2004

Parameter Selection in Genetic Algorithms

Onur BOYABATLI
Singapore Management University, oboyabatli@smu.edu.sg

Ihsan SABUNCUOGLU
Bilkent University

Parameter Selection in Genetic Algorithms

Onur BOYABATLI
Production and Operations Management Department, INSEAD
Fontainebleau, 77305, France

and

Ihsan SABUNCUOGLU
Industrial Engineering Department, Bilkent University
Ankara, 06533, Turkey

ABSTRACT

In this study, we provide a new taxonomy of parameters of genetic algorithms (GA), *structural* and *numerical* parameters, and analyze the effect of numerical parameters on the performance of GA based simulation optimization applications with experimental design techniques. Appropriate levels of each parameter are proposed for a particular problem domain. Controversial to existing literature on GA, our computational results reveal that in the case of a dominant set of decision variable the crossover operator does not have a significant impact on the performance measures, whereas high mutation rates are more suitable for GA applications.

Keywords: Simulation, Optimization, Genetic Algorithm, parameter selection, factorial design

1. INTRODUCTION

Recent advances in computational techniques have led to an increased interest in simulation-optimization (sim/opt) methodologies that are used to solve optimization problems. These methodologies differ from their antecedents by using simulation as a prescriptive tool, which has traditionally been used descriptively to estimate performance of complex stochastic systems. Sim/opt methodologies have been successfully applied to various combinatorial optimization problems.

A recent trend in sim/opt research is the use of meta-heuristic techniques, in particular Genetic Algorithms (GA's). GA is an iterative procedure, taking its inspiration from natural genetics. GA starts with a group of feasible solutions to the problem under consideration and proceeds to a new set of solutions at each iteration with the high aim of achieving better fitness (objective function) values. The performance of GA depends on several parameters. In this study, we propose a new taxonomy of these parameters based on their effects on the structure of the algorithm. We argue that the two parameter categories, structural and numerical, have different characteristics and thus have different impacts on GA structure. Even for a GA with same structural parameters (coding scheme, operator types, stopping criterion), a different combination of numerical parameters (initial population type, population size, maximum generation number and the crossover and mutation probabilities) may lead to drastic changes in the performance of the algorithm. The very problem-specific nature of GA makes it difficult to specify the best combination of these parameters.

This study examines the effects of GA numerical parameters on its performance in terms of both best fitness found and CPU time; and proposes guidelines for parameter selection for specific problem domains. A test problem of a serial assembly line taken from the literature is selected for experimentation. A GA coded in C is integrated with a simulation model using SIMAN language. 2^k factorial design is implemented.

The main contribution of our work is twofold. First, it proposes a new taxonomy for GA parameters, and second, it presents an extensive analysis on these parameters to draw general conclusions. In particular, our computational results reveal that in contrast to previous literature high mutation rates and low cross-over rates are more suitable under some specific problem domains.

The remainder of this paper is organized as follows. In §2 we give a brief summary of GA, and literature. §3 presents the proposed study, whereas in §4 experimental results are demonstrated. Finally in §5 we summarize and suggest directions for future research.

2. GA and LITERATURE REVIEW

GAs are search algorithms based on the mechanics of natural selection and natural genetics. In natural genetics, the presence/absence of genes and their order in the chromosome decide the characteristic features of individuals of a population. The different traits are passed on from one generation to the next through different biological processes, which operate on the genetic structure. By this process of genetic change and survival of the fittest, a population well adapted to the environment results. Similarly, in GA, a finite-length string coding is used to describe the parameter values of each solution for the search problem under consideration. Each string corresponds to an individual, and every individual acquires its power in the survival process in terms of its *fitness* value. Higher the fitness values, better the individuals performance in the evolution process. A fixed number of individuals correspond to a generation. GA is an iterative algorithm such that in every generation, first parents are selected depending on their fitness values, and then by some genetic operators the strings of children are produced. With their calculated fitness values, the new generation is obtained. And this procedure is repeated until some stopping criterion is met. Like the natural genetics as the generations proceed, the fitness of the whole

population (average fitness) increases, corresponding to better populations.

The power and simplicity of GA make it popular for even large-scale optimization problems. The main advantage of GA is that it does not require neither mathematical expression of response surfaces nor any derivative or gradient information. After Holland [6] has proposed GA as a search mechanism, and especially following the Goldberg's [5] book on GA, many applications of GA on various problem types are conducted.

