

DETERMINING OPTIMAL CONFIGURATION FOR TURBINE GENERATOR COOLER

Nivedita Sumi Majumdar and Dipankar Dasgupta
Division of Computer Science
Department of Mathematical Sciences
University of Memphis, TN

Abstract: Configuration problems with hierarchical decision-tree structures are difficult to encode for solution using simple Genetic Algorithms. The chromosomes typically require fitness-evaluating schemes with steep gradients to optima. Solutions get stuck at local optima. We used a GA that can control the expression of over-specified chromosomes for exploring the multilevel search-space. Some experiments to configure a turbine generator cooler are performed and results are reported.

1. INTRODUCTION

Configuration management looks for a complete set of components and their relationships such that the resulting product structure satisfies all requirements and constraints. Configuration management is usually used for highly complex systems, e.g. power generation system that has a number of sub-structures. Some of the components of these systems are so large that they might require a large manufacturing time. Accordingly they need to be discussed at the earlier stages of the design process so that they can help to eliminate any artificial constraints. A Configuration Management System allows the users to explore the various options available and permits adjustment of the overall system depending on various constraints and relationships imposed by the system. Therefore the Configuration management Problem (CMP) can be modeled to design many mechanical, electrical, and physical phenomena for their optimal solutions [1, 2, 3, 10, 11].

In configuration management, we deal with various parameters and attributes of the system that have certain admissible values subject to some constraints specific to the problem domain. The attributes are discrete options of the system. When the process runs, it must activate some of these options or attributes, and must give us as solution the complete sequence of options one needs to implement for best performance of the system, within the specified constraints. The feasibility of the solution produced is an important issue here as very often, infeasible solutions are generated. The encoding scheme and operators used to arrive at the solution thus needs to be robust to survive the pitfalls. Many constraint relationships are mutually exclusive, in the sense that, if one of the nodes in such a relationship is selected, rest of the nodes in the same relationship is automatically deactivated. Sometimes there has to be simultaneous selection of a set of nodes in the solution of the problem. The configuration module must take care that the user should not change values set by the module, and the module should not change the variables set by the user.

Design of an optimal Turbine generator cooler is an excellent example of such a problem. Each available option in the design phase of the turbine generator in terms of choice of cooling process, machines to employ and other technical details has certain pros and cons and the final design should be guided by taking all these into consideration. The domain expert can grade every available option with a number and then the problem is that of generating an optimal combination of the available options to maximize a certain measure of goodness or fitness function. MacCallum and Yu [1, 2, 3] proposed a representation of the turbine generator design considering the various physical constraints and requirements of the user. The problem has a tree structure. Such hierarchical solution structure restricts the search space severely. As such these are challenging problems for simple Genetic algorithms. This is because the fitness function has very steep maximums and in between the fitness landscape deteriorates very drastically such that it makes the chromosomes encoded for the solutions cling to local solutions and makes exploration very difficult. In this paper we propose a structured genetic algorithm solution for this configuration management problem.

2. STRUCTURED GENETIC ALGORITHM

Genetic Algorithms can be applied, virtually, to any problem that has a large search space. However, it is relatively difficult to apply Genetic Algorithms where the search space is essentially restricted to a few feasible configurations with different level of constraints. In configuration management of gas turbine coolers, there are large sets of possible solutions with a few feasible solutions. There has to be a multilevel interpretation of the chromosomes. Simple GA works in a linear fashion and is not adequate to tackle this problem. A Structured Genetic Algorithm (sGA) performs better where the problem have a hierarchical structure rather than a linear one.

The following paragraphs summarize some features of structured GAs as proposed by Dasgupta and McGregor in [4-8]: A chromosome is represented by a set of independent sub strings (genes) at different levels, with higher-level genes controlling the expression of lower level genes. During reproduction, genetic operators such as crossover and mutation modify these sub strings. In decoding to phenotype, only those genes currently active (or expressed) in the chromosome contribute to the fitness (i.e., make the phenotype) [9]. The passive (or unexpressed) genes are not used in evaluating the individual's fitness and are carried

along as dormant genetic material during the evolutionary process. Genetic operations altering high-level (i.e., control) genes result in changes to the active elements of the genomic structures. Thus, a single alteration can have a profound effect on the chromosome (such as activation of previously passive genes). Genetic operations altering the lowest level result in changes of the information-carrying genes. However, changes made to passive genes are not immediately expressed and thus are temporarily neutral. These features allow the model to solve multistage optimization problems by defining search spaces in its different levels and searching them simultaneously in a single evolutionary process.

