IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 16, Issue 3, Ver. IX (May-Jun. 2014), PP 48-53
www.iosrjournals.org

Dynamic Process Scheduling and Sequencing Using Genetic Algorithm

Priyanka Rani, Shakti Nagpal
[1,2] *CSE Department, Geeta College of Engineering Naultha, Panipat*

***Abstract*:** *This paper present the implementation of genetic algorithm for operating system process scheduling. Scheduling in operating systems has a significant role in overall system performance and throughput. An efficient scheduling is vital for system performance. The scheduling is considered as NP hard problem .In this paper, we use the power of genetic algorithm to provide the efficient process scheduling. The aim is to obtain an efficient scheduler to allocate and schedule the process to CPU. we will evaluate the performance and efficiency of the proposed algorithm using simulation results.*

Keywords: *Genetic algorithm, NP-hard, CPU, Sequencing and Scheduling.*

I. Introduction

Single Processor Scheduling Problems are classical combinatorial problems. These problems fall under the category of NP-complete problems. And a number of methods have been devised to solve them. Among them FCFS, SJF, Priority Scheduling and RR are of much importance and are widely used for scheduling of jobs in a processor. This study is an effort to develop a simple general algorithm (genetic algorithm based) for obtaining optimal or near optimal solution.

Evolutionary algorithms (EA¨s) are population-based optimization algorithms that use biology-inspired mechanisms like mutation, crossover, natural selection, and survival of the fittest in order to refine a set of solution candidates iteratively [1],[2]. The advantage of evolutionary algorithms compared to other optimization methods is their black box character that makes only few assumptions about the underlying objective functions. The most popular technique in evolutionary computation research has been the genetic algorithm[4]. The different types of operators on which genetic algorithm is made are selection operators, reproduction operators (i.e. crossover and mutation operators). These operators are the main foundations over which a genetic algorithm is based. This study is an effort to develop a simple general algorithm (genetic algorithm based) for obtaining optimal or near optimal schedules for Single Processor Scheduling Problems with minimum computation effort even for large sized problems and comparing the minimum average waiting time using different scheduling algorithms (like FCFS, SJF and scheduling using genetic algorithm).

.

II. Genetic Algorithm

A genetic algorithm (GA) is a search technique used in computing to find exact or approximate solutions to optimization and search problems [3]. Genetic algorithms are categorized as global search heuristics. Genetic algorithms are a particular class of evolutionary algorithms (EA) that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover.

[ss**Simple Genetic Algorithm** The basic genetic algorithm is as follows:

- [start]: Genetic random population of n chromosomes (suitable solutions for the problem). •
- [Fitness]: Evaluate the fitness f(x) of each chromosome x in the population. •
- [New population]: Create a new population by repeating following steps until the new population is complete,

[selection]: select two parent chromosomes from a population according to their fitness (the better fitness, bigger are the chances to get selected).
[crossover]: With a crossover probability, cross over the parents to form new offspring (children). If no crossover asperformed, offspring is the exact copy of parent.

- [Mutation]: With a mutation probability, mutate new offspring at each locus (position in chromosome).
- [Accepting]: Place new offspring in the new population.
- [Replace]: Use newly generated population for a further sum of the algorithm.
- [Test]: If the end conditions are satisfied, stop and return the best solution in current population.
- [Loop]: Go to step2 for fitness evaluation.

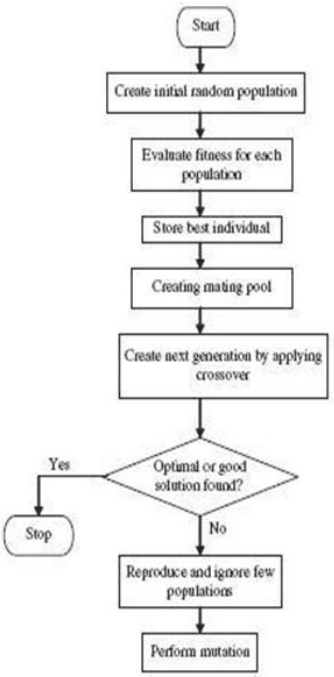

Fig.1. Flow Chart Of Genetic Algorithm

III. Problem Statement

In this, there is a number of processes (jobs) that are to waiting be processed by a single-processing system. The processor is assigned a job and remains busy until all the processes have been processed. Once a process is allocated the processor, remaining processes are forced to wait in the ready queue till their turn comes.

