AIAA-94-1256-CP

GA-OPTIMIZATION FOR RAPID PROTOTYPE SYSTEM DEMONSTRATION

Jinwoo Kim and Bernard P. Zeigler

Artificial Intelligence and Simulation Research Group
Department of Electrical and Computer Engineering
University of Arizona, Tucson, Arizona 85721
jwkim@helios.ece.arizona.edu

Abstract

This paper discusses an application of the *Genetic Algorithm*, a parallel and global search technique that emulates natural genetic operations. Real application problems often require optimization of a large number of parameters with high precision. Since the existing Genetic Algorithms do not represent the parameter sensitivities, we have devised a novel scheme of *Hierarchical Genetic Algorithm* to solve complicated engineering problems. Using this approach, the higher level GAs propose promising search spaces, while the lower level GAs search in more detail with additional parameter sets. This decreases the complexity of search and utilizes the computing resources efficiently. This scheme has been used to design an autonomous control systems for space-based resource processing plants.

1. Introduction

Applications of computer technology are expanding from pure data processing to information and knowledge processing which enables Computer-Aided System Design. Knowledge-based system applications are characterized by symbolic processing, nondeterministic computation, dynamic execution, high potential for parallel and distributed processing and knowledge management. However, fundamental physical limits of current technology have not been overcome for the more sophisticated computation-intensive problems, such as predictive modeling and forecasting, design automation, large scale, simulation and artificial intelligence. The combination of technology and economic factors make parallel and distributed computing systems attractive and effective for a large variety of intelligent machine applications [9].

The emergence of massively parallel computers has also fueled a growing interest in problem solving systems based on principles of evolution and heredity. One class of such *evolution strategies* is Genetic Algorithms [14]. The remarkable success demonstrated by Genetic Algorithms (GA) in search, optimization and learning has substantially increased interest in their potential application to modeling, simulation and design of complex real world systems [1, 6]. Such applications include identification and calibration in model construction and subsequent model-based control synthesis and policy optimization. However, complex simulations typically require large execution times to evaluate alternatives, or in GA terms, to obtain fitness values for newly generated chromosomal individuals. Such lengthy simulations present a major bottleneck to GA application since tens, or even hundreds, of individuals may need to be evaluated in every generation. Parallel processing offers promise of reducing this bottleneck along two mutually supporting avenues: 1) speeding up the simulation needed to estimate fitnesses using distributed simulation methods [5] and 2) parallelizing the evaluation and processing of fitness information. Both avenues are under active investigation and indeed a computer architecture to support their integration has been suggested [19].

Real application problems often require optimization of a large number of parameters with high precision. These parameters increase the complexity of the search problem. In existing approaches, a chromosome representing the parameters does not contain information about their sensitivity, even though parameters influence the system performance to different degrees.

We have developed a novel scheme of a Hierarchical GA optimizer which executes multiple GA modules to solve complicated problems. These GA modules are constructed hierarchically and creation/deletion is performed dynamically based on the performance of each module.

Each GA module deals with a different degree of abstracted models for evaluation and a different number of parameters for optimization. High level GA modules usually search for fewer parameters which are more sen-

sitive to the system performance. They are looking for milestones of promising search region instead of accurate solutions. The candidate individual selected from the high level GA module represents a sub search space and they are sent to the lower level GA modules for more detail search. The lower level GA takes advantage of the received information, and employs a greater number of parameters for further optimization.

The solutions found at the lower level GA module are reported back to the higher level GA module, if it is better than the candidate individual from parent module. This information is used to update the fitness of the parent. As the purpose of high level GA module is not to find actual solution, the models of this level are not necessarily accurate. In order to speed up GA search, the high level GA modules access less accurate models which can reduce simulation-based fitness evaluation time. The basic concept is that of successive approximation provided by a nested sequence of models [13].

2. Hierarchical Genetic Algorithms for Complex Problems

A simulation model of such a complex architecture is most naturally formulated as a variable structure model [21]. A Hierarchical GA is implemented on a self-organizing variable structure, where creation/deletion of modules are determined by their performance.

