Evolutionary Algorithms for MAXSAT

Authors: Nell Fusco, Ryan Donlan, Cara Nunley

Abstract: Evolutionary Algorithms aims to create a population that has specific characteristics or is best fitting to solve a problem. There exist multiple different approaches to Evolutionary Algorithms, two of which are Genetic Algorithms and Population Based Incremental Learning (PBIL). These approaches can be used in solving various types of optimization problems, one of which is MAXSAT problems. Here the effectiveness of Genetic Algorithms and PBIL on solving MAXSAT problems will be evaluated through implementation of the two algorithms and experiments on the implementations.

1 INTRODUCTION

A Satisfiability (SAT) problem is a problem of determining whether a Boolean assignment of variables satisfies an expression. While a solution to a Satisfiability (SAT) Problem satisfies all clauses in the specified problem, a solution to a MAXSAT problem satisfies not necessarily all, but the maximum number of clauses. The solutions to such problems can be generated through Evolutionary Algorithms, changing and improving an initial population over time until a good solution is found. Through solving such MAXSAT problems we will compare Genetic Algorithms(GA) and Population Based Incremental Learning (PBIL). Both Evolutionary Algorithms, GA and PBIL, have some distinct differences therefore using them on these MAXSAT problems will allow us to compare the two. Eventually, this will allow for us to make a suggestion on whether GA or PBIL should be used for MAXSAT problems.

Evolutionary algorithms aim to create an individual with certain properties. Among the different approaches to evolutionary algorithms are Genetic Algorithms (GA) and Population Based Incremental Learning (PBIL). These algorithms aim to balance randomness and greediness to emerge with a population of best-fit individuals. GA start with an initial population and then cycle through a process of evaluating an individual's fitness level, selecting a breeding bool, recombining, and mutating. It repeats this process until a predetermined stopping point such as finding the maximum fitness individual or going through a fixed number of iterations.

As opposed to GA, PBIL does not maintain a population and abstracts away from the crossover operator. PBIL attempts to evolve the genotype of a population rather than specific individuals within the population. PBIL is known to be a simpler type of optimization algorithm than a standard GA because of its focus on improving the singular population's probability vector instead of members of the population. A PBIL algorithm is run by iterating through four steps. First, assuming you have created a random probability vector, a population is generated from the probability vector. After this all of the individuals that were generated are evaluated for fitness, and the highest and lowest fitness individuals are used to tailor the probability vector. The probability vector will represent, once rounded, the most fit individual after all iterations have been completed.

Once both implementations were functioning, a set of experiments was designed to test the effectiveness of each algorithm on the MAXSAT problems. In these experiments the parameter settings were altered to test the algorithms through a large portion of the parameter space.

Next, section 2 will further discuss the details of MAXSAT problems, their applications, and the relevance of GA and PBIL to solve such problems. In sections 3 and 4 we will discuss GA and PBIL in more detail, outlining their implementations. Section 5 will outline the experiments we tested to determine the best parameters to use for testing GA and PBIL. In section 6 we will discuss and analyze the results from the experiments conducted. Finally, we will conclude our report with sections 7 and 8, exploring further experiments and ideas and summarizing our results.

2 MAXSAT

A Satisfiability (SAT) problem is a problem of determining whether a Boolean assignment of variables (X1, X2, … Xn) satisfies an expression. This expression is represented as a multitude of clauses that are each intersected with each other. To satisfy the expression, every clause must be satisfied. Each clause consists of a set of literals that are unioned with each other. These literals are either a single variable (X1) or its negation (!X1). For a clause to be satisfied, just one of the literals within the clause must be true. The clauses are represented in conjunctive normal form (CNF). An example of this form is given below.

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

This example contains three clauses, each containing 2 literals and only contains the variables X1 and X2. There are cases in which the variables are negated. If X1 is *false* and X2 is *true* then the first clause would evaluate to *false*, the second clause would evaluate to *true*, and the third clause would evaluate to *true*. However, allowing X1 to be true and X2 to be false then all of the clauses would be satisfied, and this specific Boolean assignment would be the optimal solution to the SAT example. The problems that we work with in this project are much more complex than the example, and there are no Boolean assignments that will satisfy every clause. Thus our aim will be to find a Boolean assignment that maximizes the amount of satisfied clauses.

