Multi-objective parallel machine scheduling with incompatible jobs

Proceedings for the ROADEF 2014 Conference, February 26-28 2014, Bordeaux, France

Simon Thevenin
HEC - University of Geneva
Switzerland
simon.thevenin@unige.ch

Nicolas Zufferey
HEC - University of Geneva
Switzerland
n.zufferey@unige.ch

Jean-Yves Potvin
Université de Montréal
Canada
potvin@iro.umontreal.ca

Abstract

We consider the problem of scheduling n jobs with different processing times on parallel identical machines with job incompatibility constraints and preemption possibility. Three objectives have to be minimized in lexicographical (or hierarchical) order: makespan, number of job interruptions, and sum of throughput times over all jobs. A linear programming formulation, a greedy heuristic and a tabu search are proposed to solve this problem.

1 Description of the problem

The problem, denoted (P) in the following, consists in scheduling a set of n jobs on parallel machines. Incompatibilities between jobs are represented by a conflict graph $G = (V, E)$, where V is the vertex set representing the jobs and E is the edge set. If $(i, j) \in E$, the jobs i and j cannot be scheduled simultaneously (i.e., their processing periods cannot overlap). With each job i is associated an integer processing time p_i, and preemption can only take place at integer time points. The problem is to assign p_i time slots to each job i, so that no incompatible jobs are assigned to a common time slot. There is no constraint on the number of machines.

Incompatibilities between jobs arise when some scarce resources are required to process jobs (e.g., expensive tools to equip the machines, employees with specific skills). The authors in [1] mention three concrete examples of such scarce resources: the sirup tanks in drinks bottling, the testing heads in semiconductor industries, and some specific employees in car production lines. The tooling constraints are particularly relevant in flexible manufacturing systems [8]. We consider here the special case where each resource exists in a single exemplar: two jobs are incompatible if they require a common scarce resource.

In problem (P), allowing preemption is a way to improve the makespan. It is assumed that a job can be stopped and restarted later without encountering additional costs. However, preemptions are often undesirable, either because of managerial reasons or because of the resulting work in progress. To reduce the negative impacts, the sum of throughput times (defined as the completion time minus the starting time) over all jobs should be minimized.

Problem (P) can be seen as a multicoloring problem (MP), which consists in assigning p_i different colors to each vertex of G such that no two adjacent vertices are assigned a common color, while minimizing the number of used colors. A correspondence between (MP) and (P) is straightforward. A vertex i represents a job i, an edge (i, j) means that jobs i and j are incompatible, and a color stands for a time slot. The reader is referred to [4] for pointers on the graph (multi)coloring literature. In this paper, the graph terminology (e.g., vertex, edge, color) and the scheduling terminology (e.g., job, incompatibility constraint, time slot) will be indifferently used.

The paper is organized as follows. Section 2 presents a short literature revue. A linear formulation is proposed in Section 3. Then, heuristic methods are described in Section 4, while Section 5 reports the results of numerical experiments. A conclusion and avenues for future work close the paper.

2 Literature review

The reader interested in a general and recent book on scheduling is referred to [11]. The authors in [6] study the problem of minimizing the makespan when scheduling parallel machines with job incompatibilities and give approximation methods for some special cases. An exact algorithm is also reported for two machines and job processing times of one or two time units. The problem is shown to be NP-hard with processing times in $\{1, 2, 3, 4\}$. In addition, the authors study dynamic job arrivals. The work in [2] extends these results by showing that the problem with two machines and processing times in $\{1, 2, 3\}$ is NP-hard. They also show that the problem with two machines and processing times in $\{1, 2\}$ becomes NP-hard when release dates are considered. An exact algorithm working in polynomial time is obtained for unit processing times by exploiting a bi-partite agreement graph (complement of the conflict graph). Finally, lower bounds and heuristics are derived for the general case. In [9], an exact method is proposed for scheduling two different sets of jobs on two different machines, with incompatibilities between jobs of each set (bipartite conflict graph). The three last papers do not consider preemption, as opposed to the work in [4] where the problem of scheduling parallel machines with preemption, incompatibility penalties and assignment costs is addressed. Exact methods and meta-heuristics are proposed to solve this problem. In [12], exact methods and meta-heuristics are reported for a parallel machine scheduling problem with preemption, job incompatibilities and job rejection penalties (for jobs that are not performed). The hierarchical objective function consists in minimizing the sum of rejection penalties, the number of interruptions and the sum of throughput times over all jobs. We extend this work here by considering that the makespan has to be minimized as well and that all jobs have to be performed. Since the objective function of problem (P) is very different from those in [4, 12], the existing methods cannot be easily adapted and new dedicated methods have been designed.

