike GA, there are no evolving individuals in a continuous population, rather an evolving probability vector that itself becomes the solution.

The objective of this project was to give a recommendation regarding which evolutionary algorithm (GA or PBIL) to use on MAXSAT problems. Before we could test GA and PBIL against each other, we optimized certain parameters of each algorithm. For the Genetic Algorithm, we varied the crossover and mutation probability. For the crossover probability, we started at 0.0 and worked towards 1.0 (testing every 0.1-0.2 percent). For the mutation probability, we also started at 0.0 and worked towards 1.0, but concentrated on small values around 0.01 (a typical value for mutation probability). In PBIL we varied both the positive and negative learning rate of the probability vector. For the positive learning rate, we tested values from 0.0 to 0.6, focusing on values around 0.2 (a typical positive learning rate). For the negative learning rate we focused on smaller values around 0.1. After testing these varying parameters, we used the optimized values to run GA and PBIL against each other on different MAXSAT problems. We did this to determine which algorithm could find a better solution, at what iteration that solution was found, and how long the program took to execute. These three aspects of the algorithm are critical, because they allow us to weigh solution against cost.

From the tests involving GA, we found that a small mutation probability around 0.005 and a high crossover probability in the range 0.7-1.0 was most effective. Mutation rates of 0 or greater than 0.05 were significantly less effective than the optimal of 0.005. For crossover, the effectiveness of the algorithm continued to increase as the probability rose from 0.0 to 1.0. With PBIL we found that a positive learning rate around 0.05 was most effective and that rates larger than 0.1 were much worse. Similarly for the negative learning rate, our tests show that a 0.1 rate works best, while higher rates lead to a weaker solution. Taking the optimal values we found from these tests, we ran GA against PBIL on varying MAXSAT problems. We determined that GA was not only a more effective algorithm, finding solutions that were on average 0.94% better than PBIL, it also found the optimal solution 141 iterations earlier. The GA algorithm was also executed 12 seconds faster on average than PBIL.

In the next section of this paper, we give a detailed description of the MAXSAT problem and explain why GA and PBIL are suitable methods for achieving positive results. Then in sections 3 and 4, we outline the process of Genetic and PBIL algorithms. In section 5 we proceed to explain our methodology behind our testing of the two algorithms. Section 6 contains our findings and analysis of our results. Finally, we conclude with a section on further work and a section summarizing our results.

II. MAXSAT

The MAXSAT problem stems from the NP-complete Boolean satisfiability (SAT) problems. The objective of both problems is to determine the optimal assignment of $true$ or $false$ values to variables $x_1, x_2, ..., x_n$ in a given a set of Boolean expressions. The difference between the problems is that MAXSAT does not require each Boolean expression to evaluate to $true$. In MAXSAT the optimal solution is the assignment of values that maximizes the number of satisfied expressions. In SAT problems, because each expression must evaluate to $true$, there may exist no solution.The Boolean expressions, often times referred to as clauses, are in *conjunctive normal form* (CNF). An example of this form is given below.

$$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$$

This example contains three clauses, each of which contain variables x_1 and x_2. In some of the clauses, the Boolean values of the variables are negated. Only one variable within the clause must be $true$ for the expression to be satisfied. In the example above, if x_1 is $true$ and x_2 is $true$, then the first clause evaluates to $true$, the second to $true$, and the third to $false$. However, assigning x_1 to $true$ and x_2 to $false$ satisfies all the clauses. This is the optimal solution to this problem. Though all clauses are satisfiable in this example, the problems dealt with in this project are much more complex and have no solution that satisfies all clauses. Thus the aim of our algorithms is to maximize the amount of satisfied clauses.

MAXSAT can be applied to general optimization problems. Recently in computer science literature, MAXSAT problems have been used in Cost Optimal Correlation Clustering and Causal Structure Learning. In this project, we are testing Genetic and PBIL algorithms on MAXSAT problems to determine if one method finds a better solution. Since MAXSAT is an optimization problem, applying these two evolutionary algorithms should yield positive results. This is because both algorithms balance exploration and exploitation to maneuver around the search space, while progressing towards an optimal solution. We will also be testing a variety of parameters on each algorithm in order to optimize their performance.

