

A COMPARATIVE ANALYSIS OF GENETIC ALGORITHM WITH VARIABLE CROSSOVER AND INVERSION PROBABILITY FOR OPERATING SYSTEM PROCESS SCHEDULING PROBLEM

Er.Rajiv Kumar

PhD. Scholar, Dept. of Computer Science & Engg.
Singhania University Jhunjhunu, Rajasthan, India
rajiv_kumar_gill1@yahoo.co.in

Abstract: There are numerous approaches have been developed to solve job shop scheduling problems and machines process scheduling problem. Implementation of genetic algorithm for operating system process scheduling is a new idea . Genetic Algorithm is a robust technique for solve process scheduling and optimization problem. There are many type of genetic algorithms have been developed from simple genetic algorithm to complex parallel genetic algorithm. The performance of any genetic algorithm is depend on the proper parameter setting of operators used for a problem under consideration. In this paper we will analyze the performance of modified cross over genetic algorithm for operating system process scheduling problem. As scheduling problem is defined as NP hard problem. Modified genetic algorithm is usefully implemented for operating system process scheduling problem. We saw through the simulation result that when the probability of crossover and inversion operator changes then the performance and convergence state of genetic algorithm is changed considerably.

Keywords: Genetic algorithm, NP-hard, Operating system, Scheduling, Inversion.

INTRODUCTION

Process Scheduling is a difficult problem in any operating system. The performance of any operating system is greatly depends upon the proper scheduling of the process. The scheduling problem is consider as NP-hard and All of the algorithms used to optimize this class of problems have an exponential time; that is, the computation time increases exponentially with problem size. Over many years, most of the researchers thought that optimal scheduling is very difficult task to achieve. With the growing interest in Genetic Algorithms (GAs) and the GA is another promising global optimization technique [1] be used for operating process scheduling problem. The modified cross over genetic algorithm is first proposed by Davis [2] and apply this algorithm to operating system process scheduling problem by rajiv et al [3]. Genetic algorithms (Gas) are adaptive methods, which may be used to solve search and optimization problems. Genetic algorithms (GAs) were first proposed by the John Holland [4] in the 1960s.

These Genetic algorithms have been applied to many scientific and engineering problems [5][6][7]. The performance of the genetic algorithm is limited by some problem, typically premature convergence. This happens simply because of the accumulation of stochastic errors. If by chance, a gene becomes predominant in the population, then it just as likely to become more predominant in the next generation as it is to become less he predominant. If an increase in predominance is sustained over several successive generations and population is finite, then a gene can be spread to all members of the population. Once gene has converged in this way, it is fixed then crossover cannot introduce new gene values. This produces a ratchet effect, so that as generations go by, each gene eventually becomes fixed. This diverse effect can be minimizing by applying the inversion operator with suitable probability. Here we consider the operating system process

scheduling .In this paper we vary the cross over probability with respect to the inversion probability and we will analyze the result.

THE WORK OF PAPER

The major part of this paper, contained in section 2, will explain working of genetic algorithm and their application in process scheduling problem. The GA is robust techniques and it has no. of operators which have their own properties .The parameter setting in the genetic algorithm is concerned with the setting of applicable static values of the operators used. Ie crossover probability , inversion rate , population size etc. Accessible introduction can be found in the books by Davis [8] and Goldberg[9] .Section 3 describe the proposed structure of genetic algorithm. Section 4 explain the experimental setup for analysis and Section 5 shows the experimental results of the problem under consideration and section 6 is conclusion .

INTRODUCTION OF GENETIC ALGORITHM

Overview

The evaluation function, or objective function, provides a measure of performance with respect to a particular set of parameters. The fitness function transforms that measure of performance into an allocation of reproductive opportunities. The evaluation of a string representing a set of parameters is independent of the evaluation of any other string. The fitness of that string, however, is always defined with respect to other members of the current population. In the genetic algorithm, fitness is defined by: f_i / f_A where f_i is the evaluation associated with string i and f_A is the average evaluation of all the strings in the population. Fitness can also be assigned based on a string's rank in the population or by sampling methods, such as tournament selection. The execution of the genetic algorithm is a two-stage process. It starts with the current population. Selection is applied to the current population to create an intermediate population. Then recombination and mutation are

applied to the intermediate population to create the next population. The process of going from the current population to the next population constitutes one generation in the execution of a genetic algorithm. In the first generation the current population is also the initial population. After calculating f_i/f_A for all the strings in the current population, selection is carried out. The probability that strings in the current population are copied (i.e. duplicated) and placed in the intermediate generation is in proportion to their fitness.

