Evolutionary SAT Solver (ESS)

Oscar Pérez Cruz
Polytechnic University of Puerto Rico, Hato Rey, Puerto Rico, perezcruz.oscar@gmail.com

Dr. Alfredo Cruz
Polytechnic University of Puerto Rico, Hato Rey, Puerto Rico, alfredcross@gmail.com

ABSTRACT

An NP problem is a class of problem whose solution can be found if possible in non-polynomial time with a non-deterministic algorithm. The Boolean Satisfiability Problem (SAT) is a well known NP-complete decision problem that consists in deciding whether the variables of a propositional logic formula can be given a value that satisfies the formula. This research will focus on digital testing by using this problem to find the growth faults within a programmable logic array (PLA) by analyzing its Boolean equation in conjunction normal form (CNF). The first stage of this project will be the development of a SAT solver which will incorporate genetic algorithms. Several tests are performed with different values for each parameter of the genetic algorithm to determine the best way to optimize the process of finding all the possible values that satisfy a PLA Boolean equation. Also, from the SAT library, several benchmarks will be used to determine which parameters optimize the process of finding a solution to the problem. These tests will be evaluated by how many solutions are found and the amount of time it takes to find a solution to the problem within a convergence and generation limit.

Keywords: SAT, Fitness, Selection, Crossover, Mutation

1. INTRODUCTION

Since the approximation to the new millennium, technology started to become the focus of many people who wanted to create something innovative before someone else did. During those years, new technologies were developed that now allow us to perform more complex computations that took too long to do because of the processing power at the time. One such complex computations method that has been a focus of research in recent years is evolutionary computation. Evolutionary computation uses theories from people like Charles Darwin to create algorithms that modify its function and information the way biological life forms evolve and adapt to their needs in order to survive. Some of these algorithms are called Evolutionary Programming, Genetic Programming, Evolutionary Strategies, and Genetic Algorithm (Tomassini 1995). This paper is an overview on how genetic algorithm works with binary encoding by solving the Boolean Satisfiability Problem with an equation in CNF. This is the first stage of this project, were the different parameters of genetic algorithms will be tested to identify which parameters best optimizes the process of finding the combination of values that satisfy the equation being tested.

2. THE BOOLEAN SATISFIABILITY PROBLEM

The Boolean Satisfiability Problem, also known as SAT, is a well known NP-complete decision problem that consists in deciding whether the variables of a propositional logic formula can be given a value that satisfies the formula. This formula can be represented in CNF, which is a conjunction of x clauses that contains a disjunction of y literals. A literal is the same as a variable in a Boolean equation, where its value can be either 0 or 1. A clause is a group of literals joined together by one of two propositional operators (Marques-Silva 2008). A conjunction use the AND operator ($\wedge$/*) where only if both operands are 1 does it return a TRUE (1). A disjunction uses the OR operator ($\vee$/+) where if either of the operands is a 1 does it return a TRUE. There is a third propositional operator known as the NOT ($\neg$/') which is combined with a literal to give us the complement of a literal. For example, if A is 0; its complement (A') would be 1. This problem has been widely used in many applications like model-checking of finite-state systems, design debugging, AI planning, software testing, identification of functional dependencies in Boolean functions, etc (Marques-Silva 2008). It has also been used as an exemplary problem for testing genetic algorithms because of the large search space that it can provide depending on the equation in question. This search space come from all of the possible solutions that can be found which come from the equation of two to the power of the number of literals in the equation. The following equation, $F(a,b,c,d,e,f) = (\neg a \vee c) \wedge (\neg a \vee c \vee \neg e) \wedge (\neg b \vee c \vee d \vee \neg e) \wedge (a \vee \neg b \vee c) \wedge (\neg e \vee f)$ will be use as an example to show the different techniques performed by a genetic algorithm (Luger and Stubblefield 1997). This equation contains a total of 64 combinations of values, 2^6, which are the combination of 0's and 1's in all its literals, but only 30 of these combinations satisfy the equation, returns a 1 also known as TRUE.

3. GENETIC ALGORITHMS

Genetic algorithm is a search algorithm, created by John Holland in 1975, which mimics the evolution theories created by Charles Darwin (Whitley 1994). Darwin's theory of evolution states that all life is related and come from a common ancestor. As time passes, individuals from each generation begin to mutate and become something different from their ancestors. Another part of Darwin's theory of evolution is the natural selection which states that the best genetic parts of an individual survive to aid in the survival of the race. These theories come true when the aspect of time is looked at through the different generations that appear over time. In genetic algorithm, these theories are implemented as operations or stages. There are a total of six stages in genetic algorithm: Population Generation, Fitness Calculation, Parent Selection, Mating, Mutation and Integration. The integration phase is where the offspring's created during the Mating Phase, also known as the Crossover Phase; join the new population in the next generation. After the offspring's have filled the population in the next generation, the algorithm takes the new population and continues from phase 2, Fitness Calculation, of the algorithm until either a convergence occurs or a solution to the problem is found. Before running the genetic algorithm, there are several parameters that must be chosen to determine the operation of the different phases of the algorithm. The most important parameters are: population size, crossover and mutation rate, convergence limit, and the selection, crossover and mutation technique to be used. Figure 1 is an example of how genetic algorithms evolve the information contained within an individual inside a population to find a solution to its problem.