It can be seen that sGA is the easiest way to encode both the discrete options and the variable parameters required to model a configuration management problem in a linear chromosome string, since it has an inherent tree structure.

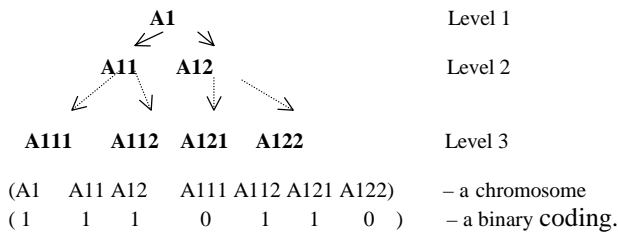


Figure 1. A simple representation of three-level sGA, where the values of A111 and A122 are set to zero.

The diagram given in Figure 1 shows how a structured genetic algorithm with three levels function. The chromosome bit positions each have a meaning mapped to the three level problem structures. The values of ‘1’ for A11 and ‘0’ for A12 bit positions, allow participation of bit positions A111 and A112 in the evaluation of the individual’s fitness and exclude participation of A121 and A122 from the evaluation process.

The structured Genetic Algorithm is especially useful in handling problems with certain GA deceptive fitness functions [9], typically seen in problems with a hierarchical structure encoded in a linear string. These problems are deceptive in the sense that feasible individual space is very restricted and the fitness landscape has many hills and valleys. When a solution is in the basin of attraction of one such peak, it becomes difficult to make it move because it

wants to cling to that peak and it cannot get any better than the local optima. The solution is driven toward a false peak dependent on the random start point, or position of the random initial string. Thus a simplistic approach will often yield sub optimal solutions.

3. PROBLEM DEFINITION

The problem is to optimally select the design parameters of a cooling system for a turbine generator. The attributes of this problem are the various parts needed to build the cooling system and the values are options for a proper configuration. The cooling fan for air-cooling can be positioned at the side, top or under the generator – each with different advantages and disadvantages. The domain expert who assigns these options with specific fitness through application interface actually provides scaled values between 0 and 1. Figure-2 illustrates the basic structure for the problem. The generator must have a cooling procedure defined for it. So *cooling option* is an attribute that must be selected. The ‘bold line’ symbolizes the ‘AND’ link that connects attributes which must be selected together (simultaneously activated) for a cooling system configuration to be feasible. The attributes *cooling option* has values: *hydrogen cooling*, *air cooling*, and *hydrogen/water cooling*. Any one of these values can be chosen, but each has to be chosen at a time. That is a turbine cannot have air-cooling and hydrogen cooling at the same time. So the relation is shown by a ‘XOR’ link or the dotted line in figure 2. If the *air cooler* is chosen as the preferred cooling option, then either *side*, *top* or *under* needs to be selected, again any one at a time. If *hydrogen cooling* is chosen for *cooling option*, then we have to choose a value for its two attributes viz. *hydrogen coolers* and *hydrogen seals*. Attribute *hydrogen coolers* can take values *Horizontal* or *Vertical* and attribute *Hydrogen Seals* can take values *Journal* or *Thrust*. If *Hydrogen or Water Cooling* is chosen, then there are four attributes viz. *hydrogen coolers*, *hydrogen seals*, *coolant* and *connections*, whose values need to be decided. Similarly, *Hydrogen coolers* like previously can be either *Horizontal* or *Vertical*. *Hydrogen Seals* can be either *Journal* or *Thrust*. *Coolant* could take values *Hose*, *Waterbox* or *Unlink* while *Passes* can be either *One* or *Two*. This is schematically illustrated in Figure 2.