The objective is to find the schedule, which follows following constrains:

1. Processor has to process every job, without leaving any job unprocessed.
2. Once a job is executed completely, processor is allocated to next job. Each job should be processed only one time.
3. No preemption is allowed, i.e. processor can‟t switch between processes before completion of currently executing process
. 4. The total waiting time that for all the jobs till processor runs all the jobs in the ready queue should be minimum.

As already discussed previously, genetic algorithm is one of the optimization techniques that can be used to solve the problems of function maximization. It is often used as an optimization method to solve problems where very little is known about the objective function[9].

Genetic Operation

The operation starts with a population of random individuals, each corresponding to a particular candidate solution to the problem to be solved. Then, the best individuals survive, mate, and create offspring, originating a new population of individuals. This process is repeated a number of times, and usually leads to better and better individuals.

The steps of applying GA relating our problem at hand are:

1. Choosing an Encoding scheme.
2. Choosing Fitness function.
3. Choosing Operators.
4. Choosing Parameters.
5. Choosing an Initialization method and Stopping criteria

A. Encoding scheme

The application of a genetic algorithm to a problem starts with the encoding. The basis of genetics in nature is a chromosome. A particular solution to the problem can then be represented by a specific assignment of values to the decision variables. The set of all possible solutions is called the search space and a particular solution represent a point in that search space.

Real/Value encoding--Direct value encoding can be used in problems where some complicated value such as real numbers are used. Use of binary encoding for this type of problems would be very difficult. In value encoding, every chromosome is a string of some values. Values can be anything connected to problem, form numbers, real numbers or chars to some complicated objects

Chromosome A 1 5 3 2 6 4 7 9 8
Chromosome B 8 5 6 7 2 3 1 4 9

The next step after encoding is to select the fitness function.

B. Fitness function

The next step is to specify a function that can assign a score to any possible solution. The score is a numerical value that indicates how well a particular solution solves the problem.

The score is the fitness of the individual solution .The task of the GA is to discover solutions that have higher fitness values among the set of all possible solutions. A fitness function must be designed for the problem under consideration. For a solution (Chromosome), the fitness function must return a single numerical value, which is proportional to the "fitness" of that solution. Our aim is to find out individual having minimum average waiting time. So the individual who has minimum average waiting time is fittest as compared to other. So, The fitness function of a Solution Si is given by

Fitness (Si)=

$$\frac{\sum_{i=1}^{N} W_i}{N}$$

Where Wi is the waiting time of the process j.
N is the total no. of processes

C. Operators

Once the encoding and the fitness function are specified, the implementer has to choose selection and genetic operators to evolve new solutions to the problem being solved. The selection operator simulates the "survival-of-the-fittest". There are various mechanisms to implement this operator, and the idea is to give preference to better individuals. Selection replicates individuals with high fitness values and removes individuals with low fitness values.

Selection

Selection is the process of choosing two parents from the population for crossing. After deciding on an encoding, the next step is to decide how to perform selection i.e., how to choose individuals in the population that will create offspring for the next The higher the fitness function, the more chance an individual has to be selected. The selection pressure is defined as the degree to which the better individuals are favored. Selection has to be balanced with variation form crossover and mutation. Too strong selection means sub optimal highly fit individuals will take over the population, and it reduces the diversity needed for change and progress, too weak selection will result in too slow evolution.

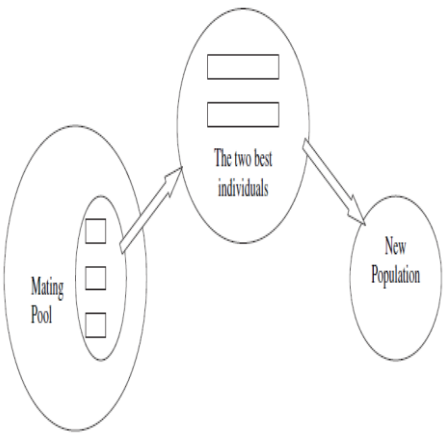

Fig.2. Selection process

Roulette wheel selection-- We use Roulette selection which is one of the traditional GA selection techniques.. The principle of roulette wheel selection is a linear search through a roulette wheel with the slots in the wheel weighted in proportion to the individual‟s fitness values[2][4]. A target value is set, which is a random proportion of the sum of the finesses in the population. The population is stepped through until the target value is reached. The expected value of an individual is that fitness divided by the actual fitness of the population. A slice of the roulette wheel is assigned to each individual, the size of the slice being proportional to the individual‟s fitness. Here N is the number of individuals in the population so the wheel is spun N times. On each spin, the individual under the wheel‟s marker is selected to be parents for the next generation.
This method is implemented as follows:
Roulette wheel selection is easier to implement but is noisy. The rate of evolution depends on the variance of fitness‟s in the population.