Schaffer *et al.* [10] study the effects of some control parameters of GA on its performance. The control parameters selected are the population size, crossover and mutation probabilities and number of crossover points used in each mating. Experimental design is used for statistical analysis. The computational results reveal that the selection and mutation operators together is very significant in the behavior of GA. Mutation appears to be more effective than crossover.

Fogarty [4] discusses the effect of varying the mutation probability over time and its effect on GA performance. A minimization type test problem is used in the analysis. Two initial population types are used, one is a seeded population containing good solutions, whereas the other is randomly created. It is observed that varying the mutation rate significantly improves the performance of seeded population case, but not when the initial population is randomly generated.

Simulation-optimization methodology integrates GA with simulation modeling during the calculation of the fitness values. For every individual of a particular generation simulation results are used to assess the fitness of the corresponding individual. The power of simulation modeling has made the research in this area popular, hence there have been several studies on GA based simulation-optimization applications.

To give an example, Nakano [8] proposes a conventional GA for job shop problems, whereas Cruz and Haddock [2] presents GA-based simulation-optimization application on five different systems, three of which are queuing and the rest are inventory systems. The results are compared with the results gathered form IPA (infinitesimal perturbation analysis). Suresh *et al.* [11] apply GA on a facility layout problem. Wellman and Gemmill [12] optimize an assembly line with GA. The results are compared with that of stochastic quasi-gradient method (SQM).

To the best of our knowledge, there is no a study in the literature analyzing all the GA parameters, initial population type, population size, maximum generation number, and mutation and crossover probabilities. This study proposes a new taxonomy of GA parameters and presents an extensive analysis of these parameters to draw conclusions for the best parameter levels for specific problem domains.

3. PROPOSED STUDY

Although GA seems to be a robust algorithm which contains same operators and has the same algorithmic logic for different applications, in fact the algorithm itself is significantly different for distinct problems. The main reason is that GA has several parameters and any combination of these parameters has different impacts on the performance on GA. We classify these parameters into two classes, structural and numerical.

Structural parameters are the main factors affecting the GA performance, and the most difficult set of parameters to be dealt with in a GA application. As understood from its categorical name, they are concerned with the structure of GA. The change in any parameter value requires significant alterations in the coding pattern of GA's, i.e. you have to rewrite the whole algorithm from scratch. The *coding scheme*, *operator types* and *stopping criterion* are the main parameters.

There have been extensive studies on selecting the levels of each structural parameter; hence GA can obtain sufficient insight from the well-established literature. For instance, it is known that the sequence representation of coding schemes is better for scheduling problems. Moreover, the applicability of the structural parameters constructs a constraint in front of decision-maker, and forces to eliminate some parameter values. For instance, the simple one-point crossover cannot be applied to the problems having sequence representation

Numerical parameters contribute to the second class of the taxonomy we proposed. The *initial population type, population size*, *maximum generation number*, *crossover* and *mutation probabilities* are the main factors considered in this category. These parameters are easy to handle when the coding structure is considered. Although alterations in these parameter levels do not require extensive coding, different combination of them leads to drastic changes in GA performance.

Similar to structural parameters, there are several studies conducted on the analysis of numerical parameters; but the very problem-specific nature of GA hinders making general conclusions about the suitable levels of them. Moreover, these studies analyze these parameters one at a time, ignoring the interaction between the parameter types. The literature on this class of parameters is more ambiguous than the previous class.

In this study, all of the numerical parameters of GA are examined with a priori known level of structural parameter setting. General insights about the best combination of these parameters are presented on particular problem domains. More detailed explanation about numerical parameters is provided in the following sections.

Testing Scheme

A test problem is taken from the literature [7] for GA based simulation-optimization application, and significance of different values of the parameters on GA performance is analyzed on this test problem by factorial designs and ANOVA (Analysis of Variance). Two types of performance measure are considered throughout the analysis:

- Best fitness value obtained
- CPU Time elapsed

The system under consideration is a manufacturing system consisting of four workstations with exponential processing times (mean 0.33, 0.5, 0.2, 0.25 respectively) and three buffers, which are located between the stations. There are 7 decision variables in the problem, 4 belong to number of machines in each workstation (M_i), and three for the number of buffer positions between the stations (B_i).

