Comparison of PBIL and PSO on MAX-SAT Problems

Jigyasa Subedi, Souleman Toure, Diyaa Yaqub

Abstract

We implemented a solution to the MAXSAT problem, an NP-complete optimization problem that aims to satisfy the maximum number of clauses for a given boolean formula, using a modified version of Particle Swarm Optimisation (PSO). We compared the solution (percentage satisfiability) found through evolving a probability vector using PSO to the MAXSAT solution found by population-based incremental learning as it uses probability-based vectors and makes adjustments to it through competitive supervised learning. We ran a set of experiments and varied the number of iterations, number of particles, and file size for PBIL, and topology as well for PSO. To compare the algorithms, we primarily focused on the percentage satisfiability each algorithm found as a solution to MAXSAT, but also their difficulty of implementation, execution time, size of problems they solve, and ease of use. We used these metrics to determine their efficacy and recommended the use of PBIL to solve the MAXSAT problem.

1 Introduction

In this project, we solved a variation of the Boolean Satisfiability problem (SAT), which determined whether there can be an assignment of variables within a Boolean formula that makes the formula true. When the interpretation/assignment did not exist, the formula was unsatisfiable and when it did exist, it was considered satisfiable. The maximum satisfiability problem (MAXSAT) is a variation of the SAT problem that looks at the maximum number of clauses that can be satisfied, as opposed to satisfiability of all clauses.

We implemented the solution to MAXSAT using Particle Swarm Optimization (PSO) and compared that to our previous implementation of Population Based Incremental Learning (PBIL). Population Based Incremental Learning is an optimization algorithm, which combines components from Genetic Algorithms and Competitive Learning. Using probability vectors representing an entire population, PBIL ranks individuals based on fitness and trains the entire population towards the most fit individual. Using a learning rate, the probability vector adjusts to become closer to the most fit individual until the population converges. After repeated iterations, mutations, and creations of new generations, the algorithm finds its optimal solution. The PSO algorithm is a stochastic optimization technique based on swarm behavior. It uses a simple mechanism that mimics swarm behavior in birds flocking and fish schooling to guide the particles to search for global optimal solutions. The neighborhood topologies we used in our algorithm were global, ring, von Neumann and random.

Since there are numerous parameters for PSO and PBIL, we varied the number iterations, the neighborhood topologies, and the number of particles/individuals. We ran experiments to compare and determine which parameters and which algorithm found the best solution to the MAXSAT problem. Based on the results from our experiments, we would recommend the use of PBIL as a solution for MAXSAT.

In Section 1, we describe what the MAXSAT problem is, and how PSO and PBIL work to solve it. In Section 2, we outline our experimental methodology. In Section 3, we analyze the results from the manipulation of variables in our experiments. Finally, we conclude with a holistic analysis and explanation of reasoning behind the recommendation of a PBIL algorithm to solve the MAXSAT problem.

2 MAX-SAT

A MAXSAT problem looks at the maximum number of clauses that can be satisfied for a given Boolean expression. A MAXSAT problem contains n variables and m clauses. The problem is in conjunctive normal form (CNF). Each clause consists of a disjunction of literals and each literal is either a variable or its

negation. The problem aims for a boolean assignment of variables that maximizes the number of clauses that are satisfied. In our case, the literals of the MAXSAT problem are integers and each clause consists of two literals. We intend to use PBIL and GA to find an assignment of the variables, as represented by literals, that lead to the maximum number of clauses evaluating to true.

The MAXSAT problem is considered to be NP-complete, which indicates that the solution can be verified in polynomial time and that an algorithm can find this decision by trying all possible solutions. The problems in this category are notoriously difficult to solve efficiently. The use of PBIL and GA can be applied to MAXSAT since they introduce randomness with certain levels of bias towards "good" solutions in a way that might lead to a faster and effective assignment of variables.

Solving the MAXSAT problem has a wide range of applicability for optimization problems. For example, large-scale event planning could have its various organizational/logistical components represented by variables. The clauses could be formed based on the desired aspects for the event and MAXSAT could help determine whether those aspects can happen based on budget-constraints. If all the clauses are satisfied, the event is within budget. If most of the clauses are satisfied, there might need to be some compromises and changes. If very few clauses are satisfied, the plan needs to change drastically. MAXSAT allows for more informed decision-making, as opposed to using the generic Boolean satisfiability problem (SAT) for this, where you would only know if the event is within budget, or not; you wouldn't be able to make smaller informed adjustments. Real-life implementations and uses of MAXSAT have included model-based diagnoses, fault localization, correlation clustering, and more.

3 Population Based Incremental Learning

Population Based Incremental Learning (PBIL) is designed to be an optimization algorithm, selecting the best or most fit elements considering some criterion. It combines genetic algorithms with competitive learning. PBIL specifically evolves probability vectors for the genotype of an entire population as opposed to more common genetic algorithms which use individual members. PBIL is a stochastic search based algorithm, meaning that it uses randomness to escape local optima when considering future states of the system.

Equation:

$$PV[i] = PV[i] \times (1 - LR) + SVB[i] \times LR$$

Where

- $PV[i]$ is i^{th} bit of probability vector in this generation
- LR is learning rate
- $SVB[i]$ is i^{th} bit of best solution vector of the current generation

PBIL starts with an arbitrary probability vector and generates a population from said vector. As we generate individuals, we also rank their fitness levels. Using these fitness levels and competitive supervised learning, we can adjust the probability vector based on the fittest individual. This adjustment should either increase the probability that we generate individuals with the alleles expressed by our fittest individual, or can alternatively decrease the probability that we express traits from the least fit individual. The labeling of the ranks and training of populations towards the fittest individual makes PBIL supervised as some data is tagged with being the "correct output".

