Journal of Information and Operations Management
ISSN: 0976–7754 & E-ISSN: 0976–7762 , Volume 3, Issue 1, 2012, pp-38-42
Available online at http://www.bioinfo.in/contents.php?id=55

A MODEL TO STUDY GENETIC ALGORITHM FOR THE FLOWSHOP SCHEDULING PROBLEM

NEELAM TYAGI[1] AND VARSHNEY R.G.[2]

[1]Graphic Era University, Dehradun, Uttarakhand, India
[2]Department of mathematics, Graphic Era University, Dehradun, Uttarakhand, India
*Corresponding Author: Email- neelam24tyagi@gmail.com

Received: December 12, 2011; Accepted: January 15, 2012

Abstract - In this paper, we have introduced the concepts of a Genetic Algorithm and we described a Genetic Algorithm –based heuristic for solving the flowshop scheduling problems. The flowshop scheduling problem is a production problem where a set of *n* jobs have to be processed with identical flow patterns on *m* machines. Heuristics play a major role in solving NP –hard combinatorial optimization problems. This paper describes a Genetic Algorithm –based heuristic to make-span minimization on flowshop scheduling. We have compared our heuristic with the NEH (Nawaz, Enscore, Ham) Algorithm which is the most popular heuristic in the literature. The computational experience shows that the Genetic Algorithm approach provides competitive results for flowshop scheduling problems.

Keywords- Flowshop scheduling; make-span; Genetic Algorithm; NEH Algorithm

Citation: Neelam Tyagi and Varshney R.G. (2012) A Model To Study Genetic Algorithm For The Flowshop Scheduling Problem. Journal of Information and Operations Management ISSN: 0976–7754 & E-ISSN: 0976–7762, Volume 3, Issue 1, pp-38-42.

Introduction

In this paper we have introduced the model to study a Genetic Algorithm. All models are mathematical abstractions of the real physical world. The more assumptions one needs to make to get the model into a form where known mathematical structures can be used to address the real problem, the more uncertainty has crept into the modeling process. To ignore this uncertainty is to ignore the real world, and our understanding of it. But, we can make the models robust and credible by addressing the fact that complexity and uncertainty are inextricably related. The flowshop scheduling has been a very active and prolific research area since the seminal paper of Johnson[27]. The flowshop scheduling problem is a production problem where a set of *n* jobs have to be processed with identical flow patterns on *m* machines. When the sequence of jobs processing on all machines is the same we have the permutation flowshop sequencing production environment. We study the flowshop problems considering the following assumptions:

- The operation processing times on the machines are known, fixed and some of them may be zero if some job is not processed on a machine.
- Set-up times are included in the processing times and they are independent of the job position in the sequence of jobs.
- At a time, every job is processed on only one machine, and every machine processes only one job.
- The job operations on the machines may not be preempted.

Several heuristic approaches for the flowshop scheduling problem have been developed. In recent years, meta-heuristic approaches, such as Simulated Annealing, Tabu Search, and Genetic Algorithms, have become very desirable in solving combinatorial optimization problems because of their computational performance.

Literature Review

A significant research effort has been devoted for sequencing jobs in a flowshop with the objective of finding a sequence that minimizes the make-span. For problems with 2 machines, or 3 machines under specific constraints on job processing times, the efficient Johnson's Algorithm 27 obtains an optimal solution for the problem. The famous "Johnson's rule" is a fast O(nlogn) method for obtaining the optimal solution for the F2/prmu/C_{max} (two machines) and for some special cases with three machines. Later, Palmer 7

