Modeling and Simulation Optimization Using Evolutionary Computation

Edwin Núñez
Paul Agarwal
COLSA Corporation
6726 Odyssey Drive
Huntsville, AL 35806
256-964-5295, 256-964-5355
enunez@colsa.com, pagarwal@colsa.com

Marshall McBride
Ron Liedel
Claudette Owens
U.S. Army Space & Missile Defense Command (USASMDC)
Huntsville, AL 35898
marshall.mcbride@us.army.mil, ronald.liedel@us.army.mil, claudette.owens@us.army.mil

Keywords:
Evolutionary computation, genetic programming, software tools, optimization techniques, modeling and simulation

ABSTRACT: *Evolutionary computation (EC) is a general term applied to a group of global optimization techniques whose main characteristics are inspired by biological evolution. Instead of working with one possible solution at a time, they usually start with a population of random solutions. The initial population evolves into a better set of solutions through three main processes: selection, recombination, and mutation. Those solutions having greater fitness are preferentially selected for recombination to produce a new set of possible solutions. Mutation is also used to maintain diversity within the newly created solutions. Through these processes, the fittest solutions transfer their characteristics to a new generation of solutions. By iterating over many generations, EC can find solutions to many complex problems.*

Models and simulations, especially those working as a federation, can serve as the fitness function to determine the value or adequacy of particular solutions. In such federations, genetic programming (GP) or other EC techniques can quickly find optimal or near-optimal solutions for particular problems or situations. The user is not required to systematically search for the optimal solution; the computer accomplishes that task. The tradeoff for accepting this advantage is the requirement for the use of high performance computing resources.

In this paper we briefly describe the fundamental characteristics of EC. We also show some of the results obtained during our research and development efforts on different problems like image noise reduction and discrimination of buried unexploded ordnance. We also provide examples of how EC can be used with models and simulations to find optimum solutions to many complicated problems. This technique has great potential for use with models and simulations in a federated environment. The modeling and simulation community needs to become more aware of these powerful EC techniques so they may be applied in a wide range of fields to quickly provide solutions to the warfighter.

1. Introduction

For some fifty years now, computer scientists have attempted to find ways to teach computers how to solve simple programs in an automated fashion. Friedberg and some collaborators conducted efforts in this direction initially. [1], [2] Artificial intelligence techniques were used in the ensuing decades with limited results. For "real world" problems the solution space is generally extremely complicated. In addition, these spaces are neither continuous nor differentiable. Classical optimization techniques are not applicable in such cases. [3]

In the last couple of decades, emphasis has shifted to evolutionary computation (EC) techniques. They can solve complex problems in an automated fashion. There are many variants of EC, although they share certain fundamental characteristics. They differ mainly in the particular techniques used to pose and solve the problem under investigation. Amongst these different EC fields we find genetic algorithms (GA), evolutionary strategies, immune systems, grammatical evolution, genetic programming (GP), gene expression programming (GEP) and many others. Succinct explanations of many of these techniques and their main characteristics are available in the books of Sait and Youssef [4], and by Glover and Kochenberg [5].

All of these techniques are inspired in evolutionary biology. John Holland of the University of Michigan was inspired by evolution's capability of solving very complex natural problems. He decided to explore the use of evolutionary principles as applied to computation. [6] As the field has matured, optimized solutions to problems of increasing complexity have been found using these techniques. Because of its roots in evolutionary biology, the terminology used in EC is borrowed from biological terms. Consequently, terms like gene, chromosome, mutation, fitness, and others are commonly used. [7]

In this paper we will expand on the work presented during the High Performance Computing Modernization Program, Users Group Conference 2005 [8]. We will provide a brief introduction to EC. Examples of these computational methods will help to elucidate the techniques used. The examples used will also show how models and simulations can be used along with EC as part of the search for optimized solutions to complex problems. In such cases, the availability of High Performance computing (HPC) resources is generally indispensable. In addition, a discussion of the special needs of users applying evolutionary computation will be presented. Possible efficient computational architectures for exploiting the HPC resources will also be offered.