2.1 Brief Review of Asynchronous Genetic Algorithm

The GA (genetic algorithm) is a probabilistic algorithm which maintains a population of individuals, $P(t) = x_1(t), \ldots, x_n(t)$ for iteration t. Each individual represents a potential solution to the problem at hand, and, in any evolution program, is implemented as some (possibly complex) data structure S. Each solution $x_i(t)$ is evaluated to give some measure of its *fitness*. Then new population (iteration $t+1$) is formed by selecting the more fit individuals (select step). Some members of new population undergo transformation (recombine step) by means of "genetic" operators to form new solutions. There are unary transformation m_i (mutation type), which create new individuals by a small change in a single individual ($m : S \to S$) and higher order transformations c_j (crossover type), which create new individuals by combining parts from several (two or more) individuals [14]. The control parameters for genetic operators (probability of crossover and mutation) need to be carefully selected to achieve acceptable performance [15]. After some number of generations the program converges and is successful if the best individual represents the optimum solution.

We have developed concepts for parallel genetic algorithms that are especially oriented to simulation-

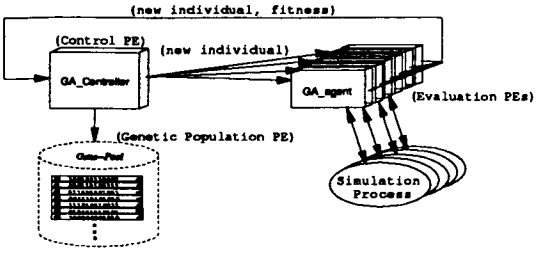

Figure 1: The Architecture for Asynchronous Genetic Algorithm Simulation

evaluated individuals on high performance computers. We have investigated a class of *Asynchronous Genetic Algorithms* (AGAs) which does not need to be synchronized by generations to create successive populations. In a multiprocessor architecture, individuals are evaluated concurrently and a central agent updates the genetic population continuously as the evaluation results become known. The motivation behind such asynchronous updating is the recognition that not only may simulation runs be time consuming, but their completion times may be highly variable. This variability is quite common in performance measuring simulations [1]. When such variability is significant, the barrier synchronization imposed by conventional GAs can greatly impede search progress since it requires that processing cannot proceed to the next generation until the slowest individual in the current population has completed its evaluation [7]. In contrast, the AGA allows new individuals to be tested as soon as both the information and the computer resources are available to do so.

A concern immediately raised in the AGA paradigm is that the blending of generations occasioned by such asynchronized processing might adversely affect the recombination schemes underlying GA search. Certainly, the supporting theory typically limits selection, mating, crossover and other operations to members of the same generation [6, 8]. Fortunately, some results in the literature suggest that search time and search success are not degraded, at least in application to typical test function suites [4]. We note that such artificial fitness functions do not include time dependence that might appropriately suggest the necessity for generational integrity.

As shown in figure 1, the processing elements (PE) in the asynchronous genetic algorithm can be categorized as : *genetic population PE, evaluation PEs* and *control PE*. While the PGA executes serial GAs in the

[1] It certainly occurs when runs are executed in conditional mode where termination depends on pre-established criteria (e.g., a run may be terminated as soon as failure to achieve a prescribed goal is obvious). However, run time variability may also arise when runs span a fixed observation interval (on the model time base). This may be due to the variability in workload encountered by the simulation engine or in the resources allocated to the particular trial

multiple processing elements with subpopulations, the asynchronous genetic algorithm evaluates a single individual in a processing elements (*evaluation PE*) at a time. The total population is always contained in the *genetic population PE* which executes genetic operations and updates the population. It also generates new individuals whenever it receives request from the *control PE*.

Message transfer between the *genetic population PE* and the *evaluation PE* is controlled by the *control PE* : it delivers evaluated individuals to the *genetic population PE* and requests new individuals for the *evaluation PE*. Thus the *evaluation PEs* keep evaluating individuals with which the *genetic population PE* continuously updates the population.

2.2 Resolution Increasing Scheme in the Hierarchical Genetic Algorithm

A binary chromosome, a unique knowledge representation scheme of Genetic Algorithms, provides a way of controlling the accuracy of parameters. The size of the binary code determines the number of points to be investigated. As more bits are employed, the search points increase dramatically (exponentially). Therefore, longer string size may provide accurate parameter values, but it also makes the GA to search through a large number of points.

As shown in figure 2, same size of binary code can increase accuracy as search space changes. At level 1, the original search space is defined by MIN and MAX. We employs 3 bits and there are 8 possible search points. If a certain point (binary code) is selected, the distance between its neighbors becomes a new sub search space. Therefore the selected candidate individual at the high level GA module has meaning as representation of its neighbors (sub search space) rather than an actual value itself. The same size of binary code is employed with the new search space, which increases the accuracy of the parameter value.