Our learning rate (alpha) would determine how much we want our vector to imitate the best individual. This helps balance the exploration and exploitation issue by taking enough information from the fit individual to lead our algorithm towards producing more fit populations; however, it limits how much we absorb from fit

individuals as to not exploit and have early convergence. After updating our probability vector we should proceed to mutating. Here we iterate over all of the probability vectors with a mutation probability condition that, if met, will randomly increase or decrease the vector at that index.

The general process is as follows: first, we iterate over the array of probability vectors. We then generate a random double and if it is less than the mutation probability, we execute the conditional which has 50% probability of increasing or decreasing the probability of said vector by the mutation amount. Then we repeat this process until our probability vectors stabilize and don't learn much from any new iterations suggesting the population has converged.

4 Particle Swarm Optimization

PSO is a stochastic optimization technique based on the movement and intelligence of swarms. It was proposed by Eberhart and Kennedy in 1995. PSO uses the concept of social interaction to solve the given problem. It uses a number of particles (candidate solutions) that constitute a swarm moving around in the search space, looking for the best position. Particles will move through a multidimensional search space to find the best position, and thus the best solution, in that space. This algorithm is inspired from swarm behavior such as bird flocking. Particles tend to keep moving in the current direction and same speed, but are biased towards their own experience and the swarm's experience. The goal is to find better and better solutions. In each iteration, each particle is updated using two "best" values: pbest and nbest. Pbest, personal best, is the best solution that particle has found so far. Nbest, the neighborhood best, is the best solution that any particle in the swarm has found so far within the particle's neighborhood. A swarm of particles fly through the search space being attracted by both, their personal best position and the neighborhood best position found so far within the swarm.

The general outline of the algorithm is as follows. Our PSO algorithm works to solve the functions by generating a user-inputted number of particles. Each particle is initialized with a random position in the solution space and velocity (which are vectors in the d-dimensional space); the initialisation range for the position was between 0 and 1 because each position in the vector corresponds to a probability and the initialization range for velocity was between -2 and 4 because these are common values used when using PSO to evaluate test function such as Rosenbrock, Ackley, and Rastrigin. In our algorithm, each particle's position acts as a potential solution to the MAXSAT problem. Therefore, each position is a probability vector whose index positions correspond to the assignment of booleans to the MAXSAT problem. Therefore, each particle's position has an associated fitness percentage, which is the percentage of clauses satisfied in the MAXSAT problem with this probability vector. During every iteration, our algorithm updates the velocity vector and position/probability vector accordingly using the PSO update velocity equation. Each particle stores and tracks its personal best fitness percentage (the maximum percentage of clauses satisfied). The particles are also divided into neighborhoods. The best solution from each of the particles in the neighborhood (the neighborhood best) and the particle's personal best, is used to update each particle's velocity vector. Although the equation is influenced by these values, it also introduces randomness through the generation of random numbers and the constriction factor. According to this updated velocity vector, the position is updated, and the new position/probability vector is further evolved for a specified number of iterations and the best solution found between all the particles with a given number of iterations is outputted.

Just a note, our PSO algorithm uses constriction factor instead of inertia as the constriction factor keeps the velocity in check.

Equation:

$$\vec{v_i} \leftarrow \chi(\vec{v_i} + \vec{u}(0, \phi_1) \otimes (\vec{p_i} - \vec{x_i}) + \vec{u}(0, \phi_2) \otimes (\vec{g_i} - \vec{x_i}))$$
$$\vec{x_i} \leftarrow \vec{x_i} + \vec{v_i}$$

Where

- $\vec{X}_i$ is the position of particle i
- $\vec{V}_i$ is the velocity of particle i
- $y_i = f(\vec{X}_i)$
- $\vec{P}_i$ is the location of personal best of particle i
- $pbest_i = f(\vec{P}_i)$
- $\vec{g}_i$ is the location of global best
- $gbest_i = f(\vec{g}_i)$

The PSO algorithm has a lot of advantages. It is easy to implement and relatively fast. It can handle dynamic problems quite efficiently. More importantly, compared to an algorithm like the genetic algorithm, it has fewer individuals and fewer parameters with better rule-of-thumb values. It is easy to hybridize with other algorithms to improve its performance. It has also been applied successfully to a wide variety of search and optimization problems. Some PSO applications include neural networks, communication networks, image and video processing. However, there disadvantages of using PSO include a local solution and leading to premature convergence.

4.1 Neighborhood Topologies

PSO topologies outline the neighbor relationship and interaction between particles, which controls the propagation of information in the particle swarm. Because the topologies describe the subset of particles with which each particle exchanges information, the topologies directly affect the swarm's optimization ability and its convergence. The global topology is the most widely used topology where all particles are directly connected to one another. So, each particle can share information with any other particle in the swarm. In the ring topology, the particles are directly connected to their m immediate neighbors. When m = 2, the particle is affected by only its immediately adjacent neighbors. The von Neumann topology operates as a grid structure, where each particle is connected to its four neighbors: top, bottom, left, and right. The random neighborhood topology is such that a random neighborhood of size k is initially created for each particle by choosing k-1 other particles randomly without repetition. We varied the traditional random topology and implemented it so that each particle has a 0.2 percent chance of being included in the particle's neighborhood.