III. GENETIC ALGORITHMS

The conceptual foundation of Genetic Algorithm comes from the intertwined process of natural-selection and evolution. The algorithm is used to solve optimization and search problems through an iterative process that evolves a population of individuals. In the scope of the MAXSAT problem, the genotype of each individual in the population is the value it assigns to the Boolean variables $x_1, x_2, ..., x_n$. Each iteration of the algorithm is called a generation and uses three nature-inspired techniques to evolve the population towards an optimal solution. These three techniques are selection, recombination (often referred to as crossover), and mutation. The use of all three of these techniques allows the Genetic Algorithm to strike a balance between exploration of the search space and exploitation of the best found solutions so far. This balance between exploration and exploitation is critical, as it allows Genetic Algorithms to overcome local maximums yet handle a large search space. A basic outline of the Genetic Algorithm is outlined below and the three evolutionary techniques are expanded on later.

1) Generate a starting population of n individuals, wherein each element of each individual is randomly generated with equal probability.
2) Evaluate and score each individual based on the fitness function.
3) Select n individuals with duplicates for the breeding pool, favoring individuals with a higher fitness.
4) Perform crossover on parent pairs with some defined probability. Otherwise move on unchanged. This produces a new population of size n.

5) Perform mutation on each element of each individual in the new population with some defined probability.
6) Iterate over steps 2 through 6 until a maximum solution is found or the defined number of generations allowed is over.

A. Selection

Selection is the process of choosing some number of individuals from the current population to act as a breeding pool for the next generation. The method for selecting these individuals is fitness-based. This means that individuals with a higher fitness are more likely to be chosen, giving them a greater opportunity to exert their influence on the next generation. The fitness of an individual is representative of how close the individual is to the optimum solution of the problem. When working with MAXSAT problems, the fitness of each individual is the amount of clauses that individual satisfies. The bias towards fitter individuals is representative of exploitation and is necessary to evolve the population towards and optimal solution. This is why duplicate individuals will be allowed in the breeding pool. However, fitness-based selection is probabilistic, so it inherently provides some randomness. In this project we will be using rank, tournament, and Boltzmann selection to determine the breeding pool.

Rank selection works by ranking the individuals 1 to N by fitness $f_1, f_2, ..., f_n$, where f_1 is the least fit individual with rank 1 and f_n is the most fit with rank N. Probabilities are then assigned to each individual by the equation given below, where i is the rank. Individuals are selected one by one based on these probabilities until the size of the breeding pool matches that of the total population.

$$\frac{i}{\sum_{j=1}^{n} i}$$

Tournament selection works by randomly choosing M individuals from the population and taking the best k of those. The k individuals chosen are the k most fit individuals in the random set of M. Typical values for M and k are 2 and 1. This means that 2 random individuals are chosen from the population and the fitter individual is placed in the breeding pool. This occurs until the breeding pool is full.

Boltzmann selection works by giving each individual a probability of being selected based on the equation shown below, where f_i is the fitness of individual i.

$$\frac{e^{f_i}}{\sum_{j=1}^{n} e^{f_j}}$$

B. Recombination

Crossover uses the breeding pool of individuals found through selection to create the next generation of individuals. Individuals from the breeding pool are randomly paired up in groups of 2 (representative of parents) and are combined to produce offspring. Crossover occurs probabilistically between each pair of individuals to introduce randomness. Typical values for crossover range from $[0.6, 0.9]$. If crossover does not occur, the two offspring are clones of the parents (parent solutions move onto next generation). This allows for big steps to be made in the solution space and allows Genetic Algorithms to escape local maximums. In this project we will be using 1-point and uniform crossover. In 1-point crossover a random crossover point between the two parents is chosen and two offspring are created. This form of crossover is represented below.

$$Parent1 : [true, false, true] \mapsto Child1 : [true, false, \textbf{false}]$$

$$Parent2 : [true, true, false] \mapsto Child2 : [true, true, \textbf{true}]$$

The crossover point in this example occurs at index 3. Thus $Child1$ receives its first two values from $Parent1$ and its last value from $Parent2$. Similarly, $Child2$ receives its first two values from $Parent2$ and its last value from $Parent1$. In uniform crossover recombination is done position by position, choosing the value for the child from either parent with equal probability. This form of crossover only produces one individual. Thus to maintain the population size, we must either double the breeding pool or force recombination to occur twice between each set of parents. Our implementation of uniform crossover uses the latter method. In the example of uniform crossover below, values 1 and 3 of the $Child$ came from $Parent2$ while value 2 came from $Parent1$. Each of the child's values had an equal opportunity of coming from either parent.