2.3 Expanding Search Parameters in the Hierarchical Genetic Algorithm

The previous section explains how search accuracy is controlled by the Hierarchical GA. If the search problems involve a large number of parameters, the GA takes longer or directs to local minima. The Hierarchical GA employs an expanding search parameter scheme. The higher level starts to search for a small number of parameters. The result obtained at the higher level is sent to the lower level GA, where extra parameters are included to the received parameters. The lower level GA takes advantage of the received information so that it need not search all the parameters from the beginning.

The expanding parameter scheme in the Hierarchical

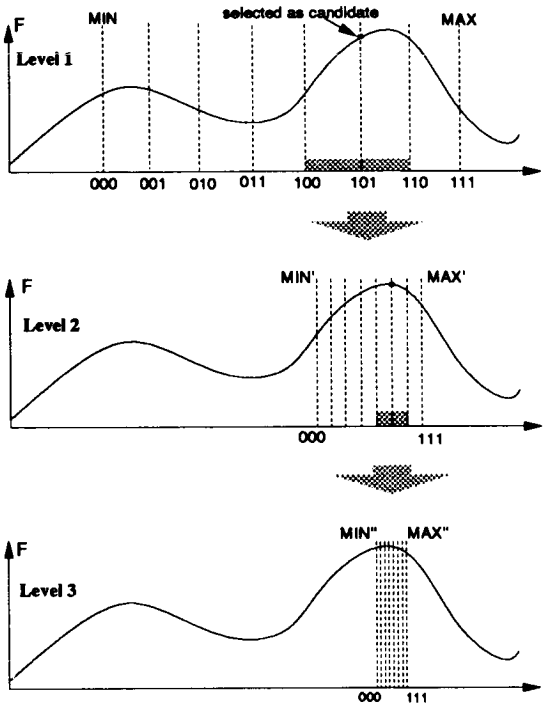

Figure 2: Resolution Control Scheme in Hierarchical Genetic Algorithms

GA also helps determine the optimal number of parameter sets. Complicated designing problems involve the evaluation of a large number of parameters, where the type of parameters as well as their appropriate values are often unknown.

2.4 Execution of Multiple GA Modules in Parallel

The selected individuals during the search operation from the root AGA create lower level GA modules as shown in figure 3. The time taken by the module for checking its population and selecting a candidate individual may be either deterministic or nondeterministic. In this experiment, we choose a variable selection interval scheme, in which the time intervals of subsequent checking become larger as the GA search continues. This strategy is based on a characteristic of GA search, the fitness of population increases faster in early search stages and saturates to a certain level. Smaller checking intervals enables the GA to choose new candidates before stagnation.

When a lower level AGA module selects an individual as a candidate for the next level, it also reports fitness to the parent AGA module. This feedback information updates the fitness of parent individual. But the individual structure of the parent and child may not be comparable to each other because they represent different parameter sets or different search space. The popu-

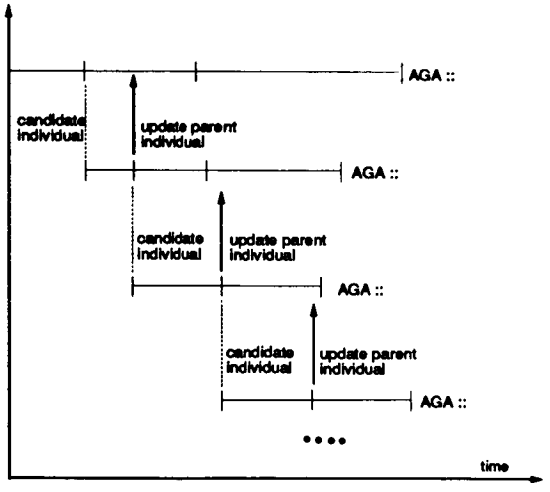

Figure 3: Creation and Execution of multiple AGA in Self-Organizing Hierarchical GA

lation of child AGA searches through the space which is suggested by a candidate individual from parent AGA. The fitness of the lower level individual, if better, updates the fitness of parent individual. If updated, the fitness of parent no longer represents an actual fitness of the binary code. But, since the fitness is increased by a certain amount, its value can not be represented in its environment. The individual now has a higher fitness and has a greater probability of being selected for the next evaluation.

Figure 4 shows the module components at each level. Each module contains a *controller* which executes *AGAs* to solve a given problem and select candidate individual(s) reported from AGAs. It creates/deletes lower level modules and communicates with higher/lower level modules. This *controller* is the intelligent component of module and is implemented by a rule-based expert system. The module is defined as a class in an object-oriented programming environment (Chez-Scheme [3]) and the same module structure is created by higher level object. This module is program-interfaced to an AGA procedure written in C. The decision making component is implemented in a symbolic processing language, while the numeric computation procedure is written in C.