3 Linear program

A linear program LP derived from [12] is given below with the following variables: Max_i (rep. Min_i) denotes the largest (resp. smallest) time unit assigned to job i; s_{it} equals 1 if job i starts or is resumed (after preemption) at time unit t, 0 otherwise; u_t equals 1 if time unit t is used, 0 otherwise; and x_{it} equals 1 if job i is processed during time unit t, 0 otherwise. The makespan is denoted C_{max}. For formulation purposes, a straightforward upper bound U on C_{max} is used which is equal to $n \cdot p_{max}$, where p_{max} denotes the largest processing time over all jobs.

The three objectives to be minimized are the following (and are considered in this order):

$$f_1 = C_{max}, f_2 = \sum_{i\in V}\sum_{t=1}^{U} s_{it} - n, f_3 = \sum_{i\in V}(Max_i - Min_i)$$

In f_2, subtracting n withdraws the number of job starts, given that $s_{it} = 1$ for each job interruption and for each job start. The problem is solved once for each objective, starting with f_1, then f_2 and finally f_3. Once the optimal value of objective f_i has been determined, it is used as a constraint to solve the next objective. The constraints of problem (P) are the following:

$$\sum_{t=1}^{U} x_{it} = p_i \qquad i \in V \qquad (1)$$
$$t \cdot x_{it} \leq Max_i \qquad 1 \leq t \leq U, \quad i \in V \qquad (2)$$
$$t \cdot x_{it} + U \cdot (1 - x_{it}) \geq Min_i \qquad 1 \leq t \leq U, \quad i \in V \qquad (3)$$
$$C_{max} \geq u_t \qquad 1 \leq t \leq U, \qquad (4)$$
$$s_{it} \geq x_{it} - x_{i(t-1)} \qquad 1 \leq t \leq U, \quad i \in V \qquad (5)$$
$$u_t = x_{it} + x_{jt} \qquad 1 \leq t \leq U, \quad (i,j) \in E \qquad (6)$$
$$s_{it},\ x_{it},\ u_t \in \{0, 1\} \qquad 1 \leq t \leq U, \quad i \in V \qquad (7)$$
$$Min_i,\ Max_i \geq 0 \qquad i \in V \qquad (8)$$

Constraint (1) states that each vertex i is colored with exactly p_i colors (i.e., each job is fully performed). Constraints (2), (3), (4) and (5) set the values of Max_i, Min_i, C_{max} and s_{it}, respectively. Constraint (6) sets the value of u_t, and forbids the assignment of a common color to incompatible jobs (such a constraint was also used for the graph coloring problem [10]). Finally, (7) and (8) are domain constraints.

4 Heuristic methods

In this section, a greedy method GR and a tabu search TS are proposed to solve (P). Both methods use a strategy where the number of available time slots (colors) is fixed to some value k. The aim is then to find a feasible coloring of the graph using only k colors (i.e. a k-coloring) which minimizes f_2 and f_3. C_{max} is then the smallest k for which a k-coloring is found. This strategy is the most efficient to solve graph coloring problems [3].

GR is an adaptation of DSATUR [5]. The saturation degree $Dsat(i)$ of a vertex i is defined as the number of different colors used by vertices adjacent to i, while the degree $deg(i)$ of i is the number of edges incident to i. GR starts from a non colored graph, and colors the vertices one by one. At each step, the vertex i which maximizes $Dsat(i)$ is the next to be colored. If there are ties, the vertex of largest degree is chosen in the subgraph obtained with only non colored vertices (ties are broken randomly, if any). To recolor the chosen vertex i, a set of p_i different colors must be selected in the set of available colors A_i (i.e., colors that are not already used by vertices adjacent to i). If there are not enough available colors (i.e., $|A_i| < p_i$), the method stops as it is not able to find a feasible k-coloring. Otherwise, the colors are selected with the recoloring method proposed in [12] to find an assignment of colors minimizing f_2 and f_3 (such a recoloring method is an implicit exhaustive enumeration method of all possible colorings for the considered vertex).

The tabu search TS is a local search metaheuristic where the neighborhood $N(s)$ of a solution s is obtained by performing moves (i.e., slight modifications to the solution structure). Starting from an initial solution, TS navigates from one neighbor solution to the next. A tabu list is used to forbid the reversal of recently performed moves. Basically, the tabu search performs the best non tabu move at each iteration. For more information on tabu search and metaheuristics in general, the reader is referred to [7, 13].

