Minimum Open Shop Scheduling

Computational intelligence project
Faculty of Mathematics
University of Belgrade

Marija Škorić
mi18063@alas.matf.bg.ac.rs
Mateja Ivančević
mi18184@alas.matf.bg.ac.rs

April 2023

Table of contents

1 Description

Open-shop scheduling is an optimization problem and a variant of optimal job scheduling. In a general job-scheduling problem, we are given n jobs J1,J2,....,Jn of varying processing times which need to be scheduled on m machines with varying processing power, while trying to minimize the makespan - the total length of the schedule. In open-shop scheduling specifically, each of n jobs consists of a set of operations O1,O2,...,Om which need to be processed in an arbitrary order on m given machines.

2 Definition

The input to the open-shop scheduling problem consists of a set of n jobs, another set of m workstations, and a two-dimensional table of the amount of time each job should spend at each workstation. Each job may be processed only at one workstation at a time, and each workstation can process only one job at a time.

The goal is to assign a time for each job to be processed by each workstation, so that no two jobs are assigned to the same workstation at the same time, no job is assigned to two workstations at the same time, and every job is assigned to each workstation for the desired amount of time. The usual measure of quality of a solution is its makespan - the amount of time from the start of the schedule (the first assignment of a job to a workstation) to its end (the finishing time of the last job at the last workstation).

3 Implementation

3.1 Input

The program is given multiple test files that all have the same format and contain the following data:

1. number of jobs - n
2. number of machines and operations of each job - m
3. array of operation lengths for each job - processing times
4. upper and lower bound of each instance

We implemented the random number generator described in Taillard[1] and generated the test instances using the given time and machine seeds. There are 10 instances for each of the 3 problem sizes: 4 machines and 4 jobs, 5 machines and 5 jobs, and 7 machines and 7 jobs.

3.2 Initial solution

The initial solution is a dense feasible schedule created by following a dispatching rule described in Liaw[2]: Whenever a machine is idle, select, among its available operations, the operation belonging to the job that has the longest total remaining processing time on other machines to process next. An operation is said to be available if it has not been processed yet, and no other operation belonging to the same job is currently being processed. If no such operation exists, the machine remains idle until the next operation has been completed on some other machine. If more than one machine is idle at the same time, select the machine with the longest total remaining processing time to schedule next.

We provide this carefully constructed schedule to an optimization method as a starting point in order to make it more efficient.

3.3 Makespan

Makespan is the amount of time that passes from the start of the schedule (the first assignment of a job to a workstation) to its end (the finishing time of the last job at the last workstation).. Given a feasible schedule, we calculate the makespan by assigning the operations to their designated machines based on the machine availability. In case of a tie situation, we choose the machine with the longest remaining processing time. If that doesn't resolve the tie, we choose the machine who's next operation is a part of the job with longer remaining processing time left after being processed on the given machine.

4 Optimization

Three optimization methods were attempted:

1. Tabu Search
2. Genetic Algorithm
3. Simulated Annealing

4.1 Tabu Search

Tabu search is an iterative improvement approach designed for getting a global optimum solution for combinatorial optimization problems. For every current solution, starting from the given initial one, we construct a neighborhood by exchanging pairs of operations in the schedule. The algorithm iteratively moves from the current solution to its best neighbor, even if it's worse than the current solution s, until a maximum number of iterations is reached. Finally, the algorithm returns the best schedule it encountered and its makespan. In order to avoid cycling to some extent, we forbid moving to recently visited solution by keeping them in the tabu list for a certain number of iteration, which is the length of the tabu list. We modified the tabu list size according to the problem size, making it about 1/10 of the neighborhood size for the given problem. Tabu list sizes were 4,6 and 18 for 4x4,5x5 and 7x7, respectively. If the tabu list is full, we remove its oldest member, making that solution available again.

4.2 Genetic Algorithm

We implemented the permutation genetic algorithm as described in Khuri, Miryala[3]. Each operation of a job is given a unique number. We first number the operations of job J1 , then the operations of job J2 , and so on. The chromosome is a permutation of numbers ranging from 0 to (number of operations - 1). We use generational genetic algorithm with roulette wheel selection, uniform crossover, and swap mutation. Since our goal is to minimize the makespan, each chromosome in the population occupies a slot on the roulette wheel of size inversely proportional to its fitness value.

4.2.1 Parameters

We used a generational genetic algorithm with uniform crossover rate 0.6, mutation rate with 0.1 and roulette wheel selection which is inversely proportional to the fitness values of each chromosome.

Number of generations was 500 with 200 chromosomes.