When the module receives a problem, it may expand search parameters as well as increase resolution. With any given problem, the controller first expands parameters and sends them to the PARA-AGA for GA search. The selected individuals at the PARA-AGA are sent to HIGH-AGA to increase resolution by the scheme explained above. The fitness of the selected individual at the PARA-AGA are also reported to the high level HIGH-AGA to update the fitness of parent individual. The candidate individuals for the next level are selected at the HIGH-AGA where the selected individual is also

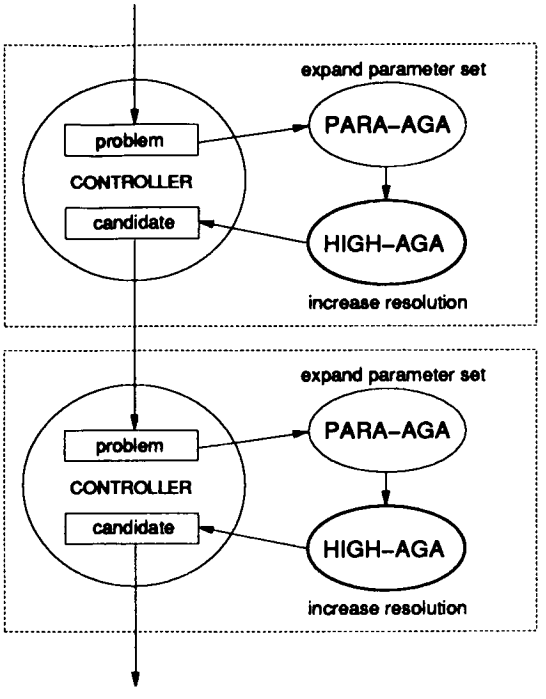

Figure 4: Module Components in the Self-Organizing Hierarchical GA

reported to the PARA-AGA of same module.

3. Design of Control System using Self-Organizing Hierarchical GA Environment

A working prototype of a plant for producing oxygen from Martian atmosphere, is constructed at NASA-UA Space Engineering Center [16]. The purpose is to evaluate the best designs and operation parameters for the Mars mission. Martian CO_2-rich atmosphere is filtered and compressed to a temperature and pressure suitable for electrocatalysis in a Zirconia-based oxygen cell. Design issues include the size of the inlet pipe, power requirements of the compressor and design of the oxygen cell including: cell configuration, material properties, electrical parameters such as operating voltage and current density, electrode materials, and method of application

In this experiment, we try design an optimal FLC to control the temperature of the oxygen production system. The basic idea of the fuzzy control centers around the *labeling* process, in which the reading of a sensor is translated into a label as done by human expert controllers [12]. With expert supplied membership functions for this labels, a reading of a sensor can be *fuzzified* and *defuzzified*. It is important to note that the transition between labels are not abrupt and a given reading might belong to several label region.

The fuzzification and defuzzification processing does not need to be sequential. The input signal can be

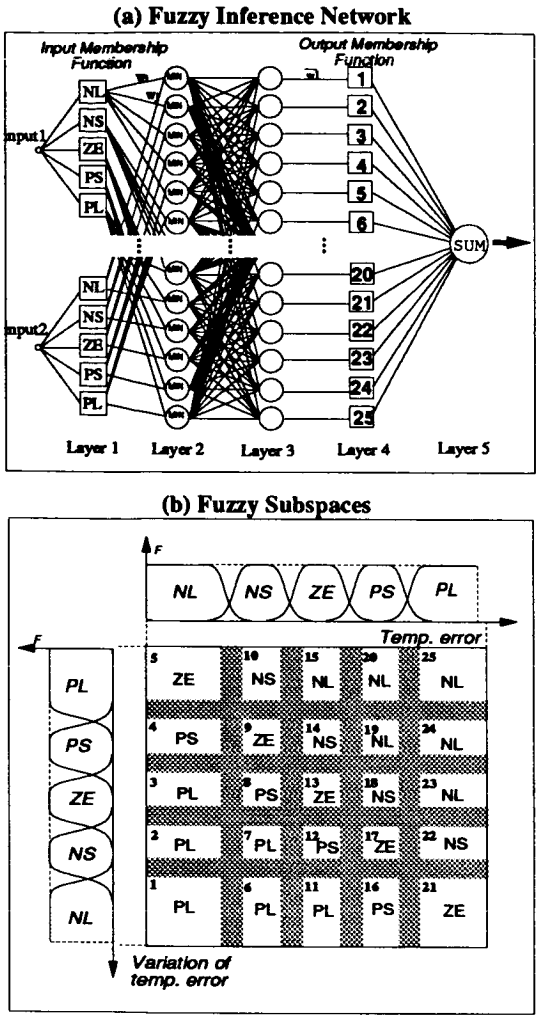

Figure 5: Fuzzy inference network and fuzzy subspaces: (PL:Positive Large) (PS:Positive Small) (ZE:Zero)(NS:Negative Small) (NL:Negative Large)

