CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

A Multi-Coloring Approach for an Order Acceptance and Scheduling Problem with Preemption and Job Incompatibilities

Simon Thevenin
Nicolas Zufferey
Jean-Yves Potvin

August 2013

CIRRELT-2013-45

Bureaux de Montréal :
Université de Montréal
C.P. 6128, succ. Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
2325, de la Terrasse, bureau 2642
Québec (Québec)
Canada G1V 0A6
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

A Multi-Coloring Approach for an Order Acceptance and Scheduling Problem with Preemption and Job Incompatibilities

Simon Thevenin[1], Nicolas Zufferey[1,2], Jean-Yves Potvin[2,3,*]

[1] HEC, Faculty of Economics and Social Sciences, University of Geneva, Boulevard du Pont-d'Arve 40, 1211 Geneva 4, Switzerland

[2] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)

[3] Department of Computer Science and Operations Research, Université de Montréal, P.O. Box 6128, Station Centre-Ville, Montréal, Canada H3C 3J7

Abstract. The order acceptance and scheduling problem occurs when the production capacity of a workshop is exceeded. The problem consists in selecting and scheduling a subset of jobs to maximize the total gain associated with the selected jobs, while possibly minimizing some costs. In this paper, the problem is considered in a parallel machine environment where preemption is allowed and where there are job incompatibilities. Preemption is particularly relevant when the production process has negligible setup times, which is typically the case for automated production systems. But even if they are negligible, setups due to preemption have a cost, and must be avoided if possible. Therefore, we consider a multi-objective approach which accounts for three objectives: total gain of the selected jobs, preemption penalties and work in progress inventory level. In this work, the problem is modeled as a graph multi-coloring problem for which a linear programming formulation is proposed to address small size instances. A constructive heuristic, as well as local search methods, are also reported to address instances of more realistic sizes.

Keywords: Scheduling, multi-coloring, multi-objectives, job incompatibilities, meta-heuristics.

Acknowledgements. Financial support for this work was provided by the Natural Sciences and Engineering Research Council of Canada (NSERC). This support is gratefully acknowledged.

---

* Corresponding author: Jean-Yves.Potvin@cirrelt.ca

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2013

1. Introduction

We consider the scheduling of n jobs with different integer processing times on a number of identical machines. Different tools can equip the machines, and jobs with the same tool requirements cannot be processed simultaneously. Such jobs are said to be incompatible. Preemption is allowed, that is, each job can be stopped at integer time points and can be resumed later. We consider that the number of machines is sufficiently large to be non constraining with regard to the number of jobs to be processed in parallel. All jobs must terminate before a global deadline D, which represents the end of the planning horizon. As it might not be possible to schedule all jobs, the workshop can reject some of them. When some job i is completed, the company earns a gain g_i (which often depends on its integer processing time p_i). There is no gain if the job is only partially performed, so it is better not to perform it at all. Our goal is to optimize the three following objectives in lexicographic order:

- f_1 : maximize the total gain associated with completely performed jobs,
- f_2 : minimize the number of job interruptions,
- f_3 : minimize the work in progress, which is the total time spent by the jobs in the production shop.

This problem is denoted P in the following and can be viewed as a multi-coloring problem, which is an extension of the well-known graph coloring problem. As the latter is NP-hard (Malaguti & Toth (2010)), problem P is also NP-hard. Therefore, heuristics and metaheuristics are the most appropriate methods to tackle large scale instances. General references on the topics studied in this paper can be found in Pinedo (2008) for scheduling problems, Slotnick (2011) for the order acceptance and scheduling problem, Malaguti & Toth (2010) for the graph coloring problem and its extensions, Gendreau & Potvin (2010) and Zufferey (2012) for metaheuristics.

A preliminary version of the work reported here has already appeared in Thevenin et al. (2013). This previous contribution is extended in the follow-

ing ways. First, a linear programming formulation is proposed and implemented with CPLEX to address small size instances. The tabu search is also much improved, in particular by considering new neighborhood structures and by integrating a diversification mechanism. Some drawbacks of the tabu search, reported in Thevenin et al. (2013) with regard to the number of job interruptions and work in progress inventory level, are now greatly alleviated in the new implementation.

The remainder of this paper is organized as follows. Section 2 provides a literature review and establishes a correspondence between problem P and the multi-coloring problem. A formal description of P and a linear programming formulation are found in Section 3. Section 4 describes some basic heuristic approaches, namely a greedy procedure and a descent method. In Section 5 a refined tabu search for problem P is described. Finally, all the proposed methods are numerically compared in Section 6. The conclusion follows in Section 7 along with future research avenues.

2. Literature review and correspondence with multi-coloring

To the best of our knowledge, the work reported in Thevenin et al. (2013) is the only attempt at solving problem P. Thus, Section 2.1 will focus on some key features of the problem. The correspondence between P and the multi-coloring problem will be explained in Section 2.2, including some relevant contributions in the graph coloring literature. A summary will follow in Section 2.3. Note that the standard three-field notation $(\alpha \mid \beta \mid \gamma)$ is used in the following, where α, β and γ correspond to the production environment, the constraints and the objectives, respectively.

2.1. Key features

In the following, the literature is reviewed with regard to three key features of our problem: preemption is discussed in Section 2.1.1, minimization of work in progress is the topic of Section 2.1.2, while multi-objective optimization is briefly introduced in Section 2.1.3.

2.1.1. Preemption penalties

Typically, when preemption is allowed, a job can be stopped and restarted later at no cost or time penalty. For most scheduling problems, allowing preemption makes them easier to solve. For example, minimizing the makespan on parallel machines without preemption ($P_m \mid\mid C_{max}$) is NP-hard, but becomes solvable in polynomial time when preemption is allowed. Preemption can thus be viewed as a constraint relaxation technique which is often used to generate lower bounds. There are some exceptions, though, as indicated in Brucker & Kravchenko (1999). If preemption is interesting from an academic point of view, the assumption that neither setup cost nor setup time is incurred is often unrealistic in production environments. A good example is the largest remaining processing time rule which is optimal for parallel machine scheduling problems consisting in minimizing the makespan with preemption ($P_m \mid prmp \mid C_{max}$), but which cannot be applied in practice because it often leads to a large number of job interruptions. Accordingly, preemption should be avoided whenever possible by introducing penalties.

Although scheduling problems with preemption have been well studied, only a few papers address preemption penalties, either explicitly or by considering setup costs. Liu & Cheng (2002) study the single machine scheduling problem with preemption, release dates, preemption penalties and delivery times, where the objective is to minimize the delivery time of the last job. It is equivalent to solving a problem where each job has a due date and where the maximum lateness must be minimized ($1 \mid prmp, r_j, d_j \mid L_{max}$). The authors show that the problem is NP-hard and propose a dynamic formulation as well as a polynomial-time approximation scheme. Schuurman & Woeginger (1999) introduce a polynomial-time approximation for the problem of scheduling n jobs on m machines with setup times and with the objective of minimizing the makespan. Heydari et al. (2010) propose an on-line heuristic for minimizing the mean flow time with preemption penalties, where the flow time is defined as the time spent by a job in the system (completion time minus release date). The authors study two different types of job interruptions: the most often encountered *resume* preemption type, where only the remaining processing time has to be performed when a job is resumed and the

restart preemption type, where the job must be restarted from the beginning (this case is only interesting in on-line scheduling problems). Shachnai et al. (2002) impose a maximum number of job interruptions in a multiprocessor scheduling environment. In another variant, the authors look for the schedule with the minimum number of job interruptions among schedules that minimize the makespan. In Mohammadi & Heydari (2011), the authors propose an exact approach for a single machine problem with release dates and deadlines. The cost function depends on the completion time of the jobs and the number of job interruptions. When a job is interrupted, a job-dependent setup cost is incurred.

2.1.2. Work in progress

Minimizing the number of job interruptions and minimizing the work in progress are clearly two different objectives. For problem P, minimizing the work in progress is equivalent to minimizing the sum of processing time and waiting time (due to preemption) over all jobs. In Hendel et al. (2009), a single machine scheduling problem is studied where preemption is allowed and where earliness and tardiness costs must be minimized. Earliness penalties are often used to represent holding costs and are also related to the time spent by the jobs in the assembly shop. The authors propose a descent method and a branch-and-bound algorithm for this problem. In Kazemi et al. (2011), earliness and tardiness penalties are considered similarly to Hendel et al. (2009). A mathematical model is proposed for a problem where the objective function includes preemption penalties and inventory costs (associated with work in progress), in addition to the earliness and tardiness penalties.

The problematic issue of unfinished products that must remain inside the workshop is also found in the blocking flow shop problem, where buffers of limited size between machines are considered. Once a buffer is full, the jobs that are still on the machine are blocked until some space is freed (see Gröflin et al. (2011) for a mathematical formulation and a tabu search metaheuristic).

2.1.3. Multi-objective

The most popular algorithms for solving multi-objective optimization problems are evolutionary algorithms, in particular genetic algorithms (GA). They are particularly well suited for these problems because they work with a population of solutions and can maintain a number of different non-dominated solutions along the Pareto front (see Marler & Arora (2004) for a survey on multi-objective optimization and Konak et al. (2006) for the most important variants of GAs for multi-objective problems). Aggregating multiple objectives into a single objective through a weighted sum or imposing a lexicographic order among the objectives, as it is done in this work, are two different ways of transforming a multi-objective problem into a (series of) single objective one(s). Three different objectives are considered in lexicographic order, where the higher level objective f_1 is optimized before f_2 which is then optimized before f_3. This approach was chosen due to a natural hierarchy among the three objectives in practical applications.

2.2. The multi-coloring problem

Section 2.2.1 first explains the correspondence between problem P and the multi-coloring problem. Then, applications of graph coloring problems in job-shop scheduling are discussed in Section 2.2.2. Finally, Section 2.2.3 is dedicated to the multi-coloring problem.

2.2.1. Correspondence between P and the multi-coloring problem

Given a non directed graph $G = (V, E)$, where V is the vertex set and E the edge set, the graph coloring problem (GCP) consists in assigning a single color (integer) to each vertex, such that no two adjacent vertices have the same color, while minimizing the total number of colors. In the more general multi-coloring problem, each vertex must be assigned a preset number of different colors such that no two adjacent vertices have a color in common, while again minimizing the total number of colors. Problem P can then be viewed as an extension of the k-multi-coloring problem, where a feasible coloring with at most k colors must be found. More precisely, each vertex stands for a job,

and there is an edge between two jobs if they are incompatible. A color then represents a time unit and color c is assigned to job i if one time unit of job i is processed during time unit c. Each job must be assigned a number of colors which is equal to its processing time, and the total number of colors k is set to D. In this extension of the k-multi-coloring problem, a weight is associated with each vertex to represent the gain of the corresponding job. The objective is to maximize the total gain over completely colored vertices, while taking into account other secondary objectives. As P is closely related to the k-multi-coloring problem, it is natural to take advantage of the literature on graph coloring.

2.2.2. Graph coloring for scheduling problems

In the following, examples of scheduling problems which are modeled as graph coloring problems are presented, with a particular emphasis on multi-coloring problems.

