Application of Genetic Algorithms to Transmit Code Problem of Synthetic Aperture Radar

Fernando Palacios Soto, James M. Stiles, and Arvin Agah

Department of Electrical Engineering and Computer Science
The University of Kansas, Lawrence, KS 66045 USA

ABSTRACT

This paper presents the design and development of a genetic algorithm to compute an upper bound to evaluate a formal algorithm developed to solve the transmit code problem of Synthetic Aperture Radar (SAR). The input to the genetic algorithm is a set of propagation matrices that contain virtual information of the transformation of a signal transmitted and received by a SAR. The output is an upper bound approximately 11 times smaller than the results provided by the formal algorithm. The contributions of this paper are twofold: the upper bound found for the transmit code problem, and a tool that can be used for further research in similar domains.

KEYWORDS

genetic algorithms, synthetic aperture radar, soft computing, transmit code problem

1. INTRODUCTION

Synthetic Aperture Radar (SAR) is commonly used to make high resolution ground maps, and can also be used from earth to map a target in the space if the target provides the necessary motion—this is called "inverse SAR" (Toomay, 1989). To form an image, a SAR emits electromagnetic waves to targets in the ground (or

Reprint requests to: Arvin Agah, Professor, Department of Electrical Engineering and Computer Science, The University of Kansas, Lawrence, KS 66045-7621 USA; Email: agah@ku.edu; URL: http://people.ku.edu/~agah/

space in the case of inverse SAR) on a specific area. When the electromagnetic signal bounces back, the SAR's antenna detects a scattered signal (or different signals) from the different targets. This signal is then processed to extract information about the targets (Fitch, 1988). One problem with this signal is that if the information about two different targets is highly correlated, then distinguishing between the two targets would not be possible. In other words, an ambiguous signal is obtained (Fitch, 1988). The challenge then is to find a vector known as the *transmit code*, which when applied to the information of the targets, minimizes the correlation among targets. Although several methods for this purpose have been developed (Chung Lin, 2000; Goodman, 2002), their results has not been compared against an upper bound, so their utility is yet to be assessed. The design of space-time codes and waveforms has become an important research topic in wireless communication systems, and is beginning to generate interest in the radar community as well.

A genetic algorithm (GA) is developed to find an upper bound to validate the quality of the results obtained from an available method (Chung Lin, 2000) to minimize the correlation between targets. More over, the GA shall be able to be initiated with good results already found to attempt to find better results when possible. The development of the GA is performed as described in (Goldberg, 1989).

This paper is organized into five sections. After Section 1 an introduction, Section 2 provides the required background, and Section 3 describes in detail the technical approach. Section 4 presents the results obtained during the process, and Section 5 concludes this paper with a discussion of the utility ofthis paper.

2. BACKGROUND

This section provides the mathematical foundation for the design of optimal transmit codes, and presents an introduction to genetic algorithms and its design and implementation.

2.1 Mathematical Foundation of the Transmit Code Problem

A radar sensing problem can generally be described in terms of five fundamental elements: the radar transmitter, the transmit propagation path, the illuminated scatterers, the scattering propagation path, and the radar receiver. Specifically, these elements can be mathematically describe in terms of a linear relation between the transmit signal function (the input) and the receive signal function (the output). The

linear system lying between the input and output consists of three parts: the transmit propagation path, the illuminated scatterers, and the scattering propagation path.

A Synthetic Aperture Radar (SAR) is generally used for high-quality imaging applications used for civil and military purposes (Chung Lin, 2000). An SAR consists of a transmitter and a receiver and may have just a single antenna (Fitch, 1988)—for instance, a single aperture SAR consists of one antenna that serves both as a transmitter and a receiver. The antenna sends a single signal to earth, and then the antenna receives back a scatter signal that carries information about the targets hit by the original signal. A multiple aperture SAR can be seen as an array of SARs where they sends different signals at different angles and then receive back the scatter signal from different angles (Chung Lin, 2000).

Once received, the scattered signal is processed to extract information about the targets. The information about the transformations of a signal emitted by a SAR is stored as a set of matrices of complex numbers, where every matrix in the set is known as a propagation matrix (Chung Lin, 2000). These transformations include the original signal emitted by the transmitter of a SAR and the signal received by the receiver. Each matrix in the set is denoted as H_i, where $1 < i \leq T$ and T is the cardinality of the set of matrices. Using this set of matrices, the received signal by a SAR over an area can be expressed as:

$$r = \sum_{i=1}^{T} \gamma_i H_i s = \sum_{i=1}^{T} \gamma_i \rho_i$$

Where γ_i is known as the reflectivity of the target; ρ_i is a vector known as the normalized response from target i and $\rho_i = H_i s$; and s is known as the transmit code and it must be a unity vector; in other words s must comply with:

$$s's = 1$$

The operator s' denotes the conjugate transpose of a vector or matrix, determined by taking the transpose of matrix, followed by applying the complex conjugate to each element of the matrix.