$$Parent1 : [true, \textbf{false}, true]$$

$$Parent2 : [\textbf{true}, true, \textbf{false}]$$

$$Child : [true, false, false]$$

C. Mutation

Mutation is used as a technique to make tiny changes to individuals in the population. Mutation occurs on each element of each child with some small probability. Typical values for mutation range from $[0.001, 0.02]$. For MAXSAT problems, if mutation occurs on an element, the value of that element is flipped ($true$ to $false$, $false$ to $true$). Mutation is probabilistic, but occurs on a very small scale. This allows Genetic Algorithms to accomplish little steps in the solution space by conducting local search around itself.

IV. Population Based Incremental Learning

Population Based Incremental Learning (PBIL) is an optimization algorithm that was proposed as a simpler alternative to traditional Genetic Algorithms (GA). Unlike GA, which relies on the evolution of individuals, PBIL maintains a probability vector which represents the genotype of the entire population. This probability vector is used to generate candidate solutions during each iteration. Competitive supervised learning (rather than crossover as in GA) adjusts these probabilities to increase the probability of generating high quality candidate solutions. Though competitive learning is often implemented through the selection of a specified number of the most fit individuals to positively update the probability vector, in this project the best and worst fit individual will be used. During each iteration the probability vector will be updated towards the best fit individual and away from the worst fit individual. If the best and worst fit individual are the same, then we only update the probability vector towards the best fit. This continuously updated probability vector will eventually become the solution (instead of the best individual in GA). The specific procedure of PBIL is further outline below.

1) Generate a probability vector wherein each element (or allele in genetics) is 0.5. The range of these elements is [0, 1].
2) Generate a population based on the probability vector. The number of individuals to generate during each iteration for a population needs to be determined.
3) Evaluate and score each individual based on the fitness function. These individuals are ranked from best to worst fit.
4) Competitive supervised learning is used to update the probability vector based on the best and worst fit individual. The probability vector learns positively from the best fit individual and negatively from the worst fit.
5) Discard the newly created population.
6) Mutate each element in the vector with probability mm. The element has equal probability (0.5) of being shifted m up or down. The mutation probability is generally small.
7) Iterate over steps 2 through 7 until the population vector converges to an optimal solution.

The implementation of competitive supervised learning is critical for the convergence of the probability vector. Because this project uses a best and worst fit individual to update the probability vector, a positive and negative learning rate must be provided. The elements in the vector are updated based on the equation below. In this function α represents the learning rate. A typical value for this rate would be 0.1.

$$PV_i = PV_i(1 - \alpha) + S_i\alpha$$

PBIL distinguishes itself from GA as it gets rids of population, storing all of its information in a single probability vector. This leads to a massive decrease in the memory footprint of the algorithm as compared to GA. Further, the evolution of one vector over a population of individuals allows PBIL to generally converge faster than GA. PBIL does not use selection and recombination operators that can be complex to implement and costly. It has been shown to work better on certain solutions, though it possesses similar issues as GA. These common issues include the fitness function of the algorithm, the number of iterations to run, and representation. In PBIL it is also difficult to determine the number of individuals to generate during each iteration, the number of individuals to select for competitive supervised learning, and the learning rate.

V. Experimental Methodology

For our Genetic Algorithm testing, we decided to investigate the effect mutation and crossover probability had on the algorithms performance. First, we investigated the performance of the algorithm with the following mutation values $[0.0, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0]$ while keeping crossover at 0.8. We choose values such as $[0.0, 0.5, 1.0]$ to get a broad scope of the effect of mutation, but focused on values around the typical mutation rate of 0.01 to determine the optimal probability. After we found the mutation probability that was most optimal, we decided to experiment with incrementing the crossover probability using the previously best found value for mutation. We tested crossover values $[0.0, 0.2, 0.4, 0.6, 0.7, 0.8, 1]$. Though the typical value for crossover is between $[0.6, 0.9]$, we wanted to test extreme values to see their effect on the algorithms proficiency. However, to find the optimal we also tested more values inside this range. To limit the possible error that arises from the probabilistic nature of Genetic Algorithms, we ran a total of three trials for each value on both of these experiments.