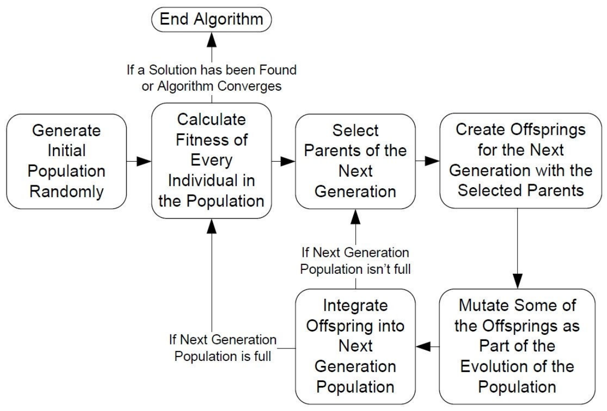

End Algorithm
If a Solution has been Found or Algorithm Converges
Generate Initial Population Randomly
Calculate Fitness of Every Individual in the Population
Select Parents of the Next Generation
Create Offsprings for the Next Generation with the Selected Parents
If Next Generation Population isn't full
Integrate Offspring into Next Generation Population
Mutate Some of the Offsprings as Part of the Evolution of the Population
If Next Generation Population is full

Figure 1: Genetic Algorithm Sample

3.1 Population and Fitness Calculation

Each population has an amount of individuals which contain an amount of genes, fitness and a rank. Gene is a term used as a part of the information contained within an individual. Another term used in genetic algorithms is chromosome which represents the individual. There are several types of encoding that the information within each individual is based on. In binary encoding, the genes of each individual are a string of bits. In permutation encoding, the genes of each individual are a string of numbers in a sequence. In value encoding, the genes of each individual are a string of values that could be specific to a problem. In tree encoding, the genes of each individual are a tree of objects like functions or commands (Obitko 1998). In order to solve the Boolean Satisfiability Problem, binary encoding is used since the literals of a Boolean equation, which also represent the genes of an individual, can only be a 0's and 1's. The fitness of an individual is used to measure how close the information contained within an individual is to the solution of the problem. The rank of the individual is used to classify the fitness of the individual in the population.

To start the algorithm, a set of individuals with their genes is generated randomly. After the first population is generated, the fitness of each individual is calculated by solving each clause in the Boolean equation. Since the equation will be in CNF; if one of the genes in the individual satisfies a part of the clause, the individual fitness increases. This process is repeated with all the clauses of the equation with each individual in the population. If the information contained in the individual satisfies all the clauses of the equation, a solution to the problem has been found. For this type of problem, it is better to have a higher fitness than a lower one, because it means that the individual has solved more clauses and is closer to finding a combination of values that satisfy the equation. A higher fitness gives an individual a much better probability of being selected as a parent of an individual in the next generation; but because in other types of problems it is better to have a lower fitness, a rank is needed to consider the class of fitness that each individual has in the population. A higher rank gives an individual a much better probability of being selected as a parent. Table 1 shows an example of a population with each individual's fitness, rank, and their probability of being selected as parents with different techniques. Also, the individuals 1, 3, 6 and 8 are a solution to the problem by having a fitness of 5 because their information contains a combination of values that satisfy the previous equation stated before. In this population, there are only three different fitness (5, 4, and 3); meaning that their will be only three different ranks in this population. Individuals with fitness 5 have a rank of 3, while those with fitness 4 have a rank of 2 and the one with fitness 3 has a rank of 1.

Table 1: Example of a Population

ID	Genes	Fitness	Rank	Probability of Selection (Roulette/Rank/Elite)
1	000001	5	3	14.3% / 15.8% / 100%
2	000100	4	2	11.4% / 10.5% / 0%
3	011001	5	3	14.3% / 15.8% / 100%
4	111010	4	2	11.4% / 10.5% / 0%
5	000111	4	2	11.4% / 10.5% / 0%
6	110011	5	3	14.3% / 15.8% / 100%
7	101011	3	1	8.6% / 5.3% / 0%
8	001011	5	3	14.3% / 15.8% / 100%

3.2 Selection

Each individual in a population has a position in their society. Normally the individuals that fit a profile would be in a very high position while others in a lower position. As already stated, in genetic algorithms, the position of an individual is determined by their fitness and rank which gives them their probability to become parents of the next generation. Each individual can be selected to be a parent of more than one individual.

