CIRRELT

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

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

Graph Multi-Coloring for a Job Scheduling Application

Simon Thevenin
Nicolas Zufferey
Jean-Yves Potvin

June 2016

CIRRELT-2016-29

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

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

www.cirrelt.ca

Graph Multi-Coloring for a Job Scheduling Application

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

[1] Geneva School of Economics and Management, University of Geneva, UNIMAIL, 40 Boulevard du Pont-d'Arve, 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. In this paper, we introduce a graph multi-coloring problem where each vertex must be assigned a given number of different colors, represented as integers, and no two adjacent vertices can share a common color. In the variant considered, the number of available colors is such that not all vertices can be colored. Furthermore, there is a bound on the number of vertices which can be assigned the same color. A gain is associated with each vertex and the first objective is to maximize the total gain over all colored vertices. Secondary objectives consider the sequence of colors assigned to each vertex. More precisely, the range and the number of interruptions must be minimized, where the range corresponds to the difference between the largest and smallest colors assigned to a vertex. This variant of the graph multi-coloring problem is of interest because it can model practical job scheduling applications. An integer linear programming formulation is first proposed to address small-size instances. A construction heuristic, as well as local search methods, are then reported to tackle larger instances. The local search methods are based on several neighborhood structures, each one focusing on a specific property of the problem. Different ways to combine these neighborhood structures are also investigated.

Keywords: Graph multi-coloring, job scheduling, local search, tabu search.

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, 2016

1 Introduction

The graph coloring problem (GCP) consists of assigning a single color (integer) to each vertex of an undirected graph, such that no two adjacent vertices share the same color, with the objective of minimizing the number of colors. The graph multi-coloring problem is an extension of the GCP, where each vertex must be assigned a preset number of different colors and no two adjacent vertices can share a common color. In this case, a vertex coloring corresponds to a sequence of colors (from the smallest to the largest). The objective is again to minimize the number of used colors.

Our goal is to solve a variant of the multi-coloring problem that is suitable for job scheduling problems with the two following main characteristics: (1) there is no feasible solution where all jobs can be performed, thus leading to a job acceptance (or job rejection) problem, and (2) each job requires the simultaneous use of two or more resources. Some enthusiasm has recently been observed in the research community for problems with characteristic (1), also called *order acceptance and scheduling problems*, because the demand often exceeds the production capacity. On the other hand, characteristic (2) refers to multiprocessor scheduling problems where two or more resources are required simultaneously to process a job. Although these problems originate from the scheduling of processors in computer systems, they are also found in other production environments. Examples include chemical plants (Rudek et al., 2013), equipment manufacturers for power supply factories (Yang et al., 2009) and production of key components of aircrafts (Mati & Xie, 2011). As far as wee know, these two characteristics have never been studied jointly, although they can model real-world problems, like the beauty care manufacturing application reported later.

We consider two types of resource: the *shared* resources (e.g., parallel machines) are involved in the processing of all jobs and limit the number of jobs that can be processed concurrently, and the *critical* resources are unique. Two jobs requiring the same critical resource, in addition to shared resources, are said to be *incompatible*. Critical resources might come in a few exemplars in practical applications, but a common approach is to simplify the problem by aggregating them and assigning them at once to a subset of jobs in a first step, so that they can be considered unique thereafter (see (Chen & Lee, 1999; Drozdowski, 1996; Edis et al., 2013)).

More formally, we assume a planning horizon divided into a number of discrete time units. We have a set of n jobs, along with an incompatibility graph $G = (V, E)$, where V is the set of jobs, and E is a set of edges between incompatible jobs. These jobs cannot be performed during a common time unit, and no more than l jobs can be scheduled concurrently because the shared resources are available only in l exemplars. Preemption is allowed, that is, each job can be stopped at integer time points and resumed later. All jobs must terminate before a preset global deadline D that stands for the end of the planning horizon. This deadline is such that only a subset of jobs can be scheduled. Each job i has an integer processing time of p_i time units, and a gain g_i is earned when it is entirely performed. There is no gain for a partially performed job, so it is better in this case not to perform it at all. The goal is to optimize the three following objectives in lexicographic order:

- f_1: maximize the total gain of the scheduled jobs;
- f_2: minimize the total number of job interruptions;
- f_3: minimize the total flow time (the flow time of a job i is the total time spent by i in the production facility).

It should be noted that a close problem was considered in (Thevenin et al., 2014), but the very sensitive job acceptance issue was ignored. The above lexicographic approach was chosen due to a natural hierarchy among the three objectives in many practical applications. More precisely: (1) minimizing the total gain of the scheduled jobs is the most important objective because it is related to the total income, (2) considering job interruptions in production scheduling is relevant when each job is made of multiple batches of the same or different products (see below). However, too many interruptions should be avoided as it leads to delays and unproductive work, (3) through minimization of the total flow time, the amount of work in progress (which generates inventory costs) is reduced. Clearly, these three objectives interact together. For instance, it was shown in the job scheduling literature that allowing preemption is a way to reduce the makespan (e.g., (Pinedo, 2012)). This was also observed in (Thevenin et al., 2014) for a problem related to (P). Although the makespan is not one of our objectives, it implies in our context that more jobs can be scheduled within a given deadline, thus increasing f_1.

An example of this type of problem is encountered at an international consumer goods company whose headquarters are in Geneva. It concerns the manufacturing of beauty care lotions, which is characterized by large quantities sold and a large variety of products. The production system uses continuous flow production and is very flexible (as it can be quickly reconfigured). To prepare a given lotion type, different resources are required simultaneously: tanks of different raw materials, a mixer, employees to perform and supervise the production process, and chemists to control the quality and safety during the production process. Due to their technical characteristics (or specific skills), the critical resources are tanks of raw products and chemists. That is, two products requiring a common critical resource cannot be processed concurrently. A time unit (typically several hours) corresponds to the time required to fill a tank of finished product (it is similar for all product types). The production system can only be reconfigured after the filling of each tank, and the changeover time is negligible. In other words, it can be assumed that preemption is only allowed at integer time points. Each job refers to the quantity (in tank units) of a product type to be delivered at a given distribution center. Each job is associated with a processing time (related to quantity) and a gain equal to the gross margin. The problem is to schedule the production for the next week, considering incompatibilities due to critical resources, and the limited amount of shared resources. Job acceptance is also an issue, as it might not be possible to schedule all of them. The objective is then to maximize the total gain associated with the scheduled jobs.

The variant of the graph multi-coloring problem that is studied in this paper to address this type of scheduling problems can be stated as follows. We have an undirected graph $G = (V, E)$, where V is the vertex set and E is the edge set. Each vertex i stands for a job and there is an edge between two vertices if the corresponding jobs are incompatible. Each color represents a particular time unit of the scheduling horizon. Color c is then assigned to vertex i if one time unit

of the corresponding job is processed during that time unit c. Each vertex i must be assigned p_i colors as defined by the processing time of the job. The total number of available colors is D and this number is such that it is not possible to fully color all vertices. In addition, each color c cannot be assigned to more than l vertices (due to the limited number of shared resources). Each vertex is associated with a gain g_i and the resulting problem (P) consists of coloring a subset of vertices in the graph, such that no two adjacent vertices share the same color, while optimizing the following objectives in lexicographic order:

- f_1 : maximize the total gain over all fully colored vertices;
- f_2 : minimize the sum of the number of interruptions in the sequence of colors assigned to each vertex, over all fully colored vertices;
- f_3 : minimize the sum of the range of colors assigned to each vertex, over all fully colored vertices, where the range is defined as the difference between the largest color and the smallest color.

These objectives correspond to those reported before for job scheduling. It can be shown that problem (P) is NP-hard by using the k-coloring problem, where one asks if a graph G can be colored with only k colors. For any instance of the k-coloring problem on a graph G, a corresponding instance of (P) can be created by setting $p_i = g_i = 1$ for every vertex i and by setting $D = k$. If the optimal solution of (P) is such that the total gain is equal to the number of vertices in graph G, then the graph is k-colorable, otherwise it is not. Therefore, if problem (P) could be solved in polynomial time, the same would be true for the k-coloring problem. But the latter is known to be NP-complete.

Due to the difficulty of (P), heuristics and metaheuristics are the most appropriate methods to tackle large-scale instances. In this regard, the contribution of this work is twofold. First, it bridges the gap between the job scheduling and the graph coloring research communities. Indeed, the graph multi-coloring variant considered in this work has some obvious job scheduling applications. Second, new and efficient methods are proposed for solving problem (P), such as a tabu search metaheuristic using different neighborhood structures, where each one addresses a particular feature of the problem. Various ways to combine these neighborhoods are also investigated. A preliminary version of the work reported here has appeared in (Thevenin et al., 2013). This previous contribution is extended in the following ways. First, an integer 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. General references on the topics studied in this paper can be found in (Malaguti & Toth, 2010) for the graph coloring problem and its extensions, in (Pinedo, 2012) for scheduling problems, in (Slotnick, 2011) for order acceptance and scheduling problems, in (Gendreau & Potvin, 2010; Zufferey, 2012) for metaheuristics.

The remainder of this paper is organized as follows. Section 2 provides a literature review. A mathematical programming formulation of problem (P) is then introduced 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

To the best of our knowledge, the work reported in (Thevenin et al., 2013) is the only attempt at solving problem (P). Section 2.1 will focus on applications of graph coloring for scheduling problems, whereas Section 2.2 will review scheduling problems sharing features with the practical application of our graph multi-coloring problem.