“AND” Links
 “Mutually XOR” Links/Constraints

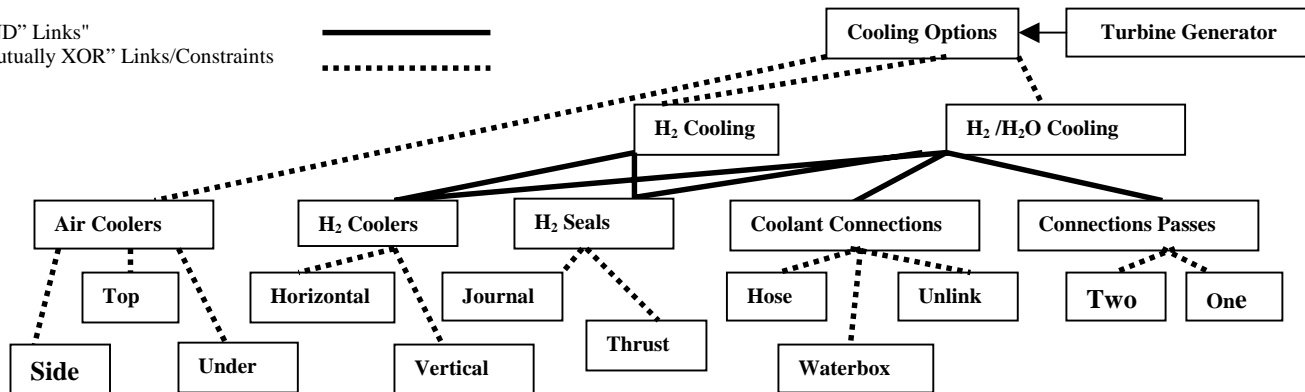


Figure 2. Hierarchical Structure of Turbine Generator Cooler Configuration Problem

4. IMPLEMENTATION DETAILS

4.1. The Encoding Scheme

The configuration space is encoded with 19 bits, to represent the chromosome. Each bit in the chromosome is set to '0' or '1' representing its activity status ('1' represents active and '0' represents passive). Figure 3 demonstrates our scheme for encoding the potential solution into chromosomes. The first three bits are the switching bits that are pivotal in controlling the part of the chromosome that is activated. A '1' in the 1st position, with two following '0' s in the 2nd and 3rd positions respectively in gene 'a' activates gene 'b' while gene 'c' and gene 'd' are inactivated. Similarly, a '0' in the 1st position, followed by a '1' in the 2nd position and a '0' again in the 3rd position of gene 'a' activates gene 'c'. As before, the other genes viz. 'b' and 'c' are inactivated at that time. This way the fitness evaluation of the string only depends on the active genes of the string. As all possible chromosomes are not feasible solutions, we generated the initial population heuristically so that high-level bits are set properly (as mentioned earlier). Then the special operators are used to preserve the feasibility.

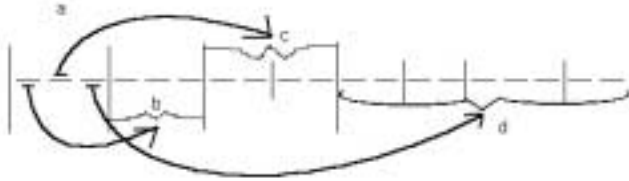


Figure 3. Chromosomal Representation and Interpretation Schema

4.2 Implementation of genetic operators

Since the search space is very sparse, the initial population is generated in semi-random fashion to maintain the feasibility constraints of the configuration model instead of allowing an unrestricted exploration of the search space. We use the values 0.8, 0.02 and 0.9 for GA parameters crossover rate, mutation rate and swap rate respectively. We tested with many parameter settings and these values generally gave a better result. Crossover is kept restricted to certain regions of the chromosome to preserve feasibility. It is applied only at the gene separation positions i.e. at positions 0,3,6,8,10,12,14,17 respectively (Figure 4) so that it never yields invalid configuration. Bit Mutation is used with specified probability at the lowest level of the decision tree. If mutation leads to an invalid string, mutated individual is accepted into the population but a high negative fitness value is given to penalize the move. At higher level (control genes) we used the swap operator. When the search space is very restricted as it is in our case, this operator is reported to perform well and we arrive at similar conclusions with our experiments. Swap occurs between two bit positions of a gene amongst itself [9, 12].

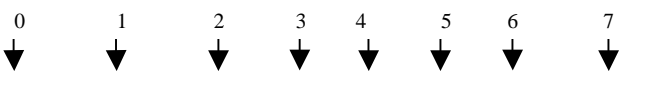


Figure 4. Gene separation points of the chromosome

4.2.1 Operator CROSSOVER

We generate a random number between 0 and 1 for each generation and check if it is lower than the crossover rate set by the user. If so, then we perform crossover to generate all the new members of the next generation in the following way. We use binary tournament Selection Method to choose the two participating parents from the population. In Binary Tournament Selection Method, any 2 members are randomly picked from the population, and their fitness is evaluated. The individual with higher fitness is selected as the first parent. Similarly the other parent is chosen.