3.2.1 Roulette Wheel Selection

The roulette wheel selection technique is a fitness proportionate technique where the fitness of all individuals in the population is added. To determine the probability of selection that each individual has, the fitness of each individual is divided by the total fitness of the population. To select the individual, a random number is generated from 0 to the fitness of the population. Then each individual is considered by giving them a range determined by their fitness. This range starts from the previous individual's range limit and ends with the current individual fitness plus the previous individual's range limit. As an example using the individuals in Table 1, the range of the first individual would extend from 0 to 5, while the range of the second individual would extend from 5 to 9, and so on. If the generated random number is 7; then the selected individual to become a parent of the next generation will be the second individual. Figure 2 is an example of the probability that each individual has of being selected using this technique with the individuals from Table 1.

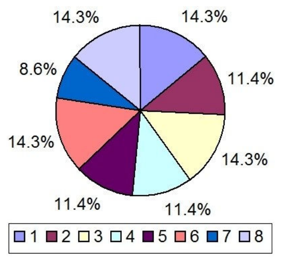


Figure 2: Probability of Selection of Every Individual in Table 1 using Roulette Wheel Selection

3.2.2 Rank Selection

The rank selection uses the same process as the roulette wheel selection. The only difference is that it considers the rank and not the fitness. Figure 3 is an example of the probability that each individual has of being selected using this technique with the individuals of Table 1.

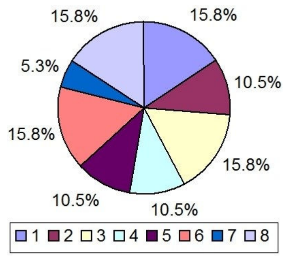


Figure 3: Probability of Selection from Each Individual in Table 1 Using Rank Selection

3.3 Mating: Cloning or Crossover

For a race to survive and prosper, the genes of the fittest individuals must continue on to the next generation. This is done by mating and creating offsprings that will carry the genes of the current generation to the next. To bring this belief in genetic algorithms, the information from the previous selected individuals is used to produce two new individuals, known as offspring's through different techniques called cloning and crossover. Cloning creates two offspring's with the exact information as their parents while crossover combines the information. To determine how the offspring will be created, a random number is generated and compared with the crossover rate. If the number is higher than the crossover rate, the cloning technique is used; if not one of the following crossover techniques is used. Each technique will be explained assuming that the selected parents are individuals 3 and 6 from Table 1.

3.3.1 Single-Point Crossover

A random number needs to be generated to determine the point where the information of each parent is divided between the offspring's. Figure 4 shows how the random number divides the information between each parent in two parts to create the offspring's. Each offspring will have the first part of one parent, while the second part is interchanged between them.

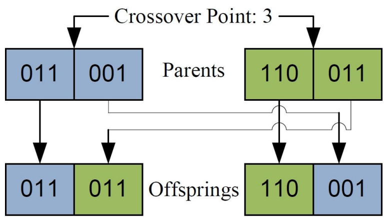


Figure 4: Single-Point Crossover Example

3.3.2 Two-Point Crossover

Two random numbers need to be generated if this technique is used, to determine the point where the information of each parent is divided between the offsprings. Figure 5 shows how the random numbers divide the information between each parent in three parts to create the offspring's. The first and last part of each parent goes to one offspring, while the middle part is exchanged between them.

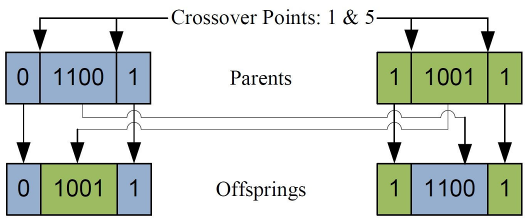


Figure 5: Two-Point Crossover Example

3.3.3 Uniform Crossover

An equal amount of random numbers to the amount of genes of each individual are generated to determine which information will be exchanged between the parents in order to create the offsprings. Figure 6 contains a random template of the genes that show which genes are exchanged between the parents. The position in the template where there is a 1, mean that the genes of the parents are exchanged.

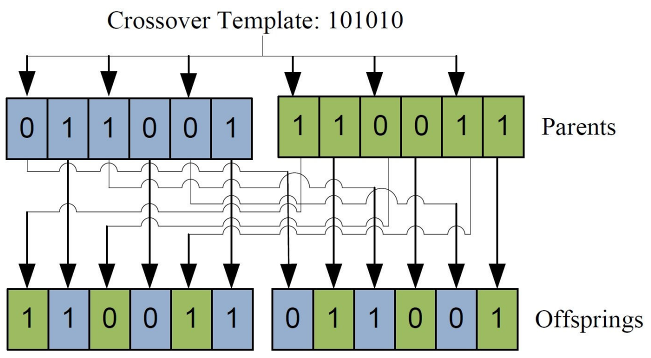


Figure 6: Uniform Crossover Example

3.4 Mutation

As the algorithm gets closer to finding one or more solutions, the information contained within many of the individuals become almost identical possibly causing a convergence in the algorithm. The convergence of the algorithm is considered when the average fitness of the population, population fitness divided by the amount of individuals, is repeated through an amount of generations. Sometimes depending on the equation or the problem, no solution is found because of this. A solution to this problem is the Mutation Phase. Through each generation in every race, a small mutation takes place. This keeps the individuals of each generation different from the others. The same theory is applied in genetic algorithms. After the Mating Phase, a mutation is considered for each offspring. This is done by generating a random number in which if the number generated is equal or less than the mutation rate, the following mutation technique is performed.