Modeling scheduling problems as GCPs is particularly relevant in the presence of job incompatibilities. In Meuwly et al. (2010), for example, the authors consider the problem of scheduling unit-time jobs with precedence constraints and job incompatibilities while minimizing the makespan. The problem is modeled as a mixed graph coloring problem. In this extension of the GCP, arc constraints are found in addition to edge constraints. In particular, if there is an arc (i, j), the color assigned to j must be larger than the one assigned to i. A greedy heuristic, a tabu search and a variable neighborhood search are proposed to solve the problem. Fukunaga et al. (2007) consider an extension of the GCP, where the cost function depends on the partition of the set of vertices into independent subsets of vertices sharing the same color (color classes). Giaro et al. (2009) study a multiprocessor task system where each task requires a fixed set of processors for its execution. Furthermore, all tasks have the same duration and machine and job unavailabilities are considered. The authors model the problem as an edge list coloring in a hypergraph and develop an exact algorithm for some special classes of graphs. In their model, an edge can link several vertices and a list of allowed colors is associated with each vertex.

In batch scheduling, a machine can process a set of jobs simultaneously, and the completion time of the batch is the completion time of its longest job. In presence of job incompatibilities, the problem can be modeled as a weighted sum-coloring problem when the objective is to minimize the mean flow time. In this extension of the GCP, the cost of a color class corresponds to its vertex of maximum weight and the objective is to maximize the sum of these costs over all problem classes. This problem is tackled in Epstein et al. (2009) and Halldórsson & Kortsarz (2004). In de Werra et al. (2005), the authors consider an extension of the batch scheduling problem, where the assignment of two or more incompatible jobs to the same batch are allowed. In this case, however, the incompatible jobs must be processed consecutively on the machine. Complexity results are given for this problem, while exact and approximation procedure are proposed for particular graph topologies.

2.2.3. Multi-coloring

Multi-coloring problems are used to model a number of scheduling applications (Marx (2004)). First, it is worth noting that any multi-coloring problem on a graph $G = (V, E)$ can be transformed into a graph coloring problem on a modified graph $G' = (V', E')$, called the split graph. G' is obtained as follows. First, for each vertex $i \in V$, p_i vertices are created. Then, an edge is created between two vertices of G' if they are associated with the same vertex in G or if they are adjacent in G. Any graph coloring algorithm can then be applied to graph G' to solve the corresponding multi-coloring problem.

The set-T-coloring problem is an extension of the multi-coloring problem, where a set of colors must be assigned to each vertex, while satisfying two different types of constraints. First, the co-node constraint forbids the difference between two colors of the same vertex i to be equal to a number taken from a given set T_i. The second constraint forbids the difference between the colors associated with two adjacent vertices i and i' to be equal to a number taken from a given set $T_{ii'}$. Dorne & Hao (1998) propose a tabu search algorithm for the set-T-coloring problem and applies it also to the graph coloring and T-coloring problems (a special case where each vertex requires only one color). Chiarandini & Stützle (2007) compare different algorithms for the

set-T-coloring problem. The authors re-implement and test different constructive heuristics and local search algorithms either on the original graph or on the split graph. The following conclusions can be drawn from the two previous papers:

- an adaptation of DSATUR (Brélaz (1979)) proves to be the best constructive heuristic, where the saturation degree of vertex i is defined as the number of different colors used by vertices adjacent to i. In DSATUR, vertices are colored sequentially in order of decreasing saturation degrees (ties are broken by considering the number of adjacent vertices which are still not colored),
- tabu search is one of the best approach to improve an initial solution produced by DSATUR,
- it is better to solve a series of k-coloring problems with decreasing k values rather than solving the problem directly,
- a solution space where the co-node constraint is always satisfied is to be preferred (i.e., a single vertex cannot be assigned conflicting colors, but two adjacent vertices can),
- a neighborhood structure where all colors of a given vertex are modified at once seems to be more efficient, but it can also be jointly used with a neighborhood where only one conflicting color in a single vertex is modified.

In Lim et al. (2005), heuristics are proposed for the set-T-coloring problem. The authors introduce a greedy heuristic, a tabu search and a squeaky wheel optimization approach. A global algorithm integrating the three methods is also reported. In Satratzemi (2004), four heuristics are developed for the multi-coloring problem. Two are DSATUR-based methods and the two others color the (still not colored) maximum stable set at each iteration. The methods differ by the way ties are broken. Due to the correspondence between problem P and the k-multi-coloring problem, the coloring terminology (e.g., vertex, color, edge) and the scheduling terminology (e.g., job, time period, incompatibility) will be used indifferently in the remainder.

2.3. Summary

As highlighted in Section 2.2.2, many extensions of the graph coloring problem have been proposed to address scheduling problems in different production systems. In particular, problem P can be related to the multi-coloring problem, for which several algorithms including tabu search are reported in the literature. In our case, however, a weight (gain) is associated with each vertex and the primary objective is to maximize the sum of the gains associated with completely colored vertices rather than minimizing the number of colors or number of conflicts. Therefore, previous methodologies for the classical multi-coloring problem must be adapted in this new setting. Section 2.1 helped to motivate our choice for the secondary objectives, as well as their integration within a lexicographic ordering scheme. Section 2.1.1 showed that, even if preemption can help to improve the performance of production systems, it must be penalized to avoid the occurrence of too many job interruptions in a solution. Then, Section 2.1.2 explained that work in progress inventories lead to non negligible costs and also need to be taken into account.

3. Formal definition of problem *P*

Problem P consists in scheduling a set of n jobs on an unconstrained number of parallel machines while allowing preemption. We also consider job incompatibilities which are modeled by a graph $G = (V, E)$, where vertex $i \in V$ is a job and edge $(i, j) \in E$ is created if jobs i and j are incompatible. With each job i is associated an integer processing time p_i and a gain g_i. A global deadline D is also given, so that no job can be processed after D. To collect the gain g_i, job i must be given p_i time units, otherwise the job is rejected. To write a mathematical formulation of the problem, the following decision variables are needed:

- $z_i = 1$ if job i is completely performed, 0 otherwise,
- Max_i = largest time unit assigned to i (equal to 0 if i is rejected),

- Min_i = smallest time unit assigned to i (equal to 0 if i is rejected),
- $s_{it} = 1$ if job i starts or is resumed (after preemption) in time unit t, 0 otherwise,
- $u_t = 1$ if time unit t is used, 0 otherwise,
- $x_{it} = 1$ if job i is processed in time unit t, 0 otherwise,

The mathematical model is then:

$$\max \ \alpha_1 \cdot \sum_{i \in V} (z_i \cdot g_i) \ - \ \alpha_2 \cdot \sum_{i \in V} \sum_{t=1}^{D} s_{it} \ - \ \alpha_3 \cdot \sum_{i \in V} (Max_i - Min_i + z_i) \quad (1)$$

s.t.

$$\sum_{t=1}^{D} x_{it} = p_i \cdot z_i \qquad i \ \in V \quad (2)$$

$$t \cdot x_{it} \leq Max_i \qquad 1 \ \leq \ t \leq D, \ \ i \in V \quad (3)$$

$$t \cdot x_{it} + D \cdot (1 - x_{it}) \geq Min_i \qquad 1 \ \leq \ t \leq D, \ \ i \in V \quad (4)$$

$$D \cdot z_i \geq Min_i \qquad i \in V \quad (5)$$

$$u_t = x_{it} + x_{jt} \qquad 1 \ \leq \ t \leq D, \quad (i,j) \in E \quad (6)$$

$$s_{it} \geq x_{it} - x_{i(t-1)} \qquad 1 \ \leq \ t \leq D, \ \ i \in V \quad (7)$$

$$s_{it}, \ z_i, \ x_{it}, \ u_t \in \{0,1\} \qquad 1 \leq t \leq D, \quad i \in V \quad (8)$$

$$Min_i, Max_i \geq 0 \ \text{ and integer} \quad i \in V \quad (9)$$

The three terms in the objective function correspond to the total gain associated with completely performed jobs, number of job interruptions and work in progress, respectively, while the corresponding weights α_1, α_2 and α_3 are chosen to obtain the desired lexicographic ordering. In the second term, the summation over the s_{it} variables also includes the first fragment of a job, but

this is equivalent to minimizing the number of job interruptions when the number of completely performed jobs is fixed (which is the case here, due to the lexicographic ordering among the three objectives). For the third term, z_i is 1 if job i is performed and $Max_i - Min_i + 1$ then corresponds to the number of time units.

Constraint (2) imposes the assignment of p_i time units to each completely performed job i. Constraints (3), (4) and (5) define Min_i and Max_i for each job i. Constraint (5) sets Min_i to 0 when job i is rejected, otherwise it indicates that Min_i should not exceed D. Note that when Min_i is forced to 0 when job i is rejected, constraint (2) forces the corresponding x_{it} variables to 0, while constraint (3) becomes $Max_i \geq 0$. Accordingly, Max_i is automatically set to 0 due to the minimization of the third term in the objective function. Constraint (6) states that two adjacent jobs in the incompatibility graph G cannot share a common time unit. It also enforces u_t to be equal to 1 if time unit t is used. Constraint (7) indicates a start or a restart by forcing s_{it} to be 1 when $x_{it} = 1$ and $x_{i(t-1)} = 0$. Otherwise the subtraction is negative or null and s_{it} gets value 0 through the minimization of the second term in the objective function.

As highlighted in Section 2.2.1, problem P can easily be viewed as a multi-coloring problem by replacing time slots by colors in the above description. This observation allows us to take advantage of the rich literature on graph coloring.

4. Basic heuristics for problem *P*

In this section, basic heuristic methods to tackle our problem are presented. A constructive method is first introduced, followed by a descent local search.

4.1. Constructive heuristic

The greedy constructive heuristic starts from an uncolored incompatibility graph and builds a complete solution by coloring a new vertex at each itera-

tion. Let U be the set of vertices that have not been considered yet. At each step, vertex $i \in U$ with the largest gain g_i is deleted from U and is either rejected or fully colored by assigning p_i admissible colors to it (i.e., colors not found in adjacent vertices). Let A_i be the set of remaining admissible colors for vertex i. If $|A_i|$ is smaller than p_i, vertex i is automatically rejected, otherwise, i is completely colored. Note that if p_i consecutive colors are available in A_i, they should be automatically selected as they lead to the minimum additional work in progress and do not induce any job interruption. Based on such observations, the following procedure is proposed to color vertex i when $|A_i| > p_i$:

1. Find the largest subset of consecutive colors in A_i and call it B_i. If the size of B_i is at least p_i, then color vertex i with p_i (randomly chosen) consecutive colors in B_i. Otherwise, assign all colors in B_i to vertex i, remove these colors from A_i, and go to step 2.
2. While vertex i is not fully colored, add a color in A_i that minimizes the additional number of job interruptions (either 0 or 1 additional interruption) and remove that color from A_i. In case of ties, choose the color that minimizes the additional work in progress.

This color assignment procedure is fast and randomized (as many ties might occur). Therefore, different runs of the greedy constructive heuristic are likely to produce different solutions. This characteristic is exploited in the following to provide different starting points for our descent method.

4.2. Descent method

Local search methods are heuristics which, starting from an initial solution, moves from the current solution s to another solution in the neighborhood of s. The latter is denoted $N(s)$ and is obtained by performing a given type of moves on s which slightly modifies its structure. In a descent method, the best neighbor solution is selected at each iteration. When the method gets trapped in the first encountered local optimum (with regard to the chosen neighborhood), a restart procedure can be applied to escape from it. Many restarts can thus be performed until a given time limit is reached to obtain

a so-called multi-start descent method. At the end, the best solution found is returned.

