MISTA 2013

Parallel Machine Scheduling with Generalized Precedence Relations

Jinil Han · Ruslan Sadykov · François Vanderbeck

1 Introduction

Parallel machine scheduling entails assigning jobs to identical machines, $k \in \{1, \ldots, m\}$, and deciding on their start time. Each job $j \in \{1, \ldots, n\}$ requires a given processing time p_j. Its starting time is denoted by S_j (its completion time is then $C_j = S_j + p_j$). The objective is typically some regular function of the completion times. Here, we additionally consider generalized precedence relations between jobs, which can be represented as

$$S_j \geq S_i + l_{ij}, \tag{1}$$

where l_{ij} is the lag between job i and j which can be positive or negative. This constraint can be used to model various types of temporal dependencies between jobs such as the requirement of synchronization, precedence, and overlap between jobs by assigning appropriate values to l_{ij}'s. The set Δ defines all job pairs (i, j) for which a generalized precedence relation exists.

For this problem, the time-indexed formulation introduced in [1] can be extended to include (1):

$$[\text{P}] \quad \min \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} c_{jt} x_{jt}, \tag{2}$$

$$\sum_{t=1}^{T-p_j+1} x_{jt} = 1, \qquad j = 1, \ldots, n, \tag{3}$$

$$\sum_{j=1}^{n} \sum_{s=t-p_j+1}^{t} x_{js} \leq m, \qquad t = 1, \ldots, T, \tag{4}$$

$$\sum_{s=t-l_{ij}+1}^{T-p_i+1} x_{is} + \sum_{s=1}^{t} x_{js} \leq 1, \qquad (i,j) \in \Delta, t = 1, \ldots, T - p_j + 1, \tag{5}$$

$$x_{jt} \in \{0, 1\}, \qquad j = 1, \ldots, n, t = 1, \ldots, T - p_j + 1, \tag{6}$$

INRIA team RealOpt and Mathematics Institute, University of Bordeaux, France
E-mail: jinil.han@inria.fr, ruslan.sadykov@inria.fr and fv@math.u-bordeaux1.fr

where the binary variable $x_{jt} = 1$ if job j is to start in period t, and 0 otherwise. Note that constraint (5) is one of the strongest way to express precedence relations between jobs, but it entails a large number of constraints. Moreover, by replacing constraint (4) with the following flow conservation constraints we can get a variant which is known to be computationally more efficient [2]:

$$\sum_{j=1}^{n} x_{j1} + y_1 = m, \tag{7}$$

$$\sum_{j=1}^{n} (x_{jt} - x_{j,t-p_j}) + y_t - y_{t-1} = 0, \qquad t = 2, \ldots, T \tag{8}$$

$$y_t \in \mathbb{Z}_+, \qquad t = 1, \ldots, T \tag{9}$$

where the integer variable y_t indicates the number of idle machines in period t. Let [F] be the formulation defined by (2)-(3) and (5)-(9). Its feasible solutions correspond to flows with value m in a network N whose nodes represent periods and where a job j is represented by arcs $(t, t + p_j)$, and idle times by arcs $(t, t + 1)$. The important advantage time-indexed formulations [P] and [F] is that they provide tight LP bounds. Their drawback is the large number of variables, especially when processing times become large.

2 Column-and-row generation approach

One possible approach to overcome the large size of [P] and [F] is to handle the variables and constraints dynamically by applying a column generation paradigm to these extended formulations [3]. In this approach, constraints (3) and (5) form the master problem, and the flow constraints (7)-(8) go to pricing subproblem. For each machine, the pricing subproblem amounts to generating a pseudo-schedule, i.e., a schedule in which jobs can be processed more than once. Solving the pricing subproblem corresponds to finding the shortest path in acyclic network N. Then, each generated pseudo-schedule is split into individual x and y variables, which are then added to the master. Once the LP relaxation of the master is solved, the most violated precedence constraints are added to it, and the process is repeated. The procedure terminates when no improving columns (variables) and no violated constraints are found. The reasoning behind adding each variable by splitting pseudo-schedule unlike standard column generation is to allow for the recombination of previously generated pricing problem solutions, and thus to accelerate the convergence of column generation. For a detailed explanation, see [3], where parallel machine scheduling is one of the test problem. However, here, the presence of the large number of additional precedence constraints reduces the great performance of the algorithm compared to a direct MIP-solver approach for [P] or [F].

