8-10-2004

Genetic Algorithm Parameter Optimization: Applied to Sensor Coverage

Ferat Sahin
Rochester Institute of Technology

Giuseppe Abbate
Harris Corp.

Recommended Citation
Ferat Sahin, Giuseppe Abbate, "Genetic algorithm parameter optimization: applied to sensor coverage", Proc. SPIE 5440, Digital Wireless Communications VI, (10 August 2004); doi: 10.1117/12.542404; https://doi.org/10.1117/12.542404

Genetic algorithm parameter optimization: applied to sensor coverage

Ferat Sahin[a]*, Giuseppe Abbate[b]
[a]Rochester Institute of Technology, 79 Lomb Memorial Dr., Rochester NY 14623
[b]Harris Corporation 1680 University Ave., Rochester NY 14610

ABSTRACT

Genetic Algorithms are powerful tools, which when set upon a solution space will search for the optimal answer. These algorithms though have some associated problems, which are inherent to the method such as pre-mature convergence and lack of population diversity. These problems can be controlled with changes to certain parameters such as crossover, selection, and mutation. This paper attempts to tackle these problems in GA by having another GA controlling these parameters. The values for crossover parameter are: one point, two point, and uniform. The values for selection parameters are: best, worst, roulette wheel, inside 50%, outside 50%. The values for the mutation parameter are: random and swap. The system will include a control GA whose population will consist of different parameters settings. While this GA is attempting to find the best parameters it will be advancing into the search space of the problem and refining the population. As the population changes due to the search so will the optimal parameters. For every control GA generation each of the individuals in the population will be tested for fitness by being run through the problem GA with the assigned parameters. During these runs the population used in the next control generation is compiled. Thus, both the issue of finding the best parameters and the solution to the problem are attacked at the same time. The goal is to optimize the sensor coverage in a square field. The test case used was a 30 by 30 unit field with 100 sensor nodes. Each sensor node had a coverage area of 3 by 3 units. The algorithm attempts to optimize the sensor coverage in the field by moving the nodes. The results show that the control GA will provide better results when compared to a system with no parameter changes.

Keywords: Genetic Algorithms, Parameter Optimization, Adaptive Genetic Algorithms.

1. INTRODUCTION

Genetic Algorithms are powerful tools, which when set upon a solution space will search for the optimal answer. These algorithms though have some associated problems such as pre-mature convergence and lack of population diversity. These problems are inherent to the criteria used for evolution of the individuals. These problems can be controlled with changes to certain evolutionary parameters such as crossover, mutation, and selection. Adaptive Genetic Algorithms (AGA) attempt to fix these problems by having preset equations and expert information detect changes in the population and then to change the parameters. This paper attempts to tackle these problems but by optimizing the evolutionary parameters by another GA. This GA will be called the ***control GA*** in this paper. The control GA will determine which parameters should be used in a generation. The control GA has three crossover operators, two mutation operators, and five selection operators. The crossover operators used are: one point, two point, and uniform. The mutation operators used are: random and swap. The selection operators used are: best, worst, roulette wheel, inside 50%, and outside 50%.

For every control GA generation each of the individuals in the population will be tested for fitness by being run through the simple GA with the assigned parameters. During these runs the population used in the next control generation is compiled. So both the issue of finding the best parameters and the solution to the problem are attacked at the same time.

The goal of the ***problem GA*** is to optimize the sensor coverage in a square field. The GA attempted to optimize the sensor coverage in the field by moving the nodes in the field. The field is a 30 x 30 grid. This provides 900 possible positions for the sensors. The field will be populated by 100 robots, which are randomly placed (dropped). Each robot will have sensor coverage of 3 units by 3 units.

Robots are small enough so that more than one can populate a single grid space, even though the system will only recognize location down to 1 by 1 grid spacing. The performance of the system will be judged by the final sensor coverage of the field. All of the coding for this problem is done in MATLAB. The fitness was dependent on the amount

* feseee@rit.edu; phone 1 585 475 2175; fax 1 585 475 5845

Digital Wireless Communications VI, edited by Raghuveer M. Rao, Sohail A. Dianat, Michael D. Zoltowski, Proceedings of SPIE Vol. 5440 (SPIE, Bellingham, WA, 2004)
0277-786X/04/$15 · doi: 10.1117/12.542404

of coverage created on the field by the robot positions. Any overlap of sensor coverage would cause this number to decrease and thus that individual would have a lower fitness. The fitness function can be found in section 3.2.

Section 2 explores literature survey on Genetic algorithms and sensor networks. The algorithm implementation will be presented in Section 3. Section 4 presents results of the proposed solution to the sensor coverage problem and comparison of the proposed method with other methods based on benchmark fitness functions. Finally, conclusion and future work are presented in Section 5 and 6 respectively.