Based on the problem-solving strategy mentioned above, where the number of colors is gradually reduced until no feasible coloring can be found, the search space of TS contains k-colorings of the graph. Each vertex i is either colored (i.e. p_i colors are assigned) or uncolored (i.e., no color is given). The first objective is then to minimize the number of uncolored vertices, given that a feasible solution is found when all vertices are colored. Objectives f_2 and f_3, however, remain unchanged.

In TS, the initial solution is generated with GR, and a move consists in completely recoloring a vertex. However, the tabu status forbids to recolor a recently recolored vertex during t iterations (where t is randomly chosen between 10 and 20 after each move). Any vertex i (colored or not) can be recolored with a (new) set of p_i colors. The way to select the colors depends on the number $|A_i|$ of available colors.

- If $|A_i| < p_i$, the move is enforced as follows. All colors of A_i are first selected. Then the $p_i - |A_i|$ missing colors are selected one by one. When a color c is assigned to vertex (job) i, the adjacent vertices using color c are uncolored (rejected). Thus, at each step, the color which minimizes the number of additional rejections is chosen.
- If $|A_i| \geq p_i$, the colors which minimize the number of interruptions and throughput time are chosen in a greedy fashion. Obviously, if contiguous colors (contiguous time units) are assigned to i, no interruption occurs and the throughput time is minimized. The method is based on this observation: while the vertex is not fully colored, the largest set of contiguous colors still available in A_i is assigned to job i.

5 Experiments

We implemented the heuristics TS and GR in C++. The linear program LP was solved with CPLEX 12.5. The methods were run on a computer with a processor Intel Quad-core i7 2.93 GHz with 8 GB of DDR3 RAM memory. The time limit for LP was set to one hour for each objective. The time limit for TS and GR was set to $n/20$ minutes, where n is the number of jobs. For each value of k, GR was restarted as long as the time limit was not reached, and the best solution obtained was returned at the end. TS and GR were run ten times for each value of k, starting with the upper bound U on C_{max}. The value of k was decreased until none of the runs could find a feasible solution.

To generate the set of test instances, n was chosen in $\{10, 25, 50, 100\}$, and for each value of n, five instances were produced and labeled a, b, c, d, e. The integer processing times were randomly chosen in $[1, 10]$. Incompatibilities between pairs of jobs were randomly generated by setting the probability to have an edge between two vertices to 0.5.

Results are shown in Table 1. For LP, the value obtained for each objective is indicated, and the results proven to be optimal by CPLEX are indicated with a * sign. For TS and GR, the values of k_{min} and k_{max} are given, where k_{min} (resp. k_{max}) is the smallest value of k such that at least one (resp. ten) successful run(s) were performed. The average values of f_2 and f_3 are given for k_{all}, which is the smallest value of k such that TS and GR found at least one feasible solution. The minimum values of k_{min} and k_{max} are indicated in bold face.

LP can tackle instances with 10 and 25 jobs, but can only guarantee the optimality for instances with 10 jobs. Given that the lower bounds returned by CPLEX for instances with $n = 50$ are poor (i.e., far from the results obtained by GR and TS), the instances with $n = 100$ were not tested. GR obtains good results on small instances: for example, optimal results are obtained for the ten runs with $n = 10$. Also, for instances with $n = 25$, it produces the best k_{min} for four instances out of five, and the values of objectives f_2 and f_3 are small. But TS is clearly the best method: it obtains the best k_{min} for all instances, and the k_{max} value is smaller than the one of GR for 12 instances out of 20. Also, the results for objectives f_2 and f_3 are clearly better than the ones obtained with GR for instances of size 50 and 100.