2.1 Graph coloring for scheduling problems

Modeling scheduling problems as GCPs is particularly relevant in the presence of incompatible jobs, and multiple extensions of the GCP have been proposed to cope with these scheduling environments (e.g., Epstein et al., 2009; Fukunaga et al., 2007; Giaro et al., 2009; Halldórsson & Kortsarz, 2004; Meuwly et al., 2010; de Werra et al., 2005). The graph multi-coloring problem has also been used to model scheduling applications when there are incompatible jobs with different processing times (Marx, 2004). For example, an extension of the graph multi-coloring problem is used in (Bloechliger & Zufferey, 2013) for a scheduling problem with parallel machines, preemption, incompatibility penalties and assignment costs. Exact methods and metaheuristics were proposed to solve the multi-coloring problem.

In the following, we introduce two studies about a particular extension of the multi-coloring problem and we summarize their main findings, given that these results have been exploited in our work.

In the set-T-coloring problem, 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 apply 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 construction heuristics and local search algorithms either on the original graph or on the split graph. The following conclusions can be drawn from these two papers.

- An adaptation of DSATUR (Brélaz, 1979) (for "Degree of SATURation") is the best construction heuristic. 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 (i.e., one by one) in order of decreasing saturation degrees (ties are broken by considering the number of adjacent vertices that are still not colored).

- Tabu search is very efficient 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 (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.

2.2 Scheduling problems

In the following, the literature is reviewed with regard to key features of scheduling problems modeled by (P). Section 2.2.1 is related to order acceptance and scheduling problems, whereas Section 2.2.2 discusses preemption and flow time. Note that the standard three-field notation ($\alpha \mid \beta \mid \gamma$) Pinedo (2012) will be used in this section, where α, β and γ correspond to the production environment, the constraints and the objectives, respectively.

2.2.1 Order acceptance and scheduling problems

In the order acceptance and scheduling problem (OAS), the decision about accepted jobs is taken at production scheduling time. This is particularly important in make-to-order production systems (Oğuz et al., 2010), because the equipment utilization needs to be optimized (ten Kate H. A., 1994). Despite the growing amount of research on OAS, it has never been considered jointly with incompatible jobs. Recent papers considering related problems involving order acceptance (or order rejection) are commented below.

The OAS problem with parallel machines and preemption has been the subject of only a few studies (e.g., Dósa & He, 2006; Hoogeveen et al., 2000; Seiden, 2001). Hoogeveen et al. (2000) consider an OAS with parallel machines and preemption, where the objective is to minimize the sum of rejection penalties and makespan. Three scheduling environments are investigated: related machines, unrelated machines, and open shop. The authors show that the problem is NP-hard in all considered settings and give approximation methods. Seiden (2001) proposes an optimal algorithm for an online OAS with parallel machines and preemption. Dósa & He (2006) also report an optimal algorithm for an online OAS with preemption based on two parallel machines with different speeds.

A larger number of studies have considered the OAS problem without preemption (e.g., Bartal et al., 1996; Cao & Yang, 2009; Fanjul-Peyro & Ruiz, 2012; Gerstl & Mosheiov, 2012; He & Min, 2000; Li & Yuan, 2010; Lu et al., 2009, 2008). Lu et al. (2008) study an unbounded parallel batch machines scheduling problem with release dates. The authors show that the problem is NP-hard, and give a dynamic programming formulation and approximation methods. In (Lu et al., 2009), the proposed methods are then adapted to the case of bounded batch machines. Cao & Yang

(2009) also propose an approximation algorithm for parallel batch machines with release dates. Gerstl & Mosheiov (2012) introduce optimal methods to solve the parallel machine scheduling problem with job rejection and deteriorating jobs. Li & Yuan (2010) also study different variants of the latter problem. They propose approximations and exact methods. Bartal et al. (1996) propose an online algorithm, upper and lower bounds for the online problem and approximation algorithms for the offline problem. The online problem is also studied by He & Min (2000) for uniform parallel machines with the objective of minimizing the makespan. Fanjul-Peyro & Ruiz (2012) report some heuristics and a mathematical programming formulation for the unrelated parallel machine scheduling problem based on the same objective.

2.2.2 Preemption penalties and flow time

To motivate our choice of objectives, this section reviews papers were preemption and work in progress are considered. It should be noted that these scheduling problems are quite different from the ones mentioned in the introduction. Therefore, we cannot really take advantage of the proposed methodologies.

Typically, when preemption is allowed, a job can be stopped and resumed 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 that 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 of 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. In various production environments, preemption must occur at integer time points due to managerial or technical constraints (c.f., the application reported in the introduction).

Although scheduling problems with preemption have been well studied, only a few papers address preemption penalties, either explicitly or by considering setup costs. 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.

Minimizing the number of job interruptions due to preemption and minimizing the overall flow time are clearly two different objectives. Minimizing the overall flow time is equivalent to minimizing the sum of processing time and waiting time (due to preemption) over all jobs. The minimization of the overall flow time has also an impact on the minimization of the total work in progress (i.e., partially finished goods waiting in the system). Such an objective is considered in various papers (e.g., Belfares et al., 2007; Hendel et al., 2009; Hsu et al., 2008; Kazemi et al., 2011). Clearly, work in progress is not desirable as it leads to inventories, which generate costs.

3 Mathematical model

In this section, we provide an integer programming formulation for our graph multi-coloring problem. To this end, the following decision variables are used:

- $z_i = 1$ if vertex i is fully colored, 0 otherwise.
- $Max_i =$ largest color assigned to vertex i (equal to 0 if vertex i is not colored).
- $Min_i =$ smallest color assigned to vertex i (equal to 0 if vertex i is not colored).
- $s_{ic} = 1$ if the sequence of colors assigned to vertex i starts or is resumed (after an interruption) with color c, 0 otherwise.
- $x_{ic} = 1$ if vertex i has color c, 0 otherwise (note that $x_{i0} = 0$ for all $i \in V$).

The mathematical model is then:

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

s.t.

$$\sum_{c=1}^{D} x_{ic} = p_i \cdot z_i \qquad i \ \in V \tag{2}$$

$$\sum_{i \in V} x_{ic} \leq l \qquad 1 \ \leq c \leq D \tag{3}$$

$$c \cdot x_{ic} \leq Max_i \qquad 1 \ \leq c \leq D, \ \ i \in V \tag{4}$$

$$c \cdot x_{ic} + D \cdot (1 - x_{ic}) \geq Min_i \qquad 1 \ \leq c \leq D, \ \ i \in V \tag{5}$$

$$D \cdot z_i \geq Min_i \qquad i \in V \tag{6}$$

$$x_{ic} + x_{jc} \leq 1 \qquad 1 \ \leq c \leq D, \quad (i,j) \in E \tag{7}$$

$$s_{ic} \geq x_{ic} - x_{i(c-1)} \qquad 1 \leq c \leq D, \quad i \in V \tag{8}$$

$$s_{ic},\ z_i,\ x_{ic} \in \{0,1\} \qquad 1 \leq c \leq D, \quad i \in V \tag{9}$$

$$Min_i, Max_i \geq 0 \ \text{ and integer } \ i \in V \tag{10}$$

The three terms in the objective function correspond to f_1 (gain), f_2 (number of interruptions) and f_3 (range), respectively, whereas the corresponding weights α_1, α_2 and α_3 are chosen to obtain the desired lexicographic ordering. Basically, α_3 is set to 1, whereas α_1 and α_2 can be arbitrary large. To carefully choose these values, some strict upper bounds on objectives f_2 and f_3 (denoted $\bar{f}_2$ and $\bar{f}_3$) can be used. By setting α_2 to $\alpha_3 \cdot \bar{f}_3$, and α_1 to $\alpha_2 \cdot \bar{f}_2 + \alpha_3 \cdot \bar{f}_3$, the lexicographic order is satisfied. For (P), we have chosen $\bar{f}_2 = \sum_{i \in V} p_i$ and $\bar{f}_3 = n \cdot D + 1$.

With regard to the second term in the objective, the summation over the s_{ic} variables also includes the first part of the sequence of colors assigned to vertex i, but this is equivalent to minimizing the number of interruptions when the number of fully colored vertices is fixed (it is the case here, due to the lexicographic ordering among the three objectives). For the third term, z_i is 1 if vertex i is colored and $Max_i - Min_i + 1$ then corresponds to the range of colors assigned to that vertex.

Constraint (2) imposes the assignment of p_i colors to each fully colored vertex i. Constraint (3) forbids having more than l vertices with the same color. Constraints (4), (5) and (6) define Min_i and Max_i for each vertex i. Constraint (6) sets Min_i to 0 when vertex i is not colored, otherwise it indicates that Min_i should not exceed D. Note that when Min_i is set to 0 when vertex i is not colored, constraint (2) forces the corresponding x_{ic} variables to 0, while constraint (4) 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 (7) states that two adjacent vertices in the incompatibility graph G cannot share a common color. Constraint (8) indicates the start of a sequence of colors assigned to vertex i or a restart by forcing s_{ic} to be 1 when $x_{ic} = 1$ and $x_{i(c-1)} = 0$. Otherwise the subtraction is negative or null, and s_{ic} gets value 0 through the minimization of the second term in the objective function.

4 Basic heuristics for problem (P)

In this section, basic heuristic methods to tackle (P) are presented. A construction method is first introduced, followed by a descent local search. Both heuristics are methods with restarts, and return the best found solution when the imposed time limit is reached.