fuzzified/defuzzified simulatenously by matching membership functions. Therefore fuzzy control processing can be adapted to a parallel neural network structure where each *neuron* represents functions (fuzzy membership) and *links* represent the weight of a fuzzy rule.

Figure 5(a) shows the structure of the Fuzzy Neural Net Control System (FNC) and its fuzzy subspace (Figure 5(b)) [10]. In this experiment, 5 input membership functions are assigned to each input signal and 5 output membership functions are used to compute fuzzy output signal.

While an earlier Fuzzy Logic Controller [16, 20] was implemented in rule-based form (*if-then*), the FNC employs a parallel inferencing network structure. Due to the parallel fuzzification/defuzzification scheme, the FNC can improve real-time performance of the control system for practical application.

The performance of the FNC is determined by the

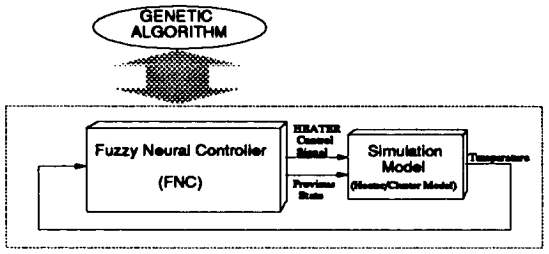

Figure 6: GA optimization of the FNC module

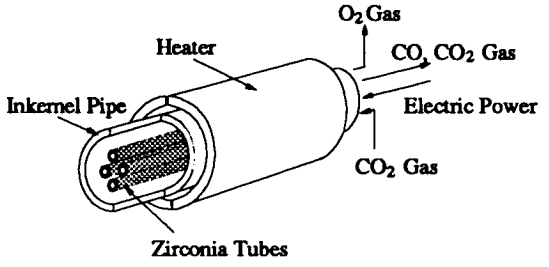

Figure 7: Oxygen Production System Cluster

input membership functions of layer 1 which fuzzify the input signals and the output membership functions of layer 4 which defuzzify normalized firing strengths. A membership function is specified by number of parameters.

In order to find a high performance fuzzy membership functions without the help of human expertise, it is necessary to employ computer-aided optimization. Since tuning the membership functions requires adjusting many parameters simultaneously, hill-climbing search methods would suffer from the complexity of the search space.

For this reason, a probabilistic optimization method utilizing evolution strategies, such as Genetic Algorithm (GA), was employed to find optimal membership functions. Since optimizing multi-parameter problems takes a long time, we developed new form of GA which is especially oriented to parallel computers that can satisfy the real-time constraints of the system.

Figure 6 shows the interaction of the FNC, simulation model and GA-optimizer. The FNC operates the simulation model, such as heater/cluster model of the Mars Oxygen Production System (OPS).

The OPS, shown as in figure 7, includes Zirconia tubes located symmetrically inside a cylinder. A radiation heater is wrapped around the outer surface. With this configuration, the majority of heat transfer between the outer surface and the oxygen gas inside the system is due to radiation. Applying the one-dimensional heat equation with lumped temperature distributions for the surface and oxygen temperatures we obtained two first order differential equations as provided below. The T_p

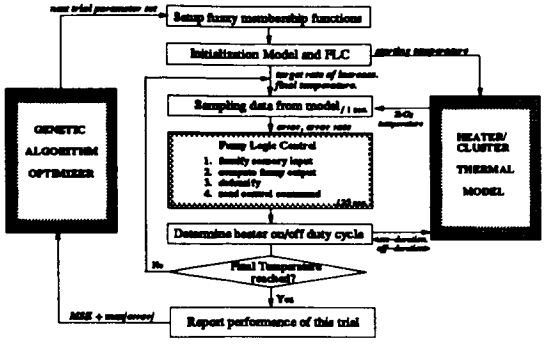

Figure 8: Individual evaluation procedure:FNC operation with heater/cluster thermal model

represents the pipe temperature and T_z is Zirconia tube temperature. A variable SW is either 0 or 1 to control the heat source (heater). The objective of the FNC is to increase the temperature of the Zirconia tubes at a constant rate until a goal temperature is reached [17].

$$\frac{dTp}{dt} = 2.75 \times SW - 4.42 \times 10^{-12}(T_p^4 - T_z^4) - 8.65 \times 10^{-4}(T_p - 278.0)$$

$$\frac{dTz}{dt} = 4.42 \times 10^{-12}(T_p^4 - T_z^4)$$

As shown in figure 5, there are two input signals to the FNC e.g., temperature increase error rate (input1) and rate of its error rate (input2). Based on two inputs, the FNC produces an output command which controls on/off duty cycle of the heater element in the model. As shown in figure 5, we employed 5 membership functions for each input signal and 5 membership functions for the output signal.