The objective is to maximize profit where there exists marginal revenue for each good produced and variable cost for each machine and buffer space used. The profit function is defined for all machine levels at 2, and buffer levels at 3 as;

$$200 * \text{throughput} - 25000 * (2+2+2+2) - 1000 * (3+3+3)$$

where throughput is computed from simulating the model for a 30 day period. Also additional warm- up period of 10 days is used for eliminating the initial bias. The simulation model is developed using SIMAN V language in UNIX environment.

GA is coded with respect to a specific set of structural variables, and experimentation is conducted at each combination of

numerical values. The basic GA model is selected for the analysis. The binary coding scheme, roulette wheel selection, objective function value as fitness, one-point crossover, bit mutation and maximum iteration numbers as stopping criterion are used in the experimental analysis.

In our test problem number of machines ranges from 1 to 4, whereas buffer spaces range from 1 to 16. In the binary coding representation, machine numbers have 2-bit scheme, where buffers have 4-bit scheme. The strings representing the individuals are produced by the concatenation of individual strings, starting from the number of machines in first workstation, and continuing with the buffer positions in the same order. 4 two-bit strings, and 3 four-bit strings lead to a binary representation of individuals with string length 20. For instance a string corresponding to the decision variables' combination;

$M_1 = 1$, $M_2 = 2$, $M_3 = 3$, $M_4 = 4$, $B_1 = 10$, $B_2 = 11$, $B_3 = 12$ is:

00	01	10	11	1001	1010	1011
1	2	3	4	10	11	12

The average of five replication results gathered from the simulation run of the system with the codified parameter levels in the string corresponds to the fitness value of the string.

The roulette wheel selection procedure is used for the reproduction process. Since this procedures requires non-negative fitness values, when there is a negative fitness value in the population, offsetting technique is used for fitness scaling.

The one-point crossover is applied with a probability of p_c. The mutation is applied by random changes in the bit values of strings. For every bit of the string, mutation occurs with probability p_m. If there are infeasible solutions reached after the operations, then these strings are discarded, and new ones are generated from the beginning instead.

Factorial Design

Since there are 5 distinct numerical parameters, 2^5 factorial design is used in this study. For each category, two levels are determined corresponding to high and low levels, respectively. For each design point, five independent GA runs are taken. ANOVA is used to determine the significance of each effect on fitness and CPU time.

For initial population type, there are two common settings in the literature, random and seeded populations. A random search heuristic of 250 points is applied, and random population is drawn among these points. The best 20 or 40 points depending on the population size are selected for the seeded population.

It is reported in Alander's study [1], that a value between n and 2n is optimal for the population size. Considering these results, the levels of population size are taken as 20 (n) and 40 (2n), where 20 equals to the string length of our GA model.

The levels of operator probabilities are also drawn from the literature. De Jong [3] suggests that the bit-mutation rate should be n^{-1} where n is the string length. It is also reported that crossover rates between 0.65 and 1, and mutation rates between 0.001 and 0,01 are useful in GA applications. In our study the crossover probability levels are set as 0.5 and 1,and the mutation levels are 0.01 and 0.05, where 0.05 is equal to 1/n, n is 20 in our model.

We cannot find consistent results about level of maximum generation number in the existing studies, therefore we make further experimentation to determine the high and low levels. According to our pilot runs and considering both fitness and CPU time response types, the levels of 50 and 100 are selected for the analysis.

4. EXPERIMENTAL RESULTS

All the computer applications are carried on Sun HPC Server 4500 with 12 400mhz CPU, 3 GB memory, 45 GB disk and 20/40 GB 8mm tape unit. In ANOVA, 95% precision level is used for both response types.

Fitness Response

Our computational results indicate that; in terms of the fitness response, the number of machines in each workstation (M_i) is the dominating decision variable set, i.e. the total profit obtained merely depends on the level of this variable. There are two reasons; first, the machine operating cost is significantly larger than the buffer cost (25 times larger), so any alteration of M_i has more significant impact on total cost of the system than the buffer, and the unit cost of machine is high enough to affect the profit of the system. Second reason is that; even a unit alteration on one of M_i 's affect the throughput of the system drastically compared to B_i.

Moreover, only some specific combinations of M_i's lead to good solutions (high profit). More than 80% of trials, GA leads to a best solution with machine combination of 3 – 4 – 2 – 3 or 4 – 4 – 2 – 3. (It is a predictable result, since the first and the second workstations have the biggest processing times, so they require more machines compared to the other workstations).