Crossover

Crossover is the process of taking two parent chromosomes and producing from them a number of offspring‟s. After the selection (reproduction) process, the population is filled with better individuals. Reproduction makes exact copies of good strings but does not create new ones. Crossover operator is applied to the mating pool with the hope that it creates a better offspring [7].

Crossover is a recombination operator that works in three steps:
i. The reproduction operator selects at random a pair of two individual strings for the mating/crossover.
ii. A cross site or cross point is selected at random along the string length.
iii. Finally, the position values are swapped between the two strings following the cross site.
Two point crossover--In two-point crossover, two crossover points are chosen and the strings between these points are exchanged between two parents involved in crossover operation. As in the Fig.below, the dotted lines indicate the crossover points. Thus the strings between these points are exchanged between the parents to produce new children for reproduction in the next generation.

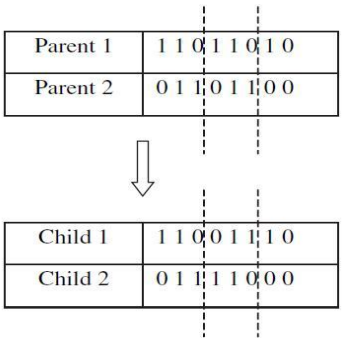

Fig.3. Two Point Crossover Process

Mutation

After crossover, the strings are subjected to mutation. Mutation prevents the algorithm to be trapped in a local minimum. Mutation plays the role of recovering the lost genetic materials as well as for randomly distributing genetic information.

Mutation is viewed as a background operator to preserve genetic diversity in the population. Mutation helps escape from local minima's trap and maintains diversity in the population. It also keeps the gene pool well stocked, and thus ensuring periodicity.

Mutation of a bit involves flipping a bit, changing 0 to 1 and vice-versa. There are various types of mutation schemes such as:
Flipping---Flipping of a bit involves changing 0 to 1 and 1 to 0 based on a mutation chromosome generated. The Figure 4.13 below explains mutation-flipping concept.

Parent	**1** 0 1 1 **0** 1 0 **1**
Mutation chromosome	**1** 0 0 0 **1** 0 0 **1**
Child	**0** 0 1 1 **1** 1 0 **0**

Table.1. Mutation Flipping

D. **Replacement Strategy**

When genetic operators (Two-point Crossover, flipping) are applied on the selected parents, one new chromosome is generated. This chromosome is added to the existing population . After that fitter chromosomes are selected from the new population .This process remain continue until we not get optimal solution.

E. Parameters

With an encoding, a fitness function, and operators in hand, the GA is ready to enter in action. But before doing that, the user has to specify a number of parameters such as population size, no. of processes [9]. For crossover the probability is 100% i.e. with every iteration/algorithm run crossover is performed always. Whereas the mutation operation performed after every five consecutive algorithm runs.

F. **Initialization method & stopping criteria**:

Following the initialization step, each individual is evaluated according to the user‟s specified fitness function.Thereafter, the GA simulates evolution on the artificial population of solutions using operators that imitate the survival-of-the-fittest and principles of natural genetics such as recombination and mutation [9]. A number of criteria can be chosen for this purpose, including among others, a maximum number of generations or time has elapsed, some predefined fitness value is reached, or the population has converged considerably [9]. For our problem, stopping criteria chosen is no. of iterations and initialization method is random generated population.