Then we generate a random point for crossover between 0 and 7 as crossover can occur only at the gene separation points as shown in Figure 5. Say, the crossover point is 3 (Figure 4). Then using Parent 1 and Parent 2, we get Child 1 and Child 2 as shown in Figure 5. We replace the evolved Parent with the Child, if the fitness of the child is better. In other words, we use one-point crossover at specific sub-sets of crossover points.

```

Parent 1: 001 100 0101 011000110
Parent 2: 001 100 1001 011010010

Child 1:  001 100 0101 011010010
Child 2:  001 100 1001 011000110
    
```

Figure 5. Crossover

4.2.2 Operator SWAP

For every member in a generation, we generate a random number between 0 and 1 and check if it is lower than the Swap rate set in the program by the user. If so, then we perform Swap to generate a new individual for the next generation in the following way. We note that Swap can occur only among the genes themselves other wise the whole purpose of trying to maintain chromosome feasibility is defeated. Therefore Swap can occur between sub strings in locations 0 through 7 from Figure 5.

We first select a random point between 0 and 7 to select the gene that we are going to transpose. Say, the gene Swap point is 6 (Figure 4). So any two points in the italicized gene is to be swapped. Now we generate 2 random numbers less than the length of the gene. Suppose that 2 and 3 are selected. Swapping occurs as shown in Figure 6.

```

Individual:  001 100 0101 011000110
Swapped:    001 100 0101 011001010
    
```

Figure 6. Swap

4.3 The tool for configuration management

We developed a java-based software to implement sGA with friendly user interface. It is shown in Figure 7. The interface is organized into four sections. The first section lists the principal three options available for the Generator Cooling viz. Air Cooling, Hydrogen Cooling or Water Cooling. The user sets a certain value between 0 and 1 as the

weightage of each of those options. Then similarly, for air-cooling, the user gives a preference value for the sub choices Side, Top or Under that is available. Likewise, every alternative for Hydrogen Cooling Or Water Cooling has to be assigned a certain preference weight between 0 and 1. The application screen pops up with the default setting as is shown in Figure 4.

This initial setting of parameters here is arbitrary and just used to demonstrate the capability of the system to arrive at the optimum based on the preferences. The user has a choice to change these settings. Then in the last section, GA parameters like population size, number of generations, crossover, mutation and Swap Rates have to be set. The “submit” button stores all the set values into a data file, which is subsequently read by the GA application to construct its array of fitness values.