Ideally the responses from different targets should be as uncorrelated as possible. In other words, it is desired that the normalized responses to be orthogonal:

$$\rho_i' \rho_j = 0 \quad where \quad i \neq j$$

Equivalently:

$$s'H_i'H_j s = 0 \quad where \quad i \neq j$$

For most practical cases there are no solutions for vector s such that the correlation between two targets is equal to zero. The problem then turns out to be finding a vector s such that the correlation among targets in minimized. For that end, two criteria have been developed. The first criterion seeks to minimize the total correlated normalized energy α between a vector ρ_i and all other vectors. It should be noted that in this paper i will always be equal to 1. Therefore, the first criterion is then defined as:

$$\alpha = \sum_{t=1}^{T} \frac{|\rho_1'\rho_t|^2}{|\rho_1|^2 |\rho_t|^2}$$

The second criterion tries to minimize the maximum normalized correlation β between ρ_1 and any other vector. This problem is known as the minimax problem. The second criterion is defined as:

$$\beta = max\left\{\beta_t : \beta_t = \frac{|\rho_1'\rho_t|^2}{|\rho_1|^2 |\rho_t|^2} \; for \; t \in \{2,3,4,....,T\}\right\}$$

Both criteria are used in the paper to evaluate the outcome of the approached based on genetic algorithms.

2.2 Genetic Algorithm

Genetic Algorithms (GAs) are defined as "search algorithms based on the mechanics of selection and natural genetics" (Goldberg, 1989). A genetic algorithm is a search algorithm combines the features of the fittest members of a population with a controlled, still randomized, information exchange. For every generation, a population of solutions represented by strings of bits, which are called chromosomes, is generated from chromosomes of fit individuals from the previous generation. Genetic Algorithms efficiently exploit the historical information of previously generated groups of individuals to improve the next generations. A GA performs its function by means of two genetic operators: *crossover* and *mutation.*

Every invocation of the crossover operator generates two offspring for a new generation from two individuals selected from the previous generation. This task is performed in two steps, namely, selection of the two individuals, and performing the

crossover. The selection process is done using a biased roulette wheel, where each individual is assigned a portion of the wheel proportional to its individual fitness value. Consequently, the random selection of an individual is made on the roulette wheel; i.e., the wheel is "spun." This selection process gives a higher probability of being selected to individuals with higher fitness values. Afterwards, the crossover itself is performed on the two selected individuals, and two offspring are generated. A number between 1 and the number of bits (or chromosomes) in an individual minus one is randomly generated. The new offspring are created by exchanging all the chromosomes from the chromosome in the position of the generated number plus one to the final chromosome. The crossover process is shown in Figure 1.

The crossover operator combined with the selection process is the bulk of the processing power of a genetic algorithm. Although the crossover operator is a very powerful operator generating new improved individuals, these individuals tend to lose potentially useful genetic material. In other word, the GA will not search for better solutions in other places of the solution space rather than where the best-so-far solutions have been found. It is here where the mutation operator takes place. The mutation operator forces the GA to search for new solutions in places where the crossover alone will not search. The mutation process is performed by executing a walk through all the bits in a string randomly turning a 1 into 0, and vice versa. The probability of bit change has been shown by empirical studies to be effective at rates close to one per thousand bits.

Genetic algorithms have been applied to a variety of domains, including radar. These include radar processing (Aydemir et al. 2003; Boyd & Glass, 1993; Daida et al. 1995; Filippidis et al. 1999; Li and Ling, 2003; Porsani et al. 2001; Stanhope

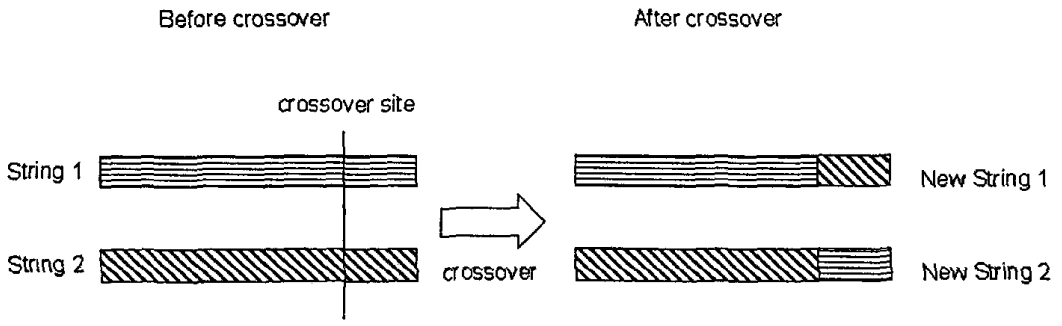

Fig. 1: The crossover process.

and Daida, 1998; Yilmaz et al. 2003), radar design (Chambers et al. 1995; Michielssen et al. 1993; Qian et al. 2001; Sanchez, 1998; Villegas et al. 2004; Weile & Michielssen, 2001); and radar modeling (Hughes, 1998; Hughes & Leyland, 2000; Sarabandi and Li, 1997).

3. TECHNICAL APPROACH

This section describes the technical approach, including the experimental setup and the experiments conducted during this project. Several tasks were performed to find an upper bound to validate the effectiveness of an algorithm currently under development to solve the transmit code problem. Two series of experiments were conducted using the two criteria described above.