3.4.1 Bit Inversion

Since the information contained within each individual in this type of encoding is a collection of 1's and 0's, all that has to be done is invert the bit of a specific set of genes. Each gene will have a 50% chance of actually being mutated. Figure 7 contains a random template which indicates which genes will be mutated. The position in the template where there is a 1, mean that the gene value will be inverted.

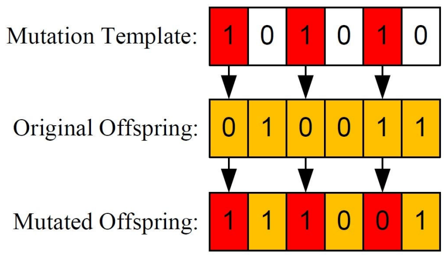


Figure 7: Bit Inversion Example

3.4.2 Forced Bit Mutation

Even with the bit inversion, there could still be a problem with the convergence of the algorithm since many applications that use genetic algorithm choose to have a very low mutation rate, like 1% or 0.1%. Because of this, many offspring may not go through a mutation phase causing the algorithm to converge really fast. For this reason, a solution for this type of encoding is to do a force mutation using bit inversion every n genes that has been passed on to the next generation regardless of the mutation rate. This is a special type of mutation which will occur to any gene of any offspring that will be integrated into the next population. This type of mutation could work very well for those algorithms that use a very low mutation rate and whose algorithm converges really fast without having found a solution; since a change in a specific bit could change the entire fitness of the individual, also changing the average fitness in the population.

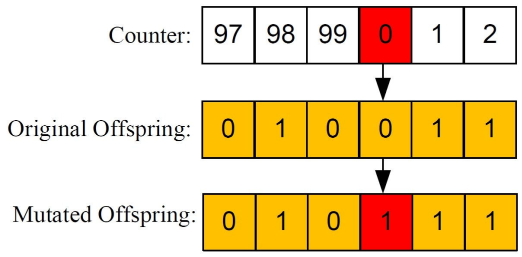


Figure 8: Force Bit Mutation Example

4. EXPERIMENT

The parameters of the genetic algorithm that are being tested are the population size, the crossover and mutation rates, and the selection and crossover techniques. The experiment will run with a generation limit of 50,000 generations and a convergence limit of 100. To avoid a fast convergence, the force bit mutation will be forced in each test every 1000 bits that is pass to a new generation. This technique may increase as it may decrease the chance of finding combination of values that satisfy the equation depending on the amount of literals of the equation and the population size. At the same time, it will avoid a fast convergence which will help during the testing of a PLA equation which contains many combinations of values that satisfy the equation. To identify which parameters best optimizes the process of finding a combination of values that satisfy the problem, two types of testing are performed.

The first type of testing will be with a PLA equation where all the combinations of values that satisfy the equation will be searched. The amount of combinations of values that satisfy the equation that are found and the amount of time it takes to find the last combination will be considered when choosing the best choice of parameters. For this type of testing, a PLA equations with 12 literals and 20 clauses will be used (Cruz 2002).

The second type of testing will be with benchmarks taken from the SAT library to test the algorithm as a SAT solver and which parameters of the genetic algorithm best optimizes the process of finding a combination of values that satisfy the equation. The amount of time and the amount of generations will be considered when choosing the best choice of parameters. For this type of testing, three benchmarks containing 20 literals and 91 clauses will be used.

5. PRELIMINARY RESULTS

After performing the testing with a population of 16 individuals, several conclusions can be gathered of the differences that some parameters of the genetic algorithm offer. The following parameters can be compared with the current gathered data.

5.1 Roulette Wheel Selection vs. Rank Selection

Rank selection offers an advantage over roulette wheel selection as it takes each fitness and ranks their value to give those individuals with a higher fitness a higher chance of being selected as parents. This is taken into consideration at later generations when the fitness of each individual is very similar. As an example, Figure 9 shows the difference of using Roulette Wheel Selection and Rank Selection with a population of eight individuals with the following fitness: 50, 51, 52, 53, 54, 55, 56, and 57. As it can be seen, the probability of an individual being selected using roulette wheel selection is almost the same for all individuals in the population; while using Rank Selection there is a higher chance to select the best individuals of the population to be parents of the next generation.