4.1 Construction heuristic

The greedy construction heuristic starts from an incompatibility graph that is not colored and builds a complete solution by coloring a new vertex at each iteration. 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, and assigned to less than l 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, it is fully colored. Clearly, when p_i consecutive colors are available in A_i, they should be automatically selected as they lead to the minimum range and do not induce any interruption in the sequence. Based on these 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 consecutive colors in B_i (randomly chosen). 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 number of additional interruptions in the sequence, which is either 0 or 1, and remove that color from A_i. In case of ties, choose the color that minimizes objective f_3 (range).

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

4.2 Descent method

Starting from an initial solution, local search methods are algorithms that 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 that 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 local optimum (with regard to the chosen neighborhood), a restart procedure can be applied to escape from it.

In the multi-start descent method developed for (P), every initial solution is provided by the greedy construction heuristic. As preliminary experiments have shown that changing only one color at a time leads to poor results, we consider moves that fully color (if the vertex is not colored) or fully recolor (if the vertex is already colored) a vertex. A neighbor solution s' of s is thus 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. 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 vertices that are colored or not).

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 vertices that are not colored. Remember that the saturation degree of a vertex is the number of different colors in its 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 vertices of vertex i that are not colored, 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 to vertex i that leads to the smallest increase

in the saturation degree of its adjacent vertices that are not colored. 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 non-saturated (i.e., assigned to less than l vertices) color that minimizes the gain loss. 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 for problem (P)

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 undoing some recently performed moves. 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, a neighborhood restriction technique, and a diversification mechanism. Finally, some ways to put more emphasis on objectives f_2 (number of interruptions) and f_3 (range) are proposed.

Initial solution. The initial solution is provided by the greedy construction 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 of 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}. However, 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., (Lü et al., 2011; Schindl & Zufferey, 2015; Thevenin et al., 2015; Wu et al., 2012)).

Tabu status. It is forbidden recoloring a recently recolored or uncolored vertex for *tab* (parameter) iterations. The larger is *tab*, the more diversified is the search.

Neighborhood reduction technique. A percentage (parameter tested in the interval $[10\%, 90\%]$ and tuned to 25%) of the whole neighborhood is considered at each iteration. The sample of neighbor solutions is randomly chosen. The larger this percentage is, the more accurate (but time consuming) is an iteration, and the smaller it is, the more diversified is the search.

Diversification strategy. Every I iterations (parameter tested in the interval $[0, 200]$ and tuned to 150) without improvement to the best solution found, $b\%$ (parameter tested in the interval $[10\%, 90\%]$ and tuned to 20%) of the colored vertices are randomly chosen and uncolored. One can observe that: (1) too small values of I do not allow the tabu search to focus enough the search in promising areas of the solution space; (2) the larger b is, the more the method will behave like restarting from scratch. Note that more sophisticated strategies (e.g., select 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 f_2 and f_3. To address this issue, an exact recoloring procedure (i.e., capable of optimally coloring 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.

Assume that we have to recolor vertex i and that $|A_i| \geq p_i$. The exact algorithm consists of finding the subset of colors in A_i that minimizes the total cost (f_2 followed by f_3) when recoloring vertex i. Remember that a vertex coloring is a sequence of colors of fixed size (from smallest to largest color). Our methodology is based on the following property.

> Let S and S' be two increasing 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 interruptions (resp. range) due to the addition of color c at the end of a sequence ending with color c' is denoted $Int(c', c)$ (resp. $Rg(c', c)$) and is defined as follows for all $c > c'$:

$$Int(c', c) = \begin{cases} 1 & \text{if } c > c' + 1 \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

$$Rg(c', c) = c - c' \tag{12}$$

A pseudo-code for identifying the optimal recoloring of vertex i is found in Algorithms 1 and 2, where $L_{int}(q, c)$ and $L_{rg}(q, c)$ are the optimal values for the number of interruptions and range, 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 f_2, as there might be more than one. These sequences are then evaluated with regard to 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 of Algorithm 2 is used to indicate that color c is added at the end of the sequence $BestS(q-1, c^*)$.

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

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

For $q = 1$ to p_i, do

For each c in A_i, do

call R(q,c);

Algorithm 2 R(q,c)

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

1. $Best_{int} \leftarrow \infty$, $Best_{rg} \leftarrow \infty$, $B \leftarrow \emptyset$
2. for each $c' \in A_i$ 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 each $c' \in B$, do
   (a) $Rg \leftarrow L_{rg}(q - 1, c') + (c - c')$
   (b) if $Rg = Best_{rg}$ then $c^* \leftarrow c'$ (with some probability)
   (c) if $Rg < Best_{rg}$ then $c^* \leftarrow c'$, $Best_{rg} \leftarrow Rg$
4. return $BestS(q, c) = BestS(q - 1, c^*) + \{c\}$, $L_{int}(q, c) = Best_{int}$, $L_{rg}(q, c) = Best_{rg}$;

R^{SD} better deals with the total gain f_1 by forbidding a smaller number of colors on adjacent, non-colored, 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 (the larger this parameter is, the slower is the method; the smaller it is, the more accurate is the method) vertices leading to the best solutions when recolored with R^{SD} become potential candidates for the exact procedure. Next, 25% of the candidate vertices are (randomly) selected, colored with R^{Exact} and added 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, whereas its neighborhood management strategy is found in Algorithm 4.

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 involved vertex v
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.

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

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

6 Experiments

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

In the following, ILP denotes the application of the exact CPLEX solver to the integer linear programming model of Section 3. *Greedy* and *Descent* are the methods 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, whereas TS^{Drop} is obtained when R^{Drop} is added to the neighborhood structure. Next, 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 (where q and γ were tested in the range $[1, 50]$ and $[0.1, 1]$, respectively). Note also that the tabu tenure tab was tested in the interval $[1, 100]$ and 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 n of vertices 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 number of colors p_i of vertex i is an integer randomly chosen in interval $[1, 10]$. The gain g_i is related to p_i (in scheduling applications, larger jobs typically bring more profits). That is, a random number β is chosen in the interval $[1, 20]$ and $g_i = \beta \cdot p_i$. For each instance, the value of parameter l was set to 30 and the number of available colors D was chosen small enough to prevent all vertices to be colored.

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, and 5 runs per instance were performed with each algorithm, except ILP 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 for each individual instance are provided in the Appendix in Tables 4 and 5 for small and large instances, respectively. In these tables, column *Instance* refers to the name *n_d_letter* of the test instance. Column *Best* contains the value of each objectives associated with the best found solution (i.e., gain, number of interruptions and range). 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 best solution, still in lexicographic order (note that negative gaps can be found due to the lexicographical ordering of the objectives). In column ILP, 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 4, in particular, shows that ILP is able to solve all instances with 10 vertices and all instances with 25 vertices when the graph density is 0.2. In the other cases, a 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 ILP are largely outperformed by the other methods for the test instances with 50 vertices. In particular, the average gap of the first objective for these instances is 8.2% for ILP versus 0.8% for TS^{Div}. Consequently, ILP was not run for the large instances. For all instances of size 10, 25 and 50, ILP obtains an average gap for the first objective of 2.8%, versus 2.1% for *Greedy*, but it is not competitive on the two other objectives. *Greedy* and *Descent* produce similar results with average gaps of 2.1%, -15.4%, -12.7% and 1.9%, -13.8%, -10.1%, respectively, for the three objectives (in lexicographic order). TS outperforms *Greedy* by 1.4% 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 4 in the Appendix). Adding R^{Drop} and the diversification mechanism lead to the best results with regard to the first objective. However, the results for the secondary objectives are poor with average gaps of 82.9% and 77.2% for TS^{Drop}, and 54.2% and 79.8% for TS^{Div}. Introducing R^{Exact} is clearly valuable for the methods, given that TS^{E1} and TS^{E2} produce gaps of 0.3%, 7.2%, 6.4% and 0.2%, 33.6%, 25.8%, respectively, for the three objectives. Overall, TS^{E1} is the best method on average. It also produces the best solutions for 28 instances out of 45.

The aggregated results for the large instances of size 100, 300 and 500 are reported in Table 2. For 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 4.5%, 6.9%, and 7.7%, respectively. TS^{Drop} outperforms TS at 3.6%. However, the diversification mechanism provides the best results with a gap of 1.2%. Finally, adding R^{Exact} does not provide an implementation that matches TS^{Div} with gaps of 3.1% for TS^{E1} and 2.8% for TS^{E2}, but huge reductions of the gaps 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 for small instances with up to 50 vertices, whereas TS^{Div} is the best for larger instances. In the latter case, however, TS^{E1} and TS^{E2} considerably reduce the values of f_2 and f_3, at the expense of a slight deterioration in f_1. These results demonstrate the effectiveness of the recoloring procedure R^{SD} for maximizing the first objective: 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 for small instances and by considerably improving the values of the secondary objectives (at the expense of a slight deterioration in the total gain for large instances).