The same combination of machine numbers, but different buffer levels does not produce significant alterations in total profit, since the cost of additional buffer position is almost nearly compensated by the increase in the throughput level.

Another experimental results is that, good solutions are highly dominant in the solution space. In other words, the solutions with good combination of machines have significantly more fitness value than the other solutions. The importance of this property reveals in GA application. As good solutions are very dominating in terms of fitness values, the selection procedure usually selects strings with these patterns for the potential parents of the following generation. Thus by the crossover operator, genetic material of these solutions is carried to the next generations. Therefore a rapid convergence to these good solutions occurs in GA, since every time children are produced from the parents with same machine number patterns. These solutions are differentiated according to their buffer levels.

According to ANOVA the significant main factors with respect to fitness response are initial population type, population size, maximum generation number and mutation probability. Following paragraphs give the interpretation of the results.

Starting with a seeded population has a positive effect on fitness response. Although it is not a general case for GA applications, there are empirical studies in the literature supporting such a conclusion [9]. With a seeded initial population GA starts with some good solutions at hand; hence like in the case of the random start, the algorithm does not loose time. Therefore there occurs a more rapid convergence to solutions with good machine levels. For specific good machine patterns, seeded GA has more time to try different combination of buffer levels, which increases the probability of hitting a better solution.

Population size and maximum generation number have also positive effects on best fitness value found. Increasing the population size (n), or generation number (m) enlarges the search space; apparently more individuals are processed, so probability of reaching better solutions increases.

Crossover probability is the only factor, which has an insignificant main effect. The reason behind this depends on the nature of

solution space. Because of the dominance characteristics mentioned above, after some initial generations, the individuals generally have 3 – 4 – 2 – 3 or 4 – 4 – 3 – 3 as machine combinations, corresponding to strings starting with:

10110110…… 11111010……

Although there are more patterns present in the solution space, and each pattern are in different amounts, since these two are the most common ones having largest of fitness values, for a rough estimate, both strings have 50% occurrences in a population. Then for 2/3 of trails, reproduction operator selects two parents with same starting strings. If selected two potential parents both have one of these patterns, then the crossover before the 8^{th} bit produce same individuals. Any cross-site after that point generates individuals with different buffer levels with same machine numbers. If somehow an inappropriate level (very low) of buffer is generated then the fitness decreases drastically and dominance of good solutions make this individual disappear in the next generation. Therefore on average crossover cannot produce poor individuals. In the other case, crossover generates individuals with some pattern of buffer levels which do not lead to low fitness values. From the preliminary analysis, we know that different buffer levels with good machine patterns do not have very significantly different fitness values. Thus crossover cannot make significant discrepancy on fitness.

If selection mechanism selects both strings as potential parents (1/3 of trials), then crossover after 1^{st}, 6^{th}, 7^{th}, and 8^{th} bit does not produce distinct children. The crossover from the 2^{nd}, 3^{rd} and 4^{th} site generates the children with machine patterns, 3 – 4 – 3 – 3, 4 – 4 – 2 – 3, 3 – 4 – 1 – 3, 4 – 4 – 4 – 3. From the generated strings all except 3 – 4 – 1 – 3 combination are present in good solutions' patterns, so they have close fitness values. The 3 – 4 – 1 – 3 combination results with a poor fitness value because of the high decline in throughput reasoning from the third workstation's bottleneck situation, and will disappear in the following generations due to the dominance of good solutions. There is no significant difference between the low level and high level of the crossover probability, since it generates similar individuals having close fitness values.

The mutation probability has a significant positive effect on fitness value. The power of mutation comes from its ability to search for various combinations of buffer levels. After the convergence to good solutions with specific machine number patterns, the improvements on fitness value is gathered by searching for different levels of buffer positions. Since mutation alters the value of a bit form 1 to 0, or vice versa, it easily provides diversity in solutions. While more combinations are searched, the probability of hitting a better solution increases automatically.

As a conclusion the initial population type, population size, maximum generation number and mutation probability presents significant behavior, and all of them have positive effect. Thus, it is more effective to use higher levels to maximize the fitness value.

CPU Time Response

Our results reveal that, the simulation time of the solutions generated by GA constitute to 95% of CPU time. Thus, the factors affecting the CPU time, in fact, alter the simulation time.