2. LITERATURE SURVEY

2.1 Genetic Algorithms

The fundamentals of Genetic Algorithms were first proposed by Holland[9]. Holland was also the first to provide an explanation for why GA works. He explained it with the Schema theorem. In this case a schema is a pattern of gene values which can be represented in a binary coding by using a string of characters in the alphabet {0, 1, #}. Genetic algorithms have a population of solutions referred to as *individuals* or *chromosomes*. A chromosome is said to contain a particular schema if it matches that schemata. For example the chromosome "1010" contains the schemata "10##", "#0#0", "##1#", and "101#". The order of the schema is the number of non # symbols it contains[3]. Individuals in the population of a GA are given reproductive trials to pass on genes from one generation to the next. Also in GA, typically the individuals with the best fitness are given more trials to produce offspring. "Holland showed that the optimal way to explore the search space is to allocate reproductive trials to individuals in proportion to their fitness relative to the rest of the population" [3]. Thus, the best schema receives more reproductive trials to create better offspring.

By using mating, mutation and killing on a population of solutions GA hope to evolve the best solution. It is very important to rank individuals against the rest of the population. Different strategies for mating and killing of individuals can be implemented by this ranking. To rank each individual in the population, a *fitness function* must be found which can represent the problem. The fitness function is able to calculate how well each individual solves the problem set before the GA. Every time a GA runs through the population creating offspring from parents a *generation* has past. These are basics of GA but there have been many advances over the years.

Beasley provides a background of current GA fundamentals. Although J.H. Holland began the GA research, many people have continued that work and refined GA. Traditional simple Genetic Algorithms have been coded as shown in figure 1[3]. The code in figure 1 makes reference too many different operators used in Genetic Algorithms. Crossover and mutation both are used in the mating process of the GA.

```
Start Genetic Algorithm
Generate initial population and compute fitness of each individual
while finished == false
        %produce new generation
        for population_size/2
                %begin reproductive cycle
                select two individuals from old generation for mating
                %biased in favor of the fitter individuals
                perform crossover to produce to new offspring
                compute fitness of two new offspring and insert in new generation
        end
        if population has converged then
                finished == true
        end
end
```

Figure 1: A traditional GA code.

In GA parents are selected and through mating operators such as crossover and mutation offspring are created. The diagram in Figure 2 shows how the crossover and mutation processes occur[3].

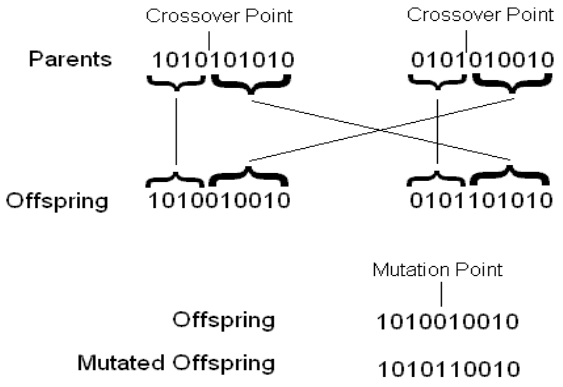

Figure 2: Crossover and mutation

In figure 2 parents, which were chosen to mate, will create two new offspring. Then a mutation is added to one bit of the offspring.

There are many choices, which can be made in determining how information is transferred from one generation to the next in GA. In a GA with *generation gaps* the parents are allowed an opportunity to continue in to the next generation[3]. In a GA that is steady state the parents will be replaced by their offspring. There are many problems that are associated with the traditional GA. One of these is the *super-fit individual*, which can dominate a population[3]. In this case an individual with a high but not optimal fitness will come to dominate the population. It will be able to dominate a population because its relatively high fitness will make it more likely to be chosen for mating. This will cause the population to have many individuals similar to the super fit individual. This will cause the population to converge and remove most of the diversity in the population. This is referred to population pre-convergence.

There have been many papers written in which the main goal is to improve the performance of GA[1, 2,4-13]. Many different approaches have been explored. Some of these approaches focus on improving the algorithms and others focus on parameter optimization. Many of the operators and parameters in a GA can be changed to have an effect on the exploration or exploitation of the search space. The *exploration/exploitation relationship* (EER) is a very important in determining how a GA will search the solution space[1]. Herrera[1] notes "different operators or parameter configurations may be necessary during the course of a run for inducing an optimal EER". A high amount of exploration will allow the GA to move quickly through the search space but may hinder it in focusing on a specific area. The level of mutation effects exploration directly. When the mutation rates are low the GA will attempt to exploit the information in the current population because many of the new offspring will be similar to their parents. Therefore the EER will also affect the number of generations that will be needed to reach a satisfactory solution[1].

One approach at optimizing GA was to implement a fuzzy logic controller, which would optimize the GA parameters[1]. Herrera adapts many of the different parameters, which he had found, would impact the EER of a GA. He found that *fuzzy logic controllers* (FLC) are able to control GA parameters and provide good results.

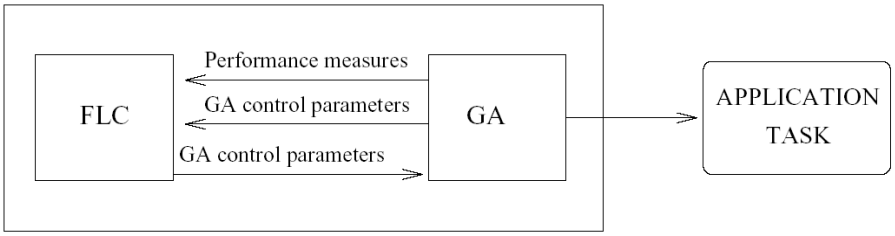

Figure 3: Adaptive GA implemented with fuzzy logic controller.