3.1 Experimental Setup

All the implementation and testing for this project was conducted on a computer model HP Pavilion 775y (HP, 2004) with an Intel® Pentium® (Intel, 2004) 4 processor at 2.40 GHz with 512 MB of RAM and an 80GB hard disk. This computer runs Windows® XP Professional (Microsoft, 2004). Microsoft Visual Studio v6.0 was used to develop the code for the GA using Visual C++ v6.0. The input to the program was provided in the form of a MATLAB file that contained a set of H matrices. Each matrix in the set was generated using random data. Three different issues were identified, two involving the basic data type used, and one due to the randomness of the GA. These implementation issues are described later in this section. After these issues were resolved, series of experiments were performed. Series of runs were executed for the two criteria. For each run, four files were generated: two text files and two binary files. The text files contained the results of each generation and the average and maximum fitness values per generation, respectively. The two binary files contained all individuals generated per every generation in a binary format, and some of the best individuals generated during the run, respectively. The binary file with the best individuals generated during a run was input to the next run, so that the GA did not have to start with a completely random population. The best individuals of the last run of each series of runs were saved and used for validation as a means to evaluate the correctness of the model. Figure 2 shows an activity diagram of the complete experimental process.

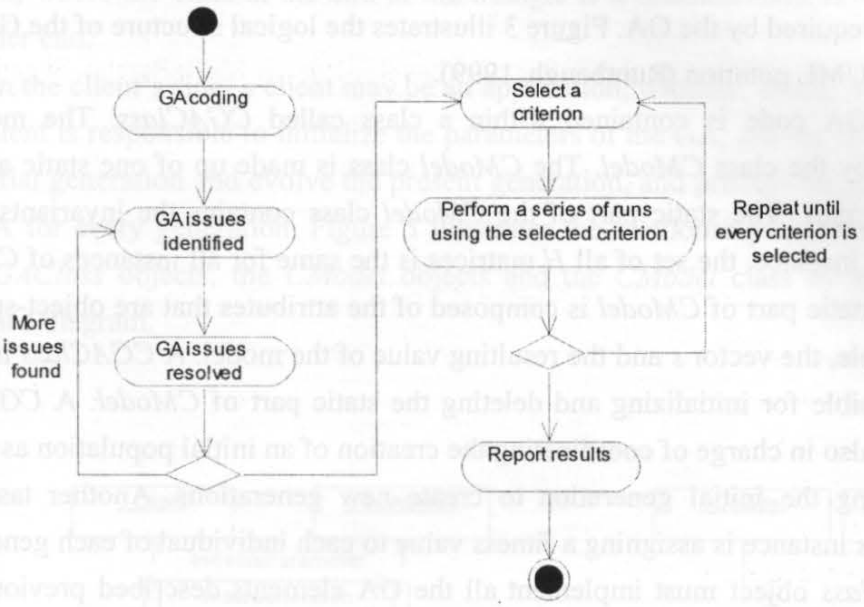

Fig. 2: Experimental process.

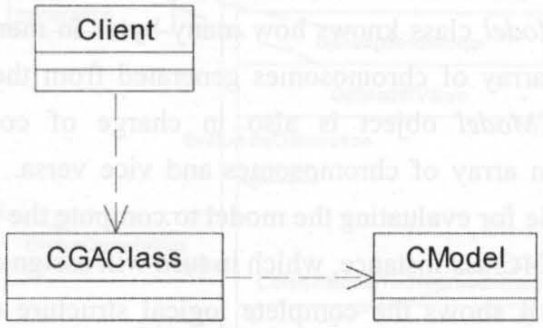

Fig. 3:GA code logical structure

3.2 Genetic Algorithm Code Structure

The GA was developed using object-oriented programming. One of the non-functional requirements for this project was to make the GA code as reusable and modifiable as possible. To achieve this end, the entire GA code was separated into two main classes: the genetic algorithm code itself, and the model. The GA code deals only with the model throughout its interface, whereas the model does not deal directly with the GA. Therefore, to replace the original model with a completely new

model is easy, and the GA will work fine; as long as the new model adheres to the interface required by the GA. Figure 3 illustrates the logical structure of the GA code using the UML notation (Rumbaugh, 1999).

The GA code is contained within a class called *CGAClass*. The model is enclosed by the class *CModel*. The *CModel* class is made up of one static and one non-static part. The static part of the *CModel* class contains the invariants of the class. For instance, the set of all H matrices is the same for all instances of *CModel*. The non-static part of *CModel* is composed of the attributes that are object-specific; for example, the vector s and the resulting value of the model. A *CGAClass* instance is responsible for initializing and deleting the static part of *CModel*. A *CGAClass* object is also in charge of coordinating the creation of an initial population as well as of evolving the initial generation to create new generations. Another task of a *CGAClass* instance is assigning a fitness value to each individual of each generation. A *CGAClass* object must implement all the GA elements described previously, in addition to methods to initialize and delete the static part of the *CModel* class.