A variation on SAT problems is MAXSAT problems. MAXSAT problems don't require the entire expression to be satisfied. Instead, MAXSAT searches for the Boolean assignment that satisfies that maximum number of clauses, and this will be considered the optimal solution. In this project we will be testing Genetic and PBIL algorithms on MAXSAT problems to determine if one of the methods outperforms the other in finding an optimal solution. Both of these algorithms are evolutionary, and so they should allow us to continuously improve our best Boolean assignment until we find one that is closer to optimal. This is because both algorithms use randomness and greediness to tailor their idea of the best Boolean assignment towards the optimal solution, which allows for an ample inspection of the search space. To optimize the performance of the two algorithms we will test different variations of the parameters to optimize their performance.

3 GENETIC ALGORITHMS

The general GA cycles through a process that mimics the process of reproduction and gene recombination within nature. A GA starts with the generation of a population of individuals. Typically, the individuals are represented as strings of symbols, such as bits or integers, and can generate a population randomly or seeded with individuals of high fitness. Once an initial population is generated, GA evaluates the fitness level of individuals and follows a selection process. Here we will be implementing and using three different selection processes, rank select, tournament select, and group select:

1) Rank Select: Rank select will rank the individuals in the population by fitness from 1 to N, where 1 is the individual with the worst fitness, and N is the individual with the best fitness. Each individual then gets a probability of being chosen that is assigned by the equation:

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

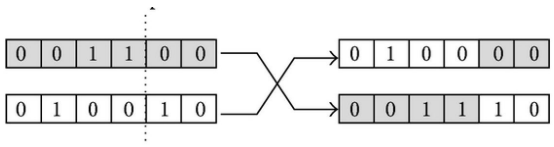
Where i is the rank of the individual. Individuals are then selected with this calculated probability until the amount of individuals chosen is the same as the initial population size.

2) Tournament Select: Tournament select chooses M individuals from the population and takes the k individuals with the best fitness. In this implementation, the tournament select method will take 2 individuals and take the one with the better fitness. This results in ½ the amount of individual, meaning the ½ that are chosen will be duplicated to replenish the population to its initial numbers.

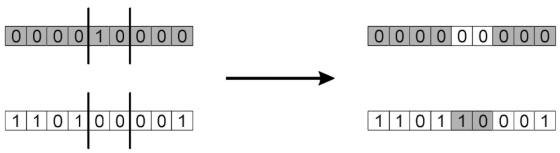
3) Group Select: Group select divides the population in half, calculates the total fitness for each half of the population and takes that half with greater fitness. Again, this results in ½ the amount of individuals, meaning the ½ that was chosen will be duplicated to get back to the amount of individuals in the initial population.

With a breeding pool selected, recombination combines pieces of the chosen individuals to spawn the new generation. Recombination is typically seen in the form of crossover - one point, two point, or uniform:

1) One Point Crossover: between the two individuals, select one point, and swap the genetic makeup from that point over.



2) Two Point Crossover: within the two individuals, select two points, and swap the genetic makeup amongst the two points.



3) Uniform Crossover: within the two individuals, look at each pair of genes and select one of the two at random. Only generates one child, therefore either double the child.

The crossover methods select two individuals from the breeding pool, then carry out some form of crossover depending on a pre-specified, given probability. After crossover is completed across the entire population, the individuals will be passed to mutation. When mutated, each symbol of the individuals has the possibility of being altered with some probability.Typically, these probabilities are small, spanning *[0.001, 0.02]*.