The FLC controlled GA is able to return results which are as good as any of the standard fixed GA's which shows that they were able to develop a robust GA.

• *Sphere model*	• *Generalized Rosenbrock's function*
$f_1(\mathbf{x}) = \sum_{i=1}^{n} x_i^2$ $-5.12 \leq x_i \leq 5.12$ $f_1^* = f_1(0, \ldots, 0) = 0$	$f_2(\mathbf{x}) = \sum_{i=1}^{n-1} (100 \cdot (x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$ $-5.12 \leq x_i \leq 5.12$ $f_2^* = f_2(1, \ldots, 1) = 0$
• *Generalized Rastringin's function*	• *Griewangk's function*
$f_3(\mathbf{x}) = a \cdot n + \sum_{i=1}^{n} x_i^2 - a \cdot \cos(\omega \cdot x_i)$ $a = 10,\ \omega = 2\pi$ $-5.12 \leq x_i \leq 5.12$ $f_3^* = f_3(0, \ldots, 0) = 0$	$f_4(\mathbf{x}) = \frac{1}{d} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$ $d = 4000$ $-600.0 \leq x_i \leq 600.0$ $f_4^* = f_4(0, \ldots, 0) = 0$

Figure 4: The functions used to test the GA in paper 1

Figure 4 shows the fitness functions, which were used to test the GA in Herrera's paper[1]. But due to difficulties in predicting the GA response there were many open issues. It was difficult to find rule bases for the FLC. The other problem was determining the frequency with which to implement the FLC to adapt the GA. This will also become an issue for the control GA in the research presented in section 3.2.

A different approach to optimize GA is the adaptation of mutation rates[2]. In a traditional GA the mutation rate is an external parameter, which is the same for every individual. In this case each individual is given a mutation rate as part of its bit string[2]. The mutation rates themselves are subject to mutation in this type of GA. It is shown that the second level learning of mutation rates is possible and helps GA find a better solution. In this case there is no global mutation rate. Mutation is one parameter that has been shown to have a significant influence on the GA convergence[1].

Herrera and Lozano are also trying to improve GA. They developed the Heterogeneous Distributed Genetic Algorithms (HDGA)[6]. In HDGA there are many different sub populations running in parallel. The difference between this and the usual Distributed GA (DGA) is in HDGA the subpopulations were each running a using a different crossover operator[6]. In this case many different fuzzy connectives based crossover operators (FCB-crossover) and BLX-a operators. The difference being that FCB-crossover operators are deterministic and will always have the same offspring from the same two parents and the BLX-a operators had a random component which would allow for differing children from the same set of parents if they were mated twice[6]. Each of the crossover operators was better at different things. Some were better at exploitation of the search space and the current populations and others were better at exploration to help get deeper into the search space. Then to allow communication between the subpopulations a migration operator is used.

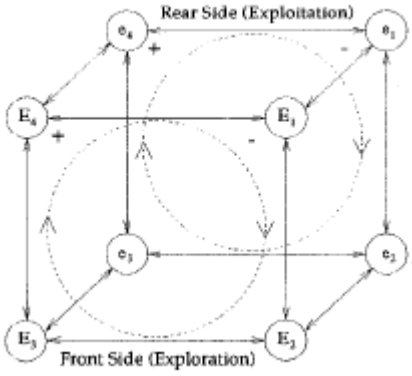

Figure 5: The figure shows the structure of the HDGA

Figure 5 shows how all of the different subpopulations are connected. This is important because it also shows how information can be communicated between subpopulations. This HDGA was shown to provide a good reliability for the convergence and kept a high diversity.

There have been many improvements in GA over the years[8]. Many different types of cross-over and mutation operators as well as dynamically changing the mutation and cross-over rates online have been developed[8].

Genetic Operations and Other Procedures

Selection	Roulette wheel Deterministic sampling Stochastic remainder sampling
Crossover	One cut point by genotype Two cut points by genotype One cut point by chromosome Uniform crossover
Mutation	Bit by bit One mutation by chromosome at a random position
Dynamic Range of pc and pm	One mutation by individual at a random position Based on a linear interpolation Based on a measure of genetic diversity exterior to some limits
Elitism	Simple Elitism Global Elitism

Figure 6: This Figure shows the different operators which were considered in paper 8

There are three main types of GA's the simple GA (SGA), the Steady State GA (SSGA), and the Replacement GA (RGA) [8]. These types of GA's can all benefit from some dynamic changes of the probability of mutation and cross over[8].

A new type of cross over called the multi step cross over (MSX) has also been developed to attempt to optimize GA[7]. This cross over utilizes a neighborhood structure. Initial results showed that the MSX provided improved results for the GA. Local search has also been shown to improve GA[7]. The local search does not put the initial offspring into the next generation but moves the offspring to the nearest locally optimal point. So in this case the cross over operator is only finding good starting points for solution and the local search is playing the main role in finding the optimal solution[7]. MSX views each of the parents that are chosen to mate in their solution space and looks at their distance from each other. Then in a step-by-step manner it gradually moves the offspring from one parent to another. As this happens the offspring will take on features of both parents but in uneven ratios[7].