2. General Characteristics of EC

Evolutionary computation is a global optimization technique that is modeled after the processes of evolutionary biology. EC techniques do not work with one solution at a time as the techniques we are most familiar with do. Instead, EC works with a population of solutions, i.e., it works with a large number of solutions at the same time. It starts with a population of random potential solutions and evolves better ones over many successive generations. Selection pressure causes each generation of solutions to improve over previous ones. To avoid getting trapped at a local optimum, the operations of recombination and mutation are applied so different regions of the solutions space are explored.

Evolutionary computing techniques are broadly applicable to a wide variety of problems. They have been used in areas as diverse as digital circuit filter design [9], Wireless LAN antenna design [10], sonic boom reduction in supersonic aircraft [11], data mining [12], quantum computer circuit design [13], and non-linear programming [14], to name only a few. In our work we have also used it to reduce image random noise, in the discrimination of buried unexploded ordnance (UXO), in determining the minimum-time-to-target for a simple missile, and in modeling the Bi-Directional Reflectance Function (BRDF) for different surfaces at lidar wavelengths. Our work and this paper are mostly concentrated on genetic programming (GP) and in particular, are inspired by a sub-field thereof known as gene expression programming (GEP). [15]

3. Requirements for Applying EC

Before using EC we must first ask which problems lend themselves to analysis with this technique. Several simple requirements must be met before a problem is considered to be a good candidate for EC techniques. We must determine

how we are going to encode the problem, what is going to be used as the fitness function and what are the computational resources available.

1. *Encoding*. The researcher must elaborate a concise representation for potential solutions to the problem. Solutions are often encoded in the form of a linear string of mathematical symbols (operators) or features that is called a *chromosome*.
2. *Fitness function*. The researcher must find a way to determine the merit of potential solutions. This is expressed as a *fitness function*. It allows one to compare the merits of any two members of the population of potential solutions. Selection of chromosomes for later reproduction can then be based on this fitness value.
3. *Computer resources*. Sufficient computational resources must exist to evaluate the fitness function over the entire population of potential solutions and over many consecutive generations.

The first condition requires a careful study of the problem by the researcher along with some of her creativity and ingenuity. The latter are used to develop the appropriate way to encode the chromosomes and to generate operators that prove fruitful. The second requirement involves an understanding of the problem's nuances. Additional creativity is essential to elaborate an adequate fitness function that properly evaluates the merits of each specific chromosome (solution). In addition, the evaluation of the fitness function generally requires very large computational resources. The fitness function may be linked to highly complex models or simulations that need to be evaluated to determine the quality of the evolved solution. In general, most of the computational time is spent evaluating the fitness function. This is particularly true when it involves the evaluation of or more models or simulations.

4. Chromosomes, Fitness Function, Recombination, and Mutation

To solve a problem using EC, we must find a way to express possible problem solutions in the form of a chromosome. We will show how this can be done by use of a simple example: the traveling-salesperson problem (TSP). The concise representation of a chromosome could simply be a string with a list of the cities to be visited in the order in which they should be visited. This would be a string of the form "ABCD…" where the letters could represent the cities of Atlanta, Boston, Chicago, Denver, etc. The string could then be interpreted as a path starting at city A and eventually returning from the last to the first city. In this case, every string permutation is a potential solution since it represents the ordered path between the different cities. The fitness function would then simply be the sum of the inter-city distances. The lower the fitness function value, the better the potential solution, making it more likely to survive to the next generation.

Figure 1 illustrates how recombination between two chromosomes is accomplished. The chromosomes of two parents are combined to create the chromosomes of the next generation. These new chromosomes then represent the "offspring" of the previous generation. Two parental chromosomes designating different TSP paths are combined to produce the chromosomes of the next generation. These offspring give new paths between the cities. The parental chromosomes are recombined between positions 6 and 7. The resultant offspring chromosomes have the head sequence of one parent and the tail sequence of the other. For TSP, care must be taken to avoid duplicate cities in the path, a characteristic of such permutation problems.