In the multi-start descent method developed for our problem, every initial solution is provided by the greedy constructive heuristic. As preliminary experiments have shown that changing only one color at a time leads to poor results, we consider moves which fully color (if the vertex has previously been rejected) or fully recolor a vertex (if the vertex has previously been colored). Thus, a neighbor solution s' of s is obtained by assigning p_i colors to a given vertex i. The neighborhood $N(s)$ is then obtained by applying this type of move to all vertices, including those that have been rejected from the current solution. The procedure for selecting the subset of colors assigned to a vertex is explained in the following (note that the term recoloring will be used for both colored and uncolored vertices).

When recoloring vertex i, two situations can occur. On the one hand, if there are at least p_i colors in set A_i, the colors are assigned one by one to vertex i by selecting the color that minimizes the saturation degree of the adjacent uncolored vertices. We recall that the saturation degree of a vertex is the number of different colors used in adjacent vertices. At each step, the color $c \in A_i$ that maximizes $\sum_{i' \in Z(i)} u_c(i')$ is chosen, where $Z(i)$ is the set of adjacent uncolored vertices of vertex i, and $u_c(i') = 1$ if color c is used in vertices adjacent to i' and 0 otherwise. The goal here is to assign a color which leads to the smallest increase in the saturation degree of the adjacent uncolored vertices. We denote this procedure by R^{SD}.

On the other hand, if it is not possible to find p_i admissible colors for vertex i (i.e., $|A_i| < p_i$), the recoloring of i is enforced as follows. First, all colors in A_i are assigned to i, and the other colors are assigned one by one, by selecting at each step the color that minimizes the gain reduction. That is, if the non admissible color c is assigned to vertex i, all adjacent vertices with color c are uncolored and the associated gain is lost. This procedure is called R^{Enf} (for enforced recoloring).

5. Tabu search

Tabu search is also a local search algorithm. In contrast with a pure descent method, a tabu list is used to prevent cycling when a local optimum is reached by forbidding to undo some recently performed moves. Then, at each iteration, the tabu search applies the best non tabu move (either improving or not). We define below the main components used in our tabu search implementation for problem P, namely the initial solution, the neighborhood structure, the tabu list as well as a neighborhood restriction technique and a diversification mechanism. Finally, some ways to put more emphasis on objectives f_2 (number of job interruptions) and f_3 (work in progress) are proposed.

Initial solution. The initial solution is provided by the greedy constructive heuristic.

Neighborhood structure. The neighborhood structure is based on the union of the three types of moves R^{SD}, R^{Enf} and R^{Drop}, where the latter simply consists in removing all colors of a given vertex. Since this move degrades solution quality, it can only be applied within a tabu search framework. Clearly, it will not be performed at all if a neighbor solution can be obtained with R^{SD}. But, it might lead to a better solution than a move of type R^{Enf} which is likely to uncolor several vertices. Note that exploiting the union of different types of moves has proven to be successful on different types of problems (e.g., Wu et al. (2012), Lü et al. (2011)).

Tabu status. It is forbidden to recolor a recently recolored or uncolored vertex for *tab* (parameter) iterations.

Neighborhood reduction technique. A fraction (tuned to 25%) of the whole neighborhood is considered at each iteration. The sample of neighbor solutions is randomly chosen.

Diversification strategy. Every I iterations (parameter tuned to 150) without improvement to the best solution found, b% (parameter tuned to 20%) of the colored vertices are randomly chosen and uncolored. It should be noted

that more sophisticated strategies (like selecting the vertices with colors that did not change for the largest number of iterations) did not provide any improvement over this simple random rule.

Despite the good performance of our tabu search with regard to the total gain f_1, the solutions obtained in preliminary experiments had quite large values for the number of job interruptions f_2 and the work in progress f_3. To address this issue, an exact recoloring procedure (i.e., able to optimally color a vertex with regard to f_2 and then f_3) was used. Our approach, as well as its integration within the neighborhood structure of our local search methods, are described below.

Let us assume that we have to recolor vertex i and that $|A_i| \geq p_i$. The exact algorithm consists in finding the subset of colors in A_i that minimizes the total cost (preemption followed by work in progress) associated with the recoloring of vertex i. Here, a vertex coloring is an ordered list or sequence of colors of fixed size. The methodology is based on the following property.

> Let S and S' be two non decreasing sequences of colors ending with the same color c'. If the cost associated with S is smaller than the cost associated with S', then adding color c at the end of S leads to a partial solution whose cost is smaller than the one obtained by adding c at the end of S'.

This property stems from the fact that the cost increase depends only on colors c and c'. The increase in the number of job interruptions (resp. work in progress) due to the addition of color c at the end of a sequence ending with color c' is denoted $Int(c', c)$ (resp. $Wip(c', c)$) and is defined as follows for all $c > c'$:

$$Int(c', c) = \begin{cases} 1 & \text{if } c > c' + 1 \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

$$Wip(c', c) = c - c' \tag{11}$$

A pseudo-code for identifying the optimal recoloring of vertex i is found in Algorithms 1 and 2, where $L_{int}(q, c)$ and $L_{wip}(q, c)$ are the optimal values

for the number of job interruptions and work in progress, respectively, for a color sequence $BestS$ of length q ending with color c. To obtain the true optimum, the procedure $R(q, c)$ must be called for all q from 1 to p_i and for all $c \in A_i$. As the complexity of the latter is $O(|A_i|)$, the exact recoloring algorithm R^{Exact} runs in $O(p_i \cdot |A_i|^2)$.

It should be noted that set B in $R(q, c)$ is used to memorize the best sequences with regard to the number of job interruptions f_2, as there might be more than one. Then, these sequences are evaluated for the work in progress f_3. If there are still two or more best sequences, ties are broken randomly as indicated by the assignment of c^* to c' so that the i^{th} sequence has probability $\frac{1}{i}$ to be selected. The + sign in the last statement is used to indicate that color c is added at the end of the sequence $BestS(q-1, c^*)$.

Algorithm 1 $R^{Exact}(i)$

```
For q from 1 to p_i do
    For c in A_i do
        call R(q,c);
```

The exact recoloring procedure is applied on a vertex i only when $|A_i| \geq p_i$. In the following, two strategies to integrate R^{Exact} into the tabu search are proposed. Clearly, R^{Exact} always leads to a better solution than R^{SD} when applied to a given vertex. But, surprisingly, the tabu search returns better solutions at the end when it is run with R^{SD}, instead of R^{Exact}. It seems that R^{SD} better deals with the total gain f_1 by forbidding a smaller number of colors on the adjacent uncolored vertices. These vertices are then more likely to be recolored in the next iterations. Based on this observation, R^{SD} must be part of the neighborhood and cannot be simply used as a filter to quickly identify promising vertices for R^{Exact}. Accordingly, the two following ways of integrating R^{Exact} into the neighborhood structure are proposed:

1. R^{SD}, R^{Enf} and R^{Drop} are first used jointly and the q (parameter) vertices leading to the best solutions when recolored with R^{SD} become potential candidates for the exact procedure. Then, 25% of the candidate vertices are (randomly) selected, colored with R^{Exact} and added

Algorithm 2 R(q,c)

If ($q = 1$) then return BestS(q,c) = $\{c\}$, $L_{int}(q, c) = 0$, $L_{wip}(q, c) = 1$
Else

1. $Best_{int} \leftarrow \infty$, $Best_{wip} \leftarrow \infty$, $B \leftarrow \emptyset$
2. for $c' \in A_i$ such that $c' < c$ do
   (a) if $(c-c') = 1$ then $Int \leftarrow L_{int}(q-1, c')$ else $Int \leftarrow L_{int}(q-1, c')+1$
   (b) if $Int = Best_{int}$ then $B \leftarrow B \cup \{c'\}$
   (c) if $Int < Best_{int}$ then $Best_{int} \leftarrow Int$, $B \leftarrow \{c'\}$
3. for $c' \in B$ do
   (a) $Wip \leftarrow L_{wip}(q-1, c') + (c - c')$
   (b) if $Wip = Best_{wip}$ then $c^* \leftarrow c'$ (with some probability)
   (c) if $Wip < Best_{wip}$ then $c^* \leftarrow c'$, $Best_{wip} \leftarrow Wip$
4. return $BestS(q, c) = BestS(q-1, c^*) + \{c\}$, $L_{int}(q, c) = Best_{int}$, $L_{wip}(q, c) = Best_{wip}$;

to the neighborhood of the current solution. This percentage is the result of a compromise: if too large, R^{SD} would never contribute to the search; if too small, the impact of the exact procedure would be negligible.

2. R^{Exact}, R^{Enf} and R^{Drop} are jointly used with a probability γ (parameter), whereas R^{SD}, R^{Enf} and R^{Drop} are jointly used with a probability $1 - \gamma$.

A pseudo-code for the tabu search is given in Algorithm 3, while its neighborhood management strategy is found in Algorithm 4.

6. Experiments

In this section, the methodologies to be compared and the test instances are presented. Then, the results obtained are reported and analyzed.

In the following, LP denotes the application of the exact CPLEX solver to the

Algorithm 3 Tabu search

Generate an initial solution s with *Greedy* (see Section 4.1).
While no stopping condition is met do:

1. generate the neighborhood $N(s)$ of solution s, with R^{Exact} integrated according to either strategy 1 or 2,
2. identify the best non tabu solution s' in $N(s)$ and the vertex v associated with this move,
3. update the tabu tenure: forbid vertex v to be recolored for *tab* iterations,
4. $s \leftarrow s'$,
5. if $I = 150$ consecutive iterations have been performed since the last improvement of the best encountered solution, uncolor $b = 20\%$ of the colored vertices in s.

Return best solution found during the search.

linear programming model of Section 3. The methods *Greedy* and *Descent* are those previously introduced in Section 4. With regard to the tabu search, the improvement provided by each algorithmic component is highlighted by adding these components one by one. More precisely, TS denotes the tabu search with R^{SD} and R^{Enf} only, while TS^{Drop} is obtained when R^{Drop} is added to the neighborhood structure. Then, TS^{Div} includes the diversification procedure with $I = 150$ and $b = 20\%$. Finally, R^{Exact} is added to TS^{Div} using strategies 1 and 2 to obtain TS^{E1} and TS^{E2}, respectively. In these two variants, parameter q is tuned to 20 and parameter γ to 0.2. Note also that the tabu tenure *tab* is tuned to 10 for instances with more than 50 jobs and to 3 otherwise.

The test instances have been generated as in Dorne & Hao (1998). Two parameters are used for this purpose: the number of jobs n and the graph density d (probability of having an edge between two vertices). A total of 90 instances were generated, with $n \in \{10, 25, 50, 100, 300, 500\}$ and $d \in \{0.2, 0.5, 0.8\}$. For each (n, d) couple, five instances were generated and labeled with letters a, b, c, d and e. The processing time p_i of job i is an

Algorithm 4 Neighborhood management strategy at each iteration

If strategy 2 is used, set R to R^{Exact} or R^{SD} with probability 0.2 and 0.8, respectively.
For each vertex $i \in V$ do:

1. if i is colored, perform R^{Drop} on i (with a probability of 25%),
2. do with a probability of 25%:
   (a) if $|A_i| < p_i$, perform R^{Enf} on i,
   (b) else,
      i. if strategy 1 is used, perform R^{SD} on i and update the set Q of the q best vertices,
      ii. if strategy 2 is used, perform R,

If strategy 1 is used, perform R^{Exact} on 25% of the vertices in Q.

integer randomly chosen in interval $[1, 10]$. The gain g_i is related to the processing time p_i, as observed in practice where larger jobs bring more profits. Basically, a random number β is chosen in the interval [1,20] and $g_i = \beta \cdot p_i$. For each instance, the deadline D is chosen small enough to prevent the scheduling of all jobs.