All of the developments listed above show how the simple GA started and that there have been many different attempts to customize and optimize GA. In many cases the methods try to optimize the GA by removing the problem of lack of diversity and premature convergence.

2.2 Summary of GA Techniques

This section will provide a general overview of some GA operators. There are many different selection methods, which can be used in GA's[3]. Rank Selection - this is the very straightforward system of choosing those individuals with the highest fitness to reproduce. Roulette Wheel Selection – In this system every individual has the opportunity to reproduce but those with higher scores are given a higher probability of being chosen. Tournament Selection- In this system the roulette wheel is used to pick a few individuals to compete to be chosen to reproduce

There are many different types of crossover three of which are explained here[3]. Single Point – cross over at one point in the bit sequence. Two Point – cross over at two points in the bit sequence. Uniform Crossover – In this scheme a map is made with 0's and 1's then the first child will receive all of the bits from parent A when the map bit is a one and from parent B when the map bit is a 0 and vise versa for the second child

There are also many different ways in which to rank the individuals of a population[3]. Ranking (no scaling) –In this case the fitness is used to rank the individuals. Linear Scaling (Fitness Proportionate) – Here a transformation is performed on the fitness based on a linear relationship using the maximum and minimum fitness. Sigma truncation – The fitness is scaled using the population's standard deviation for the fitness transformation. Sharing – This lowers the fitness of individuals that are similar to other individuals in the population.

There are also many different types of mutation operators, which were investigated in this project. Swap bits- In this case, two bits will swap positions in the bit string. Random- This mutation will randomly change the value of one bit. Swap Sequence- In this case two sequences of bit are swapped inside the bit string.

Premature Convergence and Speciation are big problem in all Genetic Algorithms[3]. Although many of the operators and techniques I have already mentioned were developed to help increase the efficiency of the GA they did not directly attack this problem. Two methods were found which did attempt to directly discourage these problems in GA[3]. De-Jong style de-crowding using replacement schemes such as one where the offspring will replace the individuals most similar to them can help increase genetic diversity. The Goldberg process of de-rating the fitness of individuals that are less unique than others in the population is another method, which is helpful.

There are many replacement schemes, which can be used to help increase the diversity of the population[3]. These replacement schemes use a straight forward ranking by fitness. Replace Worst – offspring replace those with the worst fitness. Replace Best – offspring replace those with the best fitness. Replace Randomly – offspring replace randomly. Replace Inside 50%– Offspring replace the middle 50%. This will leave the best individuals and he worst hoping to create a wider number of possibilities for the offspring. Replace Outside 50% –offspring replace the outside 50%. This will remove all of the super individuals, which may force the population into a local maximum. In the next replacement the fitness values are normalized. Replace Using a Roulette Wheel –Once the fitness have been normalized depending on that fitness they will be placed into a "roulette wheel" with the number of entries in that wheel proportionate to their normalized fitness.

All of these methods have been developed to increase the power of GA's.

2.3 Sensor Network Literature Survey

Sensors continue to get smaller with the advances in MEMS technology and are limited by battery power[15]. Wireless sensor networks have proven to be very efficient at monitoring inhospitable and dangerous environments[16]. Many different techniques have been proposed to help alleviate the strain on the battery power of these micro-sensor networks. This research investigates different clustering or communications topologies to optimize communications and power consumption. The remainder of this section reviews some of the different techniques on wireless sensor coverage problem.

Many architectures for wireless micro-sensor networks have been developed. One of these protocols is called low energy adaptive clustering hierarchy (LEACH) [15]. It is designed to help lengthen the life of micro sensors. This protocol creates clusters of nodes depending on how they are positioned. Then it assigns a cluster head and all nodes in that cluster communicate to a base station through the cluster head. In every subsequent round of communication the cluster head responsibility is moved to other member of the cluster so not to drain all of the power from one node by having it to be the cluster head all the time. LEACH is able to extend the life of these wireless micro-sensor networks[15]. Two different methods for LEACH have been developed both a distributed LEACH and a centralized LEACH[15]. This type of communication optimization is very important because as these sensors become smaller and cheaper to manufacture they can be used for many different purposes. But life of the nodes will dictate when the sensor network will fail and how much information they will be able to communicate.

Author investigates the power efficient organization in wireless sensor networks. In this paper they disperse many more sensor nodes in the area than would be needed. This is different from the assumptions made in LEACH[18]. This is done so that there is overlapping coverage. Then their algorithm creates mutually exclusive sets of sensors clusters. Each of these provides coverage of the entire area. Each one of these sets is turned on for one round successively and the

pervious on is turned off. This takes advantage of the overlapping coverage and provides longer life to the sensor network. So in this network the life of the system is directly related to the how many sets of sensors are created.