The simulation time depends on the load of the system. As the machine numbers and buffer levels increase the simulation time increases, since more production processes occur in the model. Like the fitness case, machine numbers are dominating set of decision variables according to the CPU time. The main reason is that, the production rate merely depends on the machine combinations. Buffer levels do not have significant effect on simulation time, except some extreme conditions (very low levels of buffer interrupting the production process).

GA converges to some good solutions having specific patterns of machine combinations rapidly. These solutions correspond to loaded systems, hence their simulation runs generally require more time compared to other solutions.

According to ANOVA, the significant main effects with respect to CPU time are population size, maximum generation number and mutation probability. The succeeding paragraphs give the interpretation of results.

The initial population type is insignificant with respect to CPU time. Although seeded populations converge to good solutions (loaded systems) more rapidly, a random start looses only a few generations time to converge to the same good set of solutions, and on the average these few generations' individuals' simulation time does not make significant impact on CPU time.

Population size and maximum generation number have negative effects on best fitness value found. Increasing the population size (n), or generation number (m) enlarges the search space; so CPU time increases.

The crossover probability is insignificant according to CPU time, as in the case of fitness response. The reasons are also valid for the CPU time case. Generally to cross or not does not make any sense, because in each case the produced children have similar patterns, and apparently they correspond to similar loaded systems.

Different from the other factors, mutation probability has a negative significant effect on CPU time, i.e. increasing the mutation probability p_m decreases the CPU time. The preliminary analysis on the solution space reveals that negative deviations from the good machine combinations lead to more significant decrease in simulation time, when compared with the increase in simulation time in the case of a positive alteration of machine numbers, which carry the model to a more loaded one. Another interpretation about the algorithm is that after some starting generations all the populations are composed of individuals having combinations of 3 – 4 – 2 – 3 or 4 – 4 – 2 – 3 as machine numbers leading to strings starting with:

String 1: 10110110.. String 2: 11110110..

A change from 1 to 0 leads a decrease in value of the variable, which corresponds a less-loaded system and the opposite results with an increased real value, which is a more loaded system. For string 1, the probability to go a more loaded system is 3/8 in terms of machine numbers, because there are 3 "0"'s present in the string. On the other hand a less-loaded system is achieved in 5 of 8 trials because of 5 "1"'s in the string. These probabilities are even more discriminating in string 2, because of six 1's and two 0's, corresponding to 6/8 and 2/8, respectively.

Since different levels of buffers do not have a significant impact on CPU time with good patterns of machine numbers, knowing that they have close simulation time, the CPU time decreases when the mutation probability increases, because in most of the trials, mutation carries the solution to a less-loaded system. This result is proved by comparing the probabilities, for first type of strings in 5 of each 8 trials system becomes a less-loaded one, and for second string in 6 of 8 trials same result is obtained.

The conclusion is that the population size and maximum generation number has positive effects on CPU time. Since we are trying to minimize the CPU time, low levels of these parameters are appropriate. Because the mutation has negative effect on CPU, it is better to set it at its high level. As the crossover probability

and initial population type do not have significant impact on CPU time, they can be used in any level.

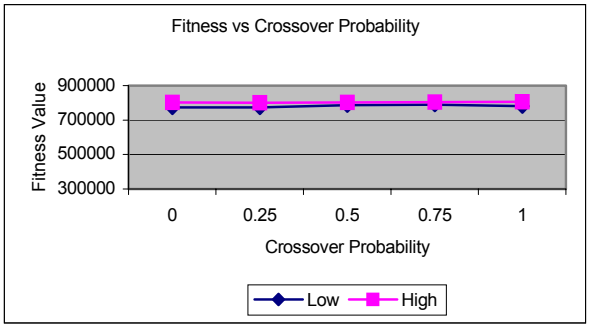
Further Experimentation

There are two conclusions which take our attention. One of them is the insignificance of crossover probability for each response types; and best performance of the high mutation probability in both cases. Thus, we decide to make further analysis of mutation probability, and crossover probability by single factor analysis. Two distinct experimental conditions are chosen, all the parameters are set to their high levels and low levels separately. Statistical inference about the results is obtained by comparing the performances using paired-t test.