The *CModel* class is responsible for allocating and deallocating memory for the chromosome arrays that a *CGAClass* object uses during generation evolving; because only the *CModel* class knows how many bytes in memory are required to allocate a complete array of chromosomes generated from the variant part of the *CModel* class. A *CModel* object is also in charge of converting the model representation into an array of chromosomes and vice versa. Finally, the *CModel* objects are responsible for evaluating the model to compute the model value that will be evaluated by a *CGAClass* instance, which in turn will assign a fitness value to that model value. Figure 4 shows the complete logical structure of the model implementation using UML notation. The asterisks denote zero to many relations. The diamonds are utilized to express aggregation relations, where the class at the end of the diamond contains the class at the other side. The triangle denotes a specialization

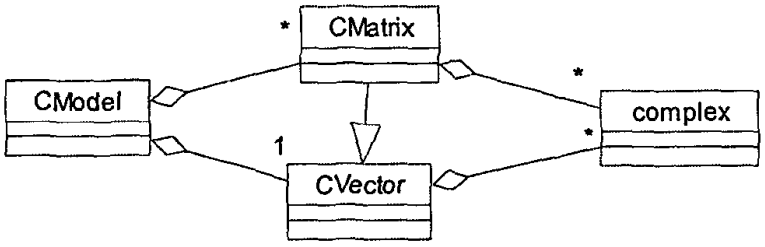

Fig. 4: Model logical structure.

relation, where the class at the end of the triangle is a specialization of the class at the other end.

On the client's side, a client may be an application, window, dialog, system, etc. The client is responsible to initialize the parameters of the GA, ask the GA to create the initial generation and evolve the present generation, and process the results from the GA for every generation. Figure 5 illustrates the relationship among the client, the *CGAClass* objects, the *CModel* objects and the *CModel* class by means of a sequence diagram.

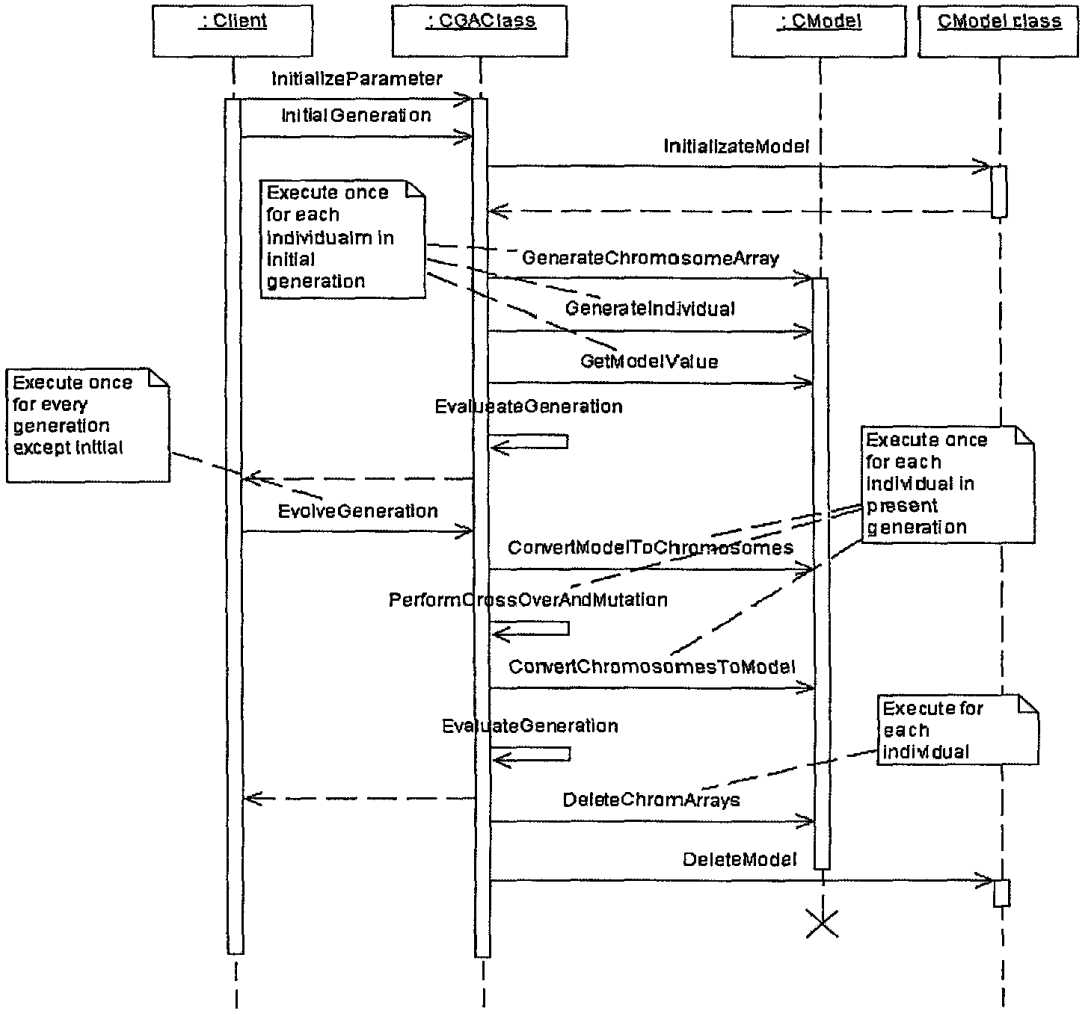

Fig. 5: Sequence diagram of the relationships between the client, CGAClass objects, CModel objects, and CModel class.

3.3 Implementation Issues

Three implementation issues were discovered during the experimentation and testing of the GA code. Because these issues are important and could be encountered by others performing similar work, they are described in this paper. The first issue was caused by changing the binary representation of the basic data type used to represent the complex numbers directly when calling the crossover or mutation operations. The second issue was caused by the limited range of this basic data type. The last issue was caused by the restrictions of the model. In this section, these issues and their resolutions are presented.