The process of fitness evaluation, breeding pool selection, recombination by means of crossover, and mutation is repeated until an optimal solution is found or the maximum specified number of iterations are completed. The main method will contain the following loop to carry out this repetition:

```
iterationNum = 0;
// if not optimal and not maxGenerations
while(numIterations != numGenerations &&
globalBestIndividual.getFitness(clauses, numClauses)
!= numClauses) {
        // selection (ts, rs, or gs)
        if (selectionType.equals("ts")) {
                this.tournamentSelect(); }
        else if (selectionType.equals("rs")) {
                this.rankSelect(); }
        else if (selectionType.equals("gs")) {
                this.groupSelect(); }
// crossover (1c, 2c, uf)
if (crossoverMethod.equals("1c")) {
                this.onePointCrossover(); }
else if(crossoverMethod.equals("uc")) {
                this.uniformCrossover(); }
//mutation
this.mutation(mutationProb);
```

GA has various advantages and disadvantages. While GA can handle large search spaces, the representation of these spaces can be problematic. Additionally, there are many gaps and questions involved in the implementation of GA. What is the fitness function and how long does it take? Which type of selection and crossover to utilize? How to set all the parameters? What will its time complexity be? Regardless, the algorithm's straightforward concept and ability to deal with diverse problems drives the wide usage of GA among Nature Inspired Algorithms.

4 POPULATION BASED INCREMENTAL LEARNING

Population Based Incremental Learning (PBIL) is an alternative optimization algorithm that was created as a different approach to solve problems for which GA were traditionally used. The PBIL's main hallmark is its probability vector. This probability vector represents the genotype of the population, so unlike GA, PBIL isn't required to maintain a population to evolve towards an optimal solution. PBIL uses competitive supervised learning to tailor the probability vector towards a final representation of an optimal individual. This is another difference between PBIL and GA. Crossover is used in GA to improve the chances of producing a potential high fitness solution whereas PBIL uses this supervised competitive learning. Competitive learning takes some specific number of high fitness individuals and updates the probability vector towards these specified individuals. In this project only the most fit individual and the least fit individual will be used to update the probability vector. However, if the most fit and least fit individuals are the same then the probability vector will only be updated towards the most fit individual. Individuals in a PBIL are

represented as bit strings. The probability vector specifies probabilistically what each bit of an individual's string could be in a range from [0,1], with 0 meaning there is a 0% chance that a randomly generated individual has the specific trait and 1 meaning there is a 100% chance that a randomly generated individual will have the specific trait. This probability vector reflects the distribution of traits within a population, and specifically the estimated optimal individual based on its learning at any given time. The specific procedure of PBIL is outlined below:

1) Generate the probability vector where each element of the vector is 0.5. These elements can range from [0, 1].

2) Generate a previously specified amount of candidate solutions from the probability vector as the population.

3) Evaluate those individuals for fitness from a previously specified fitness function that the designer deems appropriate, and then rank them according to their fitness.

4) Competitive learning will now be used to improve the probability vector based on our highest and lowest fitness individuals. The probability vector will be tailored towards the most fit individual using the pseudo code below where the LEARNING RATE is a rate between [0,1] (a standard value is 0.1) that is picked by the designer which will determine how much to shift the probability vector. The higher the LEARNING RATE the more shifted the probability vector will be to the selected individuals.

```
For i = 1 to LENGTH OF VECTOR do
    P[i] = P[i] * (1.0 - LEARNING RATE)
    + best_vector[i] * LEARNING RATE
```

Then the probability vector will tailor itself away from the least fit individual using the pseudo code below.

```
For i = 1 to LENGTH OF VECTOR do
    If (best_vector[i] != worst_vector[i])
        P[i] = P[i] * (1.0 - NEGATIVE
        LEARNING RATE) +
        best_vector[i] * NEGATIVE
        LEARNING RATE
```

5) The population may be removed from relevance and discarded.