4.2.2 Makespan

For each operation in a chromosome, we determine the job number, the machine number and duration of the operation. Then, we schedule that operation at the earliest possible time by checking for gaps between consecutive operations on the machine. We also need to check if our job has no other operations running during that time. From all the gaps larger than the duration of our operation, we choose the tightest fit. If we find no such match, we schedule the operation as soon as both the machine and the job are available.

4.3 Simulated Annealing

This method also uses an initial schedule, using the same function as the one used in Tabu search to create an initial schedule. After receiving the initial schedule, it swaps different job tasks in the same row, calculates its makespan and compares it to current and globally best solutions. To make sure we don't get stuck in a local optimum, there's a small chance to advance to a schedule that has a less optimal makespan.

5 Results

5.1 Brute force

We implemented a brute force algorithm for the problem of scheduling 3 jobs on 3 machines. It searches for the schedule with the best makespan among all possible combinations of job order permutations.

5.2 Benchmarks

For the benchmark problems of size 4x4, our method met the given lower bounds for all 10 instances(Table 1).

For the benchmark problems of size 5x5, makespan values calculated by our algorithm were generally very close to the given lower bounds, meeting it for one instance and giving a schedule with a better makespan value for one instance, as shown in Figure 3, with first row representing the suggested lower bounds and the second row showing our results(Table 2).

For the benchmark problems of size 7x7, makespan values calculated by our algorithm met the given lower bounds three times and performed better for one instance(Table 3).

Problem instances			**Genetic Algorithm**		**Simulated Annealing**		**Tabu Search**
Test file	*LB*	*UB*	*Best*	*Mean*	*Best*	*Mean*	*Best*
test440	186	193	193	193	193	193	195
test441	229	236	236	238	236	236	243
test442	262	271	271	271	271	271	284
test443	245	250	250	251	250	251	254
test444	287	295	295	296	295	296	313
test445	185	189	189	189	189	190	199
test446	197	201	201	202	201	202	210
test447	212	217	217	217	217	218	250
test448	258	261	261	262	261	264	268
test449	213	217	217	217	217	217	229

Table 1:

Problem instances			**Genetic Algorithm**		**Simulated Annealing**		**Tabu Search**
Test file	*LB*	*UB*	*Best*	*Mean*	*Best*	*Mean*	*Best*
test550	295	300	302	315	303	318	331
test551	255	262	264	276	268	280	283
test552	321	328	333	352	338	353	345
test553	306	310	318	337	326	340	367
test554	321	329	331	354	343	353	353
test555	307	312	318	336	325	339	351
test556	298	305	310	327	310	328	333
test557	292	300	306	320	305	323	317
test558	349	353	359	378	367	381	392
test559	321	326	337	351	339	353	358

Table 2:

Problem instances			**Genetic Algorithm**		**Simulated Annealing**		**Tabu Search**
Test file	*LB*	*UB*	*Best*	*Mean*	*Best*	*Mean*	*Best*
test770	435	438	490	516	470	506	464
test771	443	449	487	537	493	526	474
test772	468	479	545	582	547	566	503
test773	463	467	513	555	510	539	505
test774	416	419	468	502	448	490	474
test775	451	460	522	562	525	545	504
test776	422	435	488	529	482	513	460
test777	424	426	457	501	478	494	459
test778	458	460	518	545	517	533	464
test779	398	400	445	484	451	475	438

Table 3:

6 Conclusion

The general pattern is that individual best solutions are mostly achieved by using either genetic algorithm or simulated annealing with Tabu being the weakest out of the three methods when looking at 4x4 and 5x5 benchmarks.

However, at 7x7 Tabu search gets better results comparing the best ones to both other methods.

Comparing best values for genetic algorithm and simulated annealing, we see that they are in general very similar and are test-case dependant for discerning their quality for a certain benchmark.

Comparing the mean values of the same two methods, there seems to be barely any difference for 4x4 benchmark, for 5x5 genetic algorithm produces barely noticeable improved solutions, and for 7x7 simulated annealing produces better results.

If it's needed to work with higher benchmark open shop scheduling, we recommend using Tabu Search, but paying close attention to the size of the Tabu list as it's arguably the most important parameter for this method.

If the benchmark number is on the lower side, genetic algorithm and simulated annealing produce better results.

References

[1] Taillard, E. Benchmarks for basic scheduling problems. Eur J Oper Res, 1993, 64, 278 – 285.

[2] Liaw, CF A tabu search algorithm for the open shop scheduling problem, 1999

[3] Khuri, S., Miryala S. Genetic Algorithms for Solving Open Shop Scheduling Problems, 1999