3.3.1. *Changing the binary representation of the basic type.* To represent a complex number, the number has to be associated with a programming language's basic type; one for the real part and one for the imaginary part. The basic type that was used in this project is the double precision floating-point type or *double* type. Microsoft® Visual C++ uses the IEEE-754 standard (IEEE, 1985) to represent floating-point type. This standard includes some singular values such as NaN (Not a Number) QNaN (Quiet NaN) and Indeterminate. Since the GA can alter any bit in the binary representation of a number, it is possible that the resulting value after converting the binary representation of a number back to the representation of model be one of these singular values. This issue was solved by eliminating the individuals that contained such singular values after converting them into model representation.

3.3.2. *The range of the basic type.* One of the operations a vector must provide to implement the model is magnitude calculation. The magnitude operation returns a double value; and the magnitude of a vector is used to normalize a vector and to compute the value of the model. Even though a *double* type (the largest floating-point number in C++) was used to compute the magnitude of a vector, some times during experimentation, the limits of a double type were reached. Every time this limit was exceeded, the magnitude method returned infinity (either negative or positive infinity), causing the elements of a vector to become indeterminate during the normalization process, resulting in a non-converging run of the algorithm. Therefore, individuals that yielded an infinite magnitude had to be eliminated.

3.3.3 *The restrictions of the model.* One restriction of the model is that every vector s must be a unity vector. In other words, the result of computing the product of the conjugate transpose of the vector and the vector itself must be equal to 1. The primary goal of the GA presented in this paper is to compute a vector s that minimizes the correlation between certain matrices. Due to the randomness of the

crossover operator, obtaining vectors that meet the unity restriction is virtually impossible. This was accomplished by normalizing vector s after transforming the chromosomes array back into the model.

3.4 GA Experiments

Two series of experiments were performed, corresponding to the two criteria described earlier. Each series comprised 35 runs, with 100 generations per run and a population size of 5000 per generation. The input included 10 H matrices with 100 rows and 8 columns each. To comply with the matrix multiplication operation, the s vectors had to be row vectors of 8 elements each. This vector cardinality resulted in chromosome arrays of 1024 chromosomes (or bits); provided that a *double* type size for Microsoft Visual C++ v6.0 is 8 bytes. The *CGAClass* was set up to minimize the value of the model with a minimum value of zero and a maximum value of 1. The probabilities of crossover and mutation were 85% and 0.1%, respectively. Each series of executions lasted an average time of 12 hours.

For every run, four files were generated: two text files, and two binary files. The first text contained all vectors generated per generation, and the fitness value and model value associated with each vector. The objective of this file was to keep a log of each program's execution for analysis purposes. The second text file included the average and maximum fitness values per generation of a run, and the generation identification number which was a consecutive number in the range of 1 to the total number of generations. This file was input to a spreadsheet processor to generate charts to validate the effectiveness of the GA.

The first binary file contained all vectors generated during a run and their associated model value. This file was used for selecting the best individuals among the entire population. Although the second binary file had the same structure as the first one, its objective was somewhat different. This file can be seen as an historical file that retained the best individuals generated during a series of run. To select the best individuals, the content of the two files were compared. This process involved merging of the two binary files, resulting in a binary file containing an ordered subset of the total elements after the merge. Once created, this historical file was input to the next run of the GA so that the initial generation of a run did not have to be completely random. Moreover, this approach ensured that the best individuals found during a run would be at least as good as the best individuals of the previous

execution. By doing this, obtaining much better individuals in a run was possible, rather than when the initial population of a run was generated randomly.

3.5 Final Outcome

After a series of executions were performed, the historical binary file described in the prior section was converted into a MATLAB file format containing only s vectors in the file. Each MATLAB file (used for validation of the models operations) included a matrix whose number of rows was the number of elements in each vector s, and the number of columns was the number of vectors in the historical file. Therefore, each column of the matrix was a vector of complex numbers representing a complete vector s. The first column in the matrix had the best vector s generated during a series of runs.

4. RESULTS

This section presents the results of experiments, beginning with assessing the effectiveness of the GA by showing the progress of the fitness values with different processes to create the initial generation of the GA. Then, several of the results obtained by the GA are compared with those obtained using an algorithm based on the methodology described in (Chung Lin, 2000). In successive references this is called the formal algorithm. The development of such an algorithm is in progress.

4.1 The Effectiveness of GA

To assess the effectiveness of the GA, we analyzed the results, as shown in Figures 6, 7, and 8. The figures illustrate the typical progress of the maximum fitness values and average fitness values during a series runs of 100 generations with 1000 individuals per generation, using three different methods for generating the initial generation. The probabilities of crossover and mutation were set to 85% and 0.1%, respectively.

Figure 6 was generated with data obtained from a run when the entire initial generation is random. The maximum fitness value (MFV), shown by the solid line, exhibits very good levels from the first generation, meaning that every generation has at least one strong individual with a very good fitness value. The average fitness value (AFV), shown by the dashed line, exhibits a good performance as well. The AFV has an upward trend, i.e., in every GA cycle, the next generation was steadily improved.