Coding

Before a GA can be run, a suitable coding (or representation) for the problem must be devised. We also require a fitness function, which assigns a figure of merit to each coded solution. During the run, parents must be selected for reproduction, and recombined to generate offspring. It is assumed that a potential solution to a problem may be represented as a set of parameters (for example, the parameters that optimise a neural network). These parameters (known as genes) are joined together to form a string of values (often referred to as a chromosome. For example, if our problem is to maximise a function of three variables, $F(x; y; z)$, we might represent each variable by a 10-bit binary number (suitably scaled). Our chromosome would therefore contain three genes, and consist of 30 binary digits. The set of parameters represented by a particular chromosome is referred to as a genotype. The genotype contains the information required to construct an organism which is referred to as the phenotype. For example, in a bridge design task, the set of parameters specifying a particular design is the genotype, while the finished construction is the phenotype.

The fitness of an individual depends on the performance of the phenotype. This can be inferred from the genotype, i.e. it can be computed from the chromosome, using the fitness function. Assuming the interaction between parameters is nonlinear, the size of the search space is related to the number of bits used in the problem encoding. For a bit string encoding of length L ; the size of the search space is 2^L and forms a hypercube. The genetic algorithm samples the corners of this L -dimensional hypercube. Generally, most test functions are at least 30 bits in length; anything much smaller represents a space which can be enumerated. Obviously, the expression 2^L grows exponentially. As long as the number of "good solutions" to a problem are sparse with respect to the size of the search space, then random search or search by enumeration of a large search space is not a practical form of problem solving. On the other hand, any search other than random search imposes some bias in terms of how it looks for better solutions and where it looks in the search space. A genetic algorithm belongs to the class of methods known as "weak methods" because it makes relatively few assumptions about the problem that is being solved. Genetic algorithms are often described as a global search method that does not use gradient information. Thus, nondifferentiable functions as well as functions with multiple local optima represent classes of problems to which genetic algorithms might be applied. Genetic algorithms, as a weak method, are robust but very general.

Fitness Function

A fitness function must be devised for each problem to be solved. Given a particular chromosome, the fitness function returns a single numerical "fitness," or "figure of merit," which is supposed to be proportional to the "utility" or "ability" of the individual, which that chromosome represents. For many

problems, particularly function optimizations, the fitness function should simply measure the value of the function.

Selection

Determine which strings are "copied" or "selected" for the mating pool and how many times a string will be "selected" for the mating pool. Higher performers will be copied more often than lower performers. Example: the probability of selecting a string with a fitness value of f is f/ft , where ft is the sum of all of the fitness values in the population.

Individuals are chosen using "stochastic sampling with replacement" to fill the intermediate population. A selection process that will more closely match the expected fitness values is "remainder stochastic sampling." For each string i where f/ft is greater than 1.0, the integer portion of this number indicates how many copies of that string are directly placed in the intermediate population. All strings (including those with f/ft less than 1.0) then place additional copies in the intermediate population with a probability corresponding to the fractional portion of f/ft . For example, a string with $f/ft = 1:36$ places 1 copy in the intermediate population, and then receives a 0:36 chance of placing a second copy. A string with a fitness of $f/ft = 0:54$ has a 0:54 chance of placing one string in the intermediate population. Remainder stochastic sampling is most efficiently implemented using a method known as stochastic universal sampling. Assume that the population is laid out in random order as in a pie graph, where each individual is assigned space on the pie graph in proportion to fitness. An outer roulette wheel is placed around the pie with N equally spaced pointers. A single spin of the roulette wheel will now simultaneously pick all N members of the intermediate population.

Reproduction

After selection has been carried out the construction of the intermediate population is complete and recombination can occur. This can be viewed as creating the next population from the intermediate population. Crossover is applied to randomly paired strings with a probability denoted P_c . (The population should already be sufficiently shuffled by the random selection process.) Pick a pair of strings. With probability p_c "recombine" these strings to form two new strings that are inserted into the next population. In the proposed algorithm we use the modified crossover operator.

Good individuals will probably be selected several times in a generation; poor ones may not be at all. Having selected two parents, their chromosomes are recombined, typically using the mechanisms of crossover and mutation. The previous crossover example is known as single point crossover. Crossover is not usually applied to all pairs of individuals selected for mating. A random choice is made, where the likelihood of crossover being applied is typically between 0.6 and 1.0. If crossover is not applied, simply duplicating the parents produces offspring. This gives each individual a chance of passing on its genes without the disruption of crossover.

Mutation is applied to each child individually after crossover. It randomly alters each gene with a small probability. The next diagram shows the fifth gene of a chromosome being mutated: The traditional view is that crossover is the more important of the two techniques for rapidly exploring a search space. Mutation provides a small amount of random search, and helps ensure that no point in the search has a zero probability of being examined.