n	d	ILP	$Greedy$	$Descent$	TS	TS^{Drop}	TS^{Div}	TS^{E1}	TS^{E2}
10	0.2	**(0 ; 0 ; 0)**	(3.2 ; -2.2 ; -2.9)	(0 ; 0 ; 0)	(0 ; 4.6 ; 4.1)	(0 ; 6.8 ; 3.4)	(0 ; 2.2 ; 3.9)	**(0 ; 0 ; 0)**	(0 ; 0.4 ; 0.4)
	0.5	**(0 ; 0 ; 0)**	(0 ; 4.7 ; 9.9)	**(0 ; 0 ; 0)**	(0 ; 29.9 ; 31.6)	(0 ; 30.8 ; 29.3)	(0 ; 11.9 ; 20.5)	**(0 ; 0 ; 0)**	(0 ; 8.1 ; 4.7)
	0.8	**(0 ; 0 ; 0)**	(0.3 ; -2.9 ; -1.1)	(0.3 ; -2.9 ; -1.1)	(0 ; 56 ; 50.1)	(0 ; 57.1 ; 49.6)	(0 ; 25.3 ; 54.4)	**(0 ; 0 ; 0)**	(0 ; 12.6 ; 13.2)
	Avg.	**(0 ; 0 ; 0)**	(1.2 ; -0.1 ; 2)	(0.1 ; -1 ; -0.4)	(0 ; 30.2 ; 28.6)	(0 ; 31.5 ; 27.4)	(0 ; 13.1 ; 26.3)	**(0 ; 0 ; 0)**	(0 ; 7 ; 6.1)
25	0.2	**(0 ; 0 ; 0)**	(4.2 ; -4.9 ; -4.9)	(4.2 ; -4.9 ; -4.9)	(0 ; 68.8 ; 44.5)	(0 ; 69.4 ; 44.2)	(0 ; 38.5 ; 48.8)	(0 ; 1.1 ; 1.9)	(0 ; 18.9 ; 15.4)
	0.5	(0.5 ; 9.2 ; 3.6)	(1 ; -2 ; -6.6)	(0.9 ; 0.7 ; -1.6)	(0 ; 224.8 ; 163.5)	(0 ; 198.3 ; 151.8)	(0 ; 114.6 ; 148.1)	**(0 ; 15.4 ; 14.9)**	(0 ; 67.4 ; 54.4)
	0.8	(0.2 ; 8.3 ; 14.9)	(0.7 ; -2.6 ; -5.9)	(0.1 ; 15 ; 23)	(0 ; 218.1 ; 269.2)	(0 ; 188.5 ; 245.9)	(0 ; 140.6 ; 240.9)	**(0 ; 27.8 ; 32.2)**	(0 ; 109.6 ; 104)
	Avg.	(0.2 ; 5.9 ; 6.2)	(2 ; -3.1 ; -5.8)	(1.8 ; 3.6 ; 5.5)	(0 ; 170.6 ; 159.1)	(0 ; 152.1 ; 147.3)	(0 ; 97.9 ; 146)	**(0 ; 14.8 ; 16.3)**	(0 ; 65.3 ; 57.9)
50	0.2	**(0.2 ; 5.2 ; 5.2)**	(3.5 ; -32.5 ; -21.2)	(3.7 ; -35.5 ; -22.9)	(1.4 ; 70.4 ; 38.6)	(0.4 ; 32.3 ; 21.4)	(0.4 ; 33.5 ; 36.6)	(0.7 ; -9 ; -7.3)	(0.6 ; 11.8 ; 1.5)
	0.5	(11.2 ; 31.9 ; 31.7)	(3.6 ; -55.6 ; -35)	(4.3 ; -56.8 ; -40.1)	(3.4 ; 48.9 ; 48.4)	(1.3 ; 42.2 ; 50.4)	(1 ; 17.4 ; 54.1)	(1 ; -17.4 ; -7.4)	**(0.8 ; -5 ; -0.2)**
	0.8	(13.1 ; 5.6 ; 1)	(2.8 ; -40.7 ; -46.7)	(3.1 ; -39.9 ; -43.1)	(2 ; 141.8 ; 108.7)	**(0.6 ; 120.8 ; 98.7)**	(0.6 ; 103.8 ; 111.1)	(1 ; 47.2 ; 23.2)	(0.8 ; 78.9 ; 38.8)
	Avg.	(8.2 ; 14.2 ; 12.6)	(3.3 ; -43 ; -34.3)	(3.7 ; -44 ; -35.4)	(2.2 ; 87 ; 65.2)	(0.8 ; 65.1 ; 56.8)	**(0.7 ; 51.6 ; 67.3)**	(0.9 ; 6.9 ; 2.8)	(0.7 ; 28.6 ; 13.3)
Avg.		(2.8 ; 6.7 ; 6.3)	(2.1 ; -15.4 ; -12.7)	(1.9 ; -13.8 ; -10.1)	(0.7 ; 95.9 ; 84.3)	(0.3 ; 82.9 ; 77.2)	(0.2 ; 54.2 ; 79.8)	**(0.3 ; 7.2 ; 6.4)**	(0.2 ; 33.6 ; 25.8)

Table 1: Aggregated results for small instances

n	d	$Greedy$	$Descent$	TS	TS^{Drop}	TS^{Div}	TS^{E1}	TS^{E2}
100	0.2	(7.5 ; -52.5 ; -35)	(8.2 ; -54.4 ; -37.9)	(2.4 ; 28.6 ; 18.4)	(3.2 ; 12.4 ; 13.1)	**(1.1 ; 44.4 ; 31.9)**	(1.3 ; -15.7 ; -10.6)	(1.4 ; -3.7 ; -5.8)
	0.5	(9.3 ; -68.4 ; -57.1)	(10.2 ; -69.2 ; -59.9)	(6.3 ; 18.1 ; 14.8)	(2.8 ; 11.2 ; 10.5)	**(2.2 ; 23.6 ; 18.8)**	(2.6 ; -28.4 ; -29.2)	(2.9 ; -17.9 ; -19.5)
	0.8	(7.5 ; -74.8 ; -68.6)	(8.4 ; -73 ; -70.1)	(5.5 ; 8.5 ; 9.2)	(1.7 ; 4 ; 6.2)	**(1.7 ; 8.1 ; 10.1)**	(2.1 ; -15.3 ; -25.1)	(1.9 ; -10.5 ; -16.9)
	Avg.	(8.1 ; -65.2 ; -53.6)	(8.9 ; -65.6 ; -56)	(4.8 ; 18.4 ; 14.1)	(2.5 ; 9.2 ; 9.9)	**(1.6 ; 25.4 ; 20.3)**	(2 ; -19.8 ; -21.7)	(2.1 ; -10.7 ; -14)
300	0.2	(8.2 ; -65.1 ; -66.8)	(9.4 ; -66.8 ; -68.4)	(5.3 ; -38.6 ; -34)	(5.8 ; -43 ; -39.7)	**(0.6 ; 0.1 ; -0.1)**	(4 ; -45.4 ; -51.6)	(3.2 ; -37 ; -43.3)
	0.5	(8.1 ; -70.8 ; -78.9)	(9.4 ; -71.4 ; -79.5)	(4.6 ; -36.5 ; -39.7)	(3.5 ; -41.7 ; -41.7)	**(0.8 ; -2.9 ; -4.3)**	(3.9 ; -36.5 ; -51.6)	(3.3 ; -34 ; -47.8)
	0.8	(7.1 ; -67.8 ; -83.1)	(7.8 ; -66.8 ; -81.1)	(4.1 ; -42.1 ; -48.2)	(3.4 ; -43.7 ; -50.9)	**(0.7 ; -5.4 ; -5.3)**	(3.8 ; -36.3 ; -52.3)	(3.1 ; -32.6 ; -51.4)
	Avg.	(7.8 ; -67.9 ; -76.3)	(8.9 ; -68.3 ; -76.3)	(4.6 ; -39.1 ; -40.6)	(4.2 ; -42.8 ; -44.1)	**(0.7 ; -2.7 ; -3.2)**	(3.9 ; -39.4 ; -51.8)	(3.2 ; -34.6 ; -47.5)
500	0.2	(6.6 ; -52.4 ; -55.6)	(7.4 ; -53.8 ; -57.3)	(5.6 ; -39.2 ; -41)	(5.7 ; -41 ; -42.6)	**(1 ; 1.5 ; 2.6)**	(4.6 ; -30.6 ; -40.5)	(4.3 ; -34.5 ; -46)
	0.5	(4.6 ; -54.6 ; -70.1)	(5.5 ; -55 ; -71)	(4.2 ; -43.1 ; -56.2)	(4.3 ; -43.4 ; -56.9)	**(1.4 ; -9.8 ; -14.6)**	(3.7 ; -37.7 ; -55.7)	(3.5 ; -33.6 ; -51.6)
	0.8	(3 ; -29.9 ; -49.3)	(3.3 ; -27.3 ; -38.9)	(2.2 ; -16.6 ; -20.5)	(2 ; -14.7 ; -20.9)	**(1.6 ; -7.8 ; -8.6)**	(2.2 ; -14.9 ; -26)	(1.8 ; -16.4 ; -31.5)
	Avg.	(4.7 ; -45.6 ; -58.3)	(5.4 ; -45.4 ; -55.7)	(4 ; -33 ; -39.2)	(4 ; -33 ; -40.1)	**(1.3 ; -5.4 ; -6.9)**	(3.5 ; -27.8 ; -40.7)	(3.2 ; -28.2 ; -43)
Avg.		(6.9 ; -59.6 ; -62.7)	(7.7 ; -59.8 ; -62.7)	(4.5 ; -17.9 ; -21.9)	(3.6 ; -22.2 ; -24.8)	**(1.2 ; 5.8 ; 3.4)**	(3.1 ; -29 ; -38.1)	(2.8 ; -24.5 ; -34.9)

Table 2: Aggregated results for large instances

Although f_1 is certainly the primary objective, the ordering of f_2 and f_3 might be different depending on the decision-maker. In any case, the proposed methods can be used in both situations by only changing the evaluation of a solution. To give some insight about the impact of inverting f_2 and f_3, some experiments are now reported. Small-size instances are used to allow all heuristics to obtain similar results on f_1, in order to facilitate the comparison. Let (P') denote the problem where f_3 is infinitely more important than f_2.