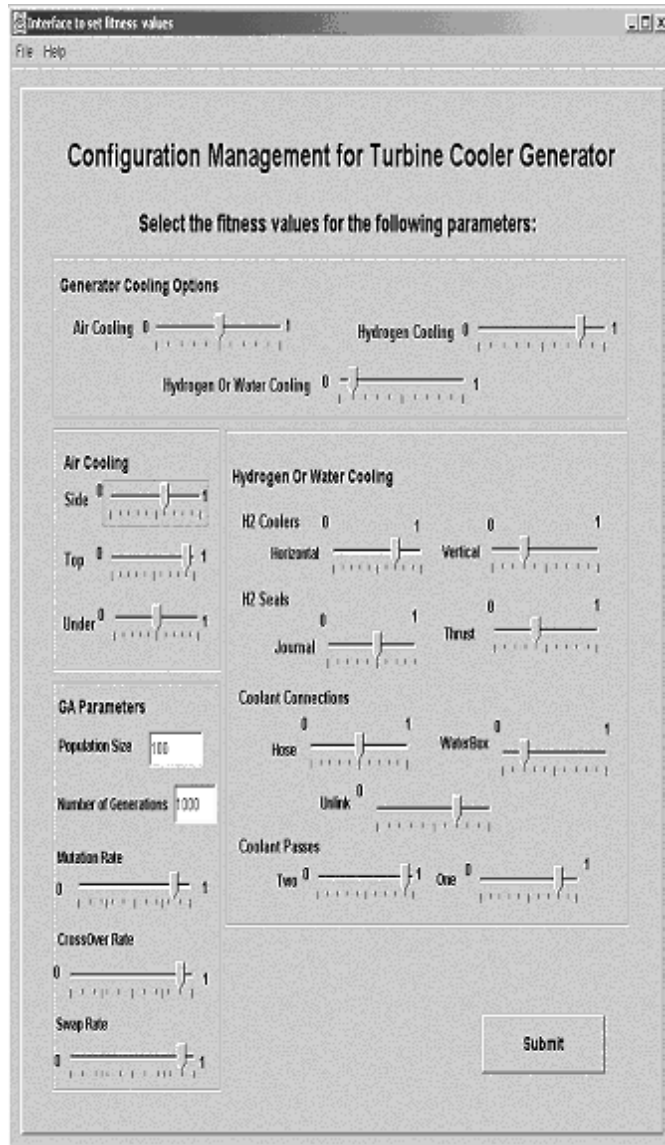


Figure 7. The Application Interface

Next, the GA determines the best path to take based on the set values. The fitness of any chromosome is the sum of the weights of nodes in its active path. Sample calculations are presented in following sections. Once the fitness of each string in the population is evaluated, the best member of the population is selected and carried over intact to the next generation. We used the elitist strategy so that the best solution evolved thus far remains in the population. Then the whole population is made to interact with each other by means of genetic operators to create the rest of the new population. For each run the population size and the number of generations are input parameters. Examples of Border crossover and Swap operators as used here are given in the following sections.

4.4 Fitness Evaluation

Table 2(a) and Table 2(b) illustrate the fitness evaluation using the best individual obtained for a typical setting of parameters. The first column lists the options available. The second column gives the value of the fitness and the third column lists the position of the chromosome that is related to that particular option. Results in Table 2(b) illustrate one sample run.

Options	Fitness Values	Chromosome Positions
Air Cooling	0.5	[0]
Side	0.6	[3]
Top	0.9	[4]
Under	0.5	[5]
Hydrogen Cooling	0.8	[1]
H2 Coolers		[6]
Horizontal	0.7	
Vertical	0.3	[7]
H2 Seals		
Journal	0.5	[8]
Thrust	0.4	[9]
Hydrogen /Water Cooling	0.1	[2]
H2 Coolers		
Horizontal	0.7	[10]
Vertical	0.3	[11]
H2 Seals		
Journal	0.5	[12]
Thrust	0.4	[13]
Coolant Connections		
Hose	0.5	[14]
Water Box	0.2	[15]
Unlink	0.7	[16]
Coolant Passes		
Two	0.9	[17]
One	0.8	[18]

TABLE 2(A). A SAMPLE SETTING OF THE OPTIONS

The best individual evolved in this run of the experiment is shown in Table 2(b). The activated bits are written in bold face. The fitness of the chromosome in Table 2(b) evaluated to 2.9.

Chromosome Position	Bit Value	Solution interpretation:
[0]	0	Hydrogen Cooling Or Water Cooling → H2 Coolers: Horizontal → H2 Seals: Journal → Coolant Connections: Unlink → Coolant Passes: Two
[1]	0	
[2]	1	
[3]	1	
[4]	0	
[5]	0	
[6]	0	
[7]	1	
[8]	0	
[9]	1	
[10]	1	
[11]	0	
[12]	1	
[13]	0	
[14]	0	
[15]	0	
[16]	1	
[17]	1	
[18]	0	

TABLE 2(B) THE BEST INDIVIDUAL FROM THE RUN WITH OPTIONS SET AS IN TABLE 2(A).

In general, the formula for evaluation of fitness is given by:

$$\begin{aligned}
Fitness &= -P_1 && \text{if } (\text{valid}(\text{gene a})==\text{False}) \\
& -P_2 && \text{if } ((\text{valid}(\text{gene b})==\text{False}) \\
& && \text{OR } (\text{valid}(\text{gene c})==\text{False}) \\
& && \text{OR } (\text{valid}(\text{gene d})==\text{False})) \\
& f_1 + f_2 && \text{otherwise}
\end{aligned}$$

Where P_1 and P_2 are penalty values for infeasible chromosomes, also $P_1 > P_2$. Moreover, $\text{valid}(a)$, $\text{valid}(b)$, $\text{valid}(c)$, $\text{valid}(d)$ are four functions that take gene a, gene b, gene c, gene d (Figure 3) respectively as their only parameters and returns a Boolean Value of 0 or 1, to indicate if the respective genes are feasible according to the constraints set in the encoding of the chromosome.