The algorithms were implemented in C++ and executed on a computer with a processor Intel Quad-core i7 2.93 GHz with 8 GB of DDR3 RAM memory. A time limit of $60 \cdot n$ seconds was used, where n is the number of jobs, and 5 runs per instance were performed with each algorithm, except LP for which a single run was performed with a time limit of 10 hours and a memory limit of 7 GB.

For the sake of completeness, the detailed results on each individual instance are provided in the Appendix in Tables A1 and A2 for small and large instances, respectively. In these tables, column *Instance* refers to the name *n_d_letter* of the test instance. Then, column *Bound* contains the best known bound for each objective in lexicographic order (i.e., total gain, number of job interruptions and work in progress). A straightforward bound for the first objective is the sum of the gains over all jobs (i.e., $\sum_{i \in V} g_i$). For the two other objectives, the bound is set to the smallest value found over all our

experiments. The other columns in these tables contain, for each method, the average gap in percentage between the solution value obtained for each of the three objectives and the corresponding bound, still in lexicographic order. In column LP, optimal solutions are indicated with $(\ldots)^*$, feasible solutions returned when the memory limit is reached are indicated with $(\ldots)^m$, whereas those returned when the time limit is reached are indicated with $(\ldots)^t$. Table A1, in particular, shows that LP is able to solve all instances with 10 jobs and all instances with 25 jobs when the graph density is 0.2. In the other cases, an upper bound is found.

Tables 1 and 2 below correspond to an aggregated version of the tables in the Appendix, by averaging the results over the five instances labeled a, b, c, d and e for each (n, d) couple. Table 1 shows that the solutions obtained with LP are largely improved by the other methods on the test instances with 50 jobs. In particular, the average gap associated with the first objective on these instances is 30% for LP versus 24.6% for TS^{Div}. Consequently, LP was not run on the large instances. Over all instances of size 10, 25 and 50, LP obtains an average gap for the first objective of 19.4%, versus 19.1% for *Greedy*, but it is not competitive on the two other objectives. *Greedy* and *Descent* produce similar results with average gaps of 19.1%, 4.1%, 7.7% and 18.8%, 5.6%, 11.6%, respectively, for the three objectives (in lexicographic order). TS improves *Greedy* by 1.1% on the first objective but, when both methods obtain the same values for this objective, *Greedy* outperforms TS most of the time on the secondary objectives (see, for example, instances 10_2_b, 10_2_c, and 10_2_d in Table A1 in the Appendix). Adding R^{Drop} and the diversification mechanism lead to the best results with regard to the primary objective. However, the results for the secondary objectives are poor with average gaps of 140.4% and 136.2% for TS^{Drop}, and 106.2% and 140.7% for TS^{Div}. Introducing R^{Exact} clearly provides a huge improvement, given that TS^{E1} and TS^{E2} produce gaps of 17.6%, 43.6%, 42% and 17.6%, 77.8%, 66.2%, respectively, for the three objectives. Overall, TS^{E1} is the best method on average. It also produces the best solutions on 28 instances out of 45.

The aggregated results for the large instances of size 100, 300 and 500 are

reported in Table 2. On this set of instances, TS is better than $Greedy$, which is slightly better than $Descent$. The average gaps with regard to the first objective are 26.4%, 28%, and 28.7%, respectively. Adding R^{Drop} leads to a small improvement at 25.9%. However, the diversification mechanism provides the best results with a gap of 23.9%. Finally, adding R^{Exact} does not provide an implementation that matches TS^{Div} with gaps of 25.5% for TS^{E1} and 25.2% for TS^{E2}, but huge improvements are observed with regard to the secondary objectives.

Overall, TS^{Div}, TS^{E1} and TS^{E2} are the best methods. If we consider the lexicographic ordering of the three objectives, TS^{E1} leads to the best average results on small instances with up to 50 jobs, while TS^{Div} is the best on larger instances. In the latter case, however, TS^{E1} and TS^{E2} considerably reduce the values of the second (number of job interruptions) and third (work in progress) objectives, at the expense of a slight deterioration in the first objective (total gain). These results demonstrate the effectiveness of the recoloring procedure R^{SD} for maximizing the total gain: the more it is used, the better are the results. We also observe that strategy 1 is better than strategy 2 when R^{Exact} is integrated within the tabu search. Finally, the graph density does not have any impact on the ranking of the proposed methods.

Based on these results, TS^{Div} is certainly a method to consider when the focus is on the total gain. But TS^{E1} is also interesting by providing the best results on small instances and by considerably improving the number of job interruptions and work in progress (at the expense of a slight deterioration in the total gain on large instances).

n	d	LP	$Greedy$	$Descent$	TS	TS^{Drop}	TS^{Div}	TS^{E1}	TS^{E2}
10	0.2	**(18.8 ; 2.5 ; 3.4)**	(21.5 ; 0 ; 0)	(18.8 ; 2.5 ; 3.4)	(18.8 ; 7.1 ; 7.5)	(18.8 ; 9.5 ; 6.9)	(18.8 ; 4.8 ; 7.4)	**(18.8 ; 2.5 ; 3.4)**	(18.8 ; 3 ; 3.9)
	0.5	**(13.1 ; 0 ; 0)**	(13.1 ; 4.7 ; 9.9)	**(13.1 ; 0 ; 0)**	(13.1 ; 29.9 ; 31.6)	(13.1 ; 30.8 ; 29.3)	(13.1 ; 11.9 ; 20.5)	**(13.1 ; 0 ; 0)**	(13.1 ; 8.1 ; 4.7)
	0.8	**(15.6 ; 3.3 ; 1.1)**	(15.8 ; 0 ; 0)	(15.8 ; 0 ; 0)	(15.6 ; 59.4 ; 51.2)	(15.6 ; 60.4 ; 50.7)	(15.6 ; 28.7 ; 55.5)	**(15.6 ; 3.3 ; 1.1)**	(15.6 ; 16 ; 14.4)
	Avg.	**(15.8 ; 1.9 ; 1.5)**	(16.8 ; 1.6 ; 3.3)	(15.9 ; 0.8 ; 1.1)	(15.8 ; 32.1 ; 30.1)	(15.8 ; 33.6 ; 28.9)	(15.8 ; 15.1 ; 27.8)	**(15.8 ; 1.9 ; 1.5)**	(15.8 ; 9 ; 7.6)
25	0.2	**(16.7 ; 5.5 ; 5.8)**	(20.1 ; 0 ; 0.2)	(20.1 ; 0 ; 0.2)	(16.7 ; 79.6 ; 54.2)	(16.7 ; 79.4 ; 53.4)	(16.7 ; 46.6 ; 58.1)	(16.7 ; 6.8 ; 8.1)	(16.7 ; 26 ; 22.5)
	0.5	(8.9 ; 12.4 ; 15.8)	(9.3 ; 0.8 ; 2.9)	(9.2 ; 3.4 ; 7.8)	(8.4 ; 236.2 ; 191.3)	(8.4 ; 207.5 ; 176.8)	(8.5 ; 121.9 ; 172.8)	**(8.4 ; 19.8 ; 28.9)**	(8.4 ; 73.7 ; 70.7)
	0.8	(11.9 ; 19.1 ; 43.8)	(12.3 ; 6.5 ; 15.1)	(11.8 ; 27.4 ; 57.7)	(11.6 ; 247.4 ; 356.8)	(11.6 ; 216.4 ; 339.5)	(11.6 ; 163 ; 323.8)	**(11.6 ; 40.5 ; 65.7)**	(11.6 ; 130.2 ; 153.7)
	Avg.	(12.5 ; 12.3 ; 21.8)	(13.9 ; 2.4 ; 6.1)	(13.7 ; 10.3 ; 21.9)	(12.2 ; 187.7 ; 200.7)	(12.3 ; 167.8 ; 189.9)	(12.3 ; 110.5 ; 184.9)	**(12.2 ; 22.4 ; 34.3)**	(12.2 ; 76.6 ; 82.3)
50	0.2	**(18.4 ; 73.5 ; 45.1)**	(21 ; 8 ; 7.6)	(21.2 ; 2.2 ; 3.8)	(19.4 ; 172 ; 89.3)	(18.6 ; 111.5 ; 66.7)	(18.6 ; 115.3 ; 86.1)	(18.8 ; 46.7 ; 26.2)	(18.7 ; 80.4 ; 39.3)
	0.5	(35.5 ; 223.6 ; 151.4)	(30.1 ; 8.4 ; 20)	(30.7 ; 5.3 ; 9.5)	(29.9 ; 265.6 ; 179.6)	(28.4 ; 246.6 ; 182.5)	(28.2 ; 187.7 ; 194.4)	(28.2 ; 102.6 ; 76)	**(28.1 ; 130.9 ; 84.4)**
	0.8	(36.2 ; 104 ; 136.7)	(28.7 ; 8.1 ; 13.4)	(28.9 ; 10 ; 21.6)	(28.1 ; 343.2 ; 344)	**(27 ; 301.1 ; 320.6)**	(27.1 ; 276 ; 347.7)	(27.4 ; 170 ; 168.4)	(27.2 ; 231.8 ; 202.2)
	Avg.	(30 ; 133.7 ; 111.1)	(26.6 ; 8.2 ; 13.7)	(26.9 ; 5.8 ; 11.7)	(25.8 ; 260.3 ; 204.3)	(24.7 ; 219.7 ; 189.9)	**(24.6 ; 193 ; 209.4)**	(24.8 ; 106.4 ; 90.2)	(24.7 ; 147.7 ; 108.6)
Avg.		(19.4 ; 49.3 ; 44.8)	(19.1 ; 4.1 ; 7.7)	(18.8 ; 5.6 ; 11.6)	(18 ; 160 ; 145.1)	(17.6 ; 140.4 ; 136.2)	(17.6 ; 106.2 ; 140.7)	**(17.6 ; 43.6 ; 42)**	(17.6 ; 77.8 ; 66.2)