Table 3 reports the results obtained by TS^{Div}, TS^{E1} and TS^{E2} for the instances with 25 vertices. The average score (over 5 runs) on each objective is given rather than the gaps. As expected, the three methods obtain similar results on f_1, with an average gain of 1206.7, 1208.3 and 1208, respectively, for TS^{Div}, TS^{E1} and TS^{E2}. TS^{Div} is outperformed by TS^{E1} and TS^{E2} on the objectives f_2 and f_3. This is in line with the conclusion drawn from the previous experiments on problem (P). Not surprisingly, solving (P) instead of (P') leads to better (resp. worse) performance on f_2 (resp. f_3). For example, TS^{Div} has an average value of 43.9 for f_2 and 333.6 for f_3 when solving (P), whereas it obtains an average value of 57.1 for f_2 and 24.3 for f_3 when tackling (P').

			(P)			(P')		
n	d	label	TS^{Div}	TS^{E1}	TS^{E2}	TS^{Div}	TS^{E1}	TS^{E2}
25	0.2	a	(1054 ; 30.8 ; 183.6)	(1054 ; 21 ; 107.8)	(1054 ; 23.6 ; 119.2)	(1054;35.8;133.2)	(1054;21;107)	(1054;23.2;111.2)
		b	(1120 ; 27.5 ; 130.8)	(1120 ; 21 ; 95)	(1120 ; 24 ; 106.6)	(1120;34.4;115.4)	(1120;21;95)	(1120;24;99.6)
		c	(1179 ; 34.1 ; 191.6)	(1179 ; 22.2 ; 122)	(1179 ; 29.4 ; 140.2)	(1179;43;164.8)	(1179;22.4;114.4)	(1179;30.4;132.4)
		d	(1070 ; 23.9 ; 126)	(1070 ; 18 ; 95)	(1070 ; 21.2 ; 109.8)	(1070;31.2;112.2)	(1070;18;95)	(1070;19.6;97)
		e	(907 ; 26.3 ; 127.2)	(907 ; 22 ; 97)	(907 ; 24.2 ; 109.2)	(907;32.8;114.6)	(907;22;97)	(907;24;101.8)
	0.5	a	(1096 ; 45.4 ; 362.8)	(1096 ; 20.8 ; 131.8)	(1096 ; 32.2 ; 233.2)	(1096;58.8;249.2)	(1096;20.2;118.6)	(1096;34.6;155.8)
		b	(1418.4 ; 49 ; 370.8)	(1419 ; 23.4 ; 160.2)	(1419 ; 33.8 ; 206.4)	(1414.8;82;276.2)	(1419;24.4;152.4)	(1416.2;31;170)
		c	(1389 ; 50.4 ; 319)	(1389 ; 28.8 ; 187.2)	(1389 ; 43 ; 232.6)	(1389;73.8;296.2)	(1389;30.2;167.2)	(1389;49.4;212)
		d	(1478.5 ; 51 ; 339.6)	(1481 ; 30.6 ; 198.6)	(1481 ; 43.8 ; 221.4)	(1481;77.8;304.8)	(1481;32.8;175.2)	(1481;53.6;213.4)
		e	(1366 ; 38.9 ; 275.6)	(1366 ; 24 ; 118)	(1366 ; 31.8 ; 154)	(1366;67.8;230.8)	(1366;24;118)	(1366;36.2;144.4)
	0.8	a	(1125.5 ; 60.4 ; 589.2)	(1126 ; 33 ; 215)	(1126 ; 47.4 ; 302.2)	(1116.2;43;227)	(1124.2;28.4;134)	(1121;37.2;158.4)
		b	(1636 ; 56.4 ; 592.8)	(1636 ; 24 ; 150.4)	(1636 ; 46.4 ; 317.6)	(1636;75.8;439.2)	(1636;25.6;147.4)	(1636;57.2;250)
		c	(961 ; 41.9 ; 230.4)	(961 ; 25.2 ; 134.8)	(961 ; 37.8 ; 172.4)	(961;56.2;202.8)	(961;26.2;115.6)	(961;42.8;159.2)
		d	(1025.5 ; 62.5 ; 574.6)	(1026 ; 36.2 ; 253.8)	(1026 ; 61 ; 388.2)	(1025.8;95.2;505.2)	(1024.8;35.8;196.2)	(1025.8;72.4;308.8)
		e	(1299 ; 60.1 ; 590.2)	(1299 ; 31 ; 234.4)	(1299 ; 52.8 ; 354.6)	(1285;49.4;279.6)	(1297.8;30.4;193)	(1297.4;49;264)
Average			(1208.3; 43.9; 333.6)	(1208.6; 25.4; 153.4)	(1208.6; 36.8; 211.2)	(1206.7;57.1;243.4)	(1208.3;25.5;135)	(1208;39;171.9)

Table 3: Comparison of the results between problems (P) and (P')

By definition, the lexicographic approach means that a lower level objective cannot be improved at the expense of an upper level objective. When switching f_2 and f_3 (i.e., f_3 becomes more important than f_2) to get problem (P'), it is therefore possible to improve f_3, when compared to its value in problem (P), because f_2 is allowed to deteriorate, again when compared to its value in problem (P). It might also happen that both f_2 and f_3 are improved in (P'), when compared to (P), for the same value of f_1. This situation can sometimes occur because the structure of two solutions may differ even if they share the same f_1 value. A careful study of Table 3 reveals, however, that this situation happens in only 4 cases out of 45. When performing additional experiments on larger instances with 50 and 100 jobs, this type of dominance was not observed at all (out of 90 instances).

Considering again Table 3 and comparing the solutions produced by TS^{Div}, TS^{E1} and TS^{E2} on problems (P) and (P'), we note that the f_1 values differ in 12 cases out of 45 (twice in favor

of (P')). Overall, the solutions produced by each method on (P') dominate those obtained on (P) in 8 cases out of 45. These results suggest that it could sometimes be beneficial to run our algorithms twice on each instance, once with the order $f_2 \rightarrow f_3$ and once with the order $f_3 \rightarrow f_2$, and to return the best solution obtained at the end.

7 Conclusion

In this paper, a graph multi-coloring problem, which is used to model practical scheduling applications, is solved with heuristic and metaheuristic methods, in particular 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 a tabu search framework are empirically demonstrated, like the joint use of several types of neighborhoods, the diversification mechanism and an exact recoloring procedure.

This work establishes a link between graph coloring and job scheduling. Because of the scheduling features considered, this link is probably among the strongest between the two research communities. As we come from the graph coloring community, our test instances are based on graph coloring references, to which scheduling features have been added. Another approach would be to consider benchmark instances reported in the scheduling literature, to which graph coloring features would be added.

Two extensions of our work are suggested. First, the number of used colors (makespan) could be part of the objectives, rather than being fixed, thus allowing all vertices to be colored. In addition, other graph multi-coloring extensions could be considered to model other scheduling environments with incompatible jobs, like a flow shop or a job shop.

References

Bartal, Y., Leonardi, S., Marchetti-Spaccamela, A., Sgall, J., & Stougie, L. (1996). Multiprocessor scheduling with rejection. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms* SODA '96 (pp. 95–103). Philadelphia, USA.

Belfares, L., Klibi, W., Lo, N., & Guitouni, A. (2007). Multi-objectives tabu search based algorithm for progressive resource allocation. *European Journal of Operational Research*, *177*, 1779 – 1799.

Bloechliger, I., & Zufferey, N. (2013). Multi-coloring and project-scheduling with incompatibility and assignment costs. *Annals of Operations Research*, *211*, 83–101.

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*, *211*.

Cao, Z., & Yang, X. (2009). A PTAS for parallel batch scheduling with rejection and dynamic job arrivals. *Theoretical Computer Science*, *410*, 2732 – 2745.

Chen, J., & Lee, C.-Y. (1999). General multiprocessor task scheduling. *Naval Research Logistics (NRL)*, *46*, 57–74.

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.

Dósa, G., & He, Y. (2006). Preemptive and non-preemptive on-line algorithms for scheduling with rejection on two uniform machines. *Computing*, *76*, 149 – 164.

Drozdowski, M. (1996). Scheduling multiprocessor tasks - an overview. *European Journal of Operational Research*, *94*, 215 – 230.

Edis, E. B., Oguz, C., & Ozkarahan, I. (2013). Parallel machine scheduling with additional resources: Notation, classification, models and solution methods. *European Journal of Operational Research*, *230*, 449 – 463.

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

Fanjul-Peyro, L., & Ruiz, R. (2012). Scheduling unrelated parallel machines with optional machines and jobs selection. *Computers & Operations Research*, *39*, 1745–1753.

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.

Gerstl, E., & Mosheiov, G. (2012). Scheduling on parallel identical machines with job-rejection and position-dependent processing times. *Information Processing Letters*, *112*, 743 – 747.

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.

Halldórsson, M. M., & Kortsarz, G. (2004). Multicoloring: Problems and techniques. In J. Fiala, V. Koubek, & J. Kratochvíl (Eds.), *Mathematical Foundations of Computer Science* (pp. 21–45). Berlin, Heidelberg, New York: Springer volume 3153 of *Lecture Notes in Computer Science*.

He, Y., & Min, X. (2000). On-line uniform machine scheduling with rejection. *Computing*, *65*, 1–12.

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.

Hoogeveen, H., Skutella, M., & Woeginger, G. J. (2000). Preemptive scheduling with rejection. *Lecture Notes in Computer Science*, *1879*, 268 – 277.

Hsu, T., Korbaa, O., Dupas, R., & Goncalves, G. (2008). Cyclic scheduling for f.m.s.: Modelling and evolutionary solving approach. *European Journal of Operational Research*, *191*, 464 – 484.

ten Kate H. A. (1994). Towards a better understanding of order acceptance. *International Journal of Production Economics*, *37*, 139–152.