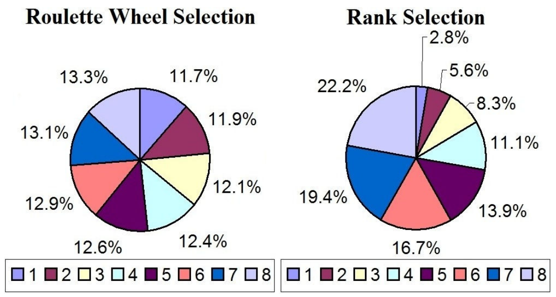


Figure 9: Difference between Roulette Wheel and Rank Selection using an advanced Boolean Equation

Even though rank selection offers an advantage over roulette wheel selection; for the PLA equation, the roulette wheel selection is much better than the rank selection. It can be seen in Figure 10, assuming that the total amount of combinations of values that satisfy the equation is 1257; most of the simulations running with the roulette wheel selection find all the combinations of values and the last combination is found much faster than those using rank selection. This is because roulette wheel selection considers all individuals in the population equally depending on the fitness. This is what is being searched for when looking for all the combinations of values that satisfy the PLA equation. But for the SAT benchmark testing, where only one combination of values that satisfy the equation is searched, rank selection is better since it offers the best individuals as parents for the next generation. As it can be seen in Figure 11, the time it takes to find a combination of values that satisfy the equation is much faster using the roulette wheel selection.

5.2 Single Point Crossover vs. Two-Point Crossover vs. Uniform Crossover

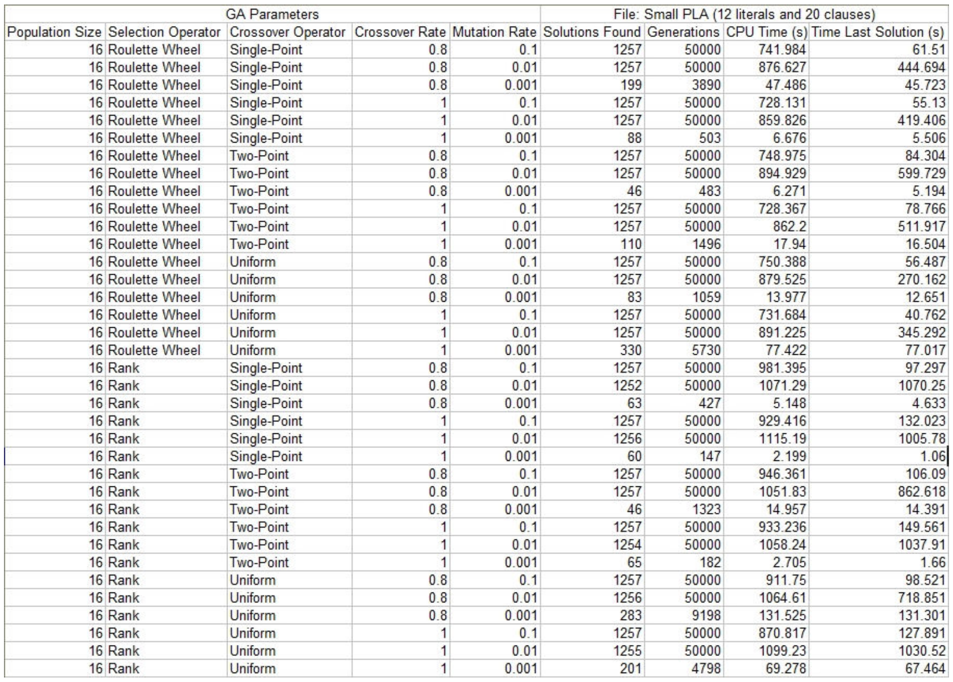
As it can be analyzed in Figure 10, by comparing the times the last combination of values was found in each simulation, the uniform crossover gives a much better performance than the other two techniques because the uniform crossover has the possibility of making many more combination of values than the other two techniques, making a lot of difference in the later generations of the simulation. However, for the SAT benchmarks, as it can be seen in Figure 11, more testing needs to be done before considering which crossover technique might be better because most of the results of the simulations are very similar for all techniques.

5.3 80% vs. 100% Crossover Rate

The main difference between having an 80% and a 100% crossover rate is that with 80% there is a small chance that the offspring have the same data as the selected parents. As it can be seen in Figure 10, it doesn’t affect much since the time it takes to finish the test and the time it takes to find the last combination of values that satisfy the equation is very close. However, for the SAT benchmark, it is better to have 100% crossover rate since it gives the new individuals of the new generation a new set of values that differentiate from the previous generation. This may bring the algorithm closer to a combination of values that satisfy the equation. This is seen in Figure 11 as many of the satisfied simulations have a 100% crossover rate.