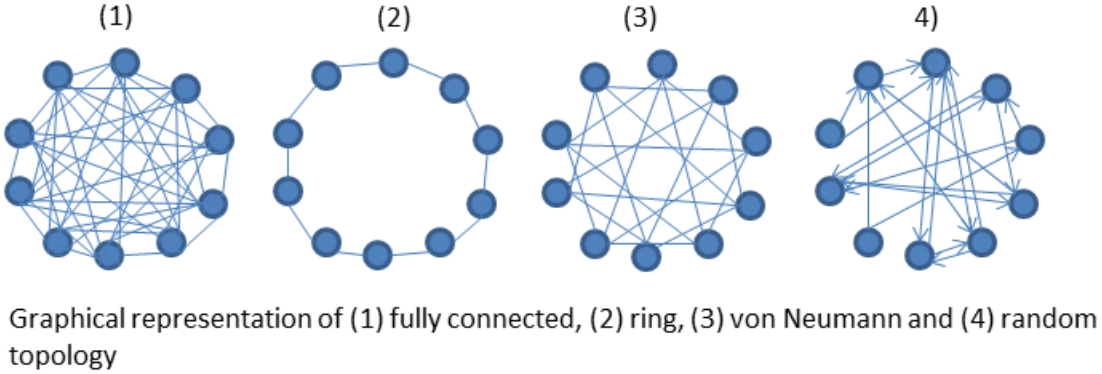

Figure 1: Different Neighborhood Topologies

5 Particle Swarm Optimization on MAX-SAT

PSO updates position and velocity. To avoid constraining the PSO algorithm, we modified it to evolve the probability vector in a way that it could still be applicable to the MAXSAT problem. The position for PSO is the position in the solution space. In our case, the position was the probability vector and we updated it as such using the PSO update velocity equation. However, as we used the update equation, the values within our probability vector could go outside of values outside of the range of 0 and 1. We allowed the probability vector to change according to the equation and go to this determined position but translated it back to a probability through normalization. For each variable, we maintained a maximum and minimum value calculated by the PSO equation over the iterations. Based on the minimum and maximum seen so far at that iteration, we normalized the value to put it in the range of 0 and 1 and used this as the probability for the probability vector. The probability vectors are used to generate an assignment to the boolean variables in the MAXSAT problem according to the probabilities. This allows us to calculate the percentage of clauses satisfied based on a probability vector evolved over iterations through PSO.

6 Experimental Methodology

Both our PBIL and PSO algorithms update/transform a probability vector for a number of iterations to find a solution to the MAXSAT problem. For PBIL, each iteration uses the learning rate to update the probability vector. For PSO, each iteration updates the position and velocity using the PSO equation to update the probability vector. Furthermore, both algorithms do this numerous times and get the best. In PBIL, there are multiple individuals that evolve their own probability vectors. In PSO, there are multiple particles that move around the solution space to evolve their own probability vectors.

To compare the algorithms, we varied neighborhood topologies, the number of iterations and the number of individuals/particles depending on the algorithm in question for 3 MAXSAT problems of varying sizes.

From previous experiments using PBIL to find a solution to MAXSAT, we concluded that the best learning rate that yielded a probability vector with the highest percentage satisfiability was 0.1. We kept this learning rate constant for all PBIL tests. Since we have never used PSO to evolve a probability vector, we didn't know how neighborhood topologies would affect solutions found and decided to experiment using Ring, Von Neumann, Global, and Random topologies. As we varied topologies, iterations and number of particles were held constant.

In terms of iterations and number of particles, we were interested in experimenting using the same parameters for both PBIL and PSO for comparison analysis. Our experimental methodology consisted of testing both PSO and PBIL with 16, 30, and 49 particles/individuals. We recorded the best percentage satisfiability to the MAXSAT problem found by each algorithm for 10, 100, 1000, and 10000 iterations for each variance in particles/individuals. Each of these experiments were done 4 times as we experimented by varying and changing it to the 4 topologies. We conducted all rounds of experiments for 3 MAXSAT problem files to see how the algorithms performed based on the number of variable assignments it had to find. We tested our algorithm on problems with filenames v140-c1200.cnf, v8385-c21736.cnf, v22136-c51563.cnf. The number after 'v' is the number of variables in the MAXSAT problem and the variable after 'c' is the number of clauses within it. Every test was run for 3 trials and we used average values for analysis and reliable results.

7 Results

7.1 PBIL Results

	v140-c1200			v8385-c21736			v22136-51563			v44079-c117720		
	16	30	49	16	30	49	16	30	49	16	30	49
10	79.38	79.47	79.88	77.76	77.78	77.95	76.86	76.94	76.98	78.56	78.67	78.49
100	86.55	87.02	87.44	80.15	80.57	80.88	78.46	78.54	78.68	79.60	79.83	80.04
1000	87.72	87.61	87.99	88.03	89.24	89.94	84.13	85.07	85.73	83.63	84.41	84.86
10000	88.08	88.00	88.16	93.99	95.00	95.66	88.72	85.98	91.04	86.94	88.04	88.80

Table 1: All PBIL results over iterations, individuals and files

Table 1 shows our results for PBIL experiments. As can be seen on....., these increases in iterations seem to have made the most difference in the 8385 variable file for all individual numbers but all other file sizes had similar increases in maximum satisfiability over iterations. No clear trends can be drawn regarding the relationship between the file size and number of iterations to find the best satisfiability. This indicates that the number of iterations necessary might not be dependent on file size but the specific assignments required and any type of MAXSAT problem. However, the graphs demonstrate that having numerous iterations is always beneficial to maximize the percentage of clauses satisfied since the best results from PBIL are obtained at iteration 10000.