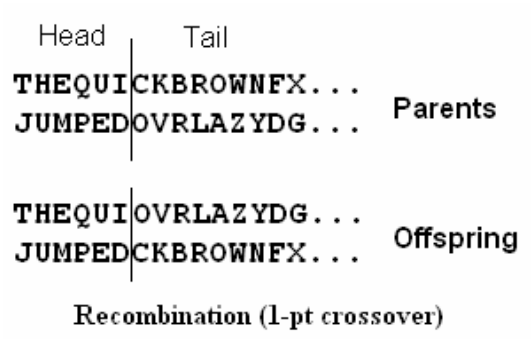

Figure 1 Chromosome mating using one-point crossover at randomly selected position.

The process of mutation can also be illustrated using figure 1. For these chromosomes, we would have one form of mutation if in the first offspring chromosome we randomly selected a position—say at R— and exchanged the letter sequence from RL to LR. Selection pressure is

achieved by giving the "most fit" parental chromosomes, i.e., those with shorter paths, a greater probability of becoming the parents for the next generation. In this manner the population moves towards better solutions.

Figure 2 illustrates this TSP example using 84 cities. Certainly genetic programming is *not* the best method for optimizing a TSP problem. This example is merely shown to illustrate the processes used in genetic computations.

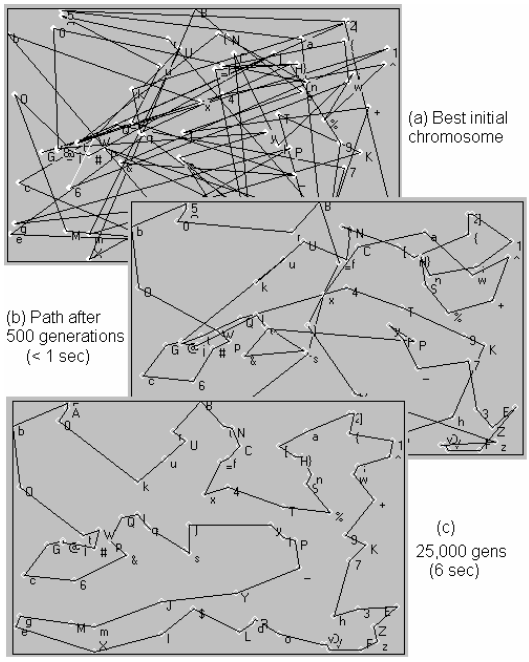

Figure 2 TSP problem. (a) Best random path (chromosome) from initial population (b) Best path after 500 generations (c) Best path after evolution through 25,000 generations (only 6 seconds of evolution for this simple case).

Care is taken to adjust the selection pressure to match the computational resource budget. High selection pressure causes improvements to spread more rapidly through the population, but at the expense of population diversity, thus possibly missing a global optimum solution. The phrase "exploration versus exploitation" is often used to describe this trade-off. When computational resources are adequate, selection pressure can be quite low.

The reader is referred to a great and rapidly growing body of literature on EC and GP for history, more techniques, theory, and examples. In particular, looking at the papers presented at the proceedings of the Genetic and Evolutionary Computation Conferences (GECCO) [16] [17] will show the wide range of topics covered (*www.isgec.org*). These describe a great variety of operations that we cannot explain here due to space limitations. They expand and improve upon the basic algorithms, and include elitism, niching, mass extinction, embedded parameters, and numerous sorts of types of selection, mutation, and recombination.

5. Work at the ARC

5.1 ARCGPL

The Advanced Research Center Genetic Programming Lab (ARCGPL) is a full-featured evolutionary computing lab that can run stand-alone (using an internal evolver) or with any number of external evolvers (see figure 3). It supports all evolutionary computing paradigms including GP, GA, finite state machine (FSM), permutation, and, through extensions, any other EC type. A wizard facilitates encoding the problem under investigation. The ARCGPL has almost 150 parameters settable through the GUI dealing with selection, recombination, mutation, population control, parsimony, logging, etc. It provides help for every panel and has a built-in tutorial and several built-in demonstrations. Classic features such as niching, mass extinction, elitism, scheduling, etc., are implemented. Commands may be issued through the GUI, by command line, or by script. Preferences can automate initialization at startup to save time. Run progress can be monitored and controlled through the GUI or by script. Statistical information is provided to the user and can be managed through the GUI.