Figure 7 was generated from the information extracted from a run where some individuals of the initial generation are the best individuals from the previous run. In other word, this instance exemplifies the progress of the fitness when the initial generation is partially random. As it can be seen, both the MFV line and the AFV line are moderately shifted up. The quality of every generation in this run was better that the one of the previous run.

Figure 8 illustrates the case when the initial population is entirely constituted by the best individuals of the previous run. The MFV line shows a constant trend and was slightly shifted up. In contrast, the AFV line goes down during the first generations and then it remains approximately constant. Although it seems that the overall quality of the AFV is lowered, the level of the AFV was improved from the one in Figure 7. The drop in the first generations is explained by the quality of the initial generation. Since the initial generation was generated from the best individuals from the previous run, the overall quality of this generation stronger than the one of the following generations. The general quality of the AFV in this run was better that the one of the previous run. We concluded that the GA developed during this project yielded good results.

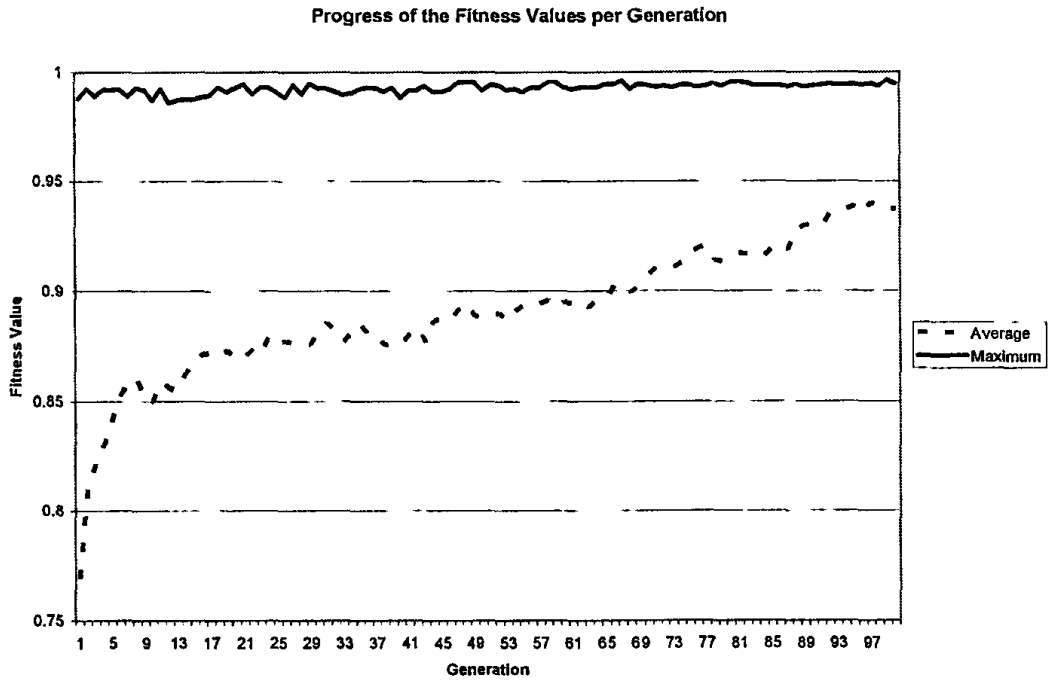

Fig. 6: The fitness value when the initial generation is totally random.

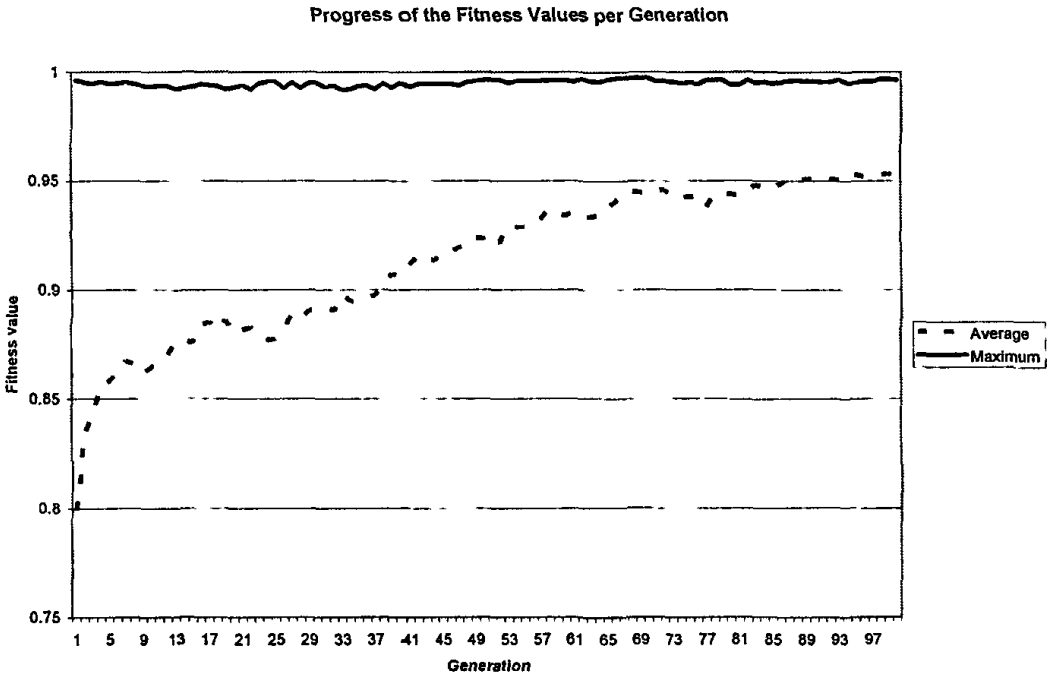

Fig. 7: Progress of the fitness value when the initial generation is partially random.