$$\begin{aligned}
f_1 &= \sum_{j=1}^{|a|} (a)_j * W_j \\
f_2 &= \sum_{i=1}^{|b|} (b)_i * W_i && \text{if } (a)_0 = 1 \\
f_2 &= \sum_{k=1}^{|c|} (c)_k * W_k && \text{if } (a)_1 = 1 \\
f_2 &= \sum_{m=1}^{|d|} (d)_m * W_m && \text{if } (a)_2 = 1
\end{aligned}$$

Where (a) , (b) , (c) and (d) represent the different gene segments as shown in Figure 3, $(x)_i$ is the i^{th} bit of gene x and

W_i is the weight set for the i^{th} by the user. $|a|$, $|b|$, $|c|$ and $|d|$ denote the length of those genes respectively.

5. RESULTS AND DISCUSSION

In this section, we report the results obtained in some experiments. Figure 8 and 9 show the best and average fitness values at different generations. Figure 8 shows how the best fitness has changed during each generation. The value converges to the optimum fitness value of 2.9 by the 800th generation with a population size of 200 chromosomes. The results were reported from an average of 6 best runs of the application. The fitness function has very steep gradient to local and global optima and therefore once a fairly good solution is found, it tries to cling to that peak. We see that the best fitness remains at 2.55 from the 67th to the 295th generation. Then it jumped to a better fitness and again clings to the value 2.7 for some time. The changes are due to mutation or swap operations.

We experimented previously without using the Swap operator. The results were not very promising. This is because, to reach from one valid configuration to the other, there had to be changes in two subsequent bit positions, which is difficult to achieve with simple mutation or crossover. For example, let us consider just the last attribute value in gene d (Figure 3). If it were activated then, the best two individuals would evaluate respectively to 2.8 and 2.9. Their last two bits could have 01 (for fitness 2.8) and 10 (for fitness 2.9) respectively. We can see that in one step by bit mutation it is highly unlikely to move an individual from one to the other. Therefore, use of swap operator has been found very useful.

The graph in Figure 9 reports how the value of the average fitness changes with the number of generations for the same runs of the experiment. The primary objective of the graph in Figure 9 is to show that the diversity in the population is maintained and every one of the individuals does not collapse to a single solution. The negative values of fitness are obtained because of the high negative penalties awarded to the individual's fitness measure if some of them reach an invalid configuration.

It is to be noted that the average population diversity is always preserved and the elitist strategy is used to track the best individual. Since the problem landscape is very rugged, we used sGA operators to enable the chromosomes to jump out of undesirable local optima attractor basins and penalized the bad individual. Therefore, it is reasonable that some bad members are generated in every generation, which causes the high negative fitness and drags down the value of the average fitness. Also, in sGA, the whole population convergence is not desired since redundancy is inherently incorporated in the encoding mechanism.

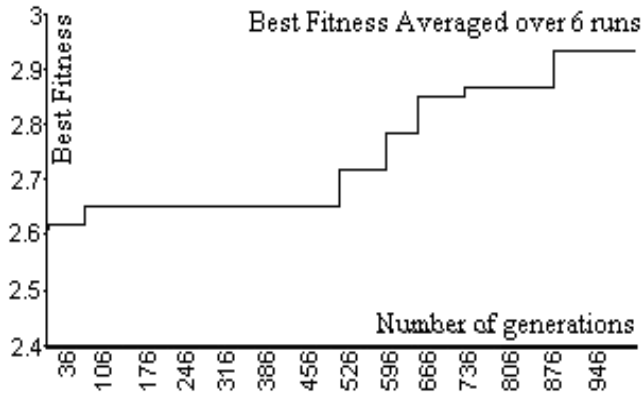


Figure 8. Best Fitness Values (producing a feasible configuration)