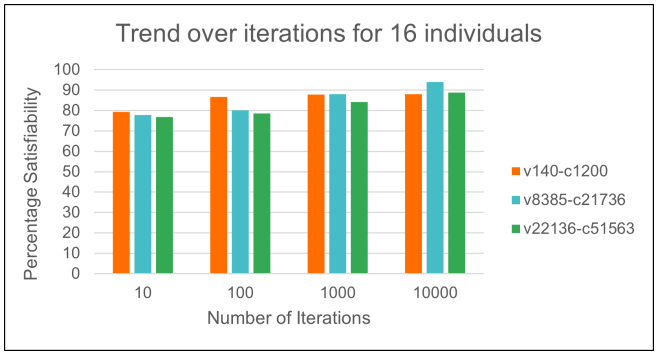

Figure 2: PBIL results for 16 individuals

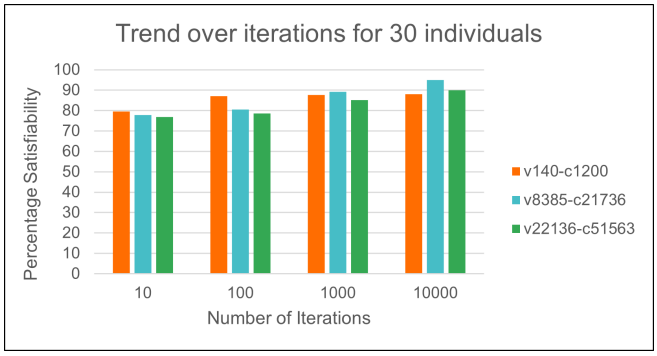

Figure 3: PBIL results for 30 individuals

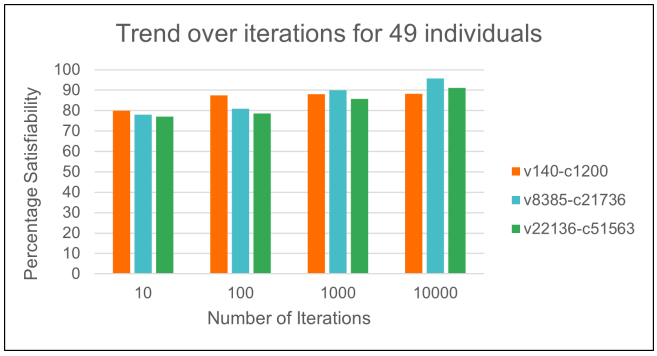

Figure 4: PBIL results for 49 individuals

The satisfiability percentage increases as iterations increase across all files and all numbers of individuals (16, 30, 49). It is clear that with more iterations and more individuals evolves the probability vector to yield higher results regarding maximum satisfiability. For most files, the largest difference between the maximum satisfiability solution from iteration 10 to iteration 10000 was with 49 particles and the smallest difference was with 16 individuals. As can be seen in Figure 4, for the file with 22136 variables, the probability vector evolved so that the maximum satisfiability between iteration 10 and iteration 10000 increased by 14% when there were 49 individuals. With 16 individuals, however, it ultimately increased only by 11% across iterations. This trend was consistent across the 8385 variables file, the increase for 16 individuals was 16% whereas the increase for 49 individuals was 17%. However, the increase in maximum satisfiability in regards to the number of individuals was not the same for the file with 140 variables. For 16 individuals, it increased 8%, and for 30 individuals it increased 8%, and for 49 it increased 8% from iteration 10 to 10000. Although this disrupts the trend, it's clear the range of values regarding the amount of increase depending on number of individuals are minimal, which indicates that the overall increase over iterations not only depends on number of individuals, but also on file size. Although the largest increase in maximum satisfiability across all numbers of individuals was for the 8385 variables file (an increase in iterations yielded the biggest difference), the number of individuals affected the range of increase the most for the largest 22136 variables file. These results indicate that more individuals may be necessary for larger problem sizes whereas they have less of an effect on maximum satisfiability when the MAXSAT problems are small. However, a limitation to this conclusion is that the range of individuals that we used was small. With bigger numbers for individuals, such as a 100 or 1000, the satisfiability trends might have been different over iterations.