For PBIL experiments, we decided to modify the positive and negative learning rates for the probability vector. First, we modified the positive learning rate by 0.05 from the typical value of 0.1 to observe any differences in the best solution found. Along with the modifications in positive learning rate, we also varied the negative learning rate by 0.005 from the typical value of 0.075 as well. The initial experiments used all permutations of the learning rate values; $[0.05, 0.1, 0.15]$ for the positive learning rate and $[0.07, 0.075, 0.08]$ for the negative learning rate. We first used a smaller MAXSAT problem (t3pm3-5555.spn.cnf) with 162 clauses and 27 different variables. After this initial exploration, in which we found that the learning rates had a significant impact on the solution found, we repeated our experiments with a bigger MAXSAT problem (HG-3SAT-V250-C1000-1.cnf). This problem had 1000 clauses and 250 variables and would display these differences in learning rates more prominently. This was conducted to see if we could achieve sub optimal or even more optimal partial solutions to the MAXSAT problems with different learning rates. In this second experiment with PBIL we tested positive learning rates of $[0.05, 0.2, 0.3, 0.4, 0.5, 0.6]$ and negative learning rates of $[0.0, 0.005, 0.01, 0.05, 0.1, 0.5, 1.0]$. Similar to experiments with GA, we wanted to conduct a broad search of the parameter by using extreme large and small values. However, we also centered most of the values around the typical learning rates for PBIL to find the optimal value.

After finding the optimal parameters for both the Genetic Algorithm and PBIL we decided to test the two functions against each other by running them with the optimal parameters we found earlier. This comparison was run on HG-3SAT problems (250 variables, 1000 clauses) in order to be able to emphasize discernible differences. We conducted three trials for GA and PBIL on five different HG-3SAT problems, giving us fifteen data points for GA and PBIL. This allowed us to make a recommendation of which algorithm was better suited to handle MAXSAT problems.

VI. Results

A. Mutation and Crossover in Genetic Algorithms

When modifying the parameters for experiments, we found that higher probabilities produced better performance in terms of best fitness. On the other hand, when modifying the mutation probability, we found that lower values produced better performance.

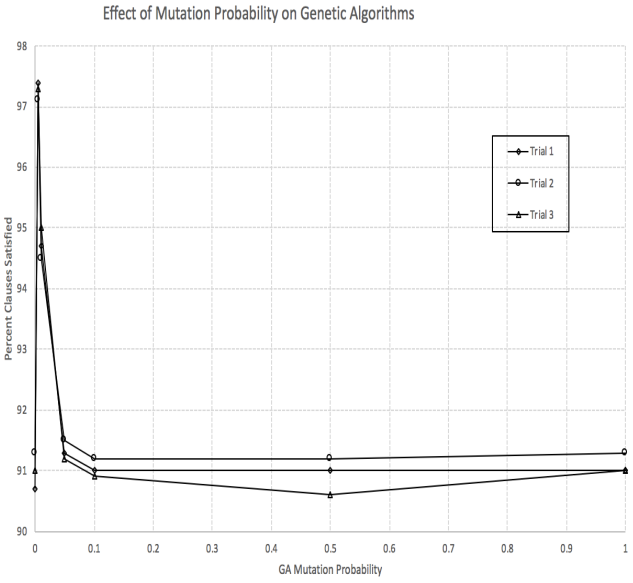

Fig. 1. Simulation results of GA when varying the mutation probability. Parameters are controlled at $Population = 50$, $Selection = Rank$, $Crossover = 1 - Point$, $CrossoverProbability = 0.8$, $MutationProbability = 0.02$, $Iterations = 750$. Simulation run on HG-3SAT-V250-C1000-1.cnf.

Figure 1 shows a negative trend in performance for increasing mutation probability in the HG-3SAT files. This negative trend is supported by a correlation coefficient of -0.393 for trial 1, -0.417 for trial 2, and -0.402 for trial 3. Through this data, we found the best value for the mutation probability to be 0.005. We believe that the reason that this value worked well for us is because we already used a higher probability of crossover which pushes for exploration, thus, we must prioritize exploitation. The higher mutation rates cause exploration to further increase which reduces exploitation of good solutions. Thus, we want to balance our solution space to further explore whilst simultaneously exploiting good solutions. In conclusion, we found the value of 0.005 to be the best balance between exploration and exploitation.