3 Preliminary computational results

We performed the preliminary computational tests to see how well a column-and-row generation approach is in comparison with other four approaches: solving [P] directly using CPLEX(`[P]-d`); solving [F] directly using CPLEX(`[F]-d`); solving [P] and [F] by cut generation approach based on CPLEX where the precedence constraints being generated as cuts, denoted respectively (`[P]-c`) and (`[F]-c`). Column-and-row generation algorithm was implemented using BapCod, a generic Branch-and-Price code developed by the INRIA RealOpt team in Bordeaux, and all LP problems were solved using CPLEX 12.4. All tests were performed on Intel 2.50 GHz PC with 4GB RAM.

Table 1 shows the performance comparison between our column-and-row generation approach and other approaches on instances with 25 jobs and a total weighted tardiness objective function. Test instances were generated using the procedure from [4] which is the most used in the literature. We report average results over 5 instances with 25 jobs since most of instances with 50 jobs cannot be solved within a hour. We consider single and two machine cases. Instance classes (given in the column `num`) vary by the parameters RDD and TF . For each pair of job i and j such that $i < j$, the precedence constraint are randomly imposed with probability `p` in percent. We only considered the case of $l_{ij} = p_i$.

The headings of Table 1: `%var`, `%cut`, `#itr`, `mast`, `sub`, and `total`. They refer respectively to the percentage of generated variables, the percentage of generated constraints of type (5), the number of iterations, the time spent in solving master problems (in seconds), the time spent in solving pricing subproblems, and the total solution time, respectively. For direct cplex approach and cut generation approach, we only report total time. The fastest approach among the five is highlighted in bold type.

The case where `%p` $= 0$ represents the standard parallel machine scheduling problem without any precedence constraint. When the number of precedence relations gets larger, the number of variables and constraints generated increases rapidly, which leads to reducing the performance of column-and-row generation approach compared to CPLEX. Indeed, as the master becomes large, the column-and-row generation approach spent a great deal of time solving it at each iteration. We can also observe that the column-and-row generation approach seems to perform better compared to other approaches for most of tested instances, whereas the CPLEX solver performs better for several instances for which the relatively large number of precedence constraints are generated.

In other words, when there are few precedence constraints, the dynamic approach is performing better than solving the formulation directly with a MIP solver, or than only handling precedence constraints dynamically. But the reverse can sometimes be observed when the precedence graph is dense. Our future research plan is to examine special precedence graph as those arising in applications with synchronization constraints.

References

1. M.E Dyer, L.A. Wolsey, Formulating the single machine sequencing problem with release dates as a mixed integer problem, Discrete Applied Mathematics, vol.26, pp.255-270 (1990)
2. Y. Pan, L. Shi, On the equivalence of the max-min transportation lower bound and the time-indexed lower bound for single-machine scheduling problems, Mathematical Programming, vol.110, pp.543-559 (2007)
3. R. Sadykov, F. Vanderbeck, Column Generation for Extended Formulations, EURO Journal on Computational Optimization, vol.1, pp.81-115 (2013)
4. C.N. Potts, L.N. Van Wassenhove, A branch and bound algorithm for the total weighted tardiness problem, Operations Research, vol.33, pp.363-377 (1985)