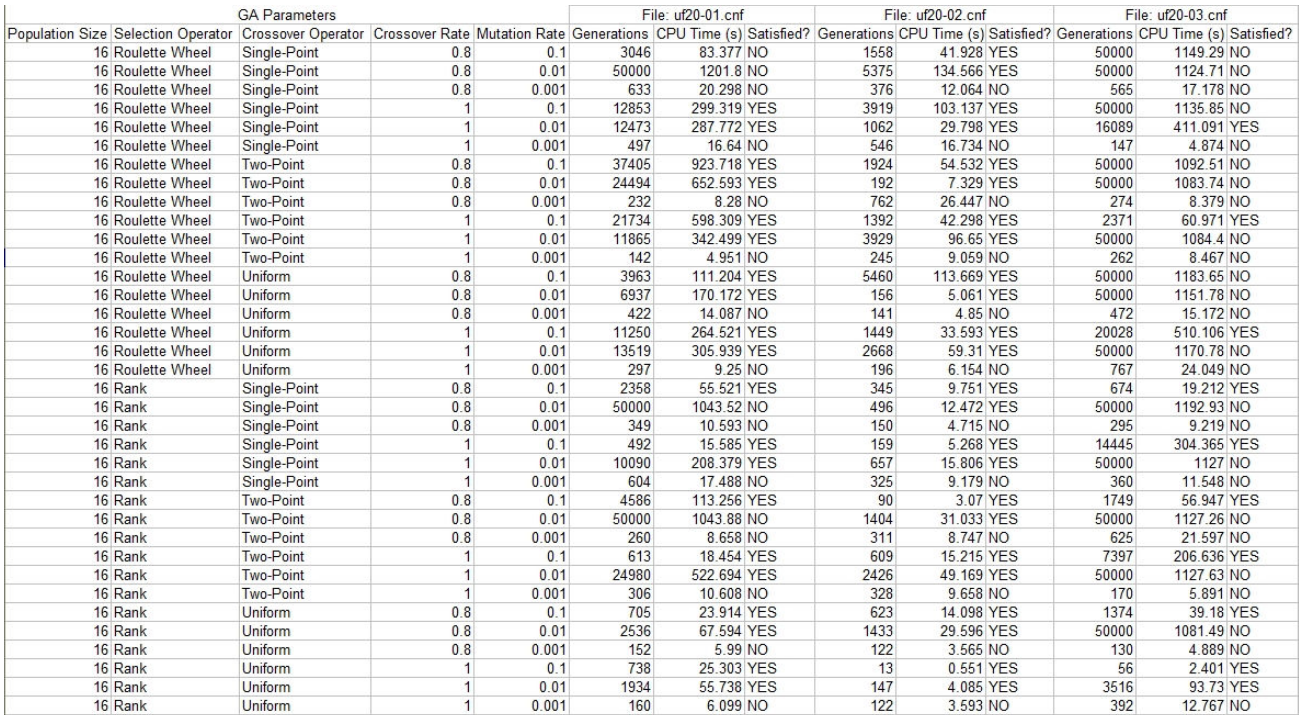
5.4 10% vs. 1% vs. 0.1% Mutation Rate

Utilizing the results of the simulation, it can be clearly seen that for both types of testing using a mutation rate of 0.1% gives neither all nor most of the combinations of values that satisfy the PLA equation and converges too fast using a benchmark from the SAT library. However, between a mutation rate of 10% and 1%, it is better to have 10% for both types of testing since the mutation takes the offspring after the crossover and gives them new values that each parent didn’t have. This is a risk because a value may be changed; decreasing the fitness of the individual or it may increase it to the point of having the combination of values that satisfy the equation. As it can be seen in Figure 10 and 11, the best times of either finding the last combination of values that satisfy the PLA equation or the combination of values that satisfy the SAT benchmark are in the simulations with a 10% mutation rate.