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.

Li, S., & Yuan, J. (2010). Parallel-machine scheduling with deteriorating jobs and rejection. *Theoretical Computer Science*, *411*, 3642–3650.

Lu, L., Cheng, T., Yuan, J., & Zhang, L. (2009). Bounded single-machine parallel-batch scheduling with release dates and rejection. *Computers and OperationsResearch*, *36*, 2748–2751.

Lu, L., Zhang, L., & Yuan, J. (2008). The unbounded parallel batch machine scheduling with release dates and rejection to minimize makespan. *Theoretical Computer Science*, *396*, 283 – 289.

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.

Marx, D. (2004). Graph coloring problems and their applications in scheduling. In *John Von Neuman PhD students conference* (pp. 11–16). volume 48.

Mati, Y., & Xie, X. (2011). Multiresource shop scheduling with resource flexibility and blocking. *Automation Science and Engineering, IEEE Transactions on*, *8*, 175–189.

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.

Oğuz, C., Salman, S. F., & Yalçın, B. Z. (2010). Order acceptance and scheduling decisions in make-to-order systems. *International Journal of Production Economics*, *125*, 200 – 211.

Pinedo, M. (2012). *Scheduling: Theory, Algorithms and Systems*. New York: Springer.

Rudek, R., Rudek, A., & Kozik, A. (2013). The solution algorithms for the multiprocessor scheduling with workspan criterion. *Expert Systems with Applications*, *40*, 2799 – 2806.

Schindl, D., & Zufferey, N. (2015). A Learning Tabu Search for a Truck Allocation Problem with Linear and Nonlinear Cost Components. *Naval Research Logistics*, *62 (1)*, 32 – 45.

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, US.

Seiden, S. S. (2001). Preemptive multiprocessor scheduling with rejection. *Theoretical Computer Science*, *262*, 437–458.

Shachnai, H., Tami, T., & Woeginger, G. (2002). Minimizing makespan and preemption costs on a system of uniform machines. *Lecture Notes in Computer Science*, *2461*, 859–871.

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.

Thevenin, S., Zufferey, N., & Potvin, J.-Y. (2014). Multi-objective parallel machine scheduling with incompatible jobs. In *ROADEF-15ème congrès annuel de la Société française de recherche opérationnelle et d'aide à la décision*.

Thevenin, S., Zufferey, N., & Widmer, M. (2015). Metaheuristics for a scheduling problem with rejection and tardiness penalties. *Journal of Scheduling*, *18 (1)*, 89 – 105.

de Werra, D., Demange, M., Monnot, J., & Paschos, V. T. (2005). A hypocoloring 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.

Yang, K.-h., Pulat, P. S., & Guan, Y. (2009). Embedded simulation on a multiprocessor job scheduling system with inspection. *Computers & Industrial Engineering*, *57*, 592 – 607. Challenges for Advanced Technology.

