International Journal of Computational Science and Mathematics.
ISSN 0974-3189 Volume 3, Number 1 (2011), pp. 93-98
© International Research Publication House
http://www.irphouse.com

Solving the Job-Shop Scheduling Problem by using Genetic Algorithm

Smita Verma, Megha Jain and Dinesh Choudhary

Department of Applied Mathematics & Computational Science,
S.G.S. Institute of Technology & Science, 23-Park Road,
Indore – 452003 (M.P.), India
E-mail: yvsmita@rediffmail.com, megha_267@rediffmail.com

Abstract

The job-Shop Scheduling is concerned with arranging processes and resources. Scheduling tools allow production to run efficiently. The goal in this paper is the development of an algorithm for the job-shop scheduling problem, which is based on genetic algorithms. Our intention is to prove, that even a relatively simple genetic algorithm is capable for job-shop scheduling.

Keywords: Job-Shop Scheduling Methodology, Genetic Algorithms, Evolutionary Computation

Introduction

Scheduling is an act of defining priority of arranging activities to meet certain requirements, constraints, or objectives. A schedule is a timetable for both jobs and machines. Time is a major constraint and we must utilize it in an optimum manner. Scheduling the production resources leads to increased efficiency, utilization and profitability for the enterprise. Job-shop is one of the most popular generalized production systems.

GA applications for JSSP have special chromosome representation as well as genetic operators to be applied to feasible schedules. In our case, the chromosomes are coded as a list of sets of numerical values for each particular schedule. A generalization of the GA is the Genetic Programming (GP) algorithm where each individual in a generation represents, with its chromosome, a feasible model solution. In this paper we need, a sequence of genetic operators that will define one genetic algorithm. There are two kinds of information defined for the GP algorithm; they are terminals (variable values and random numbers) and functions (mathematical functions used in the generated model).

Methodology

A large amount of work in JSSP has been reported over the past three decades using several approaches like optimal methods, dispatching rules, constraint-based intelligent systems, Lagrangian relaxation, neural networks, tailored heuristics, tree search techniques, inductive learning models, local search procedures and genetic algorithms.

In this paper, we have opted genetic algorithm approach by taking a 5job – 5 machine problem. We have solved it by traditional as well as by GA approach. The problem is given below. On traditionally solving this problem, we have obtained the results shown in Annexure I.

Table1: Operation Sequence.

	Sequence1	Sequence2	Sequence3	Sequence4	Sequence5
Job1	Machine3	Machine1	Machine2	Machine4	Machine5
Job2	Machine2	Machine3	Machine5	Machine1	Machine4
Job3	Machine1	Machine5	Machine4	Machine3	Machine2
Job4	Machine4	Machine3	Machine2	Mahine1	Machine5
Job5	Machine5	Machine3	Machine1	Machine2	Machine4

Genetic Algorithms in JSSP

We have generated schedules in a particular way in which the chromosome will be feasible after performing genetic operators. The decision management in JSSP distributes the jobs for each machine, selecting sometimes one task among the other alternatives so as to have a better fitness. Chromosome is coded with M*J (where M stands for machine & J for Jobs) values between 0 and 1, one for each decision.

This approach allows using the same traditional GA operators to solve the problem because the chromosome contains a sequence of numbers, all representing feasible schedules.

Crossover

Yamada and Nakano (1997) in most of their papers have introduced plenty of techniques that could be use in solving the job-shop problem. One of them is by making use of CB neighbourhood and DG distance. The idea of this technique is to evaluate a point x by the distance $d. x, p_2.$. Lets denote *parent1* and *parent2* as p_1 and p_2 . First, set x p_1 . Then, we generated the CB neighbourhood for x , $N. x.$ For each member, y_i , in $N. x$ we calculated the distance between the members and p_2 to produce $D. y_i$, $p_2.$. Then, we sort $D. y_i$, $p_2.$ in ascending order. Starting from the first index in $D_{sort.} y_i$, $p_2.$, we accepted y_i with probability one if the fitness value is less than the current fitness value$V. y_i. \leq V. x.$ Otherwise, we accepted it with probability 0.5. Starting from p_1 , we modified x step by step approaching p_2 . After some iteration, we will find that x will gradually loses p_1 's characteristics and started to inherit p_2 's characteristics although in a different ratios. We choose the *child* depending on the less DG distance between the *child* and both its parents.