GA Parameters					File: Small PLA (12 literals and 20 clauses)			
Population Size	Selection Operator	Crossover Operator	Crossover Rate	Mutation Rate	Solutions Found	Generations	CPU Time (s)	Time Last Solution (s)
16	Roulette Wheel	Single-Point	0.8	0.1	1257	50000	741.984	61.51
16	Roulette Wheel	Single-Point	0.8	0.01	1257	50000	876.627	444.694
16	Roulette Wheel	Single-Point	0.8	0.001	199	3890	47.486	45.723
16	Roulette Wheel	Single-Point	1	0.1	1257	50000	728.131	55.13
16	Roulette Wheel	Single-Point	1	0.01	1257	50000	859.826	419.406
16	Roulette Wheel	Single-Point	1	0.001	88	503	6.676	5.506
16	Roulette Wheel	Two-Point	0.8	0.1	1257	50000	748.975	84.304
16	Roulette Wheel	Two-Point	0.8	0.01	1257	50000	894.929	599.729
16	Roulette Wheel	Two-Point	0.8	0.001	46	483	6.271	5.194
16	Roulette Wheel	Two-Point	1	0.1	1257	50000	728.367	78.766
16	Roulette Wheel	Two-Point	1	0.01	1257	50000	862.2	511.917
16	Roulette Wheel	Two-Point	1	0.001	110	1496	17.94	16.504
16	Roulette Wheel	Uniform	0.8	0.1	1257	50000	750.388	56.487
16	Roulette Wheel	Uniform	0.8	0.01	1257	50000	879.525	270.162
16	Roulette Wheel	Uniform	0.8	0.001	83	1059	13.977	12.651
16	Roulette Wheel	Uniform	1	0.1	1257	50000	731.684	40.762
16	Roulette Wheel	Uniform	1	0.01	1257	50000	891.225	345.292
16	Roulette Wheel	Uniform	1	0.001	330	5730	77.422	77.017
16	Rank	Single-Point	0.8	0.1	1257	50000	981.395	97.297
16	Rank	Single-Point	0.8	0.01	1252	50000	1071.29	1070.25
16	Rank	Single-Point	0.8	0.001	63	427	5.148	4.633
16	Rank	Single-Point	1	0.1	1257	50000	929.416	132.023
16	Rank	Single-Point	1	0.01	1256	50000	1115.19	1005.78
16	Rank	Single-Point	1	0.001	60	147	2.199	1.06
16	Rank	Two-Point	0.8	0.1	1257	50000	946.361	106.09
16	Rank	Two-Point	0.8	0.01	1257	50000	1051.83	862.618
16	Rank	Two-Point	0.8	0.001	46	1323	14.957	14.391
16	Rank	Two-Point	1	0.1	1257	50000	933.236	149.561
16	Rank	Two-Point	1	0.01	1254	50000	1058.24	1037.91
16	Rank	Two-Point	1	0.001	65	182	2.705	1.66
16	Rank	Uniform	0.8	0.1	1257	50000	911.75	98.521
16	Rank	Uniform	0.8	0.01	1256	50000	1064.61	718.851
16	Rank	Uniform	0.8	0.001	283	9198	131.525	131.301
16	Rank	Uniform	1	0.1	1257	50000	870.817	127.891
16	Rank	Uniform	1	0.01	1255	50000	1099.23	1030.52
16	Rank	Uniform	1	0.001	201	4798	69.278	67.464

Figure 10: PLA Simulation Table



GA Parameters					File: uf20-01.cnf			File: uf20-02.cnf			File: uf20-03.cnf		
Population Size	Selection Operator	Crossover Operator	Crossover Rate	Mutation Rate	Generations	CPU Time (s)	Satisfied?	Generations	CPU Time (s)	Satisfied?	Generations	CPU Time (s)	Satisfied?
16	Roulette Wheel	Single-Point	0.8	0.1	3046	83.377	NO	1558	41.928	YES	50000	1149.29	NO
16	Roulette Wheel	Single-Point	0.8	0.01	50000	1201.8	NO	5375	134.566	YES	50000	1124.71	NO
16	Roulette Wheel	Single-Point	0.8	0.001	633	20.298	NO	376	12.064	NO	565	17.178	NO
16	Roulette Wheel	Single-Point	1	0.1	12853	299.319	YES	3919	103.137	YES	50000	1135.85	NO
16	Roulette Wheel	Single-Point	1	0.01	12473	287.772	YES	1062	29.798	YES	16089	411.091	YES
16	Roulette Wheel	Single-Point	1	0.001	497	16.64	NO	546	16.734	NO	147	4.874	NO
16	Roulette Wheel	Two-Point	0.8	0.1	37405	923.718	YES	1924	54.532	YES	50000	1092.51	NO
16	Roulette Wheel	Two-Point	0.8	0.01	24494	652.593	YES	192	7.329	YES	50000	1083.74	NO
16	Roulette Wheel	Two-Point	0.8	0.001	232	8.28	NO	762	26.447	NO	274	8.379	NO
16	Roulette Wheel	Two-Point	1	0.1	21734	598.309	YES	1392	42.298	YES	2371	60.971	YES
16	Roulette Wheel	Two-Point	1	0.01	11865	342.499	YES	3929	96.65	YES	50000	1084.4	NO
16	Roulette Wheel	Two-Point	1	0.001	142	4.951	NO	245	9.059	NO	262	8.467	NO
16	Roulette Wheel	Uniform	0.8	0.1	3963	111.204	YES	5460	113.669	YES	50000	1183.65	NO
16	Roulette Wheel	Uniform	0.8	0.01	6937	170.172	YES	156	5.061	YES	50000	1151.78	NO
16	Roulette Wheel	Uniform	0.8	0.001	422	14.087	NO	141	4.85	NO	472	15.172	NO
16	Roulette Wheel	Uniform	1	0.1	11250	264.521	YES	1449	33.593	YES	20028	510.106	YES
16	Roulette Wheel	Uniform	1	0.01	13519	305.939	YES	2668	59.31	YES	50000	1170.78	NO
16	Roulette Wheel	Uniform	1	0.001	297	9.25	NO	196	6.154	NO	767	24.049	NO
16	Rank	Single-Point	0.8	0.1	2358	55.521	YES	345	9.751	YES	674	19.212	YES
16	Rank	Single-Point	0.8	0.01	50000	1043.52	NO	496	12.472	YES	50000	1192.93	NO
16	Rank	Single-Point	0.8	0.001	349	10.593	NO	150	4.715	NO	295	9.219	NO
16	Rank	Single-Point	1	0.1	492	15.585	YES	159	5.268	YES	14445	304.365	YES
16	Rank	Single-Point	1	0.01	10090	208.379	YES	657	15.806	YES	50000	1127	NO
16	Rank	Single-Point	1	0.001	604	17.488	NO	325	9.179	NO	360	11.548	NO
16	Rank	Two-Point	0.8	0.1	4586	113.256	YES	90	3.07	YES	1749	56.947	YES
16	Rank	Two-Point	0.8	0.01	50000	1043.88	NO	1404	31.033	YES	50000	1127.26	NO
16	Rank	Two-Point	0.8	0.001	260	8.658	NO	311	8.747	NO	625	21.597	NO
16	Rank	Two-Point	1	0.1	613	18.454	YES	609	15.215	YES	7397	206.636	YES
16	Rank	Two-Point	1	0.01	24980	522.694	YES	2426	49.169	YES	50000	1127.63	NO
16	Rank	Two-Point	1	0.001	306	10.608	NO	328	9.658	NO	170	5.891	NO
16	Rank	Uniform	0.8	0.1	705	23.914	YES	623	14.098	YES	1374	39.18	YES
16	Rank	Uniform	0.8	0.01	2536	67.594	YES	1433	29.596	YES	50000	1081.49	NO
16	Rank	Uniform	0.8	0.001	152	5.99	NO	122	3.565	NO	130	4.889	NO
16	Rank	Uniform	1	0.1	738	25.303	YES	13	0.551	YES	56	2.401	YES
16	Rank	Uniform	1	0.01	1934	55.738	YES	147	4.085	YES	3516	93.73	YES
16	Rank	Uniform	1	0.001	160	6.099	NO	122	3.593	NO	392	12.767	NO