Schurgers et. al. also focuses on trying to extend the life of sensor networks[14]. Most of sensor nodes energy is consumed by its communications subsystem, which consists of its transceiver. Putting this subsystem into a sleep mode can save energy. Sparse Topology and Energy Management (STEM) deals with have a particular node initiate communication with another node by sending out a beacon signal with the nodes ID in it[14]. The node, which is asleep, will be periodically turning on its radio to check for communication. If it receives the Id number it will turn on and begin communication. They use one frequency band for the "wakeup plane" of communication and one frequency band for the data plane. Allowing the nodes to turn off their radios allows for a significant amount of energy conservation.

3. PROPOSED ALGORITHM IMPLEMENTATION

In this section an explanation will be provided for the main parts of this GA implementation. The three main components are the control GA, the simple GA, and the subpopulation integration. All of these components are shown in figure 7.

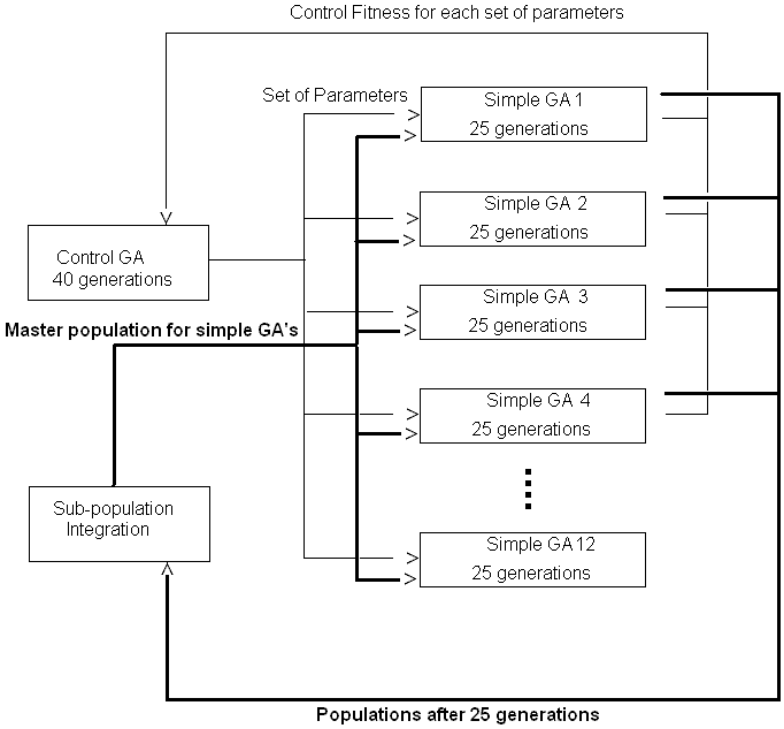

Figure 7: This a basic flow chart of the entire system described below.

3.1 Problem GA

Genetic Algorithms are evolutionary algorithms that start with a random population. The population size was 40 individuals. In the initial population for each individual the fitness was calculated. For the case of the sensor problem fitness represented the total sensor coverage of the field. That would create a maximum fitness of 900 if the entire 30 by 30 field had sensor coverage.

$$sensor\_fitness = \sum_{n=1}^{30}\sum_{m=1}^{30} x(n)y(m) \tag{1}$$

The simple GA places all of the robots on the field and then all of the sensor coverage is calculated. Each grid space, which has coverage, is assigned a 1 and those that do not have coverage are assigned zero. Next as shown in equation 1

by summing all of the grid spaces with sensor coverage a straightforward fitness is found. The simple GA would then take a set of operators from the control GA. Next using the assigned selection method the population was cut to 50 individuals, which would be randomly bred to create two children for every pair of parents. The parents were allowed to breed more than once. But children could not be breed with parents in this generation.

To insert new robot locations a high mutation rate was set. Mutation in this case is defined by moving a random robot to a new random location. The algorithm then repeats for a set number of generations. During the run the best possible solution (individual) is stored until the end of the algorithm is reached and the answer displayed both in number and graphically. Once the simple GA had completed all 25 generations its population of 40 individuals would be passed back to the sub-population integration operator, which is described later in this section.

3.2 Control GA

For this project a control GA was implemented to help optimize the parameters, which are used by the simple GA. To determine the optimal parameters the GA would run 12 different individuals with the same initial population and different parameters. Each of these individuals would be a combination of the parameters shown in figure 8.

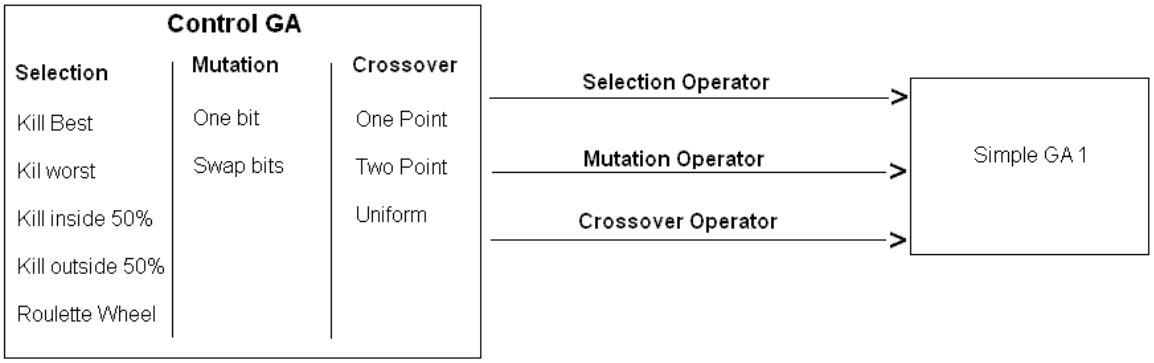

Figure 8: The figure shows the transfer of parameters and the possible parameters.