5.2 GP and buried unexploded ordnance

Expression encoding is stressed herein because it is a part of so many types of genetic programming. For example, the authors have used arithmetic expressions that are a function of sensor readings obtained for buried unexploded ordnance (UXO) to discriminate ordnance from non-ordnance. The resulting value for the expressions is compared with a threshold to decide whether the buried anomaly represents ordnance or perhaps a discarded automobile radiator. Since the dig cost is enormous, such discrimination ability can be quite valuable. The

authors recently presented a paper on this buried UXO problem at this year's Genetic and Evolutionary Computing Conference (GECCO 2005, also known as ACM SigEVO) [18]. We have also evolved filters for removing noise from images and duplicated previously reported results for a combustion problem.

6. Interaction

When doing evolutionary computation there are a large number of parameters that must be controlled by the user. It is quite important for people doing evolutionary computation to have ready access to their computations. They must frequently inspect their computations to see how their solutions are evolving. According to how well they are performing the user might change some of the parameters to achieve better results according to their judgment. Some of these parameters are: rates of the different types of mutation and recombination, frequency of population migration, mass extinction rates, and many others. This need for increased user interactivity means they will be better served by a computation center that not only has high performance computing resources, but also can accommodate them and where they can be present to quickly determine what is happening and to make necessary changes. The Space and Missile Defense Command's Advanced Research Center (ARC) in Huntsville, Alabama, is well prepared for this kind of interactivity with the HPC user.

7. HPC and EC

The third requirement, namely computer resources, is apparent in many, if not most good GP candidate problems. For example, Moore [16] describes a missile counter-measure problem where a strategy was evolved to evade an anti-aircraft missile. In this problem, a complex strategy of countermeasures optimization is investigated. A jet attempts to evade a missile through the use of several countermeasures: maneuvers, flares, and jamming of the missile's radar. Classical evasion techniques required the airplane to maneuver in a specific way when the missile reached a certain distance from the airplane. The chromosomes developed by Moore consisted of the sequence of countermeasures taken by the jet airplane that took into account uncertainties about the type and current state of the pursuing missile. Evaluation of each chromosome's fitness required running a simulation of both the attacking missile with its sensors and the airplane in flight as it maneuvered and performed the indicated sequence of countermeasures. Running these two full simulations concurrently to evaluate each chromosome representing a possible solution required a large computational resource. As a result, the researcher could test only a limited set of sequences before exhausting the number of CPU hours allocated. Complicating the results was the fact that this paucity of computational resources severely restricted the population of possible solutions examined. The population of chromosomes was kept unnecessarily small (100 individuals in each generation) and the number of generations for evolution restricted to just a few dozen. Even with the restrictions imposed by the limited computational resources, the series of countermeasures developed through the use of GP attained substantially better survivability rates than those developed analytically.

There are many such important problems that could be investigated through the use of evolutionary computation but have not been explored due to the lack of powerful enough computers. It would be quite appropriate for the HPC community to understand the needs of this kind of user and to prepare for their increasing use of computational resources.

8.0 EC and Optimization of Models and Simulations

Frequently, we must run different models and simulations in a federation, where the behavior of one affects the others. This is often done to explore and evaluate different courses of action. If a particular set of actions is taken in the first model or simulation, it will yield results that preclude certain actions in the second model or simulation and require other responses. How do we decide on the best courses of actions when more than one model or simulation is used and each interacts with the others? Currently, the users do this, even when a very complex chain of events is contemplated. The user decides beforehand how to act under a particular set of circumstances or conditions. Experience and intuition serve as the main guides to decide on the best course of action or strategy for a simulation and how to respond when events or conditions do not match what is expected. After

the simulations run and produce results, the user studies the responses to establish what new strategies or courses of action must be developed in light of what actually happened. This process is difficult, not only because it may take extensive computer resources to run the simulations each time, but because the user must decide, often in light of a very large number of variables, how exactly they must respond to the results. Attaining an optimized strategy or set of actions under those conditions is close to impossible.