Convergence

The convergence of the genetic algorithm is concern with the uniformity in the population of solution. when 95 % of the population has the same result then we can say that the gene is converge. As the population converges, the average fitness will approach that of the best individual. A GA will always be subject to stochastic errors. One such problem is that of genetic drift. Even in the absence of any selection pressure (i.e. a constant fitness function), members of the population will still converge to some point in the solution space.

This happens simply because of the accumulation of stochastic errors. If, by chance, a gene becomes predominant in the population, then it is just as likely to become more predominant in the next generation as it is to become less predominant. If an increase in predominance is sustained over several successive generations, and the population is finite, then a gene can spread to all members of the population. Once a gene has converged in this way, it is fixed; crossover cannot introduce new gene values. This produces a ratchet effect, so that as generations go by, each gene eventually becomes fixed. The rate of genetic drift therefore provides a lower bound on the rate at which a GA can converge towards the correct solution. That is, if the GA is to exploit gradient information in the fitness function, the fitness function must provide a slope sufficiently large to counteract any genetic drift. The rate of genetic drift can be reduced by increasing the mutation rate. However, if the mutation rate is too high, the search becomes effectively random, so once again gradient information in the fitness function is not exploited.

STRUCTURE OF PROPOSED GA-BASED ALGORITHM

Algorithm GA (Modified crossover GA)

- (1) Begin
- (2) Initialize Population (randomly generated);
- (3) Fitness Evaluation;
- (4) Repeat
- (5) Selection(Roulette wheel Selection) ;
- (6) Modified crossover;
- (7)Inversion();
- (8) Fitness Evaluation;
- (9) Elitism replacement with Filtration;
- (10) Until the end condition is satisfied;
- (11) Return the fittest solution found;
- (12) End

EXPERIMENTAL SETUP

The Individual are randomly generated to form an initial population. Successive generations of reproduction and crossover produce increasing numbers of individuals.

Modified crossover operator with crossover probability P_c is 0.6 is taken and then change the inversion probability with respect to crossover probability. The experiment is performing with two-crossover probability (p_c) i.e.0.6 and 1.0 Crossover operator exploited the population or you can say that it can diversify the population. But due to the genetic drift some time the population is converge to the local optimal point, At that time crossover operation cannot diversify the population. The inversion operator is explorative in nature, it diversifies the population, but in general the probability of inversion is very low. So in our simulation We first have 0.1 inversion probability then we proceed with .01,.001.

Table 1. Parameters of GA

Parameter / Strategy	Setting
Population Size	20
Population Type	Generational
Initialization	Random
Selection	Roulette wheel
Crossover	Two Parents, Modified crossover
Crossover Probability	0.6 and 1.0
Variable Inversion Probability	0.1 0.01 0.001
Replacement strategy	Keep 80 % Best
Stopping Strategy	85 % Population converge
No. of process to be Schedule	5
Fitness criterion	Minimum Turn Around Time

SIMULATION RESULTS

In this experiment we have compare-modified crossover GA with two-crossover probability P_c ie 0.6 and 1.0 also with variable inversion probability .we find that the performance and convergence state of the GA is greatly affected.

CONCLUSION

In this paper, we examine the effect of varying inversion probability with respect to variable crossover probability. The combine variation of the probability of the two operators will have enhanced the performance of GA. We have found that when the probability of inversion is very low then the GA is premature converge and also when it is high then same situation is occur. So inversion probability should not be very high or very low. This is the case of inversion probability. Now in case of crossover probability, when the cross over probability is increase then the exploited power of the GA increase and no. of iteration decrease. So the performance of the GA is depends upon the proper parameter setting of operators of GA.

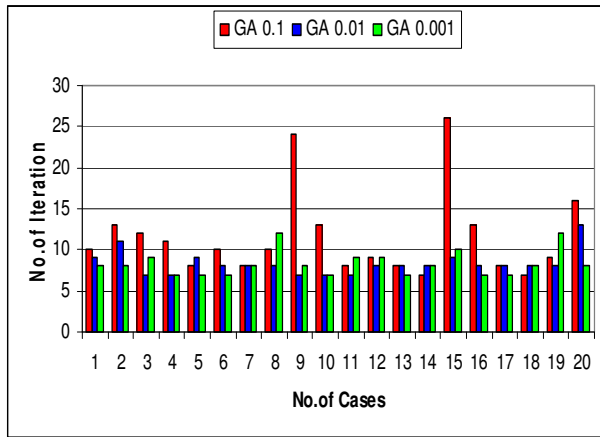


Figure 1.Comparison of Pi (0.1,0.01,0.001) with Cp=0.6,between No.of case and No. of Iteration,