The *control fitness* was the fitness, which would be returned to the control GA to determine the ranking of the sets of parameters. This fitness equation is shown in Equation 2 where Fc is control fitness, Fbf is best final fitness, and Fbi is best initial fitness.

$$Fc = Fbf - Fbi \tag{2}$$

Each simple GA would return fitness equal to the amount of increase, which they created in that populations best individual from start to finish. From that the best parameters would be determined and these would then move into the selection and crossover in the control GA. For the control GA the selection used was kill the worst and the crossover was uniform. Also in an attempt to keep the population as diverse as possible sub-population integration was used, this is explained later in this section. There were 15 possible combinations of parameters. There are 3 types of crossover, 5 types of selection and 2 types of mutation. These were described in more detail earlier in section 2.2.

3.3 Sub-population Integration

At the end of every control GA generation every set of parameters has taken the original master population and used it in a simple GA. To get the best master population for the next control GA generation these sub-populations are all combined and ranked. This process is shown in figure 7. Then the highest unique 40 individuals are chosen to be the next master population. This method was more efficient than just taking an equal number of individuals from each sub-population because each of the members is guaranteed to be unique.

4. RESULTS

4.1 Sensor Coverage Results

The main focus of this GA was the sensor coverage problem. The first step was seeing how well just a simple GA could perform. The simple GA had a single point crossover and one-bit mutation operators. First 8 samples were run at 5000 generations and those results were compiled. The average fitness after 5000 generations was 690 out of a possible 900 units. This meant that on average 76% of the field has sensor coverage after only 5,000 generations.

At this point an extended run was performed to investigate what the maximum coverage could be obtained from the simple GA. The extended run went for 50,000 generations; the fitness was able to go up to 758 units. The field now had sensor coverage of 84%. The algorithm could have continued to work but the number of generations required to get even a slight improvement in fitness would increase exponentially. These seemed like good percentages until the control GA was implemented.

With the control GA there was significant improvements in performance without having 50,000 generations, which the simple GA needed. The first step was to allow the control GA to change the type of crossover operators and selection, which was used in the subpopulations. The control GA has 12 individuals in its population each of those individuals represents a specific set of parameters. Each of the subpopulations was made up of 40 individuals and was run for 25 generations before the control GA would step in and evaluate the progress. The control GA was run for 30 generations for each of these generations the 12 control individuals were run on identical sub-populations. When the control GA was originally coded it was coded with a 5-bit configuration for the individuals but it became apparent that that was not necessary and more randomness could be had with a four bit individual configuration.

The control GA was initially run with 40 control generations. The fitness used to rank the parameters was the change in fitness over that simple population generation as explained in section 3.2. The control GA showed an improvement of 12.5% over only the simple GA. This was a significant improvement and much above the rate of the run at 50,000, which was recorded in the simple GA. The simple GA wasn't able to break 760 but in the control GA regularly reach near 800 in sensor coverage. These results are shown in run 1 of table 1. Next to help stop the pre-convergence and lack of diversity in the population a de-crowding strategy was added. At the end of every generation identical members of the population were removed. This was done because it was found that at the end of the control GA the sub-populations would have many repeated individuals. This continued to increase the convergence of the GA and the results are in run 2 of table 1. The addition of this de-crowding strategy to the control GA increases the convergence by 1.75% on average. It also helped the GA continue searching and not get over burdened by the lack of diversity. Finally to help have a more stable population and retain all of the best individuals from the parameter runs the subpopulation integration was implemented. It improved results again and made the results more consistent as can be seen in run 3 of table 1.

Table 1: The table shows the results from different phases in the implementation

	Fitness		
Sample	**Run 1**	**Run 2**	**Run 3**
1	799	810	812
2	796	812	825
3	807	824	820
4	793	807	822
5	799	824	830
Ave	798.8	815.4	821.8

4.2 Comparison of Results

In the literature, four functions were chosen to help show how the GA performed[1]. In this project two of those functions were chosen to be used help compare the performance of the proposed GA to their GA[1]. The two selected were the generalized Rosenbrock function and, generalized Rastringin function. Next the results of each of those fitness functions will be shown and their comparison:

The Rosenbrock function was used as a minimization fitness function, which is shown below.

$$f_{Rosenbrock}(x) = \sum_{i=1}^{n-1} (100 * (x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$$
$$-5.12 \le x_i \le 5.12 \qquad (3)$$
$$f^*_{Rosenbrock} = f_{Rosenbrock}(1,.....,1) = 0$$

Equation 3 above was used to help determine the effectiveness of the GA that was developed. The chart in figure 9 shows which parameters performed the best in each generation. Usually at the beginning of the GA the parameters are steady and similar. The farther into the search space the GA gets the more random the parameters, which will perform better. The different parameters will allow the GA to search in different ways by using the different cross over and selection methods.