6) Mutations may now occur. Mutations are determined using the pseudo code below where MUTATION_PROBABILITY is between the range of [0,1] (a normal value for this is 0.01) and is the chance that a singular bit in the probability vector will be shifted. The bit will be shifted by a factor called the MUTATION_SHIFT which is a value in the range of [0,1] that is the amount for mutation to affect the probability vector shifting:

```
For i = 1 to LENGTH OF VECTOR do
    If (random (0,1) <
MUTATION_PROBABILITY
    If (random (0,1) > 0.5)
    mutate_direction = 1
        Else mutate direction = 0
```

*P[i] = P[i] * (1.0 - MUTATION_SHIFT) + mutate_direction * MUTATION_SHIFT*

7) Iterate through steps 2 through 6 until the probability vector converges on what would be the optimal solution.

The end goal of a PBIL is that after enough iterations the probability vector, when rounded, will be the optimal solution. PBIL can be differentiated from GA because it doesn't require a population to be stored and instead uses a singular probability vector. This results in less memory being required to implement PBIL than GA would require. PBIL doesn't use recombination or selection like in GA, and this allows it to work more quickly as it doesn't have to run through the complicated selection and recombination code on each iteration. This usually results in a quicker convergence towards the optimal solution than that of GA. However, PBIL shares many similar issues with GA, and also has a few more that are specific to PBIL. The issues that they share are determining how to represent an individual, how to create the fitness function, and determining how many iterations will be required to get convergence. The additional issues with PBIL are the difficulty required to determine the number of individuals needed for for each iteration, the number of individuals to select for competitive learning, the learning rate, and also the different rates required for mutation.

5 EXPERIMENTAL METHODOLOGY

While we ran our algorithms against multiple files, we ran our tests to determine the best parameters for GA and PBIL and to compare the two using the file "s2v120c1200-3.cnf".

For the GA experiments, we began by first varying the mutation probability and the crossover probability. The mutation probability was tested at 0, 0.005, 0.01, 0.05, 0.1, 0.5 and 1 (*Table 1*). Each of these tests used 100 individuals, 0.7 crossover probability, 1-point crossover, tournament select, and 1,000 iterations. This resulted in 7 tests, where we did 5 trials each, therefore 35 tests. Next crossover probability was tested (*Table 2*). Using similar variables 100 individuals, 0.1 mutation probability, 1-point crossover, tournament select, and 1,000 iterations. We tested the crossover probabilities 0, 0.2, 0.4, 0.6, 0.8, and 1. This resulted in 6 tests, with 5 trials each, therefore 30 tests. Through these tests we created a pairing of 0.05 mutation probability and 1.0 crossover as a set of best variables to test the best crossover type later on.

Next the interaction between crossover probability and mutation probability was tested by using different combinations of the two (*Table 3*). This allowed us to determine three more sets of best combinations to test with later to determine the best crossover type. Using 100 individuals, 1-point crossover, tournament select, and 1,000 iterations, we tested each combination of the crossover probabilities [0.1, 0.5, 1] and the mutation probabilities [0.005, 0.01, 0.1]. This was 9 tests with 5 trials each, resulting in 45 tests.

After these 110 tests were done we were able to use these 4 couples of crossover probability/mutation probability to test the two crossover types, 1-point crossover and uniform crossover (*Table 4*). These 40 tests resulted in 2 triplets of

crossover probability/mutation probability/crossover type to test on each of the 3 selection types (*Table 5*). The tests of these 2 triplets on each of the selection types resulted in 30 tests that gave us one quad group of crossover probability/mutation probability/crossover type/selection type that satisfies the highest percentage of clauses. This group of crossover probability/mutation probability/crossover type/selection type will be used when testing GA to determine if GA or PBIL should be used for MAXSAT problems.

As suggested in the provided pseudocode, the experiments for PBIL began with the following parameters: 0.1 learning rate, 0.075 negative learning rate, 100 bits generated (population size), 0.02 mutation probability, 0.05 as the amount a mutation alters the value in the bit position, and 1,000 iterations. The first variable we varied was the mutation probability. For mutation probability we tested the values 0, 0.02, 0.05, 0.1, 0.5 and 1 with the other variables set as above. This resulted in 30 total tests (6 cases with 5 trials each) and allowed us to determine the mutation probability that should be used in PBIL to find an optimal solution (*Table 8)*.