presented a heuristic to solve the more general m -machine PFSP. Campbell, Dudek and Smith 16 developed another heuristic which was basically an extension of Johnson's Algorithm to the m machine case. The list of heuristics is almost endless. Ruiz and Maroto 25 provided a comprehensive evaluation of heuristic methods and concluded that the famous NEH heuristic of Nawaz, Enscore and Ham 21 provides the best performance for the F/prmu/C_{max} problem.
Regarding Metaheuristics, there is also a vast literature of different proposals for the PFSP under different criteria. Within these types of techniques we noticed Genetic Algorithms (GAs), Simulated Annealing (SA), Tabu Search (TS) and other procedures are considered as hybrid methods. The first Metaheuristics permutation flowshop scheduling problem (PFSP) for Simulated Annealing Algorithms was proposed by Osman & Potts 17 and Ogbu & Smith 13 Genetic Algorithms are presented in Reeves 2 and Ruiz, Maroto and Alcaraz 24. Recently, other Metaheuristics like Ant Colony Optimization, a very fast Tabu Search (TS) approach, Scatter Search, an updated and comprehensive review of flowshop heuristics and Metaheuristics, Discrete Differential Evolution, Particle Swarm Optimization or Iterated Greedy are presented in Rajendran and Ziegler 3 Ruiz and Maroto 25, Onwubolu and Davendra 14, Tasgetiren et al 20. and Ruiz and Stützle 26, respectively. Recent and high performing approaches include parallel computing methodologies, like the one presented in Vallada and Ruiz 12. Apart from make-span minimization, the PFSP has been studied under many other criteria. For example, Vallada, Ruiz 12 and Minella 15 reviewed 40 heuristics and Metaheuristics for tardiness-related criteria.

Genetic Algorithm

Genetic Algorithms were developed by Holland in 1975. The Genetic Algorithm (GA) is a search technique based on the mechanics of natural Genetics and survival of the fittest (Goldberg 9. GA simulates the biological processes that allow the consecutive generations in a population to adapt to their environment. The Genetic Algorithm object determines which individuals should survive, which should reproduce, and which should die. Since Genetic Algorithms (GAs) are adaptive and flexible, the GAs were shown to be successfully applied to several optimization problems. For example, they have been applied to routing, scheduling, adaptive control, game playing, cognitive modeling, transportation problem, traveling salesman problems, optimal control problems, database query optimization, etc.

Methodology

The general procedures of the GA are as follows:

- Initialize a population of binary or non-binary chromosomes.
- Evaluate each chromosome in the population using the fitness function.
- Select chromosome to mate (reproduction).
- Apply Genetic operators (crossover and mutation) on chromosome selected.
- Put chromosomes produced in a temporary population.
- If the temporary population is full, then go to step 7. Otherwise; go to step 3.
- Replace the current population with the temporary population.
- If termination criterion is satisfied, then quit with the best chromosome as the solution for the problem. Otherwise, go to step 2.
- GAs are different from more normal optimization and search procedures in four ways.
- GAs work with a coding of the parameter set, not the parameters themselves.
- GAs search from a population of points, not a single point.
- GAs do not use derivatives or other auxiliary knowledge.
- GAs use probabilistic transition rules, not deterministic rules.

Elements of Genetic Algorithms

Population initialization and population size

The first element of the GAs is the size of population and how to generate the initial population. The initial population of chromosomes can be generated randomly or by using some heuristics that are suitable for the problem considered. The determination of the population size is a crucial element in the GAs. Selecting a very small population size increases the risk of prematurely converging to a local optimal. Large population size increases the probability of converging to a global optimal, but it will take more time to converge. In most of the GA applications, the population size was maintained at a constant. Reeves' used the NEH Algorithm to generate the initial population. He obtained one solution from this heuristic and generated others at random. Chen et al. [5] used the CDS Algorithm (a heuristic developed by Campbell, Dudek and Smith[3]) to construct the initial population. Chen et al. [5] also stated that the initial population for their GA would be generated using other well known heuristics (for example, Dannenbring's method [6] or a job insertion-based method). We use the ***m***-1 schedules produced by the CDS method and one schedule produced by using the Dannenbring's method to generate an initial population. This operator selects a member at random and swaps two randomly selected positions of the member to generate a new member for the initial population. This procedure will be repeated until the number of the members is equal to population size. Chen et al. [5] generated some trial examples and run their heuristics with different population sizes for each example. They found that the population with the size more than 60 cannot guarantee better results than the population with the size equal to 60. Therefore, we decided to use 60 as a population size for our heuristic.