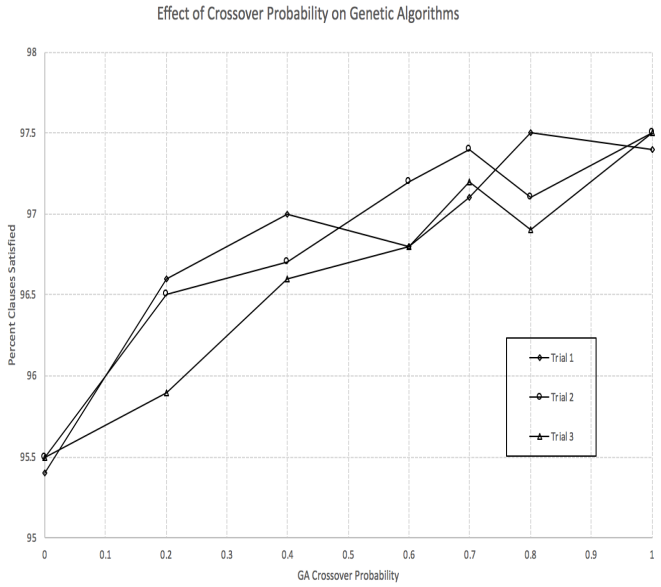

Fig. 2. Simulation results of GA when varying the crossover probability. Parameters are controlled at $Population = 50$, $Selection = Rank$, $Crossover = 1 - Point$, $MutationProbability = 0.005$, $MutationProbability = 0.02$, $Iterations = 750$. Simulation run on HG-3SAT-V250-C1000-1.cnf.

Figure 2 shows a positive trend in results for increasing crossover probability on the HG-3SAT files. This positive trend is supported by a correlation coefficient of 0.883 for trial 1, 0.922 for trial 2, and 0.967 for trial 3. While the correlation coefficient shows a positive trend between crossover probability and performance, we found that the percent difference in best fitness between 0.8 and 1 for crossover probability was not discernable. Thus, we decided to use 0.8 in later runs as it seemed to be a better value considering we wanted to keep some individuals through generations. We found that the lower crossover probabilities produced worse results as lower crossover probabilities coupled with the low rate of mutation (0.005), will result in the population reaching a local maximum with little opportunity for breaking through the local maximum. Thus, greater crossover probabilities allows for increased exploration of the solution space.

B. Positive and Negative Learning Rates in PBIL

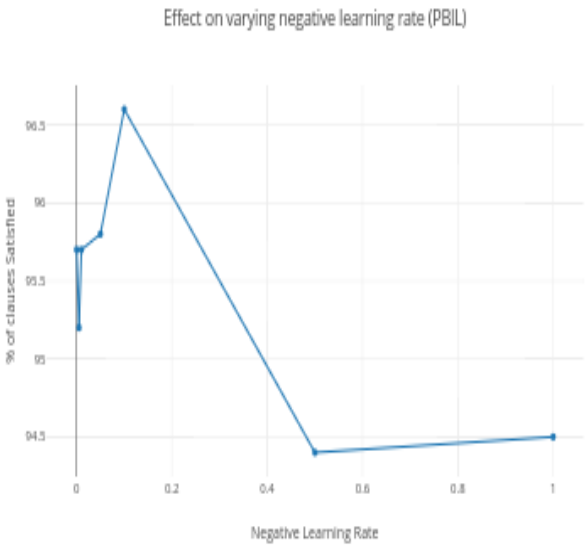

Fig. 3. Simulation results of PBIL when varying the negative learning rate. Parameters are controlled at $Individuals = 50$, $PositiveLearningRate = 0.1$, $MutationProbability = 0.02$, $MutationAmount = 0.05$, $Iterations = 750$. Simulation run on HG-3SAT-V250-C1000-1.cnf.