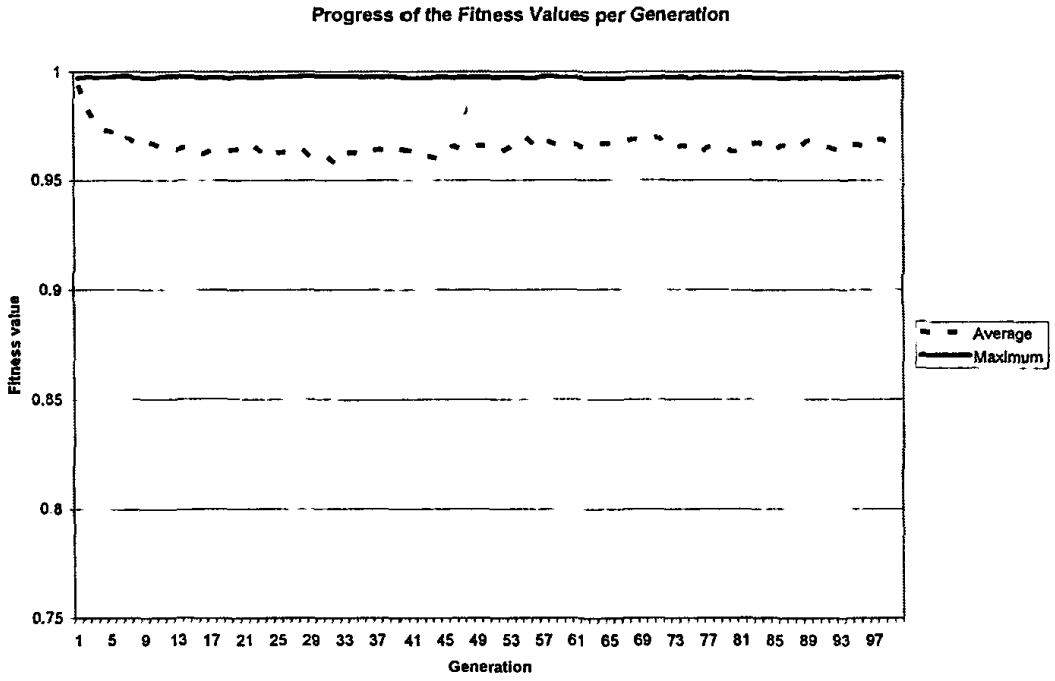

Fig. 8: Progress of the fitness value when the initial generation is generated from the best individuals of the previous run.

TABLE 1

GA And Formal Algorithm Results

Vector s from GA		Vector s from model	
-0.0272 - 0.1572i	α = 0.0028	-0.0805 - 0.1586i	α = 0.0313
-0.5866 - 0.1316i	β = 0.0011	-0.0057 + 0.0395i	β = 0.0117
0.1363 + 0.4317i		0.0102 + 0.0621i	
0.5844 - 0.0632i		0.1319 - 0.0059i	
0.1244 + 0.087i		0.0317 - 0.3372i	
-0.0346 - 0.1554i		-0.5033 + 0.0876i	
-0.0916 - 0.0446i		0.6683 + 0.0000i	
0.0021 + 0.0625i		-0.1514 + 0.3164i	

4.2 Analysis of the Results

In this section, the results obtained by the GA are compared with the results obtained by the formal algorithm. Table 1 shows the best vector s found by the GA compared with the best as computed by the formal algorithm. From Table 1 shows that the values for both α and β from the GA were smaller that those computed by the formal algorithm. Noteworthy is that the objective is to minimize these vales. For instance, both GA's α and β are approximately 11 times smaller than the values computed by the formal algorithm. Although better values were found by the GA, the GA method should not be used as a replacement of the formal algorithm for two reasons: (a) a GA is not suited for real-time applications because of time issues, and (b) there is no guarantee that the same value will be found after every run (or series of executions). For instance, the GA found the vector s presented in Table 1 during a series of runs that lasted more than 20 hours, where as the formal algorithm took a minute to produce its output. Therefore, the values obtained by the GA must be used as their original objective: to serve as upper bounds.

Consequently, more research is needed to improve the formal algorithm outcomes and integrate the strengths of the GA method with the formal method.

5. CONCLUSIONS

The objective of this project was to find an upper bound to validate the formal algorithm based on Chung Lin (2000) for the transmit code problem of Synthetic Aperture Radar. As shown, the upper bounds found for both criteria α and β were approximately 11 times smaller than the actual values found by the formal algorithm. As a result, we concluded that more research is needed to improve the formal algorithm to obtain at least the same results as those obtained from the GA.

The contributions of this project include finding an upper bound to validate the existing formal algorithms and developing a set of reusable and validated classes that can be used as a tool for further research in application of GA to radars.