Table 1: Aggregated results on small instances

n	d	$Greedy$	$Descent$	TS	TS^{Drop}	TS^{Div}	TS^{E1}	TS^{E2}
100	0.2	(40.3 ; 9.5 ; 11.1)	(40.8 ; 4.4 ; 5.6)	(37 ; 196.6 ; 102.2)	(37.5 ; 158 ; 93.4)	**(36.1 ; 233.2 ; 126)**	(36.3 ; 93.1 ; 52.6)	(36.4 ; 119.5 ; 60.8)
	0.5	(55.8 ; 10.6 ; 20.3)	(56.2 ; 6.2 ; 11.4)	(54.4 ; 314.5 ; 222.7)	(52.6 ; 297.9 ; 210.7)	**(52.4 ; 331.5 ; 233.3)**	(52.6 ; 149.4 ; 98.8)	(52.7 ; 186.1 ; 125.8)
	0.8	(54 ; 2.8 ; 21.8)	(54.5 ; 10.5 ; 18)	(53 ; 340 ; 336.3)	(51.1 ; 324 ; 328.8)	**(51.1 ; 345.1 ; 355.6)**	(51.3 ; 245.4 ; 199.9)	(51.2 ; 268.2 ; 239.7)
	Avg.	(50 ; 7.6 ; 17.7)	(50.5 ; 7 ; 11.7)	(48.2 ; 283.7 ; 220.4)	(47.1 ; 260 ; 211)	**(46.6 ; 303.3 ; 238.3)**	(46.7 ; 162.6 ; 117.1)	(46.8 ; 191.3 ; 142.1)
300	0.2	(16.3 ; 11 ; 19.2)	(17.3 ; 5.5 ; 13.4)	(13.8 ; 107.8 ; 141.2)	(14.2 ; 89.8 ; 120.3)	**(9.4 ; 224.2 ; 260.1)**	(12.5 ; 78.2 ; 75.7)	(11.8 ; 105.8 ; 106.1)
	0.5	(19.4 ; 7.8 ; 14.9)	(20.5 ; 5.6 ; 11.6)	(16.5 ; 141.5 ; 226.7)	(15.6 ; 120.6 ; 217.9)	**(13.1 ; 262.8 ; 421.6)**	(15.7 ; 136.5 ; 163)	(15.3 ; 147.6 ; 184.4)
	0.8	(16.3 ; 5.1 ; 18.4)	(16.9 ; 7.4 ; 30.2)	(13.7 ; 100.2 ; 295.6)	(13.1 ; 94.1 ; 273.9)	**(10.7 ; 223.4 ; 592.3)**	(13.4 ; 116.7 ; 246.6)	(12.8 ; 130.5 ; 260.2)
	Avg.	(17.3 ; 8 ; 17.5)	(18.2 ; 6.2 ; 18.4)	(14.7 ; 116.5 ; 221.1)	(14.3 ; 101.5 ; 204)	**(11 ; 236.8 ; 424.7)**	(13.9 ; 110.5 ; 161.8)	(13.3 ; 127.9 ; 183.6)
500	0.2	(14.9 ; 5 ; 16.8)	(15.7 ; 5.2 ; 20.4)	(14.3 ; 49.5 ; 81)	(14.3 ; 43.8 ; 75.1)	**(10 ; 182.9 ; 273.2)**	(13.3 ; 82.5 ; 99.4)	(13 ; 69.2 ; 76.8)
	0.5	(21.5 ; 4.4 ; 19.9)	(22.2 ; 3.6 ; 15.6)	(21.3 ; 42.8 ; 90.9)	(21.4 ; 41.5 ; 86.9)	**(19.1 ; 135.3 ; 277.1)**	(20.9 ; 58.1 ; 94.1)	(20.8 ; 71.9 ; 113.6)
	0.8	(13.9 ; 4.6 ; 15.2)	(14 ; 7.5 ; 35)	(13.3 ; 28.1 ; 89.5)	(13.1 ; 32.8 ; 98.7)	**(12.8 ; 44.2 ; 132.6)**	(13.3 ; 32.4 ; 82.5)	(12.9 ; 29.5 ; 68.7)
	Avg.	(16.8 ; 4.7 ; 17.3)	(17.3 ; 5.4 ; 23.7)	(16.3 ; 40.1 ; 87.1)	(16.3 ; 39.4 ; 86.9)	**(14 ; 120.8 ; 227.6)**	(15.8 ; 57.7 ; 92)	(15.5 ; 56.9 ; 86.4)
Avg.		(28 ; 6.7 ; 17.5)	(28.7 ; 6.2 ; 17.9)	(26.4 ; 146.8 ; 176.2)	(25.9 ; 133.6 ; 167.3)	**(23.9 ; 220.3 ; 296.9)**	(25.5 ; 110.3 ; 123.6)	(25.2 ; 125.4 ; 137.4)

Table 2: Aggregated results on large instances

7. Conclusion

In this paper, an order acceptance and scheduling problem is solved with different heuristic and metaheuristic methods, in particular a tabu search. Three different objectives are considered in lexicographic order to produce solutions of interest in practical settings. The effectiveness of several algorithmic components within the tabu search are empirically demonstrated like the joint use of several types of neighborhoods, the diversification mechanism, and an exact recoloring procedure.

Several extensions of our problem can be of interest. First, the number of parallel machines is unrestricted, although a more realistic variant would fix it to a given number. The makespan could also be part of the objectives. Finally, it is assumed that each resource (or tool) exists in a single exemplar, but we might want to consider different types of tools with many exemplars of each type. This would lead to an hypergraph multi-coloring problem. Each edge would represent a tool and would link that tool to all jobs that need it. In this case, the number of conflicts associated with each edge and each color would need to be smaller than or equal to the number of exemplars of the tool modeled by the edge.