Figure 8 provides detailed procedures of the FNC integrated with the GA-optimizer. An individual of a GA represents one trial set of fuzzy membership functions. The GA optimizer sends a parameter assignment to the FNC which determines its fuzzy membership functions. The model is reset to its initial conditions (starting temperatures). The operational specifications such as desired temperature increase rate and goal temperature are set inside the controller. The performance of a trial individual fitness is measured as the sum of the MSE (Mean Square Error) between actual temperature increase rate and desired one and maximum absolute value of error of temperature increase rate.

3.1 Design of the FNC for the Oxygen Production System

Our primary objective is to design optimal fuzzy membership functions that perform well with given operational specifications while utilizing minimal human expertise. The controller increases the temperature of the cell at a constant rate.

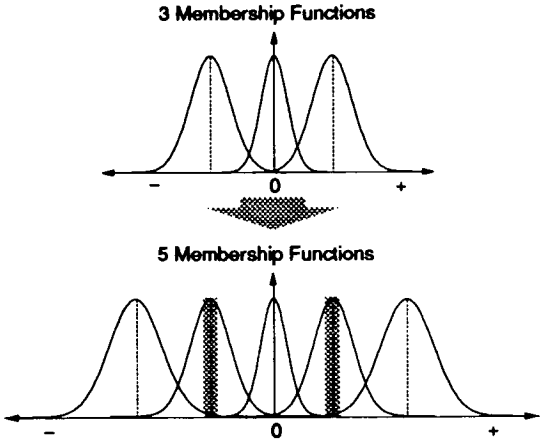

Figure 9: Expanding Fuzzy Membership Functions

Designing an optimal FLC involves the investigation of several alternatives, such as type of membership functions and the number required. A single-level GA starts to optimize the FLC based on the assumption that a given FLC specification, such as type or number of membership functions, is optimal. But real world application problem is often too complicated to determine the correct system specification.

Hierarchical GA solves this problem by changing its structure according to the performance of each module which employs a different number of fuzzy membership functions and parameter resolution. Starting from a small number of parameters, it expands search parameters and their resolution as they create lower levels. The lower levels take advantage of information found at the upper level AGA module.

Figure 9 shows how search parameters are expanded as Hierarchical GA creates lower level modules in the example of designing a FLC. The upper level module starts to design the FLC with a small number of membership functions. Designing a FLC with fewer membership functions is relatively easy compared to a large number of membership functions. Even though the upper level need not find the best membership function, it does provide some information to the lower level which supports the design of an optimal FLC. Since the membership functions found at the upper level are optimized based on constraints of a small number of parameters, the lower level GA modules give small tolerances to the received parameters. This is due to the effect of new parameters on the old parameters optimized earlier. Figure 9 illustrates how to expand membership functions, the shade area of membership function represents its tolerance.

As we increase the number of employed fuzzy membership functions, the fuzzy rule table must also be expanded. Figure 10 shows ways of adding more slots to