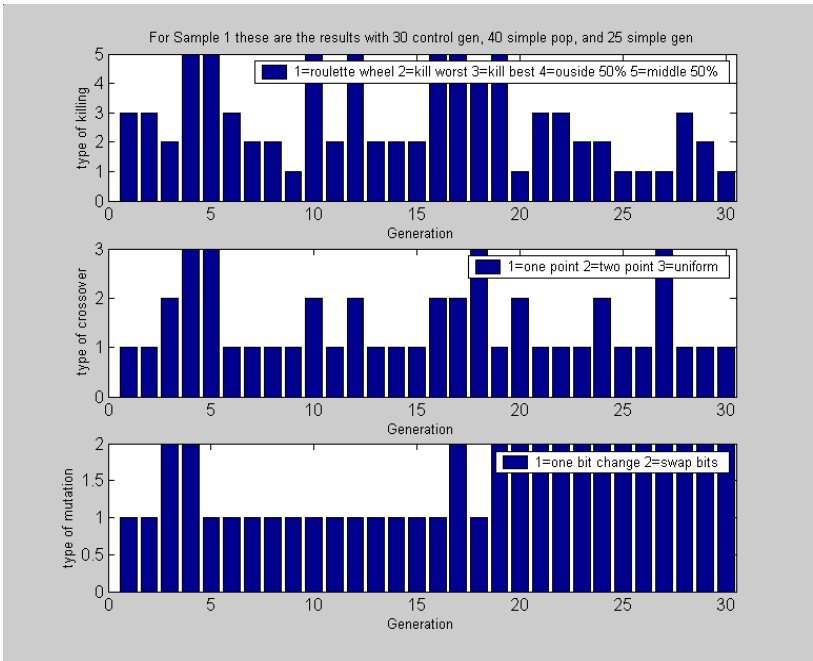

Figure 9: The above figure shows which parameters worked best over 30 generations in sample 1 of Rosenbrock.

The Rastringin function was also used as a minimization fitness function, which is shown below

$$f_{Rastringin}(x) = 10 * n + \sum_{i=1}^{n} x_i^2 - 10 * \cos(2\pi * x_i)$$
$$-5.12 \le x_i \le 5.12 \qquad (4)$$
$$f^*_{Rastringin} = f_{Rastringin}(0,.....0) = 0$$

Equation 4 above was used to help determine the effectiveness of the control GA. The chart in figure 10 shows which parameters performed the best in each generation. Herrera[1] uses the Rastringin function to test the FLC controlled GA. Many different algorithms were used in that case to determine how the FLC changed the GA parameters.

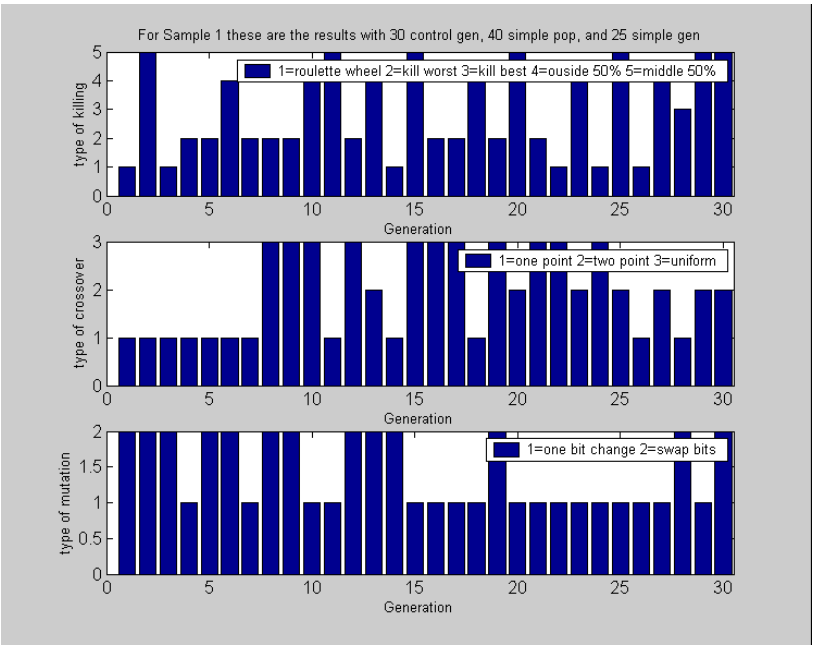

Figure 10: The above figure summarizes the parameters used during the 30 control generations of sample 1 of Rastringin.

Table 2: The table below shows the resulting fitness of the control GA from 10 runs and the average for both functions.