Fitness function

The second element of the GAs is the fitness function, which is very important of the GAs process of evolution. The GA without fitness function is blind because the GA directs its search using historical data which are the fitness values of the chromosomes. The GA will use the fitness values of each chromosome to determine if the chromosome can survive and produce offspring, or die.There are different criteria used as fitness values of a structure. The most popular of these are make-span (maximum completion time) and total flow time. We use the make-span criterion in our heuristic. For a maximization problem, the measure of performance generally constitutes the fitness function. However, our objective is to minimize the make-span. For minimization problems, the method to determine the fitness function differs from the maximization problems. Fitness function of the strings can be calculated as follows:

$f(S_i(t))=\max\{C(S_i(t)\}-C(S_it)$ (1.1)

where $S_i(t)$ is the *i*th string in *t*th generation, $C(S_i(t))$ is the make-span of $S_i(t)$ and $f(S_i(t))$ is the fitness function of $S_i(t)$. Therefore, the probability of selection for a schedule $P(S_i(t))$ with lower make-span is high (Equation 1.2).

$P(S_i(t)) = f(S_i(t)) \;/\; \Sigma f$ (1.2)

This is also the criterion used for the selection of parents for the reproduction of children.

Reproduction (or Selection)of chromosomes

The selection of chromosomes to reproduce is the third element of the GA. Reproduction (or selection) is an operator that makes more copies of better strings in a new population. Reproduction is usually the first operator applied on a population. Reproduction selects good strings in a population and forms a mating pool. This is one of the reasons for the reproduction operation to be sometimes known as the selection operator. Several selection methods have been employed by several researchers to select among the best performers. Some of these methods are: the proportional selection scheme; the roulette wheel selection; deterministic selection; ranking selection; tournament selection, etc.

Roulette wheel selection - Roulette wheel selection is chosen, where the average fitness of each chromosome is calculated depending on the total fitness of the whole population. The chromosomes are randomly selected proportional to their average fitness.

Crossover

Crossover is used as the main Genetic operator and the performance of a GA is heavily dependent on it. It's a fourth element of GA. A crossover operator is used to recombine two strings to get a better string. It is important to note that no new strings are formed in the reproduction phase.

In the crossover operator, new strings are created by exchanging information among strings of the mating pool. A crossover operator is mainly responsible for the search of new strings even though mutation operator is also used for this purpose sparingly. This heuristic uses the LOX operator that is developed by Falkenauer and Bouffouix [11].

Mutation

Mutation is nearly always regarded as an integral part of a GA. Mutation generates an offspring solution by randomly modifying the parent's feature. It helps to preserve a reasonable level of population diversity, and provides a mechanism to escape from local optima. For each child obtained from crossover, the mutation operator is applied independently with a probability p_m (Mutation Probability).

Generations (iteration)

Now that there is no practicable rule to set suitable stopping condition and it is also impossible for GA to evolve with too long time in real application, the usual way is to set a limit to a number of generations. These three operators are simple and straightforward. The reproduction operator selects good strings and the crossover operator recombines good sub-strings from good strings together, hopefully, to create a better sub-string. The mutation operator alters a string locally expecting a better string. Even though none of these claims are guaranteed and/or tested while creating a string, it is expected that if bad strings are created they will be eliminated by the reproduction operator in the next generation and if good strings are created, they will be increasingly emphasized. Further insight into these operators, different ways of implementations and some mathematical foundations of Genetic Algorithms can be obtained from GA literature.

We used the number of generations (iterations) as the termination criterion. If the number of generations is low the probability of finding the best result is low. Otherwise if the number of generations is too high, the iteration time is too long.

Genetic Algorithm Based Heuristic

Now, we describe GA-based heuristic for the flowshop problems.

Step 1: Determine the initial population S(0) as described in an earlier section. The size of the population is 60. t = 0, NG= 0

Step 2: Calculate the fitness value, $f(S_i(t))$, of each string for population. (See Equation (1.1))

Step 3: Calculate the selection probability, $P(S_i(t))$, of each string for population. (See Equation 1.2)

Step 4: Select a pair of strings (parents) according to selection probabilities of the members of S(t) (using random numbers).