Figure 11: SAT Simulation Table

6. CONCLUSION AND FUTURE WORK

For now, preliminary results lead me to believe that using the following parameters of the genetic algorithm with a PLA equation give me a better chance of finding most, if not all, the combinations of values that satisfy the equation: roulette wheel selection, uniform crossover, either 80% or 100% crossover rate, and 10% mutation rate. Also, with the SAT library benchmarks, preliminary results lead me to believe that using the following parameters of the genetic algorithm give me a better chance of finding a combination of values that satisfy the equation: rank selection, 100% crossover rate, and 10% mutation rate. There is still more testing being done with the different population sizes and other techniques for each of the operators of the genetic algorithm. More results are being gathered as time passes, and a better idea of which parameters best optimize the performance of the algorithm can be determined in the future.

The future of this project will include the implementation of analyzing the combinations of values that where found that satisfy the equation to find the growth and shrinkage faults in the equation assuming it came from a PLA. More selection, crossover, and mutation techniques will be added and tested to see if the new techniques improve the performance of finding one or more combination of values that satisfy the equation. The solver will be compared with other algorithms, specifically those that where submitted in the SAT Race, to compare the techniques used in each algorithm.

7. ACKNOWLEDGEMENTS

I would like to thank Dr. Alfredo Cruz, for introducing me into this area of research, Prof. Luis Ortiz Ortiz, for introducing me into the field of computer programming, the Nuclear Regulatory Commission (NRC grant no. 27-10-511) for providing me with a fellowship which allows me to continue with my studies and my research in AI, and my fellow grant recipient and to those professors who gave me their opinion about my work.

REFERENCES

Cruz A. (2002). Evolutionary Algorithms for VLSI Test Automation. The Graduate School of Computer and Information Sciences Nova Southeastern University

Luger G. F. & Stubblefield W. A. (1997). Artificial Intelligence Structures and Strategies for Complex Problem Solving. Boston, MA: Addison-Wesley Longman Publishing Co

Marques-Silva, J. (2008). Practical Applications of Boolean Satisfiability. *Workshop on Discrete Event Systems (WODES'08)* (May 2008), Goteborg, Sweden. DOI= http://eprints.ecs.soton.ac.uk/15340/1/jpms-wodes08.pdf.

SATLIB – Benchmark Problems, Retrieved February 28, 2011, from http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html

Obitko, M. M. (n.d.). Main page - Introduction to Genetic Algorithms - Tutorial with Interactive Java Applets. www.obitko.com. 1998. Retrieved February 6, 2011, from http://www.obitko.com/tutorials/genetic-algorithms/index.php

Tomassini, M. (1995). A Survey of Genetic Algorithms. Annual Reviews of Computational Physics III, Volume 3**.** (October 1995). DOI= http://neo.lcc.uma.es/Articles/tomassinixx_2.pdf

Whitley, D. (1994). A Genetic Algorithm Tutorial by Darrell Whitley. Statistics and Computing (4):65-85, 1994. DOI= http://www.cs.colostate.edu/~genitor/MiscPubs/tutorial.pdf

Authorization and Disclaimer