Table 1 Computational results

Instance				column-and-row generation						CPLEX			
m	n	num	%p	%var	%cut	#itr	mast	sub	total	[P]-d	[P]-c	[F]-d	[F]-c
1	25	1	0	5.0			0.3	0.1	**0.4**	8.5		23.1	
			1	4.6	0.2	8	0.5	0.1	**0.7**	1630.8	119.3	>1h	256.5
			3	5.2	0.3	11	2.0	0.1	**2.2**	>1h	>1h	>1h	1452.7
			5	3.8	0.3	11	0.6	0.1	**0.9**	>1h	>1h	>1h	>1h
			10	10.7	0.9	53	155.6	0.2	**156.7**	>1h	>1h	>1h	>1h
1	25	7	0	4.1			0.2	0.1	**0.3**	16.8		20.5	
			1	16.7	4.7	100	167.9	1.1	**171.0**	452.3	218.6	>1h	598.0
			3						>1h	>1h	>1h	>1h	>1h
			5						>1h	>1h	>1h	>1h	>1h
			10						>1h	>1h	>1h	>1h	>1h
1	25	13	0	5.3			0.5	0.1	**0.8**	9.5		22.4	
			1	4.5	0.1	12	0.9	0.2	**1.4**	57.9	25.0	471.1	30.9
			3	9.1	1.6	104	27.3	1.0	**29.5**	217.9	122.0	2213.9	166.9
			5						>1h	>1h	>1h	>1h	>1h
			10						>1h	>1h	>1h	>1h	>1h
1	25	19	0	6.6			0.3	0.1	**0.5**	9.7		20.6	
			1	6.1	0.7	53	2.9	0.4	**3.6**	52.9	21.2	615.0	47.1
			3	10.0	2.0	115	16.1	0.6	**18.4**	249.0	101.5	1159.4	240.8
			5	18.6	5.8	178	665.8	3.0	671.5	**587.5**	1030.4	>1h	1055.4
			10	21.8	4.5	204	1527.9	4.6	**1543.1**	>1h	>1h	>1h	>1h
1	25	25	0	6.2			0.7	0.1	**1.0**	35.8		45.5	
			1	5.5	0.0	1	4.5	0.3	**5.0**	125.0	77.5	1141.4	143.9
			3	4.6	0.0	1	0.4	0.1	**0.6**	128.6	276.9	1159.0	455.0
			5	10.7	1.9	79	70.3	1.2	**73.9**	1186.5	1672.6	>1h	2188.9
			10	21.0	3.9	177	2082.1	4.8	**2096.6**	>1h	>1h	>1h	>1h
2	25	1	0	5.6			0.1	0.0	**0.2**	7.5		3.6	
			1	6.9	0.8	13	1.0	0.1	**1.2**	116.6	24.3	1133.9	41.8
			3	9.4	0.8	17	5.2	0.9	**5.5**	482.5	114.7	>1h	1462.4
			5	16.4	3.8	76	125.1	0.6	**126.7**	1113.1	2493.4	>1h	1840.4
			10						>1h	>1h	>1h	>1h	>1h
2	25	7	0	3.3			0.0	0.1	**0.2**	5.2		4.3	
			1	19.2	13.7	168	242.3	4.6	254.7	158.6	**41.7**	1350.4	528.5
			3	23.2	11.7	176	1265.8	9.0	1288.1	238.2	**160.3**	1796.2	844.6
			5	25.6	10.0	191	2441.5	14.4	2471.1	**527.3**	553.8	>1h	3397.2
			10						>1h	>1h	>1h	>1h	>1h
2	25	13	0	5.5			0.2	0.1	**0.4**	4.3		4.0	
			1	9.1	3.0	58	2.2	0.2	**3.0**	26.0	10.1	424.7	9.7
			3	10.7	2.8	84	6.5	0.7	**8.1**	49.6	15.0	117.9	15.9
			5	17.0	3.6	92	25.5	0.9	**28.4**	97.9	102.2	316.7	147.8
			10	12.8	1.9	51	10.8	0.6	**13.1**	137.6	175.2	445.4	272.1
2	25	19	0	5.0			0.1	0.1	**0.3**	2.5		3.1	
			1	7.2	0.6	12	0.5	0.1	**0.8**	22.5	9.8	39.6	7.5
			3	10.1	1.1	60	1.8	0.1	**2.5**	40.7	11.6	96.1	14.5
			5	16.3	5.8	112	35.8	0.8	**39.3**	133.1	156.3	423.2	216.8
			10	16.0	3.7	92	33.0	1.0	**36.6**	254.6	277.4	611.7	363.2
2	25	25	0	4.1			0.2	0.0	**0.3**	18.3		6.0	
			1	6.2	0.7	13	0.6	0.1	**0.8**	31.5	38.3	107.5	14.8
			3	8.3	1.4	62	2.8	0.2	**3.6**	47.8	54.9	242.3	44.5
			5	11.3	1.7	35	3.8	0.6	**5.3**	132.2	179.1	523.9	314.1
			10	13.8	2.5	55	27.1	0.8	**29.6**	250.6	457.8	952.3	1459.3