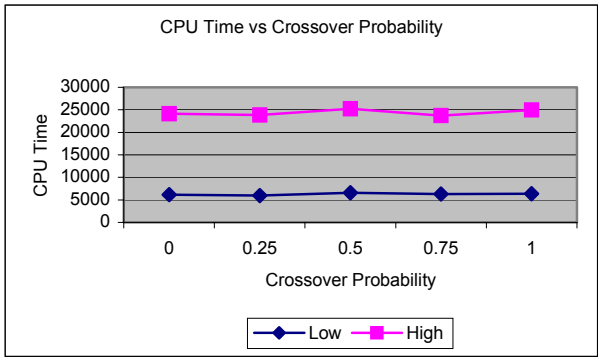
First, we investigate the significance of crossover probability. Our initial values were 0.5 and 1; we also examine the behavior of the performance measures under levels of p_c = 0, 0.25 and 0.75. Figure 1.a represents the change in fitness value due to the alteration of p_c, and Figure 1.b presents the results for CPU time.

Each point in the graphs corresponds to the average of independent GA applications. For fitness value, as observed from Figure 1.a, there is no significant difference in performance measure for different levels of crossover probability. The paired-t test statistically validates this conclusion, because the difference of means of two successive points presents an insignificant behavior. The CPU time graph also shows the insignificance of the crossover probability, which is also validated by paired-t test results of every consecutive pairs. This further analysis supports the results of our factorial design, such that crossover operator does not have a significant impact on GA performance under particular problem domains. The same type of analysis is applied for different levels of mutation probability. Figure 2.a presents fitness value according to different levels of mutation probability, while Figure 2.b shows the CPU time behavior.

According to the results of paired-t test, design point 2 (high level) presents insignificant behavior until p_m = 0.4, excluding the p_m = 0 case. But there is a significant improvement on fitness going from 0.3 to 0.4 level of mutation probability. The other respective points present insignificant behavior except p_m = 1.



a) Best Fitness with Different Levels of Crossover Probability
Figure 1: Analysis of Different Crossover Probabilities



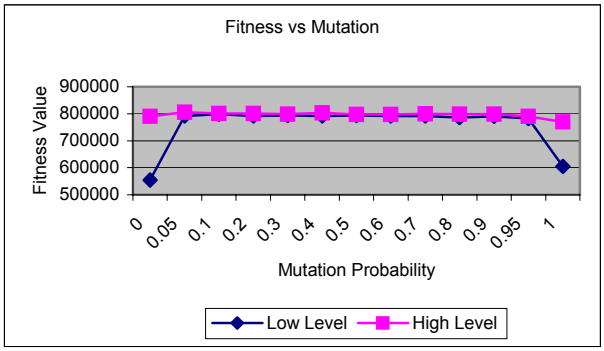
b) CPU Time with Different Levels of Crossover Probability
Figure 1: Analysis of Different Crossover Probabilities (cont'd)

The results obtained agree with the previous experimental design. The mutation operator is very effective in terms of finding better solutions. This idea can be observed best by looking at the design point 1 (low level) with p_m = 0. There is no mutation and GA cannot find even good solutions.

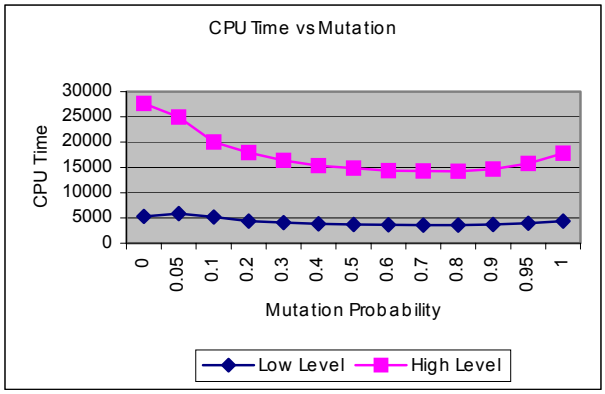
Increasing the mutation probability increases the fitness value until p_m = 0.4. The other probabilities are insignificant after this level when compared respectively until p_m = 0.95. There is a significant decrease in fitness at p_m = 1. This is because of the inadequate pattern of GA: High mutation distorts the building blocks of GA such that good characteristics of individuals cannot be properly transferred to the following generations.

Investigation of Figure 2.b shows that increments in mutation probability make significant reductions in CPU time covered. This result also agrees with our conclusions about test problem domain. As more mutation occurs more less-loaded systems are processed; hence simulation time decreases. But after point of 0.9 there is an increase in CPU time, which is in parallel with the fitness case. As the pattern of GA is disturbed, no convergence to good solutions occur, and unpredictable points from anywhere of solution space are processed. The search mechanism looses its logic, so increase in CPU time is observed.