Algorithm 1: Crossover

```
1. Let p1 and p2 be the parent solution.
2. Set x = p = q 1 .
3. Find CB Neighbourhood for x , N(x).
4. Do
      a. For each member yi N(x), calculate the distance between yi and p2 , ( )
         D yi , p2 .
      b. Sort the distance value in ascending order, ( ) Dsort yi , p2 .
      c. Starting from i =1, do
            i. Calculate the fitness value for yi , ( ) V yi .
           ii. If V(yi ) ≤ V(x) accept yi with probability one, and with
               probability 0.5
   otherwise.
          iii. If yi is not accepted, increase i by one.
   Repeat i-iii until yi is accepted.
      d. Set x = yi
      e. If V(x)≤V(q) then set q = x .
   Repeat 3-4 until number of iterations.

5. q is the child.
```

Mutation

Instead of using some random probability, we apply mutation if the DG distance between *parent1* and *parent2* are less than some predefined value. It is also defined based on the same idea as crossover. However, we choose the *child* which has the largest distance from the neighbourhood.

Algorithm 2: Mutation

```
1. Set x = p1.
2. Find Neighbourhood for x , N(x)
3. Do
      a. For each member yi N(x), calculate the distance between yi and p1 , ( )
         D yi , p1 .
      b. Sort the distance value in descending order, ( ) Dsort yi , p1 .
      c. Starting from i = 1, do
            i. Calculate the fitness value for yi , ( ) V yi .
           ii. If V(yi ) ≤ V(x) accept yi with probability one, and with
               probability 0.5
   otherwise.
          iii. If yi is not accepted, increase i by one.
```

```
Repeat i-iii until yi is accepted.
        d. Set x = yi .
        e. If V(x)≤V(q) then set q = x .
Repeat 2-3 until number of iteration.
    4. q is the child.
```

Acceptance Criterion

The final and the most important step in the GA procedure is to choose the individual to be replaced by *child*. It is widely known that we always choose the fittest individual to reproduce in the next iteration. In Yamada and Nakano (1997), they did not consider or choose the child with the same fitness value with other population members. However, by not even considering that child, we may lose the good individual without considering its abilities to be evaluated. So, in this paper, after considering the worst individual in the population, we also consider if the child has the same fitness value with the member of population. Instead of dropping that child, we replaced the old one with the child assuming that we have given chance for the old individual to reproduce. Noted that we could not take both of the individuals to avoid having problem later, we might have problem falling in the local optima. The algorithm for the whole procedure is shown

1. *Initialize population: Randomly generated a set of 10 schedules including the schedules obtained by some priority rules.*
2. *Randomly select two schedules, named them as p1 and p2 . Calculate DG distance between p1 and p2 .*
3. *If DG distance is smaller from some predefined value, apply Algorithm 2 to p1 . Generate child. Then go to step 5.*
4. *If DG distance is large, we apply Algorithm 1 to p1 and p2 . Generate child.*
5. *Apply neighbourhood search to child to find the fittest child in the neighbourhood. Noted it as child'.*
6. *If the makespan for the child' is less than the worst and not equal to any member of population, replace the worst individual with child'. If there is a member having the same makespan value, replace the member with the child'.*
7. *Repeat 2-6 until some termination condition are satisfied.*

Results and Discussions

Consider a 5 jobs and 5 machines problem with the operation sequence and the processing time for each operation have been determined in Table1. We run the program for five times using the population size = 10, number of iterations for mutation is 100 and crossover is 200. The algorithm was terminated after 200 generations. From the result, it can be shown that the combination of critical block, DG distance and genetic algorithm could provide a result as good as other methods. From Table, we could see that the last job processed is job 5 on machine 4. So, our makespan value for this problem is 34. The result also gives us the job sequence for each machine to process, the starting time and the finish time for each operation. For example, on machine 1, we start to process job 3 at time 0 and finished at 7. Then we process job 1, followed by job 4, job 5 and job 2.

We have applied both types of initial population to the data. First we used the combination of schedules we generated using the priority rules and the randomly generated schedules as the initial population. From the five runs, we find the optimum before the generation exceeded 100.

From the five runs, we could conclude that if we used the randomly generated schedules as the initial population, we will only find the optimum value at generation larger than 100. However, both results gave the same makespan value which is 34.

Conclusion

The study on GA and job shop scheduling problem provides a rich experience for the constrained combinatorial optimization problems. Application of genetic algorithm gives a good result most of the time. Although GA takes plenty of time to provide a good result, it provides a flexible framework for evolutionary computation and it can handle varieties of objective function and constraint.