Step 5: Constitute the new strings (children) by applying the LOX operator to the parents.

Step 6: Apply the shift mutation to the children with probability 0.05 (P_m = 0.05).

Step 7: Put the new strings in S(t+l). If the size of population S(t +1) = 60 then NG = NG + 1 and go to Step 8, else go to Step 4

Step 8: If NG = 20 then stop, else go to Step2

A Model Representation [4] Of Genetic Algorithms

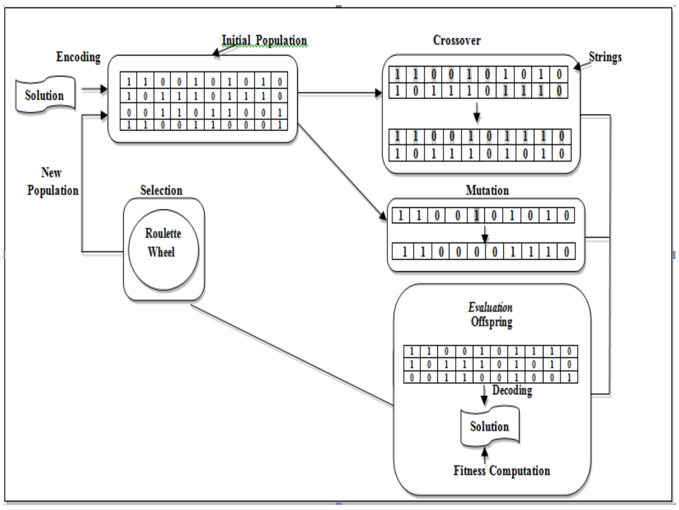

Fig.1- The fundamental cycle and operations of basic Gas (Gen and Cheng, 2000)

Neh Heuristic

The NEH heuristic was proposed by Nawaz et al 21 to solve the *m*-machine flowshop problem of minimizing make-span. The heuristic is based on the assumption that a job with more processing time on all machines will be given higher priority while a job with less processing time on all machines will receive lower priority. Accordingly, the two jobs with highest processing times are determined from the *n*-jobs problem. The best partial sequence for these two jobs is found by considering the two possible partial schedules. The relative positions of these two jobs with respect to each other are fixed in the remaining steps of the heuristic. Next, the job with the third highest processing time is determined and three partial sequences are tested in which this job is placed at the beginning, middle, and end of the partial sequence found before. The best partial sequence fixes the relative positions of these three jobs in the remaining steps of the heuristic. This procedure is repeated until all jobs are fixed and scheduled.

Computational Result

In order to examine the effectiveness of the GA-based heuristic, one comparison was made over a wide range of jobs and machines. We compared our heuristic with the NEH processing time is determined and three partial sequences are tested in which this job is placed at the beginning, middle, and end of the partial sequence found before. The best partial sequence fixes the relative positions of these three jobs in the remaining steps of the heuristic. This procedure is repeated until all jobs are fixed and scheduled.

Table 1- Relative Performance of Goa

M				
n	5	10	15	20
8	0.981	0.988	0.984	0.989
10	0.991	0.983	0.991	0.989
15	0.988	0.972	0.996	0.987
20	1.001	0.995	0.994	0.999
30	1.007	0.998	1.001	1.005

Algorithm which is the most popular heuristic in the literature. NEH is about three decades old heuristic but most