Appendix

Inst.	*Bound*	*LP*	*Greedy*	*Descent*	*TS*	TS^{Drop}	TS^{Div}	TS^{E1}	TS^{E2}
10_0.2_a	(627, 5, 28)	$(\mathbf{52.2}, \mathbf{0}, \mathbf{0})^*$	**(52.2, 0, 0)**	**(52.2, 0, 0)**	**(52.2, 0, 0)**	**(52.2, 0, 0)**	**(52.2, 0, 0)**	**(52.2, 0, 0)**	**(52.2, 0, 0)**
10_0.2_b	(295, 9, 35)	$(\mathbf{3.1}, \mathbf{0}, \mathbf{0})^*$	**(3.1, 0, 0)**	**(3.1, 0, 0)**	(3.1, 6.7, 1.7)	(3.1, 17.8, 6.3)	(3.1, 2.2, 3.4)	**(3.1, 0, 0)**	**(3.1, 0, 0)**
10_0.2_c	(668, 8, 38)	$(\mathbf{19.6}, \mathbf{0}, \mathbf{0})^*$	**(19.6, 0, 0)**	**(19.6, 0, 0)**	(19.6, 5, 5.3)	(19.6, 5, 2.1)	(19.6, 3.8, 7.9)	**(19.6, 0, 0)**	**(19.6, 0, 0)**
10_0.2_d	(366, 9, 34)	$(\mathbf{4.9}, \mathbf{0}, \mathbf{0})^*$	(4.9, 0, 0)	(4.9, 0, 0)	(4.9, 8.9, 12.9)	(4.9, 2.2, 6.5)	(4.9, 1, 5.3)	**(4.9, 0, 0)**	**(4.9, 0, 0)**
10_0.2_e	(395, 8, 35)	$(\mathbf{14.2}, \mathbf{12.5}, \mathbf{17.1})^*$	(27.8, 0, 0)	**(14.2, 12.5, 17.1)**	(14.2, 15, 17.7)	(14.2, 22.5, 19.4)	(14.2, 17, 20.6)	**(14.2, 12.5, 17.1)**	(14.2, 15, 19.4)
10_0.5_a	(351, 9, 42)	$(\mathbf{8.5}, \mathbf{0}, \mathbf{0})^*$	**(8.5, 0, 0)**	**(8.5, 0, 0)**	(8.5, 82.2, 75.7)	(8.5, 60, 47.1)	(8.5, 24.8, 53.8)	**(8.5, 0, 0)**	(8.5, 11.1, 4.3)
10_0.5_b	(600, 9, 47)	$(\mathbf{12}, \mathbf{0}, \mathbf{0})^*$	(12, 11.1, 25.5)	(12, 0, 0)	(12, 22.2, 28.1)	(12, 31.1, 39.6)	(12, 10.1, 22.6)	**(12, 0, 0)**	(12, 2.2, 0.4)
10_0.5_c	(267, 6, 21)	$(\mathbf{27.3}, \mathbf{0}, \mathbf{0})^*$	**(27.3, 0, 0)**	**(27.3, 0, 0)**	**(27.3, 0, 0)**	**(27.3, 0, 0)**	**(27.3, 0, 0)**	**(27.3, 0, 0)**	**(27.3, 0, 0)**
10_0.5_d	(672, 8, 50)	$(\mathbf{10.3}, \mathbf{0}, \mathbf{0})^*$	(10.3, 12.5, 24)	**(10.3, 0, 0)**	(10.3, 7.5, 8.4)	(10.3, 25, 19.2)	(10.3, 1.3, 0)	**(10.3, 0, 0)**	(10.3, 2.5, 0.4)
10_0.5_e	(434, 9, 44)	$(\mathbf{7.4}, \mathbf{0}, \mathbf{0})^*$	**(7.4, 0, 0)**	**(7.4, 0, 0)**	(7.4, 37.8, 45.9)	(7.4, 37.8, 40.5)	(7.4, 23.2, 25.9)	**(7.4, 0, 0)**	(7.4, 24.4, 18.2)
10_0.8_a	(442, 9, 40)	$(\mathbf{1.8}, \mathbf{0}, \mathbf{0})^*$	**(1.8, 0, 0)**	**(1.8, 0, 0)**	(1.8, 53.3, 55)	(1.8, 75.6, 85)	(1.8, 32.5, 79.5)	**(1.8, 0, 0)**	(1.8, 8.9, 10.5)
10_0.8_b	(650, 7, 49)	$(\mathbf{12.9}, \mathbf{0}, \mathbf{0})^*$	**(12.9, 0, 0)**	**(12.9, 0, 0)**	(12.9, 154.3, 105.3)	(12.9, 137.1, 100.8)	(12.9, 76.6, 147.3)	**(12.9, 0, 0)**	(12.9, 37.1, 38)
10_0.8_c	(279, 6, 18)	$(\mathbf{25.1}, \mathbf{16.7}, \mathbf{5.6})^*$	(26.2, 0, 0)	(26.2, 0, 0)	**(25.1, 16.7, 5.6)**	**(25.1, 16.7, 5.6)**	**(25.1, 16.7, 5.6)**	**(25.1, 16.7, 5.6)**	**(25.1, 16.7, 5.6)**
10_0.8_d	(493, 7, 45)	$(\mathbf{8.1}, \mathbf{0}, \mathbf{0})^*$	(8.1, 0, 0)	(8.1, 0, 0)	(8.1, 68.6, 89.3)	(8.1, 68.6, 61.3)	(8.1, 15.6, 43.6)	**(8.1, 0, 0)**	(8.1, 17.1, 17.8)
10_0.8_e	(522, 5, 26)	$(\mathbf{30.1}, \mathbf{0}, \mathbf{0})^*$	**(30.1, 0, 0)**	**(30.1, 0, 0)**	(30.1, 4, 0.8)	(30.1, 4, 0.8)	(30.1, 2, 1.5)	**(30.1, 0, 0)**	**(30.1, 0, 0)**
25_0.2_a	(1275, 21, 107)	$(\mathbf{17.3}, \mathbf{0}, \mathbf{0})^*$	(17.3, 0, 0.9)	(17.3, 0, 0.9)	(17.3, 75.2, 45.6)	(17.3, 116.2, 67.1)	(17.3, 46.9, 71.6)	(17.3, 0, 0.7)	(17.3, 12.4, 11.4)
25_0.2_b	(1208, 21, 95)	$(\mathbf{7.3}, \mathbf{0}, \mathbf{0})^*$	**(7.3, 0, 0)**	**(7.3, 0, 0)**	(7.3, 41, 27.6)	(7.3, 45.7, 30.7)	(7.3, 31.2, 37.7)	**(7.3, 0, 0)**	(7.3, 14.3, 12.2)
25_0.2_c	(1441, 18, 93)	$(\mathbf{18.2}, \mathbf{16.7}, \mathbf{20.4})^*$	(31.6, 0, 0)	(31.6, 0, 0)	(18.2, 162.2, 115.7)	(18.2, 146.7, 106.2)	(18.2, 89.4, 106)	(18.2, 23.3, 31.2)	(18.2, 63.3, 50.8)
25_0.2_d	(1513, 17, 90)	$(\mathbf{29.3}, \mathbf{5.9}, \mathbf{5.6})^*$	(31.7, 0, 0)	(31.7, 0, 0)	(29.3, 71.8, 42.4)	(29.3, 54.1, 32.2)	(29.3, 40.6, 40)	**(29.3, 5.9, 5.6)**	(29.3, 24.7, 22)
25_0.2_e	(1023, 21, 94)	$(\mathbf{11.3}, \mathbf{4.8}, \mathbf{3.2})^*$	(12.4, 0, 0)	(12.4, 0, 0)	(11.3, 47.6, 39.6)	(11.3, 34.3, 30.6)	(11.3, 25.1, 35.3)	**(11.3, 4.8, 3.2)**	(11.3, 15.2, 16.2)
25_0.5_a	(1257, 18, 109)	$(\mathbf{12.8}, \mathbf{0}, \mathbf{0})^{\mathbf{t}}$	**(12.8, 0, 0)**	(12.8, 5.6, 3.3)	(12.8, 258.9, 201.7)	(12.8, 304.4, 231.2)	(12.8, 152.1, 232.8)	(12.8, 15.6, 20.9)	(12.8, 78.9, 113.9)
25_0.5_b	(1511, 21, 131)	$(\mathbf{6.1}, \mathbf{0}, \mathbf{6.1})^{\mathbf{t}}$	(6.4, 3.8, 6.1)	(6.6, 1, 0.9)	(6.1, 316.2, 223.8)	(6.1, 261, 199.2)	(6.1, 133.3, 183.1)	(6.1, 11.4, 22.3)	(6.1, 61, 57.6)
25_0.5_c	(1514, 26, 151)	$(8.3, 46.2, 41.7)^m$	(10.1, 0, 8.6)	(9.6, 3.1, 7.2)	(8.3, 173.1, 127.4)	(8.3, 136.2, 124)	(8.3, 93.7, 111.3)	**(8.3, 10.8, 24)**	(8.3, 65.4, 54)
25_0.5_d	(1664, 19, 112)	$(13.2, 15.8, 31.3)^m$	(13.2, 0, 0)	(13.2, 0, 0)	(11, 317.9, 246.8)	(11, 235.8, 198.2)	(11.1, 168.4, 203.2)	**(11, 61.1, 77.3)**	(11, 130.5, 97.7)
25_0.5_e	(1423, 24, 118)	**(4, 0, 0)**	**(4, 0, 0)**	(4, 7.5, 27.8)	(4, 115, 156.6)	(4, 100, 131.5)	(4, 62.1, 133.6)	**(4, 0, 0)**	(4, 32.5, 30.5)
25_0.8_a	(1161, 24, 116)	$(\mathbf{3}, \mathbf{12.5}, \mathbf{49.1})^{\mathbf{t}}$	(3.3, 9.2, 24.1)	(3, 50, 100.2)	(3, 208.3, 393.8)	(3.1, 220.8, 416.4)	(3.1, 151.6, 407.9)	(3, 37.5, 85.3)	(3, 97.5, 160.5)
25_0.8_b	(1717, 23, 142)	$(4.7, 13, 27.5)^t$	(4.8, 12.2, 21.5)	(4.7, 19.1, 31.1)	(4.7, 280, 376.8)	(4.7, 209.6, 320)	(4.7, 145.1, 317.5)	**(4.7, 4.3, 5.9)**	(4.7, 101.7, 123.7)
25_0.8_c	(1557, 17, 99)	$(\mathbf{38.3}, \mathbf{0}, \mathbf{0})^{\mathbf{m}}$	(38.3, 0, 7.1)	(38.3, 0, 8.5)	(38.3, 170.6, 145.5)	(38.3, 163.5, 117.4)	(38.3, 146.5, 132.7)	(38.3, 48.2, 36.2)	(38.3, 122.4, 74.1)
25_0.8_d	(1065, 23, 131)	$(4.8, 34.8, 78.6)^t$	(5.3, 11.3, 22.9)	(4.4, 54.8, 112.5)	(3.7, 258.3, 389.6)	(3.7, 238.3, 463.2)	(3.7, 171.5, 338.6)	**(3.7, 57.4, 93.7)**	(3.7, 165.2, 196.3)
25_0.8_e	(1419, 20, 113)	$(8.5, 35, 63.7)^m$	(9.7, 0, 0)	**(8.5, 13, 36.3)**	(8.5, 320, 478.4)	(8.5, 250, 380.4)	(8.5, 200.5, 422.3)	(8.5, 55, 107.4)	(8.5, 164, 213.8)
50_0.2_a	(1419, 20, 113)	$(\mathbf{17.2}, \mathbf{71.4}, \mathbf{31.6})^{\mathbf{t}}$	(19.6, 0, 0)	(19.6, 8.1, 13.9)	(18.4, 164.8, 98.1)	(17.7, 160, 92.6)	(17.8, 114.1, 96.1)	(17.9, 54.8, 34)	(17.8, 85.2, 48.2)
50_0.2_b	(2864, 42, 231)	$(29.3, 56.8, 42.5)^m$	(33.6, 10.8, 6.4)	(33.8, 0, 0)	(29.4, 169.7, 73.3)	(29.4, 69.2, 26.9)	(29.3, 91.6, 59.4)	**(29.3, 41.6, 11.4)**	(29.3, 54.1, 17.9)
50_0.2_c	(2841, 37, 233)	$(\mathbf{10}, \mathbf{26.3}, \mathbf{19.2})^{\mathbf{m}}$	(11.6, 2.6, 0)	(11.9, 2.1, 5.4)	(12.2, 172.6, 86.6)	(10.1, 97.9, 70.4)	(10.2, 103.1, 99.1)	(10.2, 24.2, 23.7)	(10.3, 69.5, 32.7)
50_0.2_d	(2429, 38, 213)	$(21.7, 122.9, 84.9)^t$	(24.4, 14.3, 25.3)	(24.6, 0.6, 0)	(21.6, 220.6, 129.7)	(21.7, 146.9, 106.6)	**(21.4, 159.4, 116.2)**	(22, 57.7, 37.2)	(21.6, 123.4, 68)
50_0.2_e	(2872, 35, 186)	$(\mathbf{13.7}, \mathbf{90}, \mathbf{47.2})^{\mathbf{t}}$	(15.7, 12.5, 6.5)	(16, 0, 0)	(15.2, 132.5, 58.9)	(13.9, 83.5, 36.9)	(14.1, 108, 59.9)	(14.4, 55, 24.7)	(14.4, 70, 29.9)
50_0.5_a	(2628, 40, 248)	$(45.8, 221.4, 169.2)^t$	(39, 0, 0)	(39.4, 12.1, 29.8)	(40.2, 270.7, 167.8)	(39, 275.7, 188)	(38.4, 182.1, 175.5)	**(37.8, 77.9, 48.7)**	(37.9, 138.6, 82.3)
50_0.5_b	(3193, 28, 182)	$(45.1, 238.5, 136.2)^t$	(33.7, 11.5, 13.2)	(35, 3.1, 1.8)	(32.9, 346.9, 217.9)	(30.3, 309.2, 209.8)	(30.2, 236.9, 226.3)	(31.2, 130, 103.4)	**(30.2, 142.3, 74.9)**
50_0.5_c	(2827, 26, 174)	$(33.3, 182.1, 116.8)^t$	(34, 17.9, 58.1)	(34.2, 2.9, 12.6)	(34.8, 248.6, 171.7)	(33.3, 238.6, 177.1)	(33.3, 187.9, 180.2)	**(32.9, 96.4, 58)**	(33.3, 135, 95.8)
50_0.5_d	(2738, 28, 167)	$(19.8, 222.2, 225.1)^t$	(16, 5.6, 26.9)	(16.6, 0, 1.9)	(13.5, 226.1, 214.7)	(12.5, 182.2, 210.6)	**(12.5, 161.4, 265.7)**	(12.9, 90, 113.7)	(12.6, 100.6, 94.4)
50_0.5_e	(2286, 36, 171)	$(33.8, 253.6, 109.6)^t$	(28.2, 7.1, 2)	(28.4, 8.6, 1.4)	(27.9, 235.7, 125.7)	(27, 227.1, 126.7)	(26.6, 170, 124.3)	**(26.5, 118.6, 56)**	(26.5, 137.9, 74.6)
50_0.8_a	(2663, 28, 197)	$(37.5, 39.4, 78.1)^m$	(25.6, 6.7, 10.8)	(25.6, 9.1, 34)	(25, 335.2, 484.7)	(24.4, 328.5, 460.4)	(24.4, 242.7, 470.6)	**(24.2, 187.9, 218.5)**	(24.3, 187.3, 243.4)
50_0.8_b	(2898, 33, 178)	$(40.1, 145.2, 185.2)^m$	(31.3, 1.9, 5.7)	(31.7, 5.8, 24)	(30.5, 345.2, 325.7)	**(28.6, 318.7, 321.2)**	(28.8, 308.4, 341.5)	(29.3, 197.4, 170.9)	(29, 264.5, 211.3)
50_0.8_c	(3396, 31, 237)	$(44.4, 59.4, 92.1)^m$	(35.4, 9.4, 32.1)	(35.9, 8.7, 13.9)	(34.4, 328.8, 227.1)	**(33.4, 291.3, 224)**	(33.6, 282.4, 248.1)	(34, 157.5, 123.1)	(33.8, 240, 163.5)
50_0.8_d	(3627, 32, 277)	$(28.8, 86.2, 62.6)^t$	(23.2, 20, 9.8)	(23.3, 15.2, 17.5)	(22.4, 382.8, 333.9)	$(\mathbf{20.7}, \mathbf{337.2}, \mathbf{308})$	(21.1, 290.7, 348.4)	(21.7, 137.2, 127)	(21.2, 235.2, 158.2)
50_0.8_e	(1969, 29, 238)	$(29.9, 190, 265.6)^t$	(27.8, 2.7, 8.4)	(27.8, 11.3, 18.8)	(28.2, 324, 348.6)	(28.1, 230, 289.3)	**(27.6, 256, 330)**	(27.8, 170, 202.4)	(27.7, 232, 234.5)
Avg.		(19.4, 49.3, 44.8)	(19.1, 4.1, 7.7)	(18.8, 5.6, 11.6)	(18, 160, 145.1)	(17.6, 140.4, 136.2)	(17.6, 106.2, 140.7)	**(17.6, 43.6, 42)**	(17.6, 77.8, 66.2)