EC presents us with a new paradigm. Why not let the computer itself search for optimum or near-optimum sets of actions or strategies? When properly posed, GP or other particular EC techniques can search for the best strategies or behaviors to attain a particular objective. For this purpose, the user needs to search for a way to express simulation behaviors and interactions in the form of chromosomes. This may seem rather difficult at first, but with some research into the problem and experience working with EC, it turns out to be less complicated than expected. In the same manner, we also need to decide what are the desired outcomes and develop an adequate fitness function. This fitness function must evaluate the chromosomes so that solutions closer to the desired behavior yield better fitness. Such chromosomes will then have a greater probability of becoming the parents of the next generation. In this manner, an optimum or near optimum strategy can be found. The user then leaves the task of optimizing the search for the best set of actions and interactions between simulations to the computer. Their main function from then on would be to ensure there exist adequate HPC resources for the interacting simulations to work and for the evaluation of the chromosomes to take place.

This is merely an extension of the approach taken when EC is used in the design of antennas or in the optimization of other tasks. Instead of the user performing the bulk of the exploration of all possible solutions (generally an impossible task), this work is left to the computer. With adequate HPC resources, the modeling and simulation community could reach near-optimum solutions in reasonable time.

9. GP Architectures for HPC

The need for very large computational resources in many problems where GP is used has generated great interest in the parallelization of the computations. Amongst the different ways in which the parallelization can take place there are two that seem to be the most used. In the *chromosome server* approach, each node evaluates one possible solution chromosome independently. In the *population server* approach, each node can evaluate one or more populations of chromosomes. In that case, several populations are evaluated simultaneously. Each node then is equivalent to an island-like separated environment where evolution takes place. In our research we have used HPC resources (Beowulf clusters) existing at the ARC in Huntsville, AL.

Some of the most interesting results come from use of the population server approach, where each node serves one or more independent populations. In that approach, chromosome evolution is similar to what happens in an island where the local population is isolated from other populations since evolution takes place in an isolated environment. If we occasionally allow the export of the best members of one solution population to another population and have them compete with the chromosomes already existing there, it is as if you had migration from one isolated environment to another. Another effect permitted by this second approach is termed mass extinction. Intermittently, you may eliminate the population with the worst individuals and seed that population with a fresh set of chromosomes.

Which approach to use may depend on the relative time needed for fitness evaluation compared to the overhead time for selection, recombination, and mutation.

Figure 3 shows the architecture we used at the ARC. Each of the external processes represents a separate set of solution populations.

Our approach to cluster utilization for GP problems involves a proxy on the head node to serve populations of chromosomes to the working nodes and gather new best results from each node. Although our current user interface is implemented only for Microsoft Windows, the computation nodes are written in portable code and can run on Beowulf (Linux) clusters as well

as grids of Windows machines, or with a mixture thereof. The purpose of the proxy is to mediate access between the working nodes and the user interface since cluster policy often dictates isolating the working nodes in an internal net.

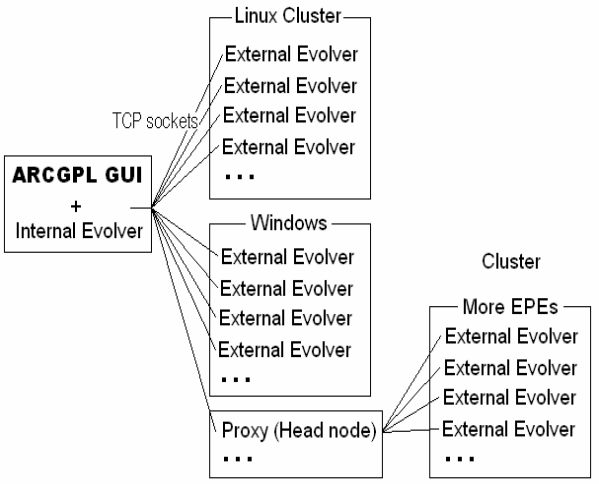

Figure 3. External parallel evolver architecture.