Table 2- Comparison of GA with the New

n	x	m	Generated Problems	Advantage GA	Even	Advantage NEH	% GA	% NEH
8	x	5	10	8	2	0	100	20
8	x	10	10	4	6	0	100	60
8	x	15	10	8	2	0	100	20
8	x	20	10	8	1	1	90	20
10	x	5	10	4	4	2	80	60
10	x	10	10	7	2	1	90	30
10	x	15	10	6	2	2	80	40
10	x	20	10	7	3	0	100	30
15	x	5	10	9	0	1	90	10
15	x	10	10	7	3	0	100	30
15	x	15	10	5	1	4	60	50
15	x	20	10	7	0	3	70	30
20	x	5	10	3	2	5	50	70
20	x	10	10	6	1	3	70	40
20	x	15	10	6	1	3	70	40
20	x	20	10	5	0	5	50	50
30	x	5	10	1	3	6	40	90
30	x	10	10	5	0	5	50	50
30	x	15	10	4	0	6	40	60
30	x	20	10	2	2	6	40	80
30	x	30	10	8	0	2	80	20
35	x	35	10	9	0	1	90	10
40	x	40	10	10	0	0	100	0
Total			230	139	35	56	76	40

Researchers still compare their heuristic with NEH or they include NEH in their Algorithms. Armentano et al 28. showed the improvement percentage of Tabu Search with diversification and intensification compared to NEH Algorithm. Koulamas 1 proposed a new heuristic called HFC (Heuristic Flowshop Scheduling with C_{max} objective) and compared HFC with NEH Algorithm. Ronconi 10 also compared his MM (Min/Max) Algorithm with NEH. Framinan et al 18. showed the excellent performance of the NEH in their Algorithm. They proposed a heuristic for mean/total flow time minimization in permutation flow shops. The heuristic exploits the idea of 'optimizing' partial schedules, already present in the NEH heuristic with respect to make-span minimization.The processing times were randomly sampled from a uniform distribution ranging from 1-20. In all, 230 problems were generated for 23 different combinations of job size and number of machines. It was not possible to solve problems larger than 40 x 40 because of software and machine limitations. This is caused by the built up computer memory requirements from the sizable GA population and the operations being carried out on it. The result of the two comparisons are presented in the following tables. In Table 1 the relative performance of the GA-based heuristic to the NEH was computed by CGA/CNEH where the C refers to the average make-span of the problems in the combination. Problems, used for calculation are the same as in Table 2 (200 runs were performed).In Table 2, the first column is the pairing of the number of jobs, ***n***, and the number of machines, m. The second column is the number of generated problems for the pairing. The third column and fifth column illustrate the number of times the best solution was obtained by the heuristic used, respectively. The fourth column shows the number of times that two heuristics in a comparison give the same make-span. The last two columns show the percentage of success of each heuristic, the total number of times that the heuristic gives the best solution (number of advantage + number of even) divided by the number of generated problems. (the method of Widmer and Hertz22. According to the results in Table-1 the GA-based heuristic obviously yields better average make-span than the NEH. Table 2 shows that in the 230 generated examples, the NEH gets better results than the GA-based heuristic only 56 times out of 230, while the GA-based heuristic is better 139 times out of 230. The NEH and the GA-based heuristic [23] give the same results 35 times out of 230.

Conclusions and Directions for Future Reserch

In this paper, we have introduced the fundamental model and described a GA-based heuristic for solving the flowshop scheduling problems. The Algorithm is easily implementable and performs quite effectively. Genetic Algorithms are easy to apply to a wide range of problems, from optimization problems like the traveling saleman problem, to inductive concept learning,scheduling, and layout problems. The results can be very good on some problems, and rather poor on others. Many scheduling problems are NP-hard problems. For such NP-hard combinatorial optimization problems, heuristics play a major role in searching for near-optimal solutions.. If only mutation is used, the Algorithm is very slow. Crossover makes the Algorithm significantly faster. In this GA-based heuristic, we generate a different parameter set for the Genetic operators. We protect the best schedule which has the minimum make-span, at each generation. Then we transfer this schedule to the next population with no change. This operation enables us to choose the higher crossover and mutation probability $p_c = 1$ (crossover probability) and $p_m = 0.05$ (mutation probability). So we increase the diversity of the population to get a better solution. We also show the excellent performance of the LOX operator.