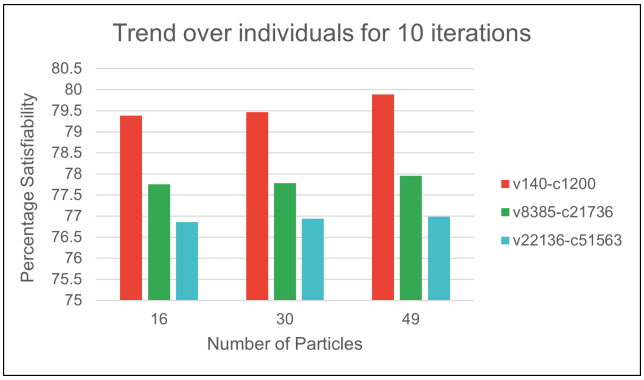

Figure 5: PBIL results for 10 iterations

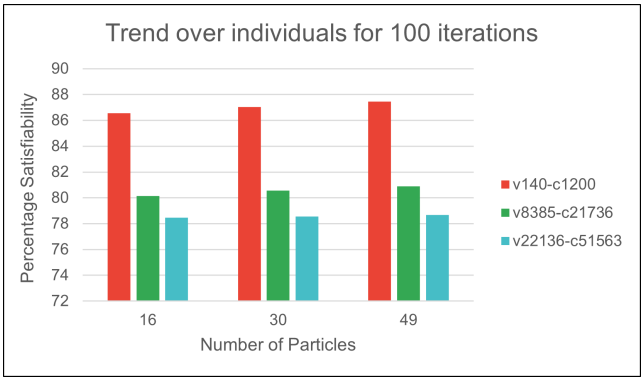

Figure 6: PBIL results for 100 iterations

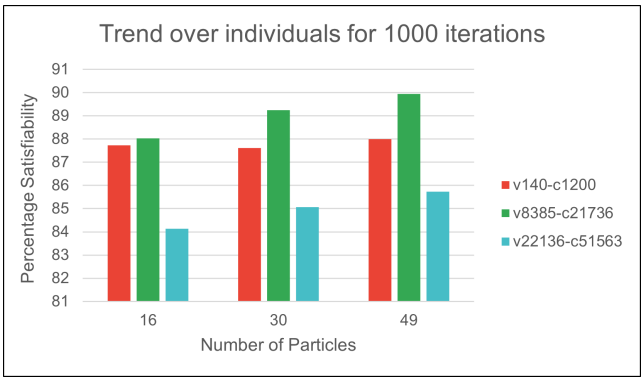

Figure 7: PBIL results for 1000 iterations

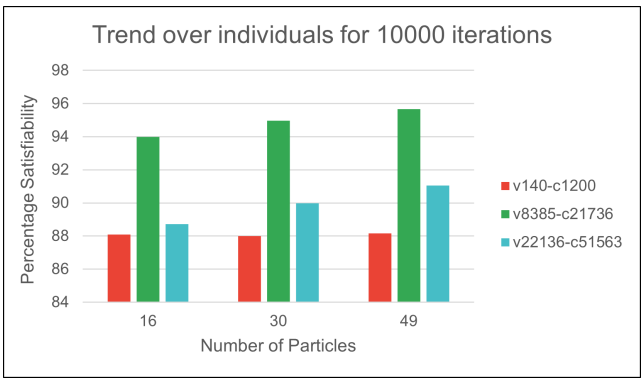

Figure 8: PBIL results for 10000 iterations

Figure 5, 6, 7 and 8 affirm the fact that varying the number of individuals (16, 30, 49) has minimal effect on the percentage satisfiability. Even while examining the differences in satisfiability at a single iteration, changing the number of individuals doesn't have significant effects. In the graph mapping the percentage satisfiability with different numbers of individuals for all the files with 10 iterations, the biggest increase of satisfiability is for the smallest file with 140 variables. However, at iteration 10000, the 8385 variable file and the 22136 variable file have higher increases when the number of individuals increase from 16 to 49 whereas the smallest file of 140 variables is stagnant. This may be indicative of earlier convergence and further demonstrates how overall, the percentage satisfiability is not affected by the range of individual numbers that we tested. The increase in iterations was much more significant on the percentage satisfiability we got. To see more of an effect with the number of individuals, they might need to be in as big of a range as that of our iterations test. However, this would be impossible to test for the PSO algorithm with the current implementation due to its inefficiency.

7.2 PSO Results

For the first section of our PSO results, we are comparing the average values across all topologies to analyze the effects of varying number of particles and number of iterations. Table 2 highlights the results of our PSO algorithm on the v140-c1200 file. Here, we see the percentage satisfiability converge between 78-80%. Table 3 highlights the results of our PSO algorithm on the v8385-c21736 file. Here, we see the percentage satisfiability converge at 77%. Table 4 highlights the results of our PSO algorithm on the v22136-c51563 file where we see the percentage satisfiability converge at 76%.

	Global			Von Neumann			Random			Ring		
	16	30	49	16	30	49	16	30	49	16	30	49
10	78.47	78.61	79.47	78.39	78.33	78.56	78.81	79.28	79.30	78.88	78.22	78.86
100	79.61	79.47	79.66	78.92	79.25	79.78	79.06	79.53	79.83	79.58	79.36	79.64
1000	79.61	79.94	79.94	80.14	79.94	80.56	79.72	80.25	80.19	79.94	80.36	80.25
10000	80.25	80.66	80.81	80.52	80.86	81.28	80.50	80.50	81.08	80.72	80.53	80.77

Table 2: PSO results for file v140-c1200

	Global			Von Neumann			Random			Ring		
	16	30	49	16	30	49	16	30	49	16	30	49
10	77.71	77.94	77.81	77.70	77.76	77.86	77.78	77.85	77.87	77.66	77.91	77.88
100	77.71	77.99	77.81	77.70	77.76	77.86	77.78	77.85	77.87	77.69	77.91	77.88
1000	77.71	77.95	77.81	77.70	77.76	77.86	77.78	77.85	77.87	77.64	77.91	77.88
10000	77.77	77.97	77.84	77.70	77.76	77.86	77.78	77.85	77.87	77.72	77.91	77.88

Table 3: PSO results for file v8385-c21736

	Global			Von Neumann			Random			Ring		
	16	30	49	16	30	49	16	30	49	16	30	49
10	76.72	76.70	76.74	76.59	76.71	76.75	76.64	76.77	76.63	76.67	76.69	76.78
100	76.74	76.77	76.74	76.73	76.77	76.76	76.76	76.77	76.96	76.74	76.77	76.79
1000	76.77	76.88	76.84	76.79	76.82	76.88	76.94	76.79	76.96	76.83	76.77	76.84
10000	76.84	76.90	76.90	76.87	76.90	76.97	76.95	76.85	76.96	76.87	76.91	76.95

Table 4: PSO results for file v22136-c51563

We further dissect these findings along two axes: trend across iterations and trend across particles. First, let's compare the trend across iterations while keeping the number of particles constant.