Next, we varied the positive learning rate. Using the variables 0.075 negative learning rate, 100 bits generated (population size), 0.02 mutation probability, 0.05 as the amount a mutation alters the value in the bit position, and 1,000 iterations we tested the learning rates 0, 0.2, 0.05, 0.1, 0.5 and 1 *(Table 9)*. These 30 tests led to the discovery of the 3 best learning rates for our implementation of PBIL, which we will later test in combinations with the best negative learning rates to find the best pairing.

Finally, we varied the negative learning rate to determine the best value. Again, using the parameters 0.1 learning rate, 100 bits generated (population size), 0.02 mutation probability, 0.05 as the amount a mutation alters the value in the bit position, and 1,000 iterations, we tested the negative learning rate values 0, 0.05, 0.075, 0.1, 0.5, and 1 *(Table 10)*. This resulted in 30 total tests and allowed us to determine the 2 best negative learning rate to use when testing with PBIL.

Using the best 3 learning rates and the best 2 negative learning rates, we tested different pairings to determine which paring should be used in PBIL when testing for an optimal solution *(Table 11)*. We selected the best performing pair and used that in testing PBIL from this point on.

Once the best variables to test both GA and PBIL with were determined, we were able to compare them with varying amount of iterations and size of populations (*Tables 6, 7, 12, and 13*). Ultimately this determined whether to use GA or PBIL based on the number of iterations allowed and population size and allowed us to make a suggestion on which algorithm to use on MAXSAT problems.

6 RESULTS

Table 1 shows the results of varying the mutation probability while keeping all other variables constant. These tests showed that 0.5 performed the best.

Table 2 displays the results of similarly keeping all variables the same and altering the crossover probability. This set of experiments showed 1.0 performed the best. This then allowed us to have one set of crossover probability, 1.0, and mutation probability, 0.5, to test later.

Table 3 shows the interaction of crossover probability and mutation probability, resulting in 3 more sets of crossover probabilities/mutation probabilities, (1.0, 0.1), (0.1, 0.5), (1.0, 1.0), to use in tests later.

Table 4 shows the results of testing these 4 pairs of crossover probability/mutation probability with the two crossover types. The result of this was 2 triplets of crossover probability/mutation probability/crossover type (1.0, 0.05, uniform crossover), (1.0, 1.0, one-point crossover), that performed the best.

Table 5 experimented with these 2 triplets of crossover probability/mutation probability/crossover type against the 3 types of selection and was able to identify a group of crossover probability/mutation probability/crossover type/selection type that outperformed all the rest (1.0, 0.05, uniform crossover, tournament select).

The next two tables, Table 6 and Table 7, used this group of crossover probability/mutation probability/crossover type/selection type to test against varying numbers of iterations and individuals, finding that 10,000 iterations and 1,000 individuals were the best two parameters. Through all these tests varying the parameters of GA, we were able to determine a set of parameters (1.0 crossover probability, 0.05 mutation probability, uniform crossover, tournament select, 10000 iterations, 1000 individuals), that led GA to perform the best that it could.

Then with PBIL, we started by varying mutation probabilities, positive learning rate, and negative learning rate.

Table 8 highlights the range of mutation probabilities (.05, .075, .1) that perform the best of the group. We selected .075 as the optimal mutation probability because it outperforms its peers. Table 9 shows the positive learning rate (.05, .075, .1) performs the best. Lastly, Table 10 gives the best negative learning rates (.05, .075).

With the best mutation probability (.075) and the combinations of the selected rates, Table 11 found that 0.05 of both positive and negative learning rates performed better compared to their peers.

In a similar manner to GA's Tables 6 and 7, PBIL's Tables 12 and 13 use the best combination of parameters to test various number of iterations and individuals. As to be expected, in both cases, as the number increases, the algorithm performs better. Through these iterations we were able to determine the optimal parameters (0.075 mutation probability, 0.05 positive learning rate, 0.05 negative learning rate, 10000 iterations, and 1000 individuals).

In conclusion, PBIL outperforms GA by an average of 6%.