Table A1: Detailed results on small instances

Inst.	*Bound*	*Greedy*	*Descent*	*TS*	TS^{Drop}	TS^{Div}	TS^{E1}	TS^{E2}
100_0.2_a	(5724 , 55 , 292)	(40.3 , 14.5 , 25.3)	(40.8 , 2.9 , 7.4)	(37.5 , 197.5 , 117.3)	(37.4 , 164.4 , 106.7)	**(36.9 , 215.6 , 122)**	(37 , 81.1 , 53)	(36.9 , 128.4 , 70.3)
100_0.2_b	(5760 , 55 , 314)	(45 , 0 , 11.5)	(45.4 , 6.5 , 14.9)	(39.5 , 194.5 , 94.1)	(42.9 , 140.4 , 80.4)	(39.3 , 246.5 , 121.3)	**(39.2 , 93.5 , 50.9)**	(39.9 , 103.3 , 51.3)
100_0.2_c	(5694 , 49 , 315)	(40.3 , 16.3 , 8.6)	(40.8 , 8.2 , 0.3)	(38.3 , 218.8 , 93.3)	**(36.8 , 164.1 , 84.8)**	(37.3 , 253.1 , 118.2)	(37.9 , 86.9 , 40.3)	(38 , 104.1 , 44.3)
100_0.2_d	(5092 , 60 , 325)	(34.2 , 8.3 , 2.2)	(34.8 , 3 , 3.1)	(30 , 169.3 , 107.1)	(30.7 , 156.7 , 100.5)	**(28.3 , 182 , 113.5)**	(28.8 , 80.3 , 51)	(28.6 , 111.3 , 64.4)
100_0.2_e	(5850 , 48 , 272)	(41.7 , 8.3 , 7.7)	(42.2 , 1.3 , 2.3)	(39.8 , 202.9 , 99.2)	(39.7 , 164.6 , 94.4)	(38.7 , 268.8 , 155)	**(38.4 , 123.8 , 67.6)**	(38.4 , 150.4 , 73.8)
100_0.5_a	(6012 , 29 , 179)	(63.8 , 3.4 , 10.6)	(64 , 0.7 , 7.4)	(57.3 , 393.1 , 280)	(57.4 , 353.1 , 274.6)	**(55 , 371.7 , 270.4)**	(55.4 , 160.7 , 113.2)	(55.3 , 214.5 , 146.9)
100_0.5_b	(5422 , 35 , 260)	(50.2 , 20 , 13.1)	(50.6 , 4.6 , 1.2)	(52.3 , 318.3 , 160.9)	**(48.3 , 355.4 , 170.2)**	(49.4 , 306.9 , 160.5)	(49.1 , 136 , 64.4)	(49.6 , 178.9 , 84.2)
100_0.5_c	(5507 , 37 , 206)	(57.6 , 0 , 0)	(58.2 , 8.1 , 22.7)	(56.2 , 303.2 , 247)	(55.5 , 256.8 , 233.7)	**(54.8 , 314.1 , 255.7)**	(55.1 , 141.1 , 93.2)	(55.1 , 183.2 , 138.8)
100_0.5_d	(5182 , 40 , 216)	(52.4 , 17.5 , 62)	(52.9 , 8.5 , 18.8)	(51.7 , 282 , 242.4)	**(49.3 , 219.5 , 191.1)**	(50.2 , 320.5 , 273.2)	(50.3 , 145 , 126.2)	(50.4 , 172 , 154.4)
100_0.5_e	(6112 , 33 , 221)	(54.9 , 12.1 , 15.8)	(55.1 , 9.1 , 7)	(54.7 , 275.8 , 183.1)	**(52.6 , 304.8 , 183.8)**	(52.8 , 344.2 , 206.6)	(53.1 , 164.2 , 97.1)	(53.2 , 181.8 , 104.7)
100_0.8_a	(5648 , 37 , 238)	(52.4 , 0 , 10.1)	(53.2 , 8.1 , 36.1)	(52.8 , 309.7 , 389.4)	**(49.3 , 326.5 , 409.5)**	(50 , 373.5 , 468.8)	(50.5 , 245.9 , 243.2)	(50 , 305.9 , 315.9)
100_0.8_b	(5936 , 38 , 280)	(53.9 , 5.3 , 16.1)	(54.3 , 4.2 , 5.9)	(53.4 , 338.9 , 320.2)	(51.5 , 328.4 , 322.4)	**(51.3 , 343.2 , 344.6)**	(51.7 , 241.1 , 203.9)	(51.3 , 265.3 , 244.9)
100_0.8_c	(5668 , 34 , 330)	(52.8 , 0 , 25.2)	(53.3 , 17.6 , 12.1)	(53.6 , 390.6 , 255.6)	**(51.4 , 322.9 , 221.8)**	(51.8 , 265.9 , 173.2)	(51.6 , 260 , 133.3)	(52 , 221.2 , 123.4)
100_0.8_d	(5506 , 35 , 260)	(55.9 , 0 , 0)	(56.2 , 7.4 , 15.4)	(52.4 , 362.9 , 371.8)	(51.1 , 334.3 , 307)	(50.9 , 381.1 , 375.3)	**(50.7 , 238.9 , 194.8)**	(50.7 , 250.9 , 222)
100_0.8_e	(5564 , 35 , 228)	(55.1 , 8.6 , 57.5)	(55.4 , 14.9 , 20.5)	(53.1 , 297.7 , 344.3)	(52.3 , 308 , 383.3)	**(51.6 , 361.7 , 416.1)**	(52.2 , 241.1 , 224.6)	(52 , 297.7 , 292.5)
300_0.2_a	(14987 , 279 , 2123)	(13.2 , 8.2 , 7.7)	(13.7 , 2.2 , 6.2)	(10.6 , 74.1 , 108)	(10.2 , 71 , 92.9)	**(6 , 219.6 , 250.8)**	(8.7 , 82.9 , 76.3)	(8.7 , 84.8 , 79.1)
300_0.2_b	(16105 , 228 , 1634)	(21.5 , 11 , 19.7)	(22.9 , 9 , 25.1)	(17 , 152.7 , 213.7)	(17.4 , 140.4 , 205.2)	**(13.7 , 256.8 , 302.3)**	(16.3 , 97.1 , 91)	(14.9 , 181.2 , 194.8)
300_0.2_c	(15869 , 191 , 1291)	(30.2 , 16.8 , 25.9)	(31.5 , 9.4 , 19.5)	(24.7 , 268.3 , 316.1)	(26.5 , 175.5 , 211.2)	**(23.8 , 285 , 286.8)**	(26 , 104 , 103.2)	(25.7 , 115.7 , 105)
300_0.2_d	(15934 , 320 , 2637)	(7.1 , 10.3 , 12.4)	(7.8 , 2.5 , 4)	(7.9 , 7.9 , 9.8)	(7.5 , 15.3 , 16.4)	**(1.4 , 162.1 , 197)**	(6.3 , 19.6 , 15.6)	(5 , 48.8 , 43.3)
300_0.2_e	(16796 , 300 , 2319)	(9.4 , 8.7 , 30.1)	(10.6 , 4.3 , 12.5)	(8.7 , 35.8 , 58.2)	(9.4 , 46.7 , 75.6)	**(2.2 , 197.3 , 263.9)**	(5.4 , 87.5 , 92.5)	(4.8 , 98.2 , 108.6)
300_0.5_a	(17804 , 214 , 2591)	(29.8 , 13.6 , 10.1)	(30.7 , 7.6 , 6.7)	(24.3 , 277.3 , 385.5)	(24.4 , 167.1 , 266)	**(22.5 , 300.7 , 418.2)**	(25.7 , 170 , 169.3)	(24.9 , 183.3 , 194.9)
300_0.5_b	(16827 , 249 , 2903)	(17.7 , 6.4 , 18.3)	(18.9 , 8 , 7.6)	(13.3 , 120.2 , 227.9)	(12.6 , 144.9 , 293.8)	**(10.5 , 291.2 , 460.9)**	(13.8 , 159.8 , 170.6)	(12.7 , 182.8 , 244.4)
300_0.5_c	(15193 , 189 , 1953)	(31.2 , 10.1 , 14.6)	(32.7 , 2.1 , 7.9)	(28.4 , 215.9 , 334.6)	(26.7 , 179.2 , 281.5)	**(26.3 , 280.1 , 406)**	(27.8 , 153.3 , 199)	(27.2 , 187.3 , 222.6)
300_0.5_d	(16002 , 287 , 3669)	(11.2 , 0 , 0)	(12.1 , 5.8 , 10.8)	(9.7 , 58.2 , 107)	(7.3 , 84.9 , 170.6)	**(4.1 , 261.7 , 429.4)**	(7.3 , 101.9 , 125.9)	(7.1 , 109.3 , 142.5)
300_0.5_e	(17490 , 316 , 3621)	(7.2 , 8.9 , 31.7)	(8.1 , 4.7 , 25)	(6.7 , 36 , 78.4)	(6.9 , 26.9 , 77.8)	**(1.8 , 180.5 , 393.7)**	(4.1 , 97.3 , 150.1)	(4.6 , 75.2 , 117.8)
300_0.8_a	(15815 , 212 , 2752)	(23.8 , 4.6 , 36.3)	(24.7 , 6 , 34.8)	(20.9 , 112.5 , 375.5)	(19.7 , 128.2 , 381.4)	**(17.9 , 288.7 , 754.9)**	(20.5 , 151.7 , 314.8)	(19.6 , 166.9 , 363.8)
300_0.8_b	(16451 , 220 , 3097)	(22.5 , 5.5 , 8.2)	(23.6 , 6 , 20.3)	(19.5 , 144.4 , 411.8)	(18.3 , 129.1 , 396.4)	**(15.7 , 297.5 , 723.2)**	(18.7 , 137.8 , 280.7)	(18.8 , 196.3 , 354.2)
300_0.8_c	(17116 , 224 , 3292)	(21.1 , 10.3 , 21.8)	(22.3 , 5 , 16)	(16 , 182.1 , 489.3)	(16 , 145.3 , 396)	**(14.2 , 253.3 , 655.2)**	(17.5 , 165.9 , 288.3)	(15.7 , 153.3 , 288.5)
300_0.8_d	(17908 , 333 , 5621)	(4.6 , 5.3 , 25.9)	(4.6 , 8.3 , 41.5)	(4.4 , 22.4 , 80.3)	(4.1 , 28.6 , 82.1)	**(1.3 , 119.7 , 363.7)**	(3.4 , 42 , 107.8)	(3 , 39.6 , 78.7)
300_0.8_e	(15985 , 288 , 4466)	(9.3 , 0 , 0)	(9.4 , 11.4 , 38.6)	(7.5 , 39.9 , 121)	(7.6 , 39.1 , 113.3)	**(4.4 , 158.1 , 464.6)**	(6.9 , 86.1 , 241.6)	(6.6 , 96.5 , 216)
500_0.2_a	(28309 , 353 , 2991)	(31.4 , 1.1 , 9.3)	(32.2 , 2.5 , 9.8)	(24.4 , 198.1 , 270.2)	(26.1 , 157.7 , 218.3)	**(24.2 , 257.5 , 321.2)**	(27.3 , 151.7 , 143.3)	(27 , 160.1 , 146)
500_0.2_b	(27884 , 560 , 4448)	(2.5 , 19.5 , 55.8)	(3 , 7.6 , 33.7)	(3.3 , 8.8 , 39.3)	(2.9 , 10.3 , 40.1)	(3.4 , 9.6 , 42.6)	(3.3 , 4.4 , 19)	**(2.4 , 4 , 13.9)**
500_0.2_c	(27691 , 468 , 4138)	(15.4 , 4.5 , 0)	(16.4 , 7 , 23.2)	(16.2 , 20.1 , 41.8)	(16 , 26.2 , 50.9)	**(9.2 , 242.5 , 361)**	(12.6 , 130.4 , 169.7)	(12.6 , 90.5 , 107.5)
500_0.2_d	(27175 , 525 , 4717)	(10.3 , 0 , 18.9)	(11.1 , 5 , 17.4)	(11.2 , 7.4 , 29.5)	(10.9 , 8 , 31.3)	**(5 , 172.6 , 306.2)**	(9.8 , 25.6 , 46.8)	(10 , 12.2 , 20.4)
500_0.2_e	(28175 , 479 , 4343)	(15.1 , 0 , 0)	(15.6 , 4.1 , 18.1)	(16.3 , 13.3 , 24.5)	(15.7 , 16.8 , 35)	**(8.2 , 232.2 , 335.1)**	(13.4 , 100.1 , 118.5)	(12.7 , 79.1 , 96.1)
500_0.5_a	(26546 , 442 , 6274)	(15.3 , 9.7 , 18.7)	(16.1 , 4.4 , 9.3)	(15.6 , 27.1 , 63)	(16.2 , 21.5 , 49.2)	**(11.5 , 138.2 , 346)**	(14.4 , 47.7 , 99.5)	(14.9 , 39.3 , 78.8)
500_0.5_b	(26070 , 224 , 2240)	(51 , 4.9 , 21.1)	(51.6 , 2.8 , 10.4)	(50.2 , 97.1 , 172.2)	(50.3 , 98.4 , 166.2)	**(48.5 , 249 , 387.8)**	(50.1 , 126 , 163.2)	(49.7 , 177.1 , 247.2)
500_0.5_c	(27312 , 348 , 4746)	(30 , 0 , 14.9)	(31.3 , 5.9 , 16.2)	(28.6 , 77.2 , 155.7)	(28.5 , 71.6 , 149.9)	**(26.1 , 197.8 , 367.8)**	(28.5 , 92.1 , 143.6)	(28.6 , 105.1 , 148.4)
500_0.5_d	(27346 , 545 , 9112)	(7.8 , 4.5 , 11.5)	(8.5 , 2.5 , 10.7)	(8.8 , 6.7 , 23.3)	(8.3 , 9.2 , 27.9)	**(6.1 , 66.4 , 177)**	(7.9 , 22.1 , 45.3)	(7.5 , 35.2 , 81)
500_0.5_e	(27282 , 607 , 8477)	(3.3 , 2.7 , 33)	(3.6 , 2.6 , 31.5)	(3.6 , 5.7 , 40.1)	(3.6 , 6.8 , 41.2)	**(3.1 , 25 , 106.9)**	(3.6 , 2.7 , 18.9)	(3.2 , 2.9 , 12.6)
500_0.8_a	(28130 , 281 , 4951)	(37 , 11.3 , 24.2)	(38 , 4.9 , 19.4)	(34.7 , 86.9 , 218.7)	**(33.7 , 112.6 , 300.3)**	(33.7 , 128.5 , 335.2)	(34.8 , 111 , 231.4)	(34.8 , 98.7 , 199.2)
500_0.8_b	(27001 , 538 , 11146)	(6 , 3.4 , 18.7)	(5.5 , 14.3 , 47.7)	(5.2 , 20.3 , 75.1)	(5.3 , 23.4 , 65.4)	**(4.8 , 28.6 , 80.3)**	(5.2 , 12.6 , 45.4)	(4.8 , 15.5 , 36.7)
500_0.8_c	(26173 , 520 , 11730)	(9.1 , 0.7 , 6.9)	(9.6 , 4.8 , 31.2)	(9.3 , 8.8 , 39.7)	(9.3 , 6.8 , 41.3)	(9.4 , 17.2 , 63)	(9.2 , 11.4 , 35.1)	**(8.6 , 7 , 26.1)**
500_0.8_d	(26228 , 471 , 10101)	**(11.6 , 3.2 , 7.8)**	(12.4 , 4 , 26.8)	(13.1 , 6.5 , 30.6)	(13.2 , 7.6 , 23)	(11.6 , 32.1 , 116.2)	(12.8 , 11.5 , 42.2)	(12.4 , 13.5 , 36.4)
500_0.8_e	(28029 , 556 , 12023)	(5.7 , 4.2 , 18.2)	(4.7 , 9.4 , 49.8)	(4.2 , 17.8 , 83.4)	(4.3 , 13.7 , 63.3)	(4.5 , 14.5 , 68.3)	(4.4 , 15.6 , 58.4)	**(3.8 , 12.8 , 44.8)**
Avg.		(28 , 6.7 , 17.5)	(28.7 , 6.2 , 17.9)	(26.4 , 146.8 , 176.2)	(25.9 , 133.6 , 167.3)	**(23.9 , 220.3 , 296.9)**	(25.5 , 110.3 , 123.6)	(25.2 , 125.4 , 137.4)