Figures 9, 10, 11 show the trend across iterations for 16 particles, 30 particles and 49 particles respectively. We see a general trend of increase in percentage satisfiability with the increase in number of iterations. For all three figures, we see that the smallest file v140-c1200 yields the highest satisfiability percentage, followed by v8385-c21736 and then v22136-c51563. Thus, we see an inversely proportional relationship between file size and satisfiability percentage. Now, let's compare the trend across particles while keeping the number of iterations constant.

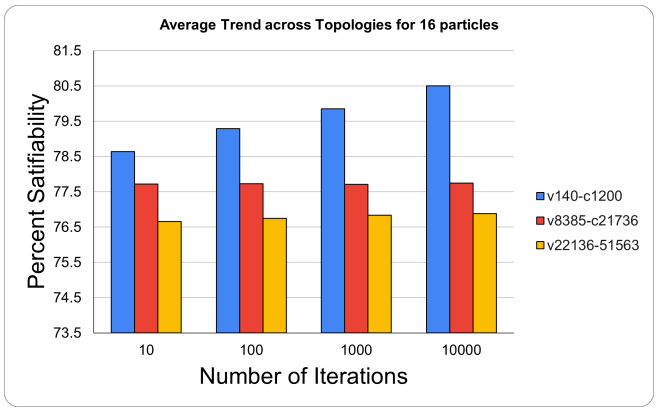

Figure 9: PSO results for 16 individuals

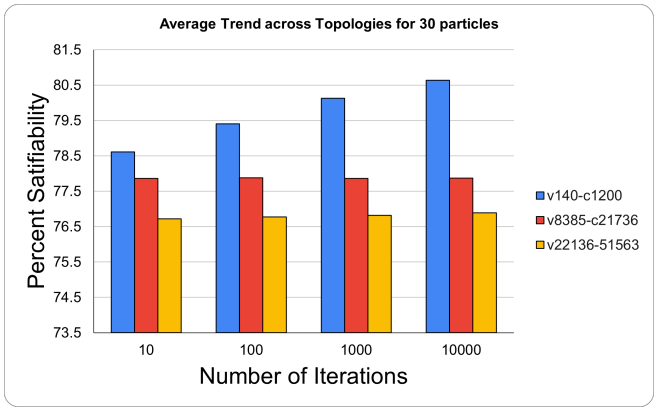

Figure 10: PSO results for 30 individuals

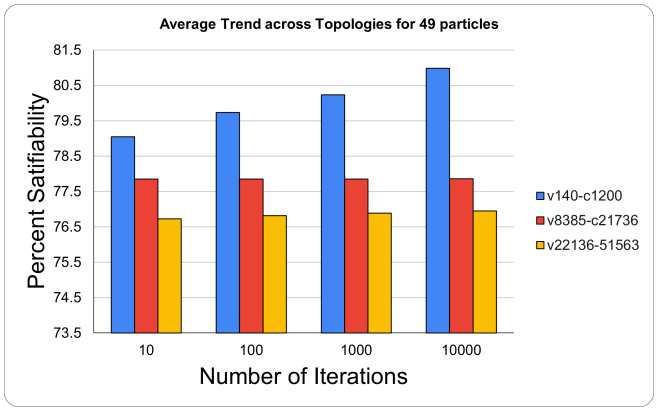

Figure 11: PSO results for 49 individuals

Figures 12, 13, 14, 15 show the trend across iterations for 10 iterations, 100 iterations, 1000 iterations and 10000 iterations respectively. Here, we see the same inversely proportional relationship between file size and satisfiability percentage. In terms of a trend between increase in the number of particles, we see some difference for the smallest file. However, for the other two, we do not see a difference in terms of increase in satisfiability percentage with increase in number of particles.