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

Inst.	*Best*	*ILP*	*Greedy*	*Descent*	*TS*	TS^{Drop}	TS^{Div}	TS^{E1}	TS^{E2}
10_2_a	(300 ; 5 ; 28)	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**
10_2_b	(286 ; 9 ; 35)	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	(0 ; 6.7 ; 1.7)	(0 ; 17.8 ; 6.3)	(0 ; 2.2 ; 3.4)	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**
10_2_c	(537 ; 8 ; 38)	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	(0 ; 5 ; 5.3)	(0 ; 5 ; 2.1)	(0 ; 3.8 ; 7.9)	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**
10_2_d	(348 ; 9 ; 34)	**(0 ; 0 ; 0)**	(0 ; 0 ; 0)	(0 ; 0 ; 0)	(0 ; 8.9 ; 12.9)	(0 ; 2.2 ; 6.5)	(0 ; 1 ; 5.3)	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**
10_2_e	(339 ; 9 ; 41)	**(0 ; 0 ; 0)**	(15.9 ; -11.1 ; -14.6)	**(0 ; 0 ; 0)**	(0 ; 2.2 ; 0.5)	(0 ; 8.9 ; 2)	(0 ; 4 ; 2.9)	**(0 ; 0 ; 0)**	(0 ; 2.2 ; 2)
10_5_a	(321 ; 9 ; 42)	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	(0 ; 82.2 ; 75.7)	(0 ; 60 ; 47.1)	(0 ; 24.8 ; 53.8)	**(0 ; 0 ; 0)**	(0 ; 11.1 ; 4.3)
10_5_b	(528 ; 9 ; 47)	**(0 ; 0 ; 0)**	(0 ; 11.1 ; 25.5)	(0 ; 0 ; 0)	(0 ; 22.2 ; 28.1)	(0 ; 31.1 ; 39.6)	(0 ; 10.1 ; 22.6)	**(0 ; 0 ; 0)**	(0 ; 2.2 ; 0.4)
10_5_c	(194 ; 6 ; 21)	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**
10_5_d	(603 ; 8 ; 50)	**(0 ; 0 ; 0)**	(0 ; 12.5 ; 24)	**(0 ; 0 ; 0)**	(0 ; 7.5 ; 8.4)	(0 ; 25 ; 19.2)	(0 ; 1.3 ; 0)	**(0 ; 0 ; 0)**	(0 ; 2.5 ; 0.4)
10_5_e	(402 ; 9 ; 44)	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	(0 ; 37.8 ; 45.9)	(0 ; 37.8 ; 40.5)	(0 ; 23.2 ; 25.9)	**(0 ; 0 ; 0)**	(0 ; 24.4 ; 18.2)
10_8_a	(434 ; 9 ; 40)	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	(0 ; 53.3 ; 55)	(0 ; 75.6 ; 85)	(0 ; 32.5 ; 79.5)	**(0 ; 0 ; 0)**	(0 ; 8.9 ; 10.5)
10_8_b	(566 ; 7 ; 49)	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	(0 ; 154.3 ; 105.3)	(0 ; 137.1 ; 100.8)	(0 ; 76.6 ; 147.3)	**(0 ; 0 ; 0)**	(0 ; 37.1 ; 38)
10_8_c	(209 ; 7 ; 19)	**(0 ; 0 ; 0)**	(1.4 ; -14.3 ; -5.3)	(1.4 ; -14.3 ; -5.3)	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**
10_8_d	(453 ; 7 ; 45)	**(0 ; 0 ; 0)**	(0 ; 0 ; 0)	(0 ; 0 ; 0)	(0 ; 68.6 ; 89.3)	(0 ; 68.6 ; 61.3)	(0 ; 15.6 ; 43.6)	**(0 ; 0 ; 0)**	(0 ; 17.1 ; 17.8)
10_8_e	(365 ; 5 ; 26)	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	(0 ; 4 ; 0.8)	(0 ; 4 ; 0.8)	(0 ; 2 ; 1.5)	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**
25_2_a	(1054 ; 21 ; 107)	**(0 ; 0 ; 0)**	(0 ; 0 ; 0.9)	(0 ; 0 ; 0.9)	(0 ; 75.2 ; 45.6)	(0 ; 116.2 ; 67.1)	(0 ; 46.9 ; 71.6)	(0 ; 0 ; 0.7)	(0 ; 12.4 ; 11.4)
25_2_b	(1120 ; 21 ; 95)	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	(0 ; 41 ; 27.6)	(0 ; 45.7 ; 30.7)	(0 ; 31.2 ; 37.7)	**(0 ; 0 ; 0)**	(0 ; 14.3 ; 12.2)
25_2_c	(1179 ; 21 ; 112)	**(0 ; 0 ; 0)**	(16.5 ; -14.3 ; -17)	(16.5 ; -14.3 ; -17)	(0 ; 124.8 ; 79.1)	(0 ; 111.4 ; 71.3)	(0 ; 62.3 ; 71.1)	(0 ; 5.7 ; 8.9)	(0 ; 40 ; 25.2)
25_2_d	(1070 ; 18 ; 95)	**(0 ; 0 ; 0)**	(3.5 ; -5.6 ; -5.3)	(3.5 ; -5.6 ; -5.3)	(0 ; 62.2 ; 34.9)	(0 ; 45.6 ; 25.3)	(0 ; 32.8 ; 32.6)	**(0 ; 0 ; 0)**	(0 ; 17.8 ; 15.6)
25_2_e	(907 ; 22 ; 97)	**(0 ; 0 ; 0)**	(1.2 ; -4.5 ; -3.1)	(1.2 ; -4.5 ; -3.1)	(0 ; 40.9 ; 35.3)	(0 ; 28.2 ; 26.6)	(0 ; 19.4 ; 31.1)	**(0 ; 0 ; 0)**	(0 ; 10 ; 12.6)
25_5_a	(1096 ; 18 ; 109)	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	(0 ; 5.6 ; 3.3)	(0 ; 258.9 ; 201.7)	(0 ; 304.4 ; 231.2)	(0 ; 152.1 ; 232.8)	(0 ; 15.6 ; 20.9)	(0 ; 78.9 ; 113.9)
25_5_b	(1419 ; 21 ; 139)	**(0 ; 0 ; 0)**	(0.3 ; 3.8 ; 0)	(0.5 ; 1 ; -4.9)	(0 ; 316.2 ; 205.2)	(0 ; 261 ; 182)	(0 ; 133.3 ; 166.8)	(0 ; 11.4 ; 15.3)	(0 ; 61 ; 48.5)
25_5_c	(1389 ; 26 ; 181)	(0 ; 46.2 ; 18.2)	(2 ; 0 ; -9.4)	(1.5 ; 3.1 ; -10.6)	(0 ; 173.1 ; 89.7)	(0 ; 136.2 ; 86.9)	(0 ; 93.7 ; 76.2)	**(0 ; 10.8 ; 3.4)**	(0 ; 65.4 ; 28.5)
25_5_d	(1481 ; 22 ; 147)	(2.5 ; 0 ; 0)	(2.5 ; -13.6 ; -23.8)	(2.5 ; -13.6 ; -23.8)	(0 ; 260.9 ; 164.2)	(0 ; 190 ; 127.2)	(0.2 ; 131.8 ; 131)	**(0 ; 39.1 ; 35.1)**	(0 ; 99.1 ; 50.6)
25_5_e	(1366 ; 24 ; 118)	**(0 ; 0 ; 0)**	**(0 ; 0 ; 0)**	(0 ; 7.5 ; 27.8)	(0 ; 115 ; 156.6)	(0 ; 100 ; 131.5)	(0 ; 62.1 ; 133.6)	**(0 ; 0 ; 0)**	(0 ; 32.5 ; 30.5)
25_8_a	(1126 ; 27 ; 173)	**(0 ; 0 ; 0)**	(0.3 ; -3 ; -16.8)	(0 ; 33.3 ; 34.2)	(0 ; 174.1 ; 231.1)	(0.1 ; 185.2 ; 246.2)	(0 ; 123.6 ; 240.6)	(0 ; 22.2 ; 24.3)	(0 ; 75.6 ; 74.7)
25_8_b	(1636 ; 23 ; 142)	(0 ; 13 ; 27.5)	(0.1 ; 12.2 ; 21.5)	(0 ; 19.1 ; 31.1)	(0 ; 280 ; 376.8)	(0 ; 209.6 ; 320)	(0 ; 145.1 ; 317.5)	**(0 ; 4.3 ; 5.9)**	(0 ; 101.7 ; 123.7)
25_8_c	(961 ; 17 ; 99)	**(0 ; 0 ; 0)**	(0 ; 0 ; 7.1)	(0 ; 0 ; 8.5)	(0 ; 170.6 ; 145.5)	(0 ; 163.5 ; 117.4)	(0 ; 146.5 ; 132.7)	(0 ; 48.2 ; 36.2)	(0 ; 122.4 ; 74.1)
25_8_d	(1026 ; 31 ; 234)	(1.2 ; 0 ; 0)	(1.7 ; -17.4 ; -31.2)	(0.7 ; 14.8 ; 19)	(0 ; 165.8 ; 174.1)	(0 ; 151 ; 215.3)	(0 ; 101.5 ; 145.6)	**(0 ; 16.8 ; 8.5)**	(0 ; 96.8 ; 65.9)
25_8_e	(1299 ; 21 ; 126)	(0 ; 28.6 ; 46.8)	(1.3 ; -4.8 ; -10.3)	**(0 ; 7.6 ; 22.2)**	(0 ; 300 ; 418.7)	(0 ; 233.3 ; 330.8)	(0 ; 186.1 ; 368.4)	(0 ; 47.6 ; 86)	(0 ; 151.4 ; 181.4)
50_2_a	(1299 ; 21 ; 126)	**(-0.4 ; 18 ; 0)**	(2.4 ; -31.1 ; -24)	(2.4 ; -25.6 ; -13.5)	(1 ; 82.3 ; 50.5)	(0.2 ; 79 ; 46.4)	(0.2 ; 47.4 ; 49)	(0.4 ; 6.6 ; 1.8)	(0.2 ; 27.5 ; 12.6)
50_2_b	(2360 ; 61 ; 304)	(0 ; 3.6 ; 24.8)	(6.1 ; -26.8 ; -6.8)	(6.3 ; -33.9 ; -12.4)	(0.1 ; 78.2 ; 51.8)	(0.1 ; 11.8 ; 11.1)	(0 ; 26.6 ; 39.6)	**(0.1 ; -6.4 ; -2.4)**	(0 ; 1.8 ; 3.2)
50_2_c	(2009 ; 56 ; 266)	**(0 ; 4.3 ; 1.2)**	(1.8 ; -15.2 ; -15.1)	(2.1 ; -15.7 ; -10.6)	(2.5 ; 125.2 ; 58.3)	(0.1 ; 63.5 ; 44.6)	(0.2 ; 67.8 ; 68.9)	(0.3 ; 2.6 ; 4.9)	(0.4 ; 40 ; 12.6)
50_2_d	(2186 ; 46 ; 251)	(1.1 ; 0 ; 0)	(4.6 ; -48.7 ; -32.3)	(4.7 ; -54.9 ; -45.9)	(1 ; 43.8 ; 24.2)	(1.1 ; 10.8 ; 11.7)	**(0.8 ; 16.4 ; 16.9)**	(1.5 ; -29.2 ; -25.8)	(1 ; 0.3 ; -9.2)
50_2_e	(2274 ; 78 ; 344)	**(0.5 ; 0 ; 0)**	(2.8 ; -40.8 ; -27.7)	(3.1 ; -47.4 ; -32.1)	(2.2 ; 22.4 ; 7.9)	(0.7 ; -3.4 ; -7)	(0.9 ; 9.5 ; 8.7)	(1.3 ; -18.4 ; -15.3)	(1.3 ; -10.5 ; -11.7)
50_5_a	(2279 ; 76 ; 365)	(13.4 ; 45.2 ; 57.1)	(2.6 ; -54.8 ; -41.7)	(3.2 ; -49.4 ; -24.3)	(4.6 ; 67.4 ; 56.2)	(2.7 ; 69.7 ; 68)	(1.6 ; 27.4 ; 60.7)	**(0.8 ; -19.7 ; -13.3)**	(0.8 ; 7.7 ; 6.3)
50_5_b	(2000 ; 62 ; 312)	(22 ; 20.5 ; 20.9)	(5.7 ; -60.3 ; -42.1)	(7.7 ; -63.3 ; -47.9)	(4.6 ; 59.2 ; 62.7)	(0.9 ; 45.8 ; 58.5)	(0.8 ; 20 ; 67)	(2.2 ; -18.1 ; 4.1)	**(0.8 ; -13.7 ; -10.5)**
50_5_c	(1989 ; 73 ; 340)	(1.6 ; 31.7 ; 36.1)	(2.6 ; -45 ; -0.8)	(2.9 ; -52 ; -29.3)	(3.8 ; 62.7 ; 70.6)	(1.6 ; 58 ; 74)	(1.6 ; 34.3 ; 75.9)	**(0.9 ; -8.3 ; -0.8)**	(1.6 ; 9.7 ; 22.9)
50_5_d	(1856 ; 60 ; 266)	(8.5 ; 0 ; 0)	(4.1 ; -67.2 ; -61)	(4.9 ; -69 ; -68.7)	(1.3 ; 1.2 ; -3.2)	(0.2 ; -12.4 ; -4.5)	**(0.2 ; -18.9 ; 12.5)**	(0.6 ; -41 ; -34.3)	(0.3 ; -37.8 ; -40.2)
50_5_e	(2004 ; 116 ; 556)	(10.4 ; 62.3 ; 44.4)	(2.8 ; -50.8 ; -29.7)	(3.1 ; -50.2 ; -30.1)	(2.5 ; 54.1 ; 55.5)	(1.3 ; 50.2 ; 56.2)	(0.8 ; 23.9 ; 54.5)	**(0.5 ; 0.3 ; 7.5)**	(0.5 ; 9.2 ; 20.3)
50_8_a	(1969 ; 61 ; 286)	(17.9 ; 0 ; 0)	(2.2 ; -23.5 ; -37.8)	(2.2 ; -21.7 ; -24.7)	(1.4 ; 212.2 ; 228.3)	(0.6 ; 207.4 ; 214.7)	(0.7 ; 145.8 ; 220.4)	**(0.3 ; 106.5 ; 78.9)**	(0.4 ; 106.1 ; 92.8)
50_8_b	(2205 ; 46 ; 317)	(17 ; 0 ; 0)	(4.8 ; -58.4 ; -62.9)	(5.4 ; -56.8 ; -56.5)	(3.7 ; 81.6 ; 49.2)	**(1.1 ; 70.8 ; 47.7)**	(1.3 ; 66.6 ; 54.8)	(2 ; 21.3 ; -5)	(1.6 ; 48.7 ; 9.1)
50_8_c	(2451 ; 76 ; 676)	(16.5 ; 0 ; 0)	(3 ; -31.4 ; -31.2)	(3.7 ; -31.8 ; -40.7)	(1.4 ; 169 ; 70.3)	**(0 ; 145.5 ; 68.7)**	(0.3 ; 139.9 ; 81.2)	(0.8 ; 61.6 ; 16.2)	(0.5 ; 113.3 ; 37.2)
50_8_d	(2414 ; 51 ; 532)	(10.2 ; 0 ; 0)	(3.2 ; -35.6 ; -32.5)	(3.3 ; -38.1 ; -27.8)	(2.2 ; 159.3 ; 166.9)	**(0 ; 134.8 ; 150.9)**	(0.4 ; 109.8 ; 175.8)	(1.2 ; 27.4 ; 39.6)	(0.6 ; 80 ; 58.8)
50_8_e	(1561 ; 54 ; 387)	(3.7 ; 27.9 ; 4.8)	(0.7 ; -54.7 ; -68.9)	(0.7 ; -50.9 ; -66)	(1.3 ; 87.1 ; 28.6)	(1.1 ; 45.6 ; 11.6)	**(0.5 ; 57.1 ; 23.2)**	(0.8 ; 19.1 ; -13.3)	(0.7 ; 46.5 ; -4.1)
Avg.		(2.8 ; 6.7 ; 6.3)	(2.1 ; -15.4 ; -12.7)	(1.9 ; -13.8 ; -10.1)	(0.7 ; 95.9 ; 84.3)	(0.3 ; 82.9 ; 77.2)	(0.2 ; 54.2 ; 79.8)	**(0.3 ; 7.2 ; 6.4)**	(0.2 ; 33.6 ; 25.8)