For further research, the technique in this paper would be applied to a larger size problem to see how it performed.

References

[1] Pinedo and X. Chao. Operation Scheduling with Applications in Manufacturing and Services. McGraw-Hill International Editions. (1999).

[2] T. Jensen and T. K. Hansen. Robust Solutions to Job Shop Problems. Proceedings of the 1999 Congress on Evolutionary Computation, pages 1138-1144. (1999).

[3] French. Sequencing and Scheduling : An Introduction to the Mathematics of the Job Shop. John Willey & Sons Inc, New York USA.(1982).

[4] Yamada and R. Nakano. Genetic Algorithms for Job-shop Scheduling Problems. Proceedings of Modern Heuristic for Decision Support. Pp. 67-81, UNICOM Seminar, 18-19 March 1997,London. (1997)

[5] T. Yamada and R. Nakano. Scheduling by Genetic Local Search with Multi-Step Crossover. The Fourth International Conference on Parallel Problem Solving from Nature, Berlin, Germany. (1996).

[6] T. Yamada and R. Nakano. A Genetic Algorithm with Multi-Step Crossover for Job- Shop Scheduling Problems. International Conference on Genetic Algorithms in Engineering Systems: Innovations and Application (GALESIA '95). (1995)

Annexure I

On traditionally solving the problem we have the following results.

List of Operations (Job.Op)	Process Time Remaining	Operations Scheduled (Machine)	Start Time	Completion Time
(1,1), (2,1), (3,1), (4,1), (5,1)	27, 20, 31, 21, 25 (3,1)	1	0	7
(1,1), (2,1), (3,2), (4,1), (5,1)	27, 20, 24, 21, 25 (1,1)	3	0	2
(1,2), (2,1), (3,2), (4,1), (5,1)	25, 20, 24, 21, 25 (5,1)	5	0	5
(1,2), (2,1), (3,2), (4,1), (5,2)	25, 20, 24, 21, 20 (1,2)	1	7	15
(1,3), (2,1), (3,2), (4,1), (5,2)	17, 20, 24, 21, 20 (3,2)	5	7	15
(1,3), (2,1), (3,3), (4,1,), (5,2)	17, 20, 16, 21, 20 (4,1)	4	0	4
(1,3), (2,1), (3,3), (4,2), (5,2)	17, 20, 16, 17, 20 (2,1)	2	0	6
(1,3), (2,2), (3,3), (4,2), (5,2)	17, 14, 16, 17, 20 (5,2)	3	5	12
(1,3), (2,2), (3,3), (4,2), (5,3)	17, 14, 16, 17, 13 (4,2)	3	12	17
(1,3), (2,2), (3,3), (4,3), (5,3)	17, 14, 16, 12, 13 (1,3)	2	15	19
(1,4), (2,2), (3,3), (4,3), (5,3)	13, 14, 16, 12, 13 (3,3)	4	15	19
(1,4), (2,2), (3,4), (4,3), (5,3)	13, 14, 12, 12, 13 (2,2)	3	17	22
(1,4), (2,3), (3,4), (4,3), (5,3)	13, 9, 12, 12, 13 (5,3)	1	15	18
(1,4), (2,3), (3,4), (4,3), (5,4)	13, 9, 12, 12, 10 (1,4)	4	19	25
(1,5), (2,3), (3,4), (4,3), (5,4)	7, 9, 12, 12, 10 (4,3)	2	19	24
(1,5), (2,3), (3,4), (4,4), (5,4)	7, 9, 12, 7, 10 (3,4)	3	22	31
(1,5), (2,3), (3,5), (4,4), (5,4)	7, 9, 3, 7, 10 (5,4)	2	24	30
(1,5), (2,3), (3,5), (4,4), (5,5)	7, 9, 3, 7, 4 (2,3)	2	22	26
(1,5), (2,4), (3,5), (4,4), (5,5)	7, 7, 3, 7, 4 (2,4)	1	26	29
(1,5), (2,5), (3,5), (4,4), (5,5)	7, 4, 3, 7, 4 (4,4)	1	29	33
(1,5), (2,5), (3,5), (4,5), (5,5)	7, 4, 3, 3, 4 (1,5)	5	26	33
(2,5), (3,5), (4,5), (5,5)	4, 3, 3, 4, (2,5)	4	29	31
(3,5), (4,5), (5,5)	3, 3, 4, (5,5)	4	31	35
(3,5), (4,5)	3,3, (3,5)	2	31	34
(4,5)	3, (4,5)	5	33	36