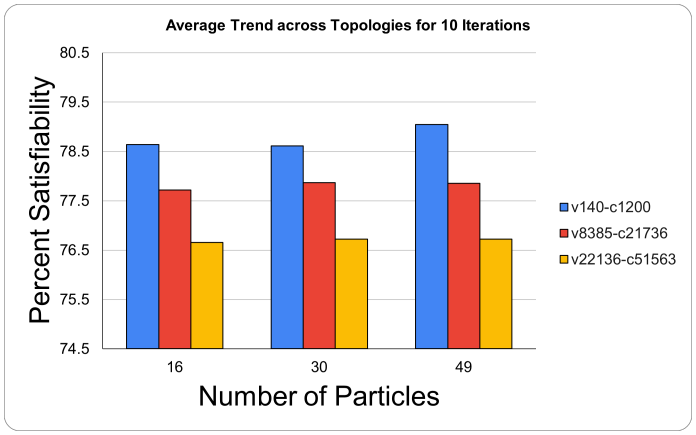

Figure 12: PSO results for 10 iterations

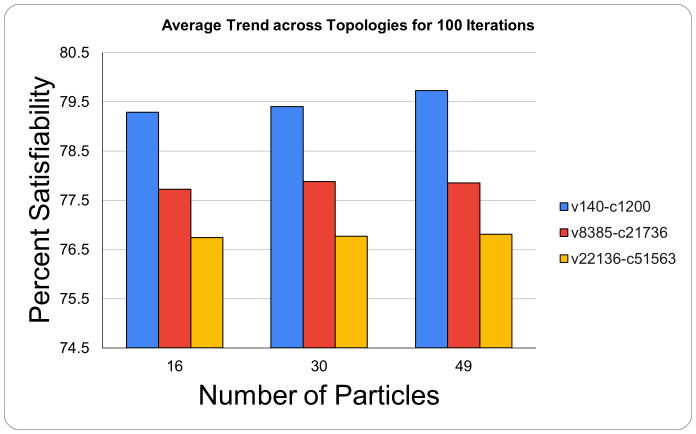

Figure 13: PSO results for 100 iterations

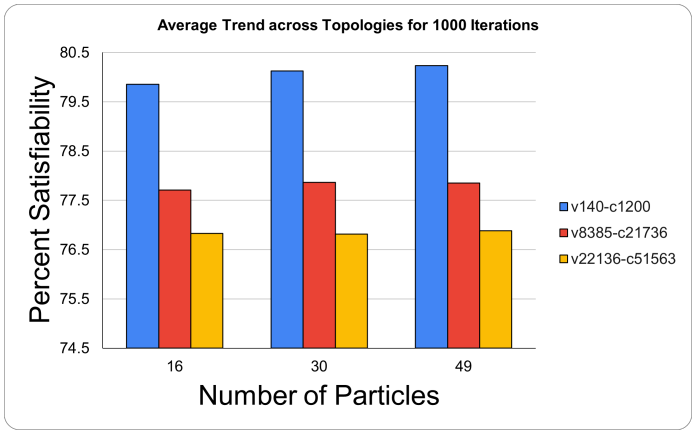

Figure 14: PSO results for 1000 iterations

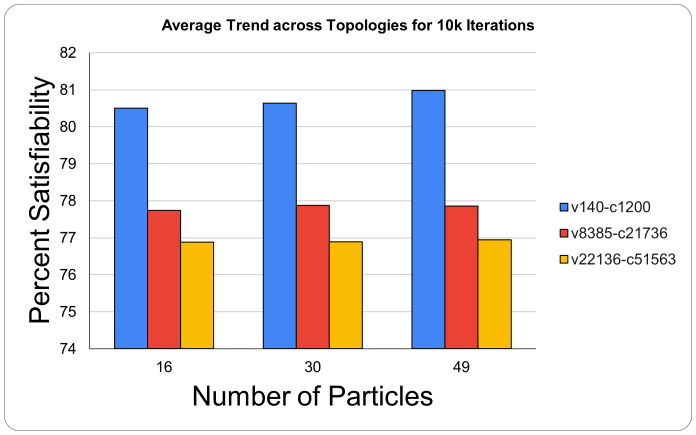

Figure 15: PSO results for 10000 iterations

For the second section of our PSO results, we are analyzing the effects of different neighborhood topologies on the percentage satisfiability. In order to do this, we are keeping the number of particles and number of iterations constant at 49 particles and 10000 iterations respectively. Although there was a slight increase in percentage satisfiability between averages values across all topologies for 16 particles to 49 particles, it was a minimal difference. Therefore, we decided to focus our analysis on only 49 particles to find the best neighborhood topology in hopes that they could be generalized to 16 and 39 particles as well. Furthermore, we found that 10000 iterations produced the best result so we decided to use that to draw comparisons between the topologies. These results are shown in Figure 16.

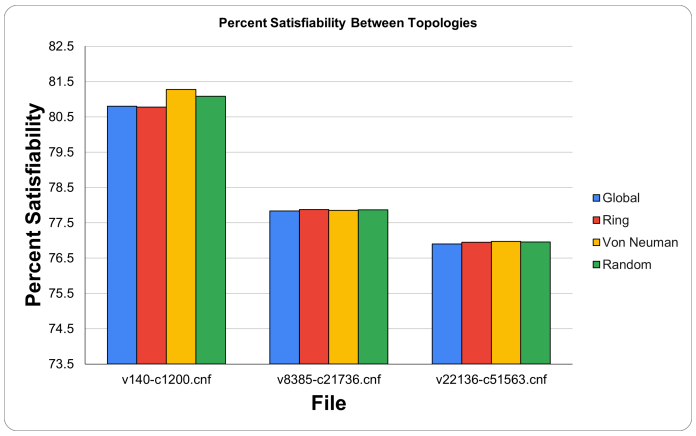

Figure 16: Different topologies at 49 particles and 10000 iterations

As before, we find that the smallest file produced the highest satisfiability percentage and this was true across all topologies. As evidenced by this same figure, for the v140-c1200 file, we find that Von Neumann topology produced the highest satisfiability percentage, followed by random topology. However, these differences are not highly significant as they are all still within the 1% range between 80.5-81.5%. For the other two files, we found that all the topologies produced almost, if not exactly, the same results. This is in line with our previous findings where the smallest file produced the most variance and there were minimal differences for the other files. Thus, we cannot conclusively say that there was one optimal topology which produced the best results as all four topologies produced similar results.

7.3 Comparison of PSO with PBIL

7.3.1 Iterations

Increasing the number of iterations had the most significant difference on maximizing percentage satisfiability for both PSO and PBIL algorithms. Testing with both algorithms demonstrated that it had a more significant difference than the number of individuals, file sizes, or topologies on the solution.