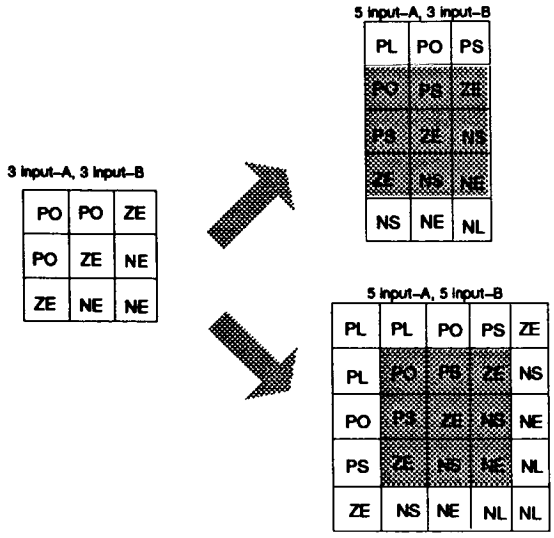

Figure 10: Expanding Fuzzy Rule Table

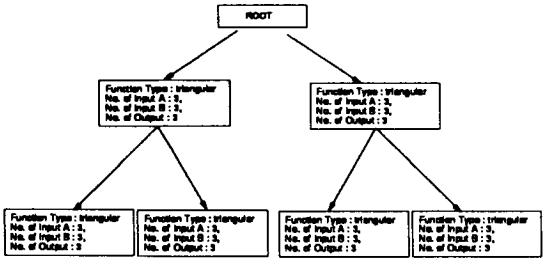

Figure 11: Tree of Various FLC Specifications

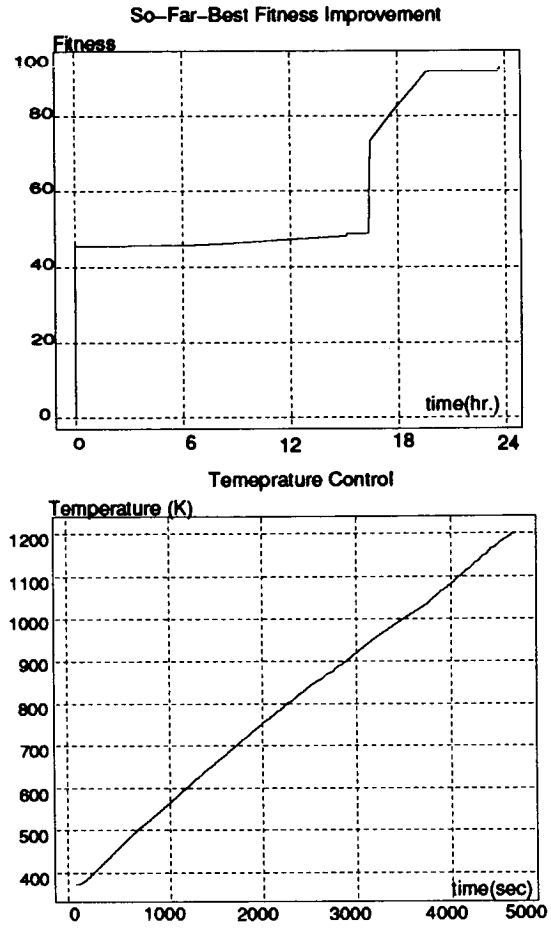

Figure 12: Simulation Results from Self-Organizing Hierarchical GA to design optimal FLC

the fuzzy rule tables. Since the fuzzy rules are optimized based on constraints of fewer membership functions (for example, 3 input A,B, and 3 output membership functions), the expanded fuzzy rules need to be optimized with not only more slots but also some degree of tolerance of suggested rule parameters.

Figure 11 shows a tree of various specifications of the FLC in which the Hierarchical GA searches through optimal design. Hierarchical GA first optimizes fuzzy rules which are more sensitive to the FLC performance at the root module. The small number of fuzzy membership functions with small parameter variance were used in order to maximize the sensitivity of fuzzy rules. The fuzzy rules found at the root module are sent to lower levels, where two different membership function types, such as triangular and bell shape are employed. The lower levels have wider search ranges in the parameters and utilize the fuzzy rule information received from the root. A greater number of fuzzy membership functions are employed when the lower level GA modules are created.

Figure 12 shows the simulation results that illustrate how the Hierarchical GA investigates various FLC specifications to design an optimal controller. The fitness improvement of Hierarchical GA shows that when a certain GA module is executed, the fitness increases suddenly. The FLC specifications of the module provides the best performance among other FLC specifications. The population of each module represents different species, because they expresses different number of parameters and search spaces. When a certain species (the correct one) is created, the performance of the Hierarchical GA improves in a step like manner. The temperature profile shown in the figure 12 is that of by the suggested optimal FLC in the Hierarchical GA.

4. Conclusions

Real world application problems often require optimization of a large number of parameters with high precision. These parameters increase the complexity of the search problem. In existing GA, a chromosome representing the parameters does not contain information about their sensitivity, even though they influence the system performance to different degrees.

We have devised a novel scheme of Hierarchical Genetic Algorithms in self-organizing variable structure

environment for complex real world problems. The design of a temperature control system for the oxygen production plant was selected as an experiment. Since conventional control schemes are limited their functionality to relatively simple applications, Fuzzy Logic/Neural Net control methods are received more attention for the sophisticated applications. The parameters embedded in the controller need to be optimized for the required control performance.