Run	Rosenbrock	Rastringin
1	100.6	32.5
2	62.2	36.5
3	129.2	42.6
4	73.7	29.9
5	68.5	42.7
6	21.3	11.9
7	22.8	47.07
8	21.9	45.6
9	20.3	21.7
10	22.5	46.2
Averages	54.3	35.667

Table 3: Table shows comparison to results in [1]

		Results from [1]	Control GA
Rosenbrock Function	**Best**	9.04E-04	20.3
	Worst	36.8	129
	Average	17.01	54.3
Rastringin Function	**Best**	5.68E-15	11.9
	Worst	227	47.07
	Average	25.58	35.667

Herrera uses the Rosenbrock and Rastringin function to test the FLC controlled GA[1]. Many different algorithms were used in that case to determine how the FLC changed the GA parameters. The average of all of the algorithms for the Rosenbrock function was 17.01. This was only slightly lower than the best result for the control GA, which was 20.3. The average of all of the algorithms for the Rastringin function was 25.58[1]. This was only slightly higher than the best result for the control GA, which was 11.9. In this case, the control GA performed comparably on average to the results in [1].

5. CONCLUSION

There have been significant improvements gained from the control GA optimizing the parameters over the simple GA. The improvement was over 20%. When the GA was tested against fitness function used in other papers it performed on a comparable level to some very robust GA's. Overall this GA performed very well although there are improvements, which could make it much stronger.

6. FUTURE WORK

There is still much work, which can be done to improve the proposed control GA method. There are more parameters, which can be added to the control GA to help create more diversity. Also FCB-crossover operators[1] could be used to help with the exploitation/exploration of the search space. Also there needs to be more work done to determine the correct amount of simple GA generations which should be run for every individual of the control GA.

REFERENCES

1. Francisco Herrera and Manuel Lozano, "Adaptation of Genetic Algorithm Parameters Based on Fuzzy Logic Controllers", *Genetic Algorithms and Soft Computing*, Physica-Verlag, 1996, 95-125
2. Thomas B¨ack., "Self-adaptation in genetic algorithms", in *Towards a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, Cambridge, MA, 1992. MIT Press.
3. David Beasley, David R. Bull, and Ralph R. Martin, "An Overview of Genetic Algorithms Part 1", *Fundamentals in University Computing*, 1993
4. N. Chaiyaratana and A.M.S. Zalzala, "Recent Developments in evolutionary and Genetic Algorithms: Theory and Applications", in *Genetic Algorithms in Engineering Systems: Innovations and Applications*, 2-4 September 1997 Conference Publication No.446 IEE
5. Lei Wang, Licheng Jiao, "A Novel Genetic Algorithm Based on Immunity", *National Key Lab for Radar Signal Processing, IEEE International Symposium on Circuits and Systems* May 28-31 2000, Geneva Switzerland
6. Francisco Herrera and Manuel Lozano, "Heterogeneous Distributed Genetic Algorithms Based on the Crossover Operator", *Genetic Algorithms in Engineering Systems: Innovations and Applications*, 2-4 September 1997 Conference Publication No.446 IEE
7. Takehsi Yamada and Ryohei Nakano, "A genetic algorithm with multi-step crossover for job-shop scheduling problems", *IEE Genetic Algorithms in Engineering Systems: Innovations and Applications* September 12-14 1995
8. J.A. Vasconcelos, J.A. Ramirez, R.H.C. Takahashi, and R.R. Saldanha, "Improvements in Genetic Algorithms", *IEEE Transactions on Magnetics* Vol. 37 No.5 September 2001
9. J.H. Holland, *Adaptation in Natural and Artificial Systems*, MIT Press 1975
10. K. A. De Jong, "Analysis of the behavior of a class of genetic adaptive Systems," Ph.D. dissertation, Univ. of Michigan, Ann Arbor, 1975.
11. D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning,* Addison Wesley, 1989.
12. H. Muhlenbein, "Parallel genetic algorithms, population genetics and combinatorial optimization", *in 3rd ICGA*, pages 416-421 1989
13. N.L.J. Ulder E. Pesch, P.J.M. van Laarhoven, J. Bandelt, and E.H.L. Aarts, "Genetic local search algorithm for the traveling salesman problem", *In PPSN 1st* pages 109-116 1994
14. Curt Schurgers, Vlasios Tsiatsis, and Mani B. Srivastava, "STEM: Topology Management for Energy Efficient Sensor Networks", *IEEEAC* paper #260, Updated Sept 24, 2001 2 Page(s) 1-10
15. Heinzelman, W.B.; Chandrakasan, A.P.; Balakrishnan, H., "An application-specific protocol architecture for wirelessmicro-sensor networks", *Wireless Communications, IEEE Transactions on*, Volume: 1 Issue: 4, Oct 2002 Page(s): 660 –670
16. Slijepcevic, S.; Potkonjak, M., "Power efficient organization of wireless sensor networks", Communications, 2001. ICC 2001. IEEE International Conference on , Volume: 2 , 2001 Page(s): 472 -476 vol.2
17. W. Heinzelman, A. Chandrakasan and H. Balakrishnan, "Energy efficient Communication Protocol for Wireless Micro-sensor Networks", *Proceedings of the 33rd Hawaii International Conference on System Sciences (HICSS '00)*, January 2000.