In Figure 3, we modify the negative learning rate. We found that the negative learning rate that that we found produced the best solution (0.1) was relatively close to the rule of thumb value in popular literature (0.075). Although modifying the negative learning rate had limited effects on best fitness, there was enough of an effect on best fitness (2 percent between best and worst performance) to indicate negative learning rate of 0.1 had the best percentage of clauses satisfied. While initially increasing the negative learning rate increases the percent of clauses fulfilled, this performance plateaus and decreases after a negative learning rate value of 0.1. Interestingly, there was no variation in run time while modifying the negative learning rate which

shows that PBIL is very consistent timing wise. While testing the different variations in negative learning rate, we found that as the negative learning rate increased, the best fitness performance decreased. With the increases in negative learning rate, we found that the iteration the best fit individual was found decreased as well. The decrease in the iteration which found the best fit individual time can be explained by the simple fact that higher negative learning rates increased the speed at which the probability vector was modified to reach the zero value.

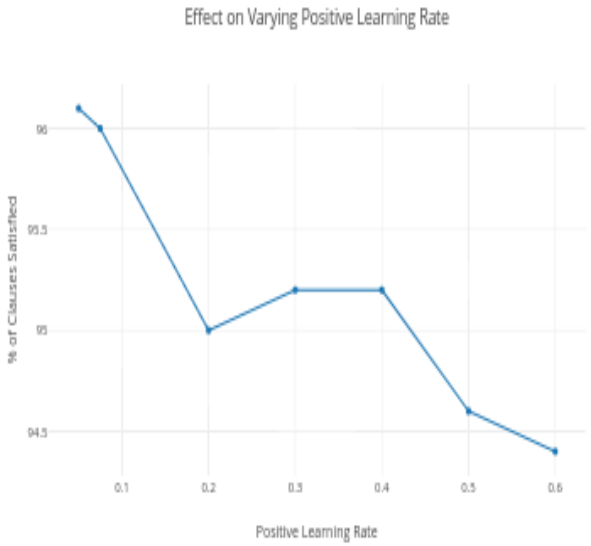

Fig. 4. Simulation results of PBIL when varying the positive learning rate. Parameters are controlled at $Individuals = 50$, $NegativeLearningRate = 0.1$, $MutationProbability = 0.02$, $MutationAmount = 0.05$, $Iterations = 750$. Simulation run on HG-3SAT-V250-C1000-1.cnf.

In Figure 4, we modify the positive learning rate using the previously found optimal negative learning rate. This figure showcases that a lower value for the positive learning rate achieved much better results than higher values. The optimal value for positive learning rate proved to be more difficult than finding the optimal value for negative learning rate. Nevertheless, similarly to the negative learning rate experiments, the variance in the percent of clauses satisfied was very minimal. The low level of variance in performance suggests that PBIL is quite consistent in finding solutions regardless of positive learning rate. Through varying the positive learning rate, we found the number of iterations necessary for the algorithm to find the maximum value of the probability vector greatly increased. This increase was somewhat expected after the results from finding the optimal negative learning rate. The increased positive learning rate results in the algorithms two learning rates pushing against each other in the probability vector which results in more iterations to converge to the best-found solution.

In furthering our results we determined if while conducting experiments while PBIL is more consistent if you do not know the optimal parameters to use and therefore could offer more accurate results. If you had enough time to explore the parameter space of the GA algorithm it would be much more beneficial as it gives better values for clauses satisfied and takes significantly less time to run the program.