In this paper, a Hierarchical GA investigates various FLC specifications using variable structure simulation. More sensitive parameters, such as fuzzy rules, are optimized before other parameters. Higher level GAs search for candidate individuals that might contain the optimum in a given search space. These candidates are sent to the lower level to be investigated in greater detail. If better solutions are found at a lower level, they are reported back to the higher level and incorporated into its on-going GA search. The higher level GAs search in a sparse space with fewer parameters that influence the system performance significantly. In order to reduce GA search time, higher levels also utilize less accurate models for which simulation-based evaluation time is reduced.

The simulation exhibited interesting search behavior : when a good GA module is discovered, the performance increases suddenly. This suggests that not only has a good design been found, but that all other design frameworks can be eliminated.

References

[1] Back, T. and Schwefel, H-P.,"An Overview of Evolutionary Algorithms for Parameter Optimization", *Evolutionary Computation*, Vol.1, No.1, MIT Press, 1993.

[2] Collins, R. J. and Jefferson, D. R., "Selection in Massively Parallel Genetic Algorithms", Proceedings of the 4th ICGA, Morgan Kaufman, San Mateo, CA, 1991.

[3] Daniel, S. M., "Motorola Software Tools for Chez Scheme : User's Guide and Reference Manual", Motorola inc. Scottsdale AZ, 1991.

[4] DeJong, K. *An Analysis of the Behavior of a Class of Genetic Adaptive System*, Ph.D Dissertation, Dept. of Computer and Communication Sciences, Univ. of Michigan, Ann Arbor, 1975.

[5] Fuji, R. M., "Parallel Discrete Event Simulation", Communication of the ACM, Vol.33, No.10, Oct. 1990.

[6] Goldberg, D. E., "Genetic algorithms in search, optimization, and machine learning", Addison-Wesley, Reading, MA, 1989.

[7] Grefenstette, J.J., "Parallel Adaptive Algorithms for function optimization", Tech. Report No. CS-81-19, Vanderbilt University, Computer Science Department, Nashville. 1981.

[8] Holland, J., "Adaptation in Natural and Artificial System", University of Michigan Press, Ann Arbor, 1975.

[9] Hwang, K. and Degroot, D., "Parallel Processing for Supercomputer and Artificial Intelligence", McGraw-Hill, New York, 1988.

[10] Jang, J.S., "ANFIS:Adaptive-network-based fuzzy inference systems" *IEEE trans. on System, Man, and Cybernetics*, 1992

[11] Kim, J. and Zeigler, B. P., "Designing Fuzzy Neural Net Controllers using GA Optimization", accepted to International Joint Symposium IEEE/IFAC on Computer-Aided Control System Design. 1994.

[12] Lee, C. C., "Fuzzy logic in control systems : fuzzy logic controller-part 1.". *IEEE trans. on System, Man, and Cybernetics*, 20(2):404-418, 1990

[13] Maumov, Y. and Meystel, A., "Optimum Design of Multiresolutional Hierarchical Control System", Proceedings of the 7th International Conference of Intelligent Control. Glasgow, UK. 1992.

[14] Miachalewicz, Z., *Genetic Algorithm + Data Structure = Evolution Programming*, Springer-Verlag, New York, 1992

[15] Schaffer, J., Caruana, R., Eshelman, L. and Das, R., "A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization", *The Third International Conference on Genetic Algorithms*, Morgan Kaufmann Publisher, Los Altos, CA, 1989

[16] Schooley L., Zeigler B., Cellier F., Marefat M., Wang F., *"High Autonomy Control of Space Resource Processing Plants"*, Control Systems Magazine, June, 1993

[17] Stephens, W. P, Cummings, D and Vincent, T.L. "Automation of a Nonmechanical Oxygen Compressor", Progress Report in the Dept. of Aerospace and Mechanical Eng. University of Arizona, Nov. 1, 1991.

[18] Zeigler, B.P., *Object-Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems*, San Diego, CA. Academic press, 1990.

[19] Zeigler, B. P. and Louri, A. "A Simulation Environment for Intelligent Machine Architecture", Journal of Parallel and Distributed Computing, No. 18, 1993.

[20] Zeigler, B.P. and Kim, J., "Event-based Fuzzy Logic Control System", *Proceedings of 8th IEEE International Symposium on Intelligent Control*, Columbus, OH, 1993.

[21] Zeigler, B.P., Kim, T.G., Lee, C., "Variable Structure Modelling Methodology : An Adaptive Computer Architecture Example", Transactions of The Society for Computer Simulation, Vol. 7, No. 4, 1991.