7 FURTHER WORK

Given more time, one of the first things we would explore more in regard to this experiment would be to test on more MAXSAT problems, both smaller files and larger files. While we have run our code on multiple files, we only ran our experimental tests on one file so that we were able to explore the parameter space as much as possible while keeping the difficulty of the MAXSAT problem constant.

We would also further explore the parameter space. Given our limited time we were able to explore the parameter space to a large enough degree for us to make conclusions about GA versus PBIL on MAXSAT problems, but there is a still a large portion of the parameter space that we were unable to explore. Further exploration of the parameter space would allow us to

determine the exact best parameters for both GA and PBIL that lead to their optimal solutions and therefore would make our tests more accurate.

Finally, we would explore the time complexity of our algorithms and how long it takes them to run. We expect that with larger and more complex MAXSAT problems there could be a large difference in the time it takes each algorithm to run and that would be worth looking further into.

8 CONCLUSIONS

In addition to looking at the results from the tests we conducted, we found it significant to report that the implementation of PBIL was much quicker and more straight forward than the implementation of GA, adding to its case for being a more viable option for solving MAXSAT problems.

We hoped to determine which algorithm would be better suited to finding the optimal solution of a MAXSAT problem with different population sizes and maximum iterations allowed. Tables 6 and 7 show our results of this testing on our optimized GA, and tables 12 and 13 should our results of this testing on PBIL. These tables showed us that PBIL was able to find a better solution for every instance of changing population, and changing the maximum number of iterations. PBIL's average best solution ranged from 1%-6% more clauses satisfied than the best solution from the GA. This means that PBIL could create a solution that satisfied up to 72 more clauses that a GA could.

The difference of clauses satisfied was greater the more amount of iterations were completed: for 1000 iterations there was a 6% difference in favor of PBIL, and for 10 iterations there was a difference of around 3% in favor of PBIL. It is also interesting to note that GA only improved by about 2% from 10 iterations to 1000, while PBIL improved over 5% in that same gap.

The difference was about 6% higher for PBIL for all comparisons of population size that we tested. It is interesting to note that both algorithms improved by about 1% percent from 10 individuals to 1000.

TABLES AND GRAPHS:

GA TESTING:

Table 1: Varying Mutation Probability: 100 individuals, 0.7 crossover prob, 1-point crossover, tournament select, 1000 iterations

Mutation Probability	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
0	80.58%	80.25%	80.00%	80.17%	79.67%	79.56%
0.005	80.25%	79.58%	79.67%	79.42%	79.83%	79.45%
0.01	80.08%	79.42%	79.50%	79.83%	79.58%	79.51%
0.05	79.92%	79.50%	80.08%	80.67%	79.92%	80.37%
0.1	79.58%	79.67%	80.17%	80.08%	79.58%	79.94%
0.5	80.17%	79.92%	79.92%	79.75%	79.92%	79.73%
1	80.50%	79.50%	80.00%	79.67%	80.17%	79.82%

Table 2: Varying Crossover Probability: 100 individuals, 0.01 mutation prob, 1-point crossover, tournament select, 1000 iterations

Crossover Probability	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
0	79.83%	79.17%	79.50%	79.25%	79.42%	79.43%
0.2	79.50%	79.75%	79.50%	79.67%	79.42%	79.57%
0.4	79.67%	79.67%	79.33%	80.25%	79.92%	79.77%
0.6	80.00%	80.17%	79.58%	79.58%	79.67%	79.80%
0.8	79.75%	79.25%	79.75%	79.92%	79.75%	79.68%
1	79.67%	80.08%	79.83%	79.25%	80.33%	79.83%

Table 3: Varying BOTH Crossover and Mutation Probability: 100 individuals, 1-point crossover, tournament select, 1000 iterations