C. Comparison of GA and PBIL

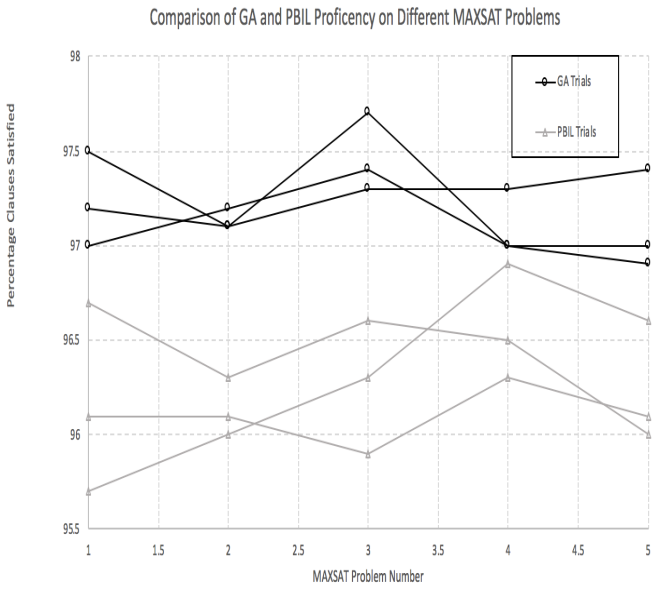

Fig. 5. Simulation results for percent clauses satisfied when comparing GA and PBIL on different MAXSAT problems. Parameters are controlled at $Population = 50$ and $Iterations = 750$. GA parameters are $Selection = Rank$, $Crossover = 1 - Point$, $CrossoverProbability = 0.8$, $MutationProbability = 0.005$, and $MutationProbability = 0.02$. PBIL parameters are $PositiveLearningRate = 0.05$, $NegativeLearningRate = 0.1$ $MutationProbability = 0.02$, and $MutationAmount = 0.05$. Simulation run on HG-3SAT-V250-C1000 files 1-5.

In comparing algorithm performance between GA and PBIL, we found that after finding the optimal parameters for both functions, the Genetic Algorithm performed significantly better in best fitness

percentage. In Figure 5, the best fitness results are shown when comparing the performance of PBIL and GA on the 5 different HG-3SAT files. As Figure 5 showcases, GA generally outperforms the PBIL algorithm by a significant margin when it comes to the number of clauses satisfied. While GA outperforms PBIL in terms of best fitness at optimal parameters, GA had more variance in performance across crossover and mutation probability parameters while PBIL had fairly consistent performance across all variations of positive and negative learning rates. Thus, we conclude that favorable GA performance requires more exploration of the parameter space. Accordingly, GA requires more time to properly set up, but once given the time to explore the parameter space, GA results in better best fitness.

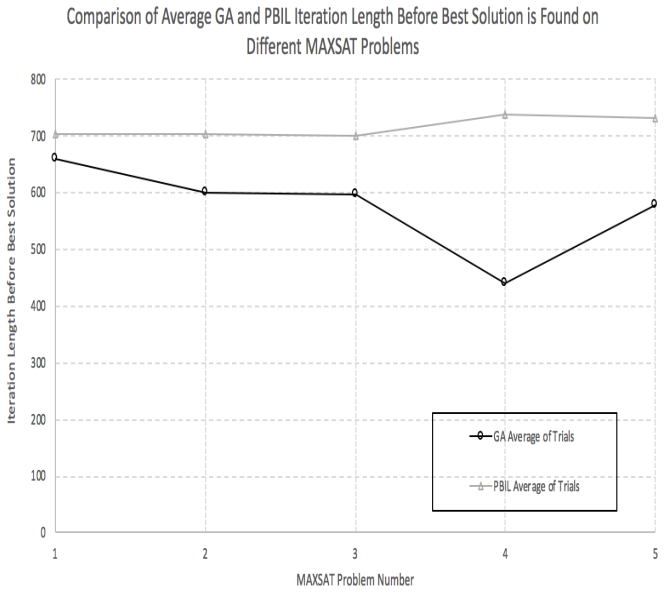

Fig. 6. Simulation results for iteration length before best found solution when comparing GA and PBIL on different MAXSAT problems. Parameters are controlled at same values as Figure 5. Simulation run on HG-3SAT-V250-C1000 files 1-5.

Figure 6 showcases GA finding the best-fit individual earlier than PBIL. On average, GA finds the best-fit individual 100 iterations faster. This figure takes the value from three trials and averages them into one data point for each file. The fact that the best solution was found towards the end of the allowed number of iterations suggests that both algorithms could have found better solutions given more time. This possibility is explored in our further works section. The fact that GA does require less iterations than PBIL, suggests that we could lower the iterations allowed and cut down the execution time of GA even further.