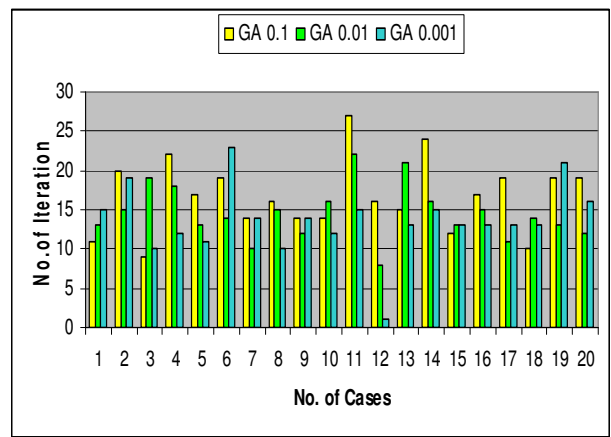


Figure 2.Comparison of Pi (0.1,0.01,0.001) with Cp=1.0 between No.of case and No. of Iteration

Table 2. Comparison Results

Sr.No.	Job Service Time in Second					C _{P=0.6}			C _{P=1.0}		
	J1	J2	J3	J4	J5	I _{Pi=0.1}	I _{Pi=0.01}	I _{Pi=0.001}	I _{Pi=0.1}	I _{Pi=0.01}	I _{Pi=0.001}
1	17	21	5	10	24	11	13	15	10	9	8
2	12	24	4	16	19	20	15	19	13	11	8
3	11	27	3	23	18	9	19	10	12	7	9
4	14	19	21	2	7	22	18	12	11	7	7
5	22	11	10	19	1	17	13	11	8	9	7
6	12	14	17	27	13	19	14	23	10	8	7
7	15	7	24	28	30	14	10	14	8	8	8
8	17	11	17	26	10	16	15	10	10	8	12
9	24	15	16	23	18	14	12	14	24	7	8
10	10	9	27	21	19	14	16	12	13	7	7
11	25	16	15	23	12	27	22	15	8	7	9
12	20	17	11	14	21	16	8	1	9	8	9
13	9	29	22	10	14	15	21	13	8	8	7
14	21	15	24	16	20	24	16	15	7	8	8
15	14	24	13	28	17	12	13	13	26	9	10
16	26	30	29	19	20	17	15	13	13	8	7
17	23	17	20	16	27	19	11	13	8	8	7
18	23	22	17	28	21	10	14	13	7	8	8
19	14	20	9	10	20	19	13	21	9	8	12
20	18	27	5	29	23	19	12	16	16	13	8
Sum						334	290	273	230	166	166
Mean						16.7	14.5	13.65	11.5	8.3	8.3

C_p = Crossover Probability=0.6,1.0
P_i =Probability of Inversion =0.1,0.01,0.001

REFERENCES

[1] David E.Goldberg.Genetic Algorithm in Search, Optimization and Machine Learning.Addision- Wesley Publishing Company,Inc.,1989.

[2] L.Davis,” Applying Adaptive Algorithms to Epistactic Domains”, in Proceeding of the Int. Joint Conf. on Artificial Intelligence (IJCAI’85), Los Angeles, CA. pp.162-164.

[3] R.Kumar , Dr. R .Kumar, S.Gill, A.Kaushik,” Genetic Algorithm Approach to Operating system Process scheduling problem ” ,International Journal of Engineering Science and Technology,2010,pp 4247-4252.

[4] Holland, J.H., 1975. “Adaptations in natural and artificial systems”, Ann Arbor: The University of Michigan Press

[5] M.Srinivas and L.M.Patnaik,”Genetic Algorithms: A survey”, IEEE computer Magazine,pp.17-26,june 1994

[6] L.Davis. Job-shop scheduling with genetic algorithms.Van Nostrand Reinhold,1990.

[7] S. French, (1982). Sequencing and Scheduling: And introduction to the Mathematics of the Job Shop, Ellis Horwood, Chichester.

AUTHOR



Er. Rajiv Kumar is the Assistant Professor in Computer Science & Engineering Department NC College of Engineerg , Israna (Panipat). He did B.Tech in

Computer Engineering from JMIT Radaur Yamuna Nagar , Kurukshetra university in 2002 and M.Tech in Computer Science & Engineering from Kurkshetra university in 2004. He is GATE scholar and now currently doing PhD research in the field of Genetic algorithm and their application in Scheduling problems from Singhania university Rajasthan .he has over 7 year of teaching & research Experience. He has design the GA simulator for operating system process scheduling.