Most researchers use 0.01 mutation probability in their heuristic. This heuristic uses 0.05 value of mutation probability and achieve good results. Using a mutation probability higher than 0.05 may reduce the convergence. Investigation of this would lead to a further study of GAs. According to the computational results, the GA-based heuristic success rate is 76% (in Table 2). Therefore, this heuristic is quite effective for flowshop scheduling problems. Also, the GA-based heuristic can be easily extended to solve flowshop problems with other criteria, such as total flow time, maximum tardiness, total tardiness, etc.
Future research directions [19] suggested here are intended to bridge the gap between the development of theory and practical applications of theory. Three areas of research are identified:

- Theoretical,
- Computational,
- Empirical research

References

[1] Koulamas C. (1998) *Eur J Opl Res* 105, pp. 66-71.
[2] Reeves C.R. (2004) *Improving the efficiency of tabu search for machine scheduling problems.*
[3] Rajendran C. and Ziegler H. (2004) *European Journal of Operational Research* 155 (2), pp. 426-438.
[4] Cengiz Kahraman., Orhan Engin., Ihsan Kaya. and Mustafa Kerim Yilmaz. (2008) *International Journal of Computational Intelligence Systems*, 1(2), pp. 134-147.
[5] Chen C.L., Vempati V.S. and Aljaber N. (1995) *Eur J Opl Res* 80, pp. 389-396.
[6] Dannenbring D. (1977) *Mngt Sci* 23, pp. 1174-1182.
[7] Palmer D.S. (1965) *Operational Research Quarterly*, 16 (1), pp. 101-107.
[8] Whitley D., Starkweather T. and Fuguay D. (1989) *Third International Conference on Genetic Algorithms.* pp. 133-140.
[9] Goldberg D.E. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning* (Addison Wesley: Reading, MA).
[10] Ronconi D.P. (2003) *Int J Prod Econ* 6, pp. 1-10.
[11] Falkenauer and S. Bouffoix. (1991) *International Conference on Roboticsa nd Automation. USA, pp*. 824-829.
[12] Vallada E. and R. Ruiz. (2009) *European Journal of Operational Research*, 193 (2), pp. 365-376.
[13] Ogbu F.A. and Smith D.K. (1990) *Computers & Operations Research*, 17(3), pp. 243–253.
[14] Onwubolu G.C. and Davendra D. (2006) *European Journal of Operational Research*, 171 (2), pp. 674-692.
[15] Minella G., Ruiz R., and Ciavotta M. (2008) *Journal on Computing,* 20 (3) , pp. 451-471.
[16] Campbell H.G., Dudek R.A., and Smith M.L. (1970) *Management Science Series B-Application,* 16(10), pp. 630-637.
[17] Osman I.H. and Potts C.N. (1989) *Omega-International Journal of Management Science,* 17 (6), pp. 551-557.
[18] Framinan J.M., Leisten R. and Rajendran C. (2003) *International Journal of Production Research*, 41(1), pp. 121-148.
[19] Jatinder N.D. Gupta. Edward F., Stafford Jr. (2006) *European Journal of Operational Research* ,169, pp. 699–711.
[20] Tasgetiren M.F., Liang Y.C., Sevkli M. and Gencyilmaz G. (2007) *European Journal of Operational Research*, 177 (3), pp.1930-1947.
[21] Nawaz M., Enscore E.E. and Ham I. (1983) *Omega-International Journal of Management Science,* 11 (1), pp. 91-95.
[22] Widmer M. and Hertz A. (1989) *Eur J Opl Res* 41, pp. 186-193.
[23] Etiler O., Toklu B., Atak M. and Wilson J. (2004) *Journal of the Operational Research Society* 55: pp. 830-835.
[24] Ruiz R., Maroto C. and Alcaraz J. (2006) *Omega-International Journal of Management Science*, 34 (5), pp. 461-476.
[25] Ruiz R. and Maroto C. (2005) *European Journal of Operational Research,* 165(2), pp. 479-494.
[26] Ruiz R. and Stützle T. (2007) *European Journal of Operational Research,* 177 (3), pp. 2033-2049.
[27] Johnson S.M. (1954*) Naval Research logistics Quarterly* pp. 61–68.
[28] Armentano V.A. and Ronconi D.P. (1999) *Comput Opns Res* 26, pp. 219-235.