Every GP problem is different and requires custom coding to solve it. Rather than develop a special language, we simply assume the power of the C++ programming language. The programming burden is quite low, however. Through a wizard that prompts for essential item names, a shell evaluator we call an external fitness evaluator (or EFE) is created. A programmer then simply fills in the stubbed portions. Only two stubs require implementation: the first is the fitness function. It is called with a chromosome and returns a fitness value that allows comparison with other chromosomes. The second function simply names the operators that are to be used. The wizard-generated code is portable to either Linux or Windows and is delivered as a Windows DLL or Linux shared object file. The graphical user interface allows the user to specify the DLL or shared object file to load.

This arrangement can be fast. In a combustion problem we examined, an initial solution was obtained in a short time once we understood the problem (less than one hour). However refinements and a full problem solution necessitated substantially longer time (several days). As indicated earlier, initial GP calculations require great interactivity with the computational process to modify several parameters that control the evolution process. This interaction is necessary to attain a more efficient evolution. Once a satisfactory set of parameters is obtained, the process can run with less interaction.

Other stubs created by the wizard allow access to the user interface for presentation of problem-specific data and gathering data from the user, printing results, handling events, managing interrupts, performing your own mutation and recombination steps, and more. In one image-processing problem we simply re-wrapped an existing image processing lab as an EFE and had near-immediate results for evolving an image noise-reduction filter.

10. Summary

Evolutionary Computing, coupled with HPC resources, is a robust optimization technique that is surely under-utilized and not as widely recognized as it should be. Many complex problems are well suited for the use of GP in the search for optimum or near-optimum solutions. This technique will prove very fruitful when coupled with the availability of large computational resources. When using GP, users will not only require HPC resources, but will be best served by facilities and environments that allow large interactivity with the computations to modify them as results are available. The computer itself can evolve optimized strategies and behaviors interacting models or simulations. Facilities like the ARC in Huntsville, AL, are already well organized to accomplish this task. The modeling and simulation community should look closely at evolutionary computations and be prepared to facilitate its use.

10. References

[1] Friedberg, R.: “A Learning Machine, Part I.” IBM J. Research and Development, Vol. 2, pp. 2-13, 1958.

[2] Friedberg, R., Dunham, B., and North, J.A.: “A Learning Machine, Part II.” IBM J. Research and Development, Vol. 3, pp. 282-287, 1959.

[3] Langdon, W.B., and Poli, R.: Foundations of Genetic Programming, Springer-Verlag, Heidelberg, 2002, p.3.

[4] Sait, S.M., and Youssef, H.: Iterative Computer algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems, IEEE Computer Society, Los Alamitos, CA, 1999.

[5] Glover, F., and Kochenberg, G.A. (editors): Handbook of Metaheuristics, Kluwer Academic Publishers, Boston, MA, 2003.

[6] Holland, J.: Adaptation in Natural and Artificial Systems, MIT Press, Cambridge, MA, 1975.

[7] Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning, Addison Wesley, Reading, MA, 1989.

[8] Núñez, E., Banks, E.R., Agarwal, P., McBride, M., and Liedel, R.: "High Performance Evolutionary Computation", Proceedings of the High Performance Computing Modernization Program, Users Group Conference 2005, June 27-30, Nashville, TN, p. 308-313.

[9] Koza, J., Bennett (III), F.H., Andre, D., and Keane, M.A.: Genetic Programming III: Darwinian Invention and Problem Solving, Morgan Kaufman, 1999.

[10] Sanderson, R.: "Automatic Synthesis of an 802.11a Wireless LAN Antenna Using Genetic Programming: A Real World Application", Genetic and Evolutionary Computation – GECCO 2004 Conference Proceedings, Part II, Springer. Seattle, WA, June 2004. , p. 1201-1213.

[11] Karr, C.L., Bowersox, R., and Singh V.: "Minimization of Sonic Boom on Supersonic Aircraft Using an Evolutionary Algorithm", Genetic and Evolutionary Computation – GECCO 2003 Conference Proceedings, Part II, Springer, Chicago, IL, July 2003, p. 2157-2167.

[12] Araujo, D.L.A., Lopes, H.S., and Freitas, A.A.: "A Parallel Genetic Algorithm for Rule Discovery in Large Databases", Proc. 1999 IEEE Systems, Man and Cybernetics Conf., Vol. III, Tokyo, Oct. 1999, p. 940-945.