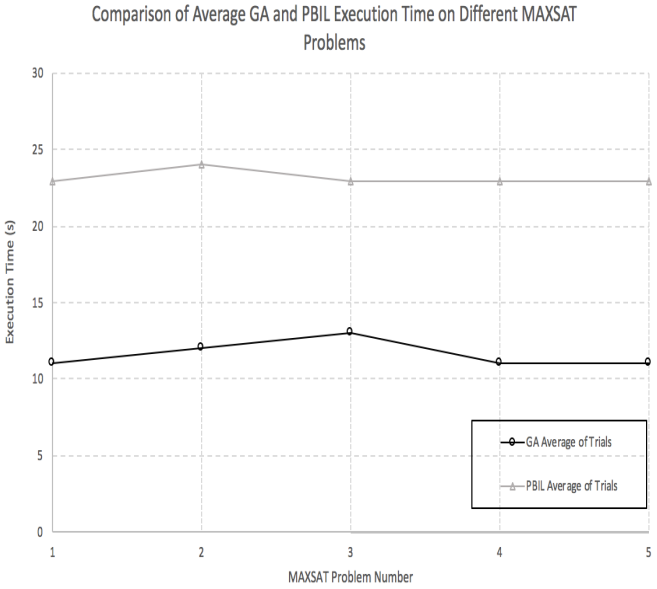

Fig. 7. Simulation results for execution time when comparing GA and PBIL on different MAXSAT problems. Parameters are controlled at same values as Figure 5. Simulation run on HG-3SAT-V250-C1000 files 1-5.

Figure 7 showcases GAs runtime being significantly lower than PBIL for the HG-3SAT files. Thus, on average, GA runs faster and more efficiently than PBIL given GA having good parameters. However, its important to note that its possible that there is a configuration of PBIL parameters which would result in faster runtimes which may come at the cost of best fitness performance.

TABLE I
COMPARISON OF GA AND PBIL ON DIFFERENT MAXSAT PROBLEMS

	GA	PBIL
Percent Clauses Satisfied	97.21	96.27
Iteration Length	574	715
Execution Time (s)	11	23

Tbl. 1. Average result over 15 trials for GA and PBIL in terms of percent of clauses satisfied by the best solution, iteration length to find the solution, and execution time of the algorithm.

This table summarizes our results found when comparing the two algorithms on different files. Since we tested the algorithms on five different files, with three trials each, we have fifteen data points for both algorithms. Table 1 above displays the average value of our findings. The table shows

that the Genetic Algorithm is significantly better in all three aspects than PBIL. Most importantly is the fact that the Genetic Algorithm found on average a 0.94% more optimal solution than PBIL.

In conclusion, the data showed that conducting experiments in PBIL is less reliant on optimal parameters. However, if you have time to explore the parameter space of the GA algorithm and find optimal parameters, it results in two main benefits. First, our data showed GA had a lower runtime than PBIL. Second, GA finds a higher percentage of clauses satisfied in the best-fit individual for the MAXSAT problem. Thus, based off these aforementioned benefits, we recommend GA be used when there is time to explore the parameter space. However, if there are time constraints and one would like a general estimate, we recommend PBIL as it didnt had a lower level of best-fitness performance variability in comparison to GA.

VII. Further Work

In further work, one goal would be to obtain more data to further support our conclusion. In our current model, we ran each data point three times. While three data points does account for the level of variability between each experiment, many scientists contend that any experiment needs 5 trials to be viable. Thus, given more time to work on this project, we would like to run each result a minimum of 5 times to get the average results of a configuration of algorithm parameters. Additionally, given more time to work on this, we would like to see the effects of population size and number of generations on time and best fitness. Viewing our data, all of our experiments converge on a best fit individual before the number of maximum generations. Thus, we know that it is possible to cut down the time necessary to run these experiments by lowering the maximum number of iterations. Given more time, we would like to explore how the different parameters effect time and best fitness in their entirety. Viewing the fitness and time output of the best parameter settings would give us a better understanding of which algorithm works better for the MAXSAT problem. Also, another explorable idea would be to examine the runtime of the algorithms as the problems reach bigger size as its possible PBIL might run faster than GA given a bigger problem. Furthermore, the time necessary to find the best parameters would also be something to investigate going forward. In conclusion, we would like to get more time to gather more data to be able to better compare the algorithms.

VIII. Conclusions

After analyzing our data, we found that the ideal algorithm to be Genetic Algorithms for MAXSAT problems. The Population Based Incremental Learning (PBIL) algorithm creates much less variance in the percentage of clauses satisfied. Thus, you do not need to explore the parameter space of the PBIL algorithm to the same extent as GA to achieve good results. However if you have the time and resources to explore the parameter space surrounding GA than the runtime and best-fitness performance benefits are clear. In conclusion, in the files that we tested GA and PBIL, we found that GA outperforms PBIL in almost every aspect: finding the best-fitness, the iteration the best-fit individual is found, and runtime.