From Figures 2.a and 2.b, it is concluded that even more increments than 0.05 in mutation probability lead better solutions in terms of maximum fitness and minimum CPU time. Mutation probability of 0.4 is the most convenient choice, since it has the maximum fitness and significantly less CPU time.



a) Best Fitness with Different Levels of Mutation Probability
Figure 2: Analysis of Different Mutation Probabilities



b) CPU Time with Different Levels of Mutation Probability

Figure 2: Analysis of Different Mutation Probabilities

5. CONCLUSION

In this study, we have proposed a new taxonomy of parameters of GA, numerical and structural, and examine the effects of numerical parameters on the performance of the algorithm in GA based simulation-optimization applications by the use of a test problem. Our aim is not to find the best parameter combination for a particular problem, but to come up with general conclusions. We start with the characteristics of the problem domain. The main characteristic features of our problem domain are:

- There is a dominance of a set of decision variables with respect to the objective function value of the optimization problem: The objective function value is directly related with the combination of this dominant set of variables. Alteration of the values of dominant variable alters the solution's performance substantially when compared with the other decision variables.

- The good solutions are highly dominant over *other solutions* with respect to the objective function value, but not significantly diverse among each other.

- Combining the above two features; good solutions have specific set of patterns in dominant set of decision variables, and these solutions are dominant over other feasible solutions among the solution space.

These properties of the problem domain generate a rapid convergent behavior of GA. According to our computational results, high mutation rates give better performance. GA mechanism creates a lock-in effect in the search space, i.e. the processed solutions have similar string patterns, hence high mutation rates decreases the risk of premature convergence and provides diversification in the search space in this particular problem domain. Due to the dominance crossover operator does not have significant impact on the performance of GA. Moreover, starting with a seeded population generates more efficient results. Because of the rapid convergent behavior of GA high levels of population size and maximum generation number parameters are inappropriate when the tradeoff between fitness improvement and CPU time covered is considered.

We conduct same type of factorial design analysis on the extensions of our test problem. Variations of objective function values and a constrained version of the problem are issues, but the experimental results agree with the previous statements about the best parameter levels for the particular problem domain under consideration.

As a future research direction, the same analyses can be carried out for different problem domains, and with different structural parameter settings, and even the interaction between the numerical and structural parameters could be investigated.

6. REFERENCES

[1] J.T. Alander, "On optimal population size of genetic algorithms",**Proceedings of CompEuro 92,** 1992, pp.65-70.

[2] J.A. Cruz, and J. Haddock, "A general scheme of simulation optimization using a genetic algorithm," Technical Report, Rensselaer Polytechnic Institute, New York, 1993.

[3] K.A. De Jong An analysis of the behavior of a class of genetic adaptive systems. Doctoral dissertation, University of Michigan, 1975.

[4] T.C. Fogarty, "Varying the probability of mutation in the genetic algorithm," **Proceedings of the Third International Conference on Genetic Algorithms**, 1989, pp.104-109.

[5] D. Goldberg, **Genetic algorithms in search, optimization and machine learning**, Addison-Wesley, reading, M.A, 1989.

[6] J.H. Holland, **Adaptation in Natural and Artificial Systems**. University of Michigan Press, Ann Arbor, 1975.

[7] A.M. Law, and M. G. McComas, "Simulation-based optimization," **Proceedings of the 2000 Winter Simulation Conference**, 2000, pp. 46-49.

[8] R. Nakano, and T. Yamada, "Conventional genetic algorithm for job shop problems, "**Proceedings of the 4th Int. Conference on Genetic Algorithms**, 1991, pp.474-479.

[9] C.R. Reeves, **Modern Heuristic Techniques for Combinatorial Problems**, Wiley, New York, 1993.

[10] J.D. Schaffer, R.A. Caruna, L.J. Eshelman and R. Das, "A study of control parameters affecting online performance of genetic algorithms for function optimization," **Proceedings of the Third International Conference on Genetic Algorithms**, 1989, pp. 51-60.

[11] G. Suresh, V.V. Vinod and S. Sahu, "A genetic algorithm for facility layout," **International Journal of Production Research**, Vol. 33, 1995, pp. 3411-3423.

[12] M.A. Wellman and D.D. Gemmill, "A genetic algorithm approach to optimization of asynchronous automatic assembly systems, "**The International Journal of Flexible Manufacturing Systems**, Vol. 7, 1995, pp. 27-46.