[13] Barnum, H., Bernstein, H.J., and Spector, L.: "Quantum Circuits for OR and AND for Ors", Journal of Physics A: Mathematical and General, Vol. 33, 2000, p. 8047-8057.

[14] Koza, J.R., Mydlowec, W., Lanza, G., Yu, J., and Keane, M.A.: "Automatic Synthesis of Both the Topology and Sizing of Metabolic Pathways using Genetic Programming", Proceedings of the Genetic and Evolutionary Computation Conference, San Francisco. Morgan Kaufmann, July 7-11, 2001.

[15] Ferreira, C.: "Gene Expression Programming: A New Adaptive Algorithm for Solving Problems", Complex Systems, Vol. 13, 2001, p. 87-129.

[16] Banshaf, W., Daida, J., Eiben, A.E., Garzon, M., Honavar, V., Jakiela, M., and Smith, R.E. (editors): Proceedings of the Genetic and Evolutionary Computation Conference, Vols. 1 and 2, July 13-17, 1999, Orlando, FL, Morgan Kaufmann Publishers, San Francisco, CA.

[17] Deb, K., Poli, R., Banshaf, W., Beyer, H.G., Burke, E., Darwen, P., Dasgupta, D., Floreano, D., Foster, J., Harman, M., Holland, O., Lanzi, P.L., Spector, L., Tettamanzi, A., Tierens, D., and Tyrrell, A. (editors), Genetic and Evolutionary Computation – GECCO 2004, Proceedings, Parts I and II, June 26-30, 2004, Seattle, WA, Springer-Verlag, Berlin Heidelberg, Germany.

[18] Banks, E.R., Núñez, E., Agarwal, P., Owens, C., McBride, M., and Liedel R.: "Genetic Programming for Discrimination of Buried Unexploded Ordnance (UXO)" GECCO 2005, June 25-29, Washington, DC.

Author Biographies

DR. EDWIN NÚÑEZ is a Senior Scientist at COLSA Corporation where he has managed

projects in applying evolutionary computing and genetic programming techniques. He is a graduate of the University of Puerto Rico and Colorado State University. He is also a member of the American Association for Artificial Intelligence, International Society for Genetic and Evolutionary Computation, American Geophysical Union, American Meteorological Society, and the American Physical Society.

PAUL AGARWAL of COLSA Corporation is the Software Engineering Manager for the U.S. Army Space and Missile Defense Command USASMDC Advanced Research Center. He has over 20 years experience in the public and private sectors with computer software design, development, implementation, and evaluation, and in systems analysis. He is currently involved in program management.

MARSHALL McBRIDE is the government technical monitor responsible for the day-to-day operations of the U.S. Army Space and Missile Defense Command (USASMDC) Advanced Research Center, in Huntsville, AL. He has 15 years experience in research & development, modeling & simulation, program management, test & evaluation, and high performance computing center management. He is a graduate of Auburn University with a BS degree in Aerospace Engineering

RON LIEDEL is a senior engineer with the Future Warfighter Center of the U.S. Army Space and Missile Defense Command (USASMDC) in Huntsville, AL, and has authored numerous SMDC and MDA policy forming software documents. These documents include the SMDC Software Development Plan, the SMDBL 10 Year Software Plan, as well as the Command Software Mission and Function Statement. Mr. Liedel founded and chaired the SMDC/SDIO Computer Resource Working Group and has presented several papers nationally on Software Sizing for Mega Systems. He serves as the Technology Products and Services Director for SMDC.

DR. CLAUDETTE OWENS is Acting Chief of the Information and Computational Engineering Division of the Future Warfighter Center, U.S. Army Space and Missile Defense Command (USASMDC) in Huntsville, AL. Dr. Owens oversees the Advanced Research Center and the Simulation Center, and manages the DOD High Performance Computing Management Program (HPCMP) Shared Resource Center located at SMDC. She has over 20 years experience in basic R&D, engineering and simulation in government. Dr. Owens holds a Ph.D. and Masters in Physics from Alabama A&M University and BS degrees in Electrical Computer Engineering and Electrical Engineering Technology from the University of Alabama, Huntsville and Alabama A&M University, respectively.