Pseudocode

START GA
Step 1. Initialize nop and population size [input no. of processes and population size from the user].
Step 2. Initialize brust time of each process randomly .
Step 3. Initialize population randomly [Using cpu_createperm function].
Step 4. Evaluate the fitness of each individual [Using fitness function].
Step 4. FOR no. of iterations DO
Step 5. Select two parents from the randomly generated population[Perform Roulette Wheel Selection using wheel function].
Step 6. Crossover two parents [Perform Two-Point Crossover using cyclic_crossover function].
Step 7. If i/5==0, Mutate two offspring produced. [Perform mutation at every fifth iteration using mutation_flipping function].
Step 8. Evaluate Fitness and return the two best fitted individual to the population [Perform weak parent replacement using replaceweak function].
Step 9. Is done=Optimization criteria met?
Step 10. If (not) then, i=i+1.[increase iteration number].
END FOR[If optimization criteria met, stop the algorithm].
Step 11. Output the best solution
END GA

G. **Result:**

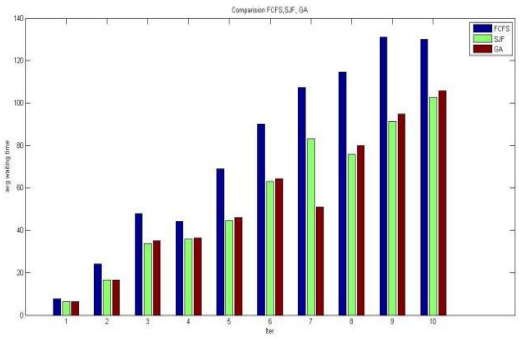

Fig.4.Comparison Of Scheduling Algorithms

Three algorithms for CPU scheduling are implemented The procedure followed while implementing it is first started with randomly generating the population. This process in GA has been called as encoding the input population. Then a selection technique has been employed over the encoded population and after that crossover is performed. After the crossover operation, mutation is performed, but after every 5 iterations. Then this operation is repeated for the required number of iterations. **The bar graph shown here** is about the results

showing the **average waiting time** of various algorithms. The results that came out from genetic algorithm are same or nearly same result as SJF.

IV. Conclusion

Waiting time optimization/minimization is the main point of discussion in this paper. This paper is based on implementing a scheduling based Genetic algorithm concepts and making a comparison of it with SJF and FCFS scheduling algorithms based on the average waiting time of these algorithms at different number of iterations.

Basically GA is one of the better function optimization methods generally employed now days. Population is randomly generated. The encoding scheme for this problem of function maximization is value/real encoding. And so a generated. different type of crossover method is applied for crossing the chromosomes in the population for producing better offspring's. And finally interchanging mutation is used while implementing this algorithm. In this paper mainly we solve the problem of CPU scheduling using the genetic algorithm. Since SJF gives provably optimal solution, but it has a drawback that we must know in advancelength of next CPU burst. And our algorithm gives same or nearly same result as SJF.After all the experimentation and implementation, results that came out are like, best one is genetic algorithm.

In future, further this work may be enhanced to incorporate multiprocessor system environment.

References

[1]. H.Nazif(2009). A Genetic Algorithm on Single Machine Scheduling Problem to Minimise Total Weighted Completion Time. European Journal of Scientific Research,Vol.35 No.3, pp.444-452

[2]. Garey, M. & Johnson, D. (1979). Computers and Intractability: a theory of NP- Completeness. W.H.Freeman, San Francisco Proceedings of the Second International Conference on Genetic Algorithms, pp. 252-256

[3]. Back, T., Fogel, David B. & Michalewicz, Z. (Eds.) (1997). Handbook of Evolutionary Computation. Computational Intelligence, Library Oxford University Press in cooperation with the Institute of Physics Publishing / CRC Press, Bristol, New York, ring bound edition. ISBN: 0-7503-0392-1, 978-0-75030-392-7, 0-7503-0895-8, 978-0-75030-895-3. [Also available online at:http://books.google.de/books?id=n5nuiIZvmpAC]

[4]. Back, T. (1996). Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms. Oxford University Press US. ISBN: 0-1950-9971-0, 978-0-19509-971-3.[Also available online at: http://books.google.de/books?id=EaN7kvl5coYC]

[5]. Tomassini, M. (1999). Parallel and Distributed Evolutionary Algorithms: A Review. In Miettinen, K., Makela, M., & Periaux, J. (Eds.), Evolutionary Algorithms in Engineering and Computer Science (pp. 113 - 133). Chichester: J. Wiley and Sons.

[6]. Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. Boston: Addison-Wesley

[7]. Sivanandam, S. N. & Deepa, S. N. (2008). Introduction to Genetic Algorithms. Springer

[8]. Davis, L. (1991). Handbook of Genetic Algorithm. Von Nostrand Reinhold, Newyork.

[9]. Lobo, Fernando Miguel (2000).The parameter-less genetic algorithm: rational and automated parameter selection for simplified genetic