Although both algorithms demonstrated the same trends (with more iterations, better satisfiability found), the scale of increase in percentage satisfiability across iterations was drastically different. PBIL evolved the probability vector much more effectively as our percentage satisfiability increased by a maximum of 17.7% between 10 and 10000 iterations (average of topologies for 49 particles for 8385 variables). However, the increases for PSO were so similar across files, particles, and topologies and the maximum was approximately 2%.

7.3.2 File Size

Although the number of iterations was most influential in evolving the probability vector to yield maximum satisfiability, its influence was affected by the file size. With PBIL, a linear trend couldn't be determined. The highest increase from 10 to 10000 iterations was for the 8385 variable file and the lowest increase in percentage satisfiability was for the 140 variables (smallest) file. With PSO, the file size was inversely proportional to percentage satisfiability. The PSO algorithm evolved the probability vector so that the range from 10 to 10000 iterations was the largest for the smallest file but as the file sizes get larger, the maximum satisfiability would not improve as much over iterations.

An interesting file that we experimented on was the one with 8385 variables. Using PBIL, the solution was a significantly higher satisfiability value than the solutions found for other files. For PSO, this was the file in which none of the iterations changed the maximum satisfiability found; the best satisfiability at 10 iterations was the same as that of 10000 iterations, which indicates premature convergence.

These results may be indicative of how the use of learning rate to evolve the vector in PBIL allowed more exploration of the solution space that eventually led to a better result. PSO, whereas, may have been stuck at a local optimum and was potentially getting worse results over iterations.

7.3.3 Number of Individuals/Particles

For both PSO and PBIL algorithms, the number of individuals or number of particles that we tested didn't significantly affect the solutions. Increasing the number of particles didn't seem to find better satisfiability for PSO but it would take much longer to run. The reason we didn't test with larger individual/particle number ranges was because it would take hours to find a solution to one test file with PSO. Given this information, finding MAXSAT solutions by evolving probability vectors using PSO may not be the best application of the PSO.

7.3.4 Neighborhood Topologies and Learning Rate

When testing various learning rates for PBIL in our previous project, we found that 0.1 yielded the best satisfiability. To compare the PSO and PBIL algorithms, we were interested in finding the neighborhood topology that yielded the best results. However, there was too much variance across iterations, file types, and particle numbers, and no clear trend was found. Therefore, the learning rate was much more effective in evolving the probability vector to the optimal solution than the neighborhood topologies were. We couldn't compare the best result of neighborhood topology with the best learning rate, 0.1, because of this inconclusive result.

That being said, even with the best combination of topology, iteration, and particle number, PBIL still performs better across all file sizes.

8 Further Work

PBIL performed significantly better than PSO with the parameters we tested. However, our choice of parameters for testing was limited. For example, we used the best learning rate found for PBIL from our previous project, 0.1, but varying this learning rate to find something better could have led to an algorithm recommendation that is better for finding the MAXSAT solution.

When writing the PSO algorithm, we varied the initialisation ranges for velocity and noticed minor changes. Further exploring the impact of the range for velocity initialisation may have found more interesting results as it may have affected the extent to which the algorithm was exploiting and exploring the solution space. Since our choices regarding velocity and position initialization were arbitrary, further experiments could include changing the range of velocity or position initialisation to see if it would have an effect on the MAXSAT solution but also how the probability vector would change over iterations. Further work could include initializing velocity values to a negative range, a large range, a small range (between 0 and 1), and a constant of 0.5. Similarly, running additional tests by initializing the probability vector, position, by setting all elements to 0.5 (which is what is done in PBIL) while keeping the velocity range the same as what was used for the other tests (-2, 4) to isolate the variables and determine influence could yield interesting results, especially for the files that prematurely converged.

Overall for the project, we had four files that we began testing with. We began with PBIL testing which took over three hours for 49 particles and 10k iterations. We attempted the same test with PSO; no results appeared after almost 5 hours, so we decided to forgo testing for the file with 44079 variables and 117720 clauses. We couldn't draw comparisons for this file and therefore excluded it from our analysis. However, further work could include testing these algorithms on a larger number of files and with numerous files of similar sizes to get more reliable results regarding how file size affects the efficacy of the algorithms.

With increased iterations, the satisfiability increased for both PSO and PBIL. We stopped iterations at 10000, but if it was increased, the probability vector may have continued evolving to find a better solution. Investigating the point at which there is convergence could be valuable as it would demonstrate how effective each algorithm is at finding the optimum result with the least work done. However, with the current algorithms and their implementations, these tests may take an extreme amount of time and an alternative to finding a solution to the MAXSAT problem may be best.

Lastly, our experiments would benefit from comparisons of an additional metric - execution time. As we ran the experiments, we noticed that PBIL was solving the MAXSAT problems faster than PSO, but this observation would be further strengthened if we recorded execution time for each test. It would allow us to determine more specific trends and the recommendation would have focused on both execution time and percent satisfiability, which is important since we want efficiency and an optimal solution.

9 Conclusion

This project aimed to compare results of applying PSO to evolve a probability vector and finding a solution for MAXSAT with the results from PBIL when applied to the MAXSAT problem. Across all files, PBIL performed better than our PSO implementation regardless of the other parameters we changed (iteration, topology, particle numbers). Compared to our PSO, the solution found by PBIL improved more as we increased the number of iterations. We found increasing particles had very little effect on maximum satisfiability found but this may have been due to the range of values we tested. PSO also had significantly longer execution time than PBIL. PBIL performed better on important metrics such as difficulty of implementation, execution time, size of problems it can solve, and ease of use. On this basis, we recommend the use of PBIL for solving MAXSAT problems.