Job Shop Scheduling with Multi-shot ASP

Mohammed M. S. El-Kholany and Martin Gebser

Alpen-Adria-Universität Klagenfurt, Austria

1 Introduction

The job shop scheduling problem consists of many jobs that must be processed by a set of machines. It is one of the most complicated combinatorial optimization problems [3]. However, in the literature, there are relatively few studies that focus on large-scale job shop scheduling. A rolling horizon approach has been proposed in [5]. It divides the problem into sub-problems (time windows) and solves each sub-problem using a shift bottleneck heuristic while minimizing the total weighted tardiness. A decomposition heuristic based on multi-bottleneck machines was proposed in [8], where each sub-problem was solved by a genetic algorithm. In this study, we investigate another decomposition method, balancing the number of operations per time window.

2 Approach

The main idea of our proposed method is to split the set of all operations into time windows based on the earliest starting time for each operation, given by the sum of processing times of predecessor operations belonging to the same job. That is, each job is a sequence of operations, all of which could be processed at their respective earliest starting time if the machine to use is free. However, machines may be occupied by operations of other jobs, so that the earliest starting time merely provides a lower bound on the actual starting time of an operation.

Given a job shop scheduling instance with n operations, ordered by their earliest starting times, and a number m of time windows, the i-th time window for $1 \leq i \leq m$ includes the operations at positions from $(i-1)*\lceil n/m \rceil+1$ to $\min\{i*\lceil n/m \rceil, n\}$ in the order by earliest starting times. This static approach makes sure that the number of operations per time window is balanced, amounting to at most $\lceil n/m \rceil$ operations and less for the last time window in case n is not a multiple of m. After this decomposition, we schedule operations time window wise, using the multi-shot Answer Set Programming (ASP; [4]) system *clingo*[DL] [2], an extension of *clingo* [1] with difference constraints.

We performed experiments with job shop scheduling instances of different size [7, 6], where the numbers of jobs and machines range from 30 to 100 or 10 to 20, respectively, using between 1 and 10 time windows, as indicated in the first column of Table 1. The second column provides averages for compared features, including the makespan to minimize, runtime in seconds, and number of interrupted calls, over ten instances per size, except for just five instances each in two groups with 50 jobs and 10 machines, denoted by 50×10(e) and 50×10(h), as [6] noted a subdivision into easy/hard instances.

Considering the number of interrupted calls when using just 1 or 2 time windows, the entries 1.0 or 2.0, respectively, tell us that all calls during multi-shot solving time

Table 1. Results for job shop scheduling instances with different numbers of time windows

Window(s)	Comparison	30×15	30×20	50×10(e)	50×10(h)	50×15	50×20	100×20
1	Makespan	**2144.7**	2450.0	**3035.4**	**4462.6**	3542.1	3879.4	46786.1
	CPU(Sec)	1000.0	1000.0	1000.0	1000.0	1000.0	1000.0	1000.0
	Interrupted Calls	1.0	1.0	1.0	1.0	1.0	1.0	1.0
2	Makespan	2228.9	**2345.0**	3167.8	5001.8	**3524.8**	**3540.8**	23977.5
	CPU(Sec)	1000.0	1000.0	1000.0	1000.0	1000.0	1000.0	1000.0
	Interrupted Calls	2.0	2.0	2.0	2.0	2.0	2.0	2.0
3	Makespan	2396.7	2542.1	3439.6	4914.0	3621.5	3605.7	7002.6
	CPU(Sec)	921.5	804.8	1000.0	1000.0	1000.0	1000.0	1000.0
	Interrupted Calls	2.7	2.2	3.0	3.0	3.0	3.0	3.0
4	Makespan	2481.0	2632.9	3556.4	5483.4	3870.0	3918.6	**6964.4**
	CPU(Sec)	345.9	482.9	1000.0	1000.0	1000.0	1000.0	1000.0
	Interrupted Calls	1.1	1.7	4.0	4.0	4.0	4.0	4.0
5	Makespan	2537.9	2750.1	3572.4	5598.2	3908.1	4006.3	7094.9
	CPU(Sec)	190.5	233.3	952.3	1000.0	978.8	974.3	1000.0
	Interrupted Calls	0.7	1.1	4.6	5.0	4.8	4.8	5.0
6	Makespan	2600.3	2856.4	3803.0	5742.4	4043.5	4116.5	7357.9
	CPU(Sec)	75.9	94.1	781.6	1000.0	919.6	855.4	1000.0
	Interrupted Calls	0.3	0.3	4.2	6.0	5.4	4.9	6.0
7	Makespan	2659.7	2868.6	3669.6	5612.4	4064.3	4144.1	7519.4
	CPU(Sec)	16.5	6.4	656.1	885.3	706.4	668.0	1000.0
	Interrupted Calls	0.0	0.0	4.6	6.2	4.1	3.8	7.0
8	Makespan	2656.8	2938.8	3764.8	5790.6	4269.0	4228.0	7587.0
	CPU(Sec)	40.5	41.3	355.5	911.4	522.6	420.8	1000.0
	Interrupted Calls	0.3	0.1	2.4	7.2	3.5	2.6	8.0
9	Makespan	2642.9	2946.0	3691.0	5546.8	4146.4	4366.2	7681.6
	CPU(Sec)	13.2	**3.1**	308.4	854.1	321.9	392.9	1000.0
	Interrupted Calls	0.1	0.0	2.2	7.6	2.5	2.9	9.0
10	Makespan	2679.1	2982.2	3619.0	5804.8	4127.3	4397.1	7925.6
	CPU(Sec)	**2.6**	6.3	**209.4**	**772.6**	**239.7**	**245.2**	1000.0
	Interrupted Calls	0.0	0.0	1.8	7.4	1.9	2.0	10.0

out, which means that finding (and proving) an optimal schedule was infeasible within the allotted 1000.0 seconds of runtime, divided evenly among all time windows. One should note that each job is a sequence of 15 operations, so that the number of operations to schedule amounts to 450 for the 10 instances in the smallest group, i.e., 30×15. For this group, the average runtime drops from 1000.0 to just **2.6** seconds when switching from 1 to 10 time windows, while the average makespan of best schedules found increases from **2144.7** to 2679.1. In general, looking at the smallest average makespans and runtimes, highlighted in boldface, we see that fewer or more time windows, respectively, are advantageous. For the 100×20 group, the average makespan is best with 4 time windows, which indicates noticeable benefits due to decomposing large instances.

References

1. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Multi-shot ASP solving with clingo. Theory and Practice of Logic Programming **19**(1), 27–82 (2019)
2. Janhunen, T., Kaminski, R., Ostrowski, M., Schaub, T., Schellhorn, S., Wanko, P.: Clingo goes linear constraints over reals and integers. Theory and Practice of Logic Programming **17**(5-6), 872–888 (2017)
3. Lenstra, J., Kan, R.: Computational complexity of discrete optimization problems. Annals of Discrete Mathematics **4**, 121–140 (1979)
4. Lifschitz, V.: Answer Set Programming. Springer-Verlag (2019)
5. Singer, M.: Decomposition methods for large job shops. Computers & Operations Research **28**(3), 193–207 (2001)
6. Storer, R., Wu, D., Vaccari, R.: New search spaces for sequencing problems with application to job shop scheduling. Management Science **38**(10), 1495–1509 (1992)
7. Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operational Research **64**(2), 278–285 (1993)
8. Zhai, Y., Liu, C., Chu, W., Guo, R., Liu, C.: A decomposition heuristics based on multi-bottleneck machines for large-scale job shop scheduling problems. Journal of Industrial Engineering and Management **7**(5), 1397–1414 (2014)