IE 517 HW3
Alican Yılmaz(2016402093)
JUNE 2021

1.Problem Introduction

Permutation flow-shop problem is a well-known combinatorial optimization problem in the domain of scheduling problems. Finding optimal solution by commercial solvers can be too hard when the problem involves high number of jobs and machines. To find a reasonably good solution in limited time, several heuristics have been developed. In this work, the aim is to find a permutation of jobs that will minimize the makespan of the flow-shop by using Genetic Algorithm (GA), which is a well-known population-based metaheuristic that mimics biological evolution. In section 2, the applied genetic algorithm will be briefly explained with its solution representation and its necessary operators (e.g., mutation, crossover etc.). In section 3, numerical results will be discussed with given summary tables for each of the 3 instances. In section 4, final conclusion about the implemented algorithm and its limitations will be commented.

2. Genetic Algorithm

The algorithm was implemented by the combination of genetic operators defined in Section 2.2. The selection of parents is conducted such that the selection probability of parents is a decreasing function of makespan. In the literature, there are several crossover/mutation operators that are known to work well in this problem type. Here, in the selection process of the operators, I refer to the studies [1] and [2] by Murata T. et. al and Reeves C.R. In those, several different cross-over and mutation operators have been experimented in flow-shop scheduling problem and good performing operators were commented. Thus, in this work, I preferred to follow the recommendations of them and used the operators whose success has been experimentally proved before. For the stopping condition, I preferred to use CPU time-limitation instead of total number of generations elapsed as it might be more convenient if the problem has dynamic aspects [2]. P_c and P_m are selected as 1, (i.e., %100 probability of crossover and mutation) as those specifications seemed to work best in this problem setting [1]. Below, high level pseudo-code of the implemented algorithm is explained:

GA Pseudocode:

1. **Initialization:** Choose N_{pop}, P_c, and P_m. Stopping condition is chosen.
2. **Generation:** Generate N_{pop} initial solutions randomly.
3. *Repeat* the following *until* stopping condition is met:
4. **Selection:** Select N_{pop} pair of parents according to the selection probability.
5. **Crossover:** Apply crossover operator with probability P_c to each pair of parents to obtain N_{pop} children.
6. **Mutation:** Apply mutation operator with probability P_m to each child to obtain N_{pop} children.
7. **Update Population:** Randomly remove one solution from the current population and add the best solution from the previous population to the current population.
8. *End*
9. **Calculate Statistics:** Calculate the statistics of the population.

2.1. Solution Representation

Since the problem includes sequencing of the jobs, "*permutation representation*" is preferred in this problem. For instance, [1,3,5,2,7,6,4] means that the jobs are processed in all of the machines in the order of 1->3->5->2->7->6->4. Crossover and mutation operators are selected so that there is no infeasibility risk after each operation.

2.2. Genetic Operators

In this section, the genetic operators used in the algorithm will be examined. The function that calculates the objective of the solution will also be explained, as it is a key part of the problem.

2.2.1. Selection

The selection of parents is based on the probability distribution described below:

$$P([k]) = \frac{2k}{Npop * (Npop+1)} \quad ,$$

where k refers to the kth order of the solution in the population in ascending order of fitness [2]. (i.e., descending order of makespan.). Thus, it requires the ordering of the solutions in their corresponding population based on their fitness values before the implementation.

2.2.2. Crossover

1-point crossover is applied as it has been found to be one of the successful operators in this problem setting. However, due to fast convergence problem, it is recommended to be used in combine with mutation operator. 1-point crossover emphasizes *"exploitation"* over *"exploration"*. So, it is expected that incorporation with mutation restores the balance [2]. In 1-point crossover, one crossover point X is selected randomly, and pre-X points of Parent-1 are taken as they are. The rest of the points are selected from Parent-2 in ordered way, and by taking each "legitimate" element from it. The figure-1 from [2] illustrates the 1-point crossover process:

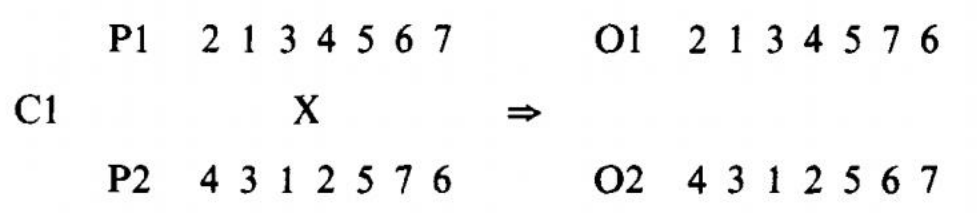

Figure-1: 1-point crossover process

2.2.3. Mutation

Shift-change mutation is used in this problem, as it was experimentally proved to be a successful operator in this problem setting [1]. In this mutation, a random job at position i is removed and inserted at another position randomly as shown in Figure-2 below:

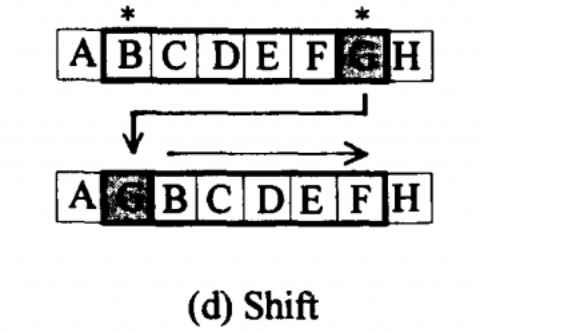

Figure-2: Shift-change mutation process

2.3. Calculating Objective of the Solution

The total makespan of the given job-order can be modeled recursively as follows:

$$F(j,i)= \text{Max}\{F(j,i-1) , F(\text{job before } j, i)\} + p(j,i)$$

where F(j,i) is the time at which job j leaves machine i and p(j,i) is the processing time of job j at machine i. here F(j,i-1) shows the time at which job j leaves previous machine and F(job before j, i) shows the time at which machine i is available to process job j. However, the stopping (discontinuity) conditions must be specified in that recursive formula. There is discontinuity at first machine and at first job. The implementation of the objective calculation function can be found on the Python code in Appendix.

3. Numerical Results

Instance: 8 jobs 4 machines				
Best Solution (sequence of jobs processed by the machines)	Obj. val. of the best chromosome	Avg. obj. val. of the best chromosomes	% Gap of the best chromosome	CPU Time (s)
[7, 4, 2, 5, 6, 1, 0, 3]	4534	4955.4	0	180

Instance: 13 jobs 5 machines				
Best Solution (sequence of jobs processed by the machines)	Obj. val. of the best chromosome	Avg. obj. val. of the best chromosomes	% Gap of the best chromosome	CPU Time (s)
[2, 4, 0, 5, 7, 1, 6, 9, 10, 11, 8, 12, 3]	937	954	1.8	300
Instance: 20 jobs 5 machines				
Best Solution (sequence of jobs processed by the machines)	Obj. val. of the best chromosome	Avg. obj. val. of the best chromosomes	% Gap of the best chromosome	CPU Time (s)
[3, 15, 19, 11, 4, 2, 5, 8, 0, 14, 16, 9, 17, 6, 1, 13, 18, 7, 12, 10]	1323	1361.5	1.6	300

In the first instance N_{pop} is specified as 5 and the stopping time is selected as 3 minutes. The model performed quite well on the first instance. In the second and third instances the stopping condition is selected as 5 minutes and it managed to find a solution within %2 Gap.

4. Conclusion & Further Remarks

In all three instances, the developed genetic algorithm gave good results; it found solutions within %2 gap in all of the instances. However, the model still could be improved further. As discussed in [1], construction heuristic could be applied to obtain better initial population instead of choosing randomly. P_m and P_c could be updated adaptively so that more solution space could be searched. Also, local search could be applied after each crossover operation in terms of better population management. Still, I believe, the results were satisfactory enough and gave relatively good results, if not the global optimum.

5.References

[1] Murata, T., Ishibuchi, H., & Tanaka, H. (1996). Genetic algorithms for flowshop scheduling problems. *Computers & Industrial Engineering*, *30*(4), 1061–1071. https://doi.org/10.1016/0360-8352(96)00053-8

[2] Reeves, C. R. (1995). A genetic algorithm for flowshop sequencing. *Computers & Operations Research*, *22*(1), 5–13. https://doi.org/10.1016/0305-0548(93)e0014-k