REFERENCES

Aydemir, M.E., Günel, T., Erer, I. and Kurnaz S. 2003. A novel approach for synthetic aperture radar image processing based on genetic algorithm. *Proceedings of International Conference on Recent Advances in Space Technologies RAST 2003* , November 2003, Istanbul, Turkey, 351-4.

Boyd, R.V. and Glass, C.E. 1993. Interpreting ground-penetrating radar images using object-oriented, neural, fuzzy, and genetic processing. *Proceedings of SPIE*, **1941**, 169-80.

Chambers, B., Anderson, A.P. and Mitchell, R.J. 1995. Application of genetic algorithms to the optimisation of adaptive antenna arrays and radar absorbers. In *Proceedings of the First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, Sheffield, UK, September 1995, 94-9.

Chung Lin, S. 2000. *Design of Space-Time Transmit Codes for Optimizing Multi-static Radar Performance.* Technical Report 18221-2, University of Kansas, Center for Research, Inc.

Daida, J.M., Hommes, J.D., Ross, S.J. and Vesecky, J.F. 1995. Extracting curvilinear features from synthetic aperture radar images of Arctic Ice: algorithm discovery using the genetic programming paradigm. *Proceedings of the 1995 International Geoscience and Remote Sensing Symposium: Quantitative Remote Sensing for Science and Applications*, July 1995, Firenze, Italy.

Filippidis, A., Jain, L.C. and Martin, N.M. 1999. Using genetic algorithms and neural networks for surface land mine detection. *IEEE Transactions on Signal Processing*, **47(1)**, 176-86.

Fitch, P.J. 1988. *Synthetic Aperture Radar*, New York, Springer-Verlag, Inc.

Goldberg, D.E. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning*, Reading, Massachusetts, Addison-Wesley Publishing Company, Inc.

Goodman, N.A. 1995. *SAR and MTI Processing of Sparse Satellite Clusters*. M.S. Thesis, Department of Electrical Engineering and Computer Science, University of Kansas.

Hughes, E.J. 1998. *Radar Cross Section Modelling using Genetic Algorithms*. PhD thesis, Department of Aerospace, Power, & Sensors, Cranfield University, Royal Military College of Science, Shrivenham, UK.

Hughes, E.J., and Leyland, M. 2000. Using multiple genetic algorithms to generate radar point-scatterer models. *IEEE Transactions on Evolutionary Computation*, **4(2)**, 147-63.

IEEE. 1985. ANSI/IEEE Standard 754, *IEEE Standard for Binary Floating-Point Arithmetic*, IEEE, New York.

Li, J. and Ling, H. 2003. Use of genetic algorithms in ISAR imaging of targets with higher order motions. *IEEE Transactions on Aerospace Electronic Systems*, **39(1)**, 343-51.

MathWorks 1999. *MATLAB*. The MathWorks, Inc.

Michielssen, E., Sajer, J.-M., Ranjithan, S. and Mittra, R. 1993. Design of lightweight, broad-band microwave absorbers using genetic algorithms. *IEEE Transactions on Microwave Theory and Techniques*, **41(6)**, 1024-31.

Porsani, M.J., Stoffa, P.L., Sen, M.K. and Chunduru, R.K. 2001. Fitness functions, genetic algorithms and hybrid optimization in seismic waveform inversion. *Journal of Seismic Exploration*, **9**, 143-64.

Qian, J., Wang, X., Wu, R. and Pei M. 2001. The multi-zone scheme for designing radar-absorbing materials using GA. *Proceedings of the Genetic and Evolutionary Computation Conference Late-Breaking Papers*, San Francisco, CA, 347-51.

Rumbaugh, J., Jacobson, I. and Booch, G. 1999. *The Unified Modeling Language Reference Manual,* Addison-Wesley, 1999.

Sanchez, J.L.M. 1998. *Radar Waveform Design using Genetic Algorithms*. Dissertation, The University of Texas at El Paso.

Sarabandi, K. and Li, E.S. 1997. Characterization of optimum polarization for multiple target discrimination using genetic algorithms. *IEEE Transactions on Antennas and Propagation*, **45(12)**, 1810-7.

Stanhope, S.A. and Daida, J.M. 1998. Genetic programming for automatic target classification and recognition in synthetic aperture radar. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, San Mateo, CA, Morgan Kaufmann, 303-9.

Toomay, J.C. 1989. *Radar Principles for the Non-Specialist*, New York,Van Nostrand Reinhold.

Villegas, F.J., Cwik, T., Rahmat-Samii, Y. and Manteghi, M. 2004. A parallel electromagnetic genetic-algorithm optimization EGO application for patch antenna design. *IEEE Transactions on Antennas and Propagation*, **52(9).**

Weile, D.S. and Michielssen, E. 2001. The control of adaptive antenna arrays with genetic algorithms using dominance and diploidy. *IEEE Transactions on Antennas and Propagation*, **49(10)**, 1424-33.

Yilmaz, A.S., McQuay, B.N., Wu, A.S. and Sciortino, J.C. 2003. Evolving sensor suites for enemy radar detection. *Proceedings of the Genetic and Evolutionary Computation Conference*, Chicago, IL, Springer-Verlag.