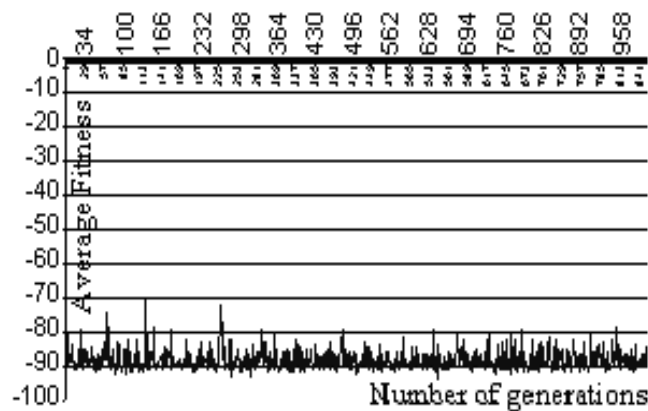


Figure 9. Average Fitness Values

6. CONCLUSIONS

Problems of a hierarchical nature are better to encode in a Structured GA representation. This is because the hierarchical nature of the problem is well reflected by the activation of a particular gene dependent on a higher-level bit value, which is the essence of the sGA. But in sGAs, many options can be activated; we always run a risk of producing infeasible solutions unless some restrictions are in place either in the form of special operators or in the replacement scheme. Accordingly, in a hierarchical structure, a small change in the high-level can drastically deteriorate or improve the individual solution. We used the structured GA with modified genetic operators devised using the domain knowledge, which shows good performance for this complex problem. Many variations to the techniques implemented here is possible and we continue to investigate the possibilities with more complex problems of this same nature.

7. ACKNOWLEDGEMENTS

This work was supported by NSF grant number NSF-IIS-0104251. We would also like to thank Jonathan Gomez and Fabio Gonzalez for their help and useful comments.

8. REFERENCES

- [1]. Bei Yu and Ken MacCallum. A knowledge based system for product configuration management. CONFIGMAN. <http://www.cad.strath.ac.uk/Research/projects/config.html>
- [2]. Bei Yu and Ken MacCallum, A Virtual Configuration Workbench for Product Development, presented at the International conference of Engineering Design (ICED'97), Tampere, August 19-21, 1997.
- [3]. Bei Yu and Ken MacCallum, A Product Structure Methodology to Support Configuration Design, in Proc. of the First International Workshop on Product Structuring, Technical University of Delft, The Netherlands, pp83-98, June 22-23, 1997.
- [4]. Dipankar Dasgupta and Douglas R. McGregor, 'A more Biologically Motivated Genetic Algorithm: The Model and some Results'. In *Cybernetics and Systems: An International Journal*, Vol. 25, issue 3, May-June, 1994, pp447-469.
- [5]. Dipankar Dasgupta and D. R. McGregor. 'Designing Neural Networks using the Structured Genetic Algorithm'. In *Proceedings of International Conference on Artificial Neural Network (ICANN)*, 4-7 September 1992, Brighton (UK), pp 263-268.
- [6]. Dipankar Dasgupta and D. R. McGregor. 'Engineering Optimizations Using the Structured Genetic Algorithm'. In *Proceedings of European Conference Artificial Intelligence (ECAI-92)*, 3-7 August 1992, Vienna (Austria), pp 608-609.
- [7]. Dipankar Dasgupta and D. R. McGregor. 'Designing Application-Specific Neural Networks Using the Structured Genetic Algorithm'. In *proceedings of International workshop on Combination of Genetic Algorithms and Neural Networks (COGANN-1992)*, USA, published by IEEE computer Society, pp 87-96. Available in technical report form IKBS-9-92.ps.Z, 1992.
- [8]. Dipankar Dasgupta and D. R. McGregor, 'sGA: A Structured Genetic Algorithm', Technical Report no. IKBS-11-93.ps.Z, Dept. of Computer Science, University of Strathclyde, UK, April 1993.
- [9]. Dipankar Dasgupta, 'Handling Deceptive Problems using the Structured Genetic Algorithm' In *proceedings of IEEE World Conference on Computational Intelligence (Evolutionary Computation session)*, Florida, USA, June 26-July 2, 1994, pp 807-811.
- [10]. Ian Parmee. "Diverse Evolutionary Search for Preliminary Whole System Design". *Developments in Neural Networks and Evolutionary Computing for Civil and Structural Engineering*, CIVIL-COMP Press, Edinburgh, UK, pp. 199-204, 1995.
- [11]. Ian Parmee. "The Maintenance of Search Diversity or Effective Design Space Decomposition using Cluster-Oriented Genetic Algorithms (CoGAs) and MultiAgent Strategies (GAANT)", *Adaptive Computing in Engineering Design and Control '96*, Plymouth Engineering Design Centre, Plymouth, UK, pp. 128-138.
- [12]. M. Voss and Foley. "Evolutionary Algorithm for Structural Optimization". *Proceedings of GECCO 1999 (Genetic and Evolutionary Computation Conference)*, July 13-17, Orlando, FL, pp. 678-685.