Crossover Probability	Mutation Probability	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
0.1	0.005	80.00%	80.00%	79.75%	79.50%	79.58%	79.77%
0.5	0.005	79.33%	80.33%	79.83%	80.08%	79.58%	79.83%
1	0.005	79.67%	79.83%	79.33%	80.25%	79.83%	79.78%
0.1	0.01	79.08%	79.67%	79.75%	79.92%	79.33%	79.55%
0.5	0.01	79.67%	80.08%	79.75%	79.75%	80.00%	79.85%
1	0.01	79.92%	80.08%	79.58%	79.42%	79.58%	79.72%
0.1	0.1	79.67%	80.25%	79.75%	80.08%	79.92%	79.93%
0.5	0.1	79.75%	79.92%	79.33%	80.50%	79.75%	79.85%
1	0.1	80.92%	80.17%	80.25%	79.58%	80.17%	80.22%
0.1	0.5	79.83%	79.75%	80.83%	80.67%	80.17%	80.25%
0.5	0.5	80.08%	79.75%	80.08%	79.60%	79.42%	79.79%
1	0.5	79.67%	79.92%	79.67%	80.08%	80.17%	79.90%
0.1	1	79.50%	79.50%	80.41%	79.84%	79.50%	79.75%
0.5	1	79.42%	79.83%	79.83%	79.83%	80.00%	79.78%
1	1	80.25%	79.83%	80.08%	80.17%	80.17%	80.10%

Table 4: Varying Crossover Type with Crossover prob/Mutation prob pairs: 100 individuals, tournament select, 1000 iterations

mutation/cross	Crossover Type	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
0.05/1.0	1c	79.50%	79.66%	80.25%	79.67%	79.50%	79.72%
	uc	79.92%	80.08%	80.08%	79.75%	80.00%	79.97%
mutation/cross	Crossover Type	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
1.0/0.1	1c	79%	79.08%	79.42%	79.58%	79.50%	79%
	uc	77.83%	78%	77.83%	77.33%	77.58%	77.71%
mutation/cross	Crossover Type	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
0.1/0.5	1c	79.75%	79.33%	79.83%	79.67%	80.42%	79.80%
	uc	80.08%	79.42%	79.75%	79.92%	79.75%	79.78%
mutation/cross	Crossover Type	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
1.0/1.0	1c	79.25%	79.83%	80.00%	80.25%	80.50%	79.97%
	uc	77.67%	78.08%	77.92%	77.92%	77.75%	77.87%

Table 5: Varying Selection Type with Crossover prob/Mutation prob/Selection type triplets: 100 individuals, 1000 iterations

Cross/ Mutation	Selection Type	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
1.0/0.05 uc	rank select	78.83%	78.67%	78.92%	79.58%	78.67%	78.93%
	group select	79.67%	80.17%	79.58%	79.67%	79.83%	79.78%
	tournament select	79.58%	80.00%	80.08%	80.58%	80.08%	80.06%
1.0/1.0 1c	rank select	78.33%	77.83%	77.92%	78.50%	78.83%	78.28%
	group select	79.92%	79.75%	79.42%	79.58%	79.58%	79.65%
	tournament select	80.17%	79.75%	80.33%	79.58%	79.75%	79.92%

Table 6: Varying Iterations: 100 individuals, 1.0 crossover prob, 0.05 mutation prob, uniform crossover, tournament select

Iterations	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
10	78.33%	79.00%	78.25%	78.25%	78.25%	78.42%
100	79.08%	79.58%	78.83%	80.17%	80.25%	79.58%
500	79.58%	80.42%	79.75%	80.08%	79.83%	79.93%
1000	79.98%	79.75%	79.83%	79.96%	79.33%	79.77%
10000	80.33%	80.42%	80.25%	80.33%	80.16%	80.30%

Table 7: Varying Individuals: 1.0 crossover prob, 0.05 mutation prob, uniform crossover, tournament select, 500 iterations

Individuals	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
10	79.83%	79.50%	78.75%	79.33%	79.17%	79.32%
50	79.58%	79.25%	79.75%	79.08%	79.33%	79.40%
100	79.50%	79.58%	79.58%	79.33%	79.25%	79.45%
1000	80.17%	80.67%	80.17%	80.17%	80.42%	80.32%

PBIL TESTING:

Table 8: Varying Mutation Probability: 100 individuals, 0.1 learning rate, 0.075 negative learning rate, 0.05 as the amount the mutation alters the value, 1000 iterations

Mutation Probability	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
0	85.75%	85.42%	86.25%	86.42%	86.25%	86.02%
0.02	85.83%	86.67%	85.83%	86.67%	86.08%	86.22%
0.05	86.67%	86.67%	86.67%	86.08%	86.50%	86.52%
0.075	86.67%	86.50%	86.67%	86.67%	86.42%	86.58%
0.1	86.67%	86.67%	86.67%	86.08%	86.67%	86.55%
0.5	84.83%	85.17%	85.33%	85.00%	84.92%	85.05%
1	82.92%	82.92%	83.58%	83.42%	83.42%	83.25%

Table 9: Varying Learning Rate: 100 individuals, 0.02 mutation probability, 0.075 negative learning rate, 0.05 as the amount the mutation alters the value, 1000 iterations

Learning Rate	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
0	84.83%	83.75%	84.83%	84.58%	84.67%	84.53%
0.05	86.67%	86.67%	86.67%	86.50%	86.50%	86.60%
0.075	86.67%	86.08%	86.67%	86.67%	86.67%	86.55%
0.1	86.67%	86.67%	86.67%	86.08%	86.50%	86.52%
0.5	84.75%	85.75%	85.33%	85.50%	84.92%	85.25%
1	85.25%	85.42%	85.50%	85.50%	85.50%	85.43%

Table 10: Varying Negative Learning Rate: 100 individuals, 0.02 mutation probability, 0.1 learning rate, 0.05 as the amount the mutation alters the value, 1000 iterations

Negative Learning Rate	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
0	86.25%	86.25%	85.83%	86.50%	85.83%	86.13%
0.05	86.67%	85.92%	86.67%	86.50%	86.33%	86.42%
0.075	86.67%	85.75%	86.67%	86.67%	86.42%	86.43%
0.1	86.08%	86.08%	85.83%	86.67%	85.83%	86.10%
0.5	86.33%	86.08%	85.25%	86.50%	86.33%	86.10%
1	86.42%	86.25%	86.42%	85.75%	86.08%	86.18%

Table 11: Varying BOTH Positive and Negative Learning Rates: 100 individuals, 0.075 as the amount the mutation alters the value, 1000 iterations

pos, neg learning	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
0.05, 0.05	86.50%	86.67%	86.67%	86.50%	86.67%	86.60%
0.05, 0.075	85.92%	86.08%	86.67%	86.50%	86.50%	86.33%
0.075, 0.05	86.33%	86.50%	86.67%	86.00%	85.42%	86.18%
0.075, 0.075	85.92%	86.08%	86.50%	86.67%	86.50%	86.33%
0.1, 0.05	85.92%	86.67%	86.67%	86.08%	86.67%	86.40%
0.1, 0.075	86.00%	86.67%	86.25%	85.50%	86.67%	86.22%

Table 12: Varying Iterations: 100 individuals, 0.05 learning rate, 0.05 negative learning rate, 0.075 as the amount the mutation alters the value

Iterations	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
10	80.83%	82.25%	80.75%	81.25%	80.67%	81.15%
100	85.33%	84.92%	85.25%	84.75%	85.42%	85.13%
500	85.42%	86.67%	86.50%	86.67%	86.50%	86.35%
1000	86.50%	86.67%	86.67%	86.67%	86.67%	86.64%
10000	86.67%	86.67%	86.67%	86.08%	86.67%	86.55%

Table 13: Varying Individuals: 0.05 learning rate, 0.05 negative learning rate, 0.075 as the amount the mutation alters the value, 500 iterations

Individuals	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Average
10	84.83%	85.83%	85.33%	85.08%	85.42%	85.30%
50	86.50%	86.08%	86.67%	86.67%	86.67%	86.52%
100	86.08%	86.67%	86.67%	86.67%	86.08%	86.43%
1000	86.67%	86.50%	86.67%	86.67%	86.67%	86.63%