Table A2: Detailed results on large instances

References

Brélaz, D. (1979). New methods to color vertices of a graph. *Communications of the ACM*, *22*, 251–256.

Brucker, P., & Kravchenko, S. (1999). Preemption can make parallel machine scheduling problems hard. *OsnabrTucker Schriften zur Mathematik, Reihe P (Preprint)*, *211*.

Chiarandini, M., & Stützle, T. (2007). Stochastic local search algorithms for graph set T-colouring and frequency assignment. *Constraints*, *12*, 371–403.

Dorne, R., & Hao, J.-K. (1998). Tabu search for graph coloring, T-colorings and set T-colorings. In S. Voss, S. Martello, I. Osman, & C. Roucairol (Eds.), *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization* (pp. 77 – 92). Norwell, MA: Kluwer.

Epstein, L., Halldórsson, M. M., Levin, A., & Shachnai, H. (2009). Weighted sum coloring in batch scheduling of conflicting jobs. *Algorithmica*, *55*, 643–665.

Fukunaga, T., Halldórsson, M. M., & Nagamochi, H. (2007). "Rent-or-buy" scheduling and cost coloring problems. In *Proc. of the 27th Int. Conf. on Foundations of Software Technology and Theoretical Computer Science* (pp. 84–95). Berlin, Heidelberg: Springer-Verlag.

Gendreau, M., & Potvin, J.-Y. (2010). *Handbook of Metaheuristics, 2nd Edition*. New York, NY: Springer.

Giaro, K., Kubale, M., & Obszarski, P. (2009). A graph coloring approach to scheduling of multiprocessor tasks on dedicated machines with availability constraints. *Discrete Applied Mathematics*, *157*, 3625–3630.

Gröflin, H., Pham, D., & Bürgy, R. (2011). The flexible blocking job shop with transfer and set-up times. *Journal of Combinatorial Optimization*, *22*, 121–144.

Halldórsson, M. M., & Kortsarz, G. (2004). Multicoloring: Problems and techniques. In J. Fiala, V. Koubek, & J. Kratochvíl (Eds.), *Mathematical*

Foundations of Computer Science 2004 (pp. 21–45). Berlin, Heidelberg, New York: Springer, volume 3153 of *Lecture Notes in Computer Science*.

Hendel, Y., Runge, N., & Sourd, F. (2009). The one-machine just-in-time scheduling problem with preemption. *Discrete Optimization*, *6*, 10–22.

Heydari, M., Sadjadi, S., & Mohammadi, E. (2010). Minimizing total flow time subject to preemption penalties in online scheduling. *The International Journal of Advanced Manufacturing Technology*, *47*, 227–236.

Kazemi, M., Mahdavi, I., Aalaei, A., Kia, R., & Nikoofarid, E. (2011). Just-in-time preemptive one-machine problem with costs of earliness-tardiness, interruption and work in process: A mathematical programming. In *Proc. of Industrial Engineering and Engineering Management* (pp. 800 – 804). Changchun, China.

Konak, A., Coit, D. W., & Smith, A. E. (2006). Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering & System Safety*, *91*, 992–1007.

Lim, A., Zhu, Y., Lou, Q., & Rodrigues, B. (2005). Heuristic methods for graph coloring problems. In *Proceedings of the 2005 ACM Symposium on Applied Computing* (pp. 933–939). New York, NY: ACM.

Liu, Z., & Cheng, T. C. E. (2002). Scheduling with job release dates, delivery times and preemption penalties. *Information Processing Letters*, *82*, 107–111.

Lü, Z., Glover, F., & Hao, J.-K. (2011). Neighborhood combination for unconstrained binary quadratic problems. In M. Caserta, & S. Voss (Eds.), *Metaheuristics: Intelligent Decision Making* (pp. 49–61). New York, NY: Springer.

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

Marler, R. T., & Arora, J. S. (2004). Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, *26*, 369–395.

Marx, D. (2004). Graph coloring problems and their applications in scheduling. *Periodica Polytechnica Ser. El. Eng.*, *48*, 11–16.

Meuwly, F.-X., Ries, B., & Zufferey, N. (2010). Solution methods for a scheduling problem with incompatibility and precedence constraints. *Algorithmic Operations Research*, *5*, 75 – 85.

Mohammadi, E., & Heydari, M. (2011). Single machine scheduling problem with minmax criteria and preemption penaties. In *Proc. of the IEEE Conf. on Computer Science and Automation Engineering* (pp. 440 – 444). Shanghai, China.

Pinedo, M. (2008). *Scheduling: Theory, Algorithms and Systems*. New York, NY: Springer.

Satratzemi, M. (2004). A heuristic algorithm for the set T-coloring problem. In *Proc. of Information and Communication Technologies: From Theory to Applications* (pp. 531 – 532). Damascus, Syria.

Schuurman, P., & Woeginger, G. J. (1999). Preemptive scheduling with job-dependent setup times. In *Proc. of the 10th annual ACM-SIAM Symposium On Discrete Algorithms* (pp. 759–767). Baltimore, MD.

Shachnai, H., Tami, T., & Woeginger, G. (2002). Minimizing makespan and preemption costs on a system of uniform machines. In R. Möhring, & R. Raman (Eds.), *Algorithms - ESA 2002* (pp. 859–871). Berlin, Heidelberg : Springer-Verlag, volume 2461 of *Lecture Notes in Computer Science*.

Slotnick, S. A. (2011). Order acceptance and scheduling: A taxonomy and review. *European Journal of Operational Research*, *212*, 1–11.

Thevenin, S., Zufferey, N., & Potvin, J. -Y. (2013). Tabu search for a preemptive scheduling problem with job incompatibilities. In *Proc. of IFAC Conference on Manufacturing Modelling, Management, and Control*. Saint Petersburg, Russia.

de Werra, D., Demange, M., Monnot, J., & Paschos, V. T. (2005). A hypo-coloring model for batch scheduling. *Discrete Applied Mathematics*, *146*, 3–26.

Wu, Q., Hao, J.-K., & Glover, F. (2012). Multi-neighborhood tabu search for the maximum weight clique problem. *Annals of Operations Research*, *196*, 611–634.

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