		LP			GR					TS				
		C_{max}	f_2	f_3	k_{min}	k_{max}	k_{all}	f_2	f_3	k_{min}	k_{max}	k_{all}	f_2	f_3
10	a	30*	0*	51*	**30**	**30**	30	0	51	**30**	**30**	30	0.4	53.6
	b	23*	0*	56*	**23**	**23**	23	0	56	**23**	**23**	23	1.3	64.2
	c	25*	0*	64*	**25**	**25**	25	0	64	**25**	**25**	25	0	64
	d	30*	0*	68*	**30**	**30**	30	0	68	**30**	**30**	30	0	68
	e	27*	0*	45*	**27**	**27**	27	0	45	**27**	**27**	27	0.7	48.5
25	a	41*	0	129	**41**	**41**	41	5.5	184.1	**41**	**41**	41	8.9	223
	b	41*	7	206	42	**43**	42	16.5	270.5	**41**	46	42	22.25	273.3
	c	40	7	235	**40**	**40**	40	18.7	334.9	**40**	41	40	13.7	254.7
	d	37	11	238	**37**	**37**	37	16.5	297.4	**37**	38	37	29.75	310.5
	e	40	55	320	41	**41**	41	13.1	289.6	**40**	42	41	18.7	295.4
50	a	493	0	261	51	52	51	65.2	829.8	**48**	**50**	51	31.4	585.6
	b	498	0	264	62	**63**	62	48.5	833.25	**61**	64	62	52.4	831.2
	c	498	0	302	62	**63**	62	60.8	1066.8	**61**	66	62	55.6	892
	d	493	0	284	55	**56**	55	58	919	**54**	58	55	54.5	848.5
	e	493	0	277	54	**55**	54	62	789	**53**	56	54	68	911
100	a				93	95	93	170	3086	**87**	**89**	93	119.2	2345.2
	b				98	100	98	147	2553.2	**92**	**95**	98	117.6	2513.8
	c				98	100	98	176	3057.5	**89**	**91**	98	106.1	2310.9
	d				92	95	92	173	2713	**86**	**89**	92	111.9	2351.2
	e				107	108	107	149.5	3302.8	**99**	**104**	107	108.8	2547.8

Table 1: Comparison of the methods.

To see how the objectives are competing, Figure 1 shows – for a representative instance with $n = 100$ and label d – the variation of f_2 (left part) and f_3 (right part) for different values of k. As expected, fewer interruptions are required when the makespan is larger. Also, we can see that f_2 and f_3 are linked even if theoretically, minimizing one does not mean minimizing the other. Such graphics can be helpful for practitioners.

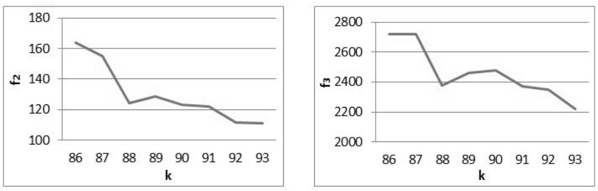

Figure 1: Variation of f_2 and f_3 depending on k, for instance d with $n = 100$.

6 Conclusion

We consider in this work a parallel machine scheduling problem with job incompatibilities and three objectives: makespan, number of job interruptions, and sum of throughput times over all jobs. We propose efficient methods taking advantage of the graph coloring literature, namely, a linear program, a greedy heuristic and a tabu search. Future work includes adaptation of these methods, as well as new methods, for a problem where the number of machines is limited.

References

[1] C. Almeder and B. Almada-Lobo. Synchronisation of scarce resources for a parallel machine lotsizing problem. *International Journal of Production Research*, 49(24):7315–7335, 2011.

[2] M. Bendraouche and M. Boudhar. Scheduling jobs on identical machines with agreement graph. *Computers & Operations Research*, 39(2):382 – 390, 2012.

[3] I. Blöchliger and N. Zufferey. A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Computers & Operations Research*, 35(3):960 – 975, 2008.

[4] I. Blöchliger and N. Zufferey. Multi-coloring and job-scheduling with assignment and incompatibility costs. *Annals of Operations Research*, 2013. doi : 10.1007/s10479-013-1397-1.

[5] D. Brélaz. New Methods to Color Vertices of a Graph. *Communications of the Association for Computing Machinery*, 22:251–256, 1979.

[6] G. Even, M. M. Halldórsson, L. Kaplan, and D. Ron. Scheduling with conflicts: online and offline algorithms. *Journal of Scheduling*, 12(2):199–224, 2009.

[7] M. Gendreau and J. -Y. Potvin, editors. *Handbook of Metaheuristics 2nd Edition*. Springer, 2010.

[8] A. Hertz and M. Widmer. An improved tabu search approach for solving the job shop scheduling problem with tooling constraints. *Discrete Applied Mathematics*, 65(1–3):319 – 345, 1996.

[9] I. N. Lushchakova and V. A. Strusevich. Scheduling incompatible tasks on two machines. *European Journal of Operational Research*, 200(2):334 – 346, 2010.

[10] E. Malaguti and P. Toth. A survey on vertex coloring problems. *International Transactions in Operational Research*, 17 (1):1 – 34, 2010.

[11] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems Third Edition*. Springer, 2008.

[12] S. Thevenin, N. Zufferey, and J.-Y. Potvin. A multi-coloring approach for an order acceptance and scheduling problem with preemption and job incompatibilities. Technical report, CIRRELT-2013-45, August 2013.

[13] N. Zufferey. Metaheuristics: Some principles for an efficient design. *Computer Technology and Application*, 3:446 – 462, 2012.