Table 4: Detailed results for small instances

Inst.	*Best*	*Greedy*	*Descent*	*TS*	TS^{Drop}	TS^{Div}	TS^{E1}	TS^{E2}
100_2_a	(3651 ; 106 ; 491)	(6.5 ; -40.6 ; -25.5)	(7.1 ; -46.6 ; -36.1)	(2 ; 54.3 ; 29.2)	(1.9 ; 37.2 ; 22.9)	**(1.2 ; 63.8 ; 32)**	(1.3 ; -6 ; -9)	(1 ; 18.5 ; 1.3)
100_2_b	(3533 ; 103 ; 450)	(10.4 ; -46.6 ; -22.2)	(11 ; -43.1 ; -19.8)	(1.3 ; 57.3 ; 35.4)	(6.9 ; 28.3 ; 25.9)	(1.1 ; 85 ; 54.4)	**(0.9 ; 3.3 ; 5.3)**	(2.1 ; 8.5 ; 5.6)
100_2_c	(3614 ; 206 ; 713)	(5.9 ; -72.3 ; -52)	(6.7 ; -74.3 ; -55.7)	(2.9 ; -24.2 ; -14.6)	**(0.4 ; -37.2 ; -18.4)**	(1.3 ; -16 ; -3.6)	(2.2 ; -55.5 ; -38)	(2.4 ; -51.5 ; -36.3)
100_2_d	(3670 ; 124 ; 508)	(8.7 ; -47.6 ; -34.6)	(9.5 ; -50.2 ; -34.1)	(2.9 ; 30.3 ; 32.5)	(3.8 ; 24.2 ; 28.3)	**(0.5 ; 36.5 ; 36.6)**	(1.2 ; -12.7 ; -3.4)	(0.9 ; 2.3 ; 5.2)
100_2_e	(3632 ; 116 ; 495)	(6.1 ; -55.2 ; -40.8)	(6.9 ; -58.1 ; -43.8)	(3.1 ; 25.3 ; 9.5)	(2.8 ; 9.5 ; 6.8)	(1.3 ; 52.6 ; 40.1)	**(0.8 ; -7.4 ; -7.9)**	(0.8 ; 3.6 ; -4.5)
100_5_a	(2752 ; 128 ; 637)	(20.8 ; -76.6 ; -68.9)	(21.4 ; -77.2 ; -69.8)	(6.7 ; 11.7 ; 6.8)	(7 ; 2.7 ; 5.3)	**(1.6 ; 6.9 ; 4.1)**	(2.6 ; -40.9 ; -40.1)	(2.5 ; -28.8 ; -30.6)
100_5_b	(2850 ; 173 ; 672)	(5.3 ; -75.7 ; -56.3)	(5.9 ; -78.8 ; -60.8)	(9.3 ; -15.4 ; 1)	**(1.7 ; -7.9 ; 4.6)**	(3.7 ; -17.7 ; 0.8)	(3.1 ; -52.3 ; -36.4)	(4.1 ; -43.6 ; -28.8)
100_5_c	(2504 ; 87 ; 489)	(6.8 ; -57.5 ; -57.9)	(8.1 ; -54 ; -48.3)	(3.6 ; 71.5 ; 46.2)	(2.2 ; 51.7 ; 40.6)	**(0.7 ; 76.1 ; 49.9)**	(1.2 ; 2.5 ; -18.6)	(1.2 ; 20.5 ; 0.6)
100_5_d	(2652 ; 120 ; 617)	(7 ; -60.8 ; -43.3)	(8 ; -63.8 ; -58.4)	(5.6 ; 27.3 ; 19.9)	**(1 ; 6.5 ; 1.9)**	(2.6 ; 40.2 ; 30.7)	(2.9 ; -18.3 ; -20.8)	(3.1 ; -9.3 ; -10.9)
100_5_e	(2959 ; 130 ; 625)	(6.8 ; -71.5 ; -59)	(7.3 ; -72.3 ; -62.2)	(6.5 ; -4.6 ; 0.1)	**(2.1 ; 2.8 ; 0.4)**	(2.4 ; 12.8 ; 8.4)	(3.2 ; -32.9 ; -30.3)	(3.3 ; -28.5 ; -27.6)
100_8_a	(2878 ; 172 ; 1348)	(6.6 ; -78.5 ; -80.6)	(8.1 ; -76.7 ; -76)	(7.4 ; -11.9 ; -13.6)	**(0.6 ; -8.3 ; -10)**	(1.9 ; 1.9 ; 0.4)	(2.9 ; -25.6 ; -39.4)	(2 ; -12.7 ; -26.6)
100_8_b	(2962 ; 130 ; 879)	(7.6 ; -69.2 ; -63)	(8.4 ; -69.5 ; -66.3)	(6.5 ; 28.3 ; 33.9)	(2.8 ; 25.2 ; 34.6)	**(2.3 ; 29.5 ; 41.6)**	(3.2 ; -0.3 ; -3.2)	(2.4 ; 6.8 ; 9.9)
100_8_c	(2770 ; 123 ; 851)	(3.4 ; -72.4 ; -51.5)	(4.4 ; -67.5 ; -56.5)	(5 ; 35.6 ; 37.9)	**(0.5 ; 16.9 ; 24.8)**	(1.4 ; 1.1 ; 5.9)	(1 ; -0.5 ; -9.5)	(1.9 ; -11.2 ; -13.4)
100_8_d	(2763 ; 152 ; 1196)	(12.1 ; -77 ; -78.3)	(12.7 ; -75.3 ; -74.9)	(5.2 ; 6.6 ; 2.6)	(2.5 ; 0 ; -11.5)	(2.1 ; 10.8 ; 3.3)	**(1.7 ; -22 ; -35.9)**	(1.8 ; -19.2 ; -30)
100_8_e	(2710 ; 166 ; 1185)	(7.9 ; -77.1 ; -69.7)	(8.4 ; -75.8 ; -76.8)	(3.6 ; -16.1 ; -14.5)	(2 ; -14 ; -7)	**(0.6 ; -2.7 ; -0.7)**	(1.9 ; -28.1 ; -37.6)	(1.4 ; -16.1 ; -24.5)
300_2_a	(14146 ; 893 ; 7752)	(8.1 ; -66.2 ; -70.5)	(8.6 ; -68.1 ; -70.9)	(5.3 ; -45.6 ; -43)	(4.9 ; -46.6 ; -47.2)	**(0.4 ; -0.1 ; -3.9)**	(3.3 ; -42.9 ; -51.7)	(3.2 ; -42.3 ; -51)
300_2_b	(14015 ; 839 ; 6334)	(9.8 ; -69.8 ; -69.1)	(11.4 ; -70.4 ; -67.7)	(4.6 ; -31.3 ; -19.1)	(5.1 ; -34.7 ; -21.3)	**(0.8 ; -3.1 ; 3.8)**	(3.8 ; -46.4 ; -50.7)	(2.2 ; -23.6 ; -23.9)
300_2_c	(12279 ; 731 ; 4683)	(9.8 ; -69.5 ; -65.3)	(11.5 ; -71.4 ; -67.1)	(2.7 ; -3.8 ; 14.7)	(5.1 ; -28 ; -14.2)	**(1.6 ; 0.6 ; 6.6)**	(4.4 ; -46.7 ; -44)	(3.9 ; -43.6 ; -43.5)
300_2_d	(15734 ; 763 ; 7972)	(5.9 ; -53.7 ; -62.8)	(6.7 ; -57 ; -65.6)	(6.8 ; -54.7 ; -63.7)	(6.3 ; -51.7 ; -61.5)	**(0.1 ; 9.9 ; -1.8)**	(5.1 ; -49.9 ; -61.8)	(3.8 ; -37.6 ; -52.6)
300_2_e	(16458 ; 959 ; 8890)	(7.5 ; -66 ; -66.1)	(8.8 ; -67.4 ; -70.7)	(6.9 ; -57.5 ; -58.7)	(7.5 ; -54.1 ; -54.2)	**(0.2 ; -7 ; -5.1)**	(3.5 ; -41.4 ; -49.8)	(2.9 ; -38 ; -45.6)
300_5_a	(14026 ; 864 ; 13639)	(10.9 ; -71.9 ; -79.1)	(12.1 ; -73.4 ; -79.7)	(3.9 ; -6.6 ; -7.8)	(4 ; -33.8 ; -30.5)	**(1.7 ; -0.8 ; -1.6)**	(5.6 ; -33.1 ; -48.8)	(4.7 ; -29.8 ; -44)
300_5_b	(15120 ; 952 ; 16849)	(8.4 ; -72.2 ; -79.6)	(9.8 ; -71.7 ; -81.5)	(3.5 ; -42.4 ; -43.5)	(2.7 ; -35.9 ; -32.1)	**(0.4 ; 2.3 ; -3.4)**	(4.1 ; -32.1 ; -53.4)	(2.9 ; -26 ; -40.7)
300_5_c	(11303 ; 747 ; 10335)	(7.5 ; -72.2 ; -78.3)	(9.5 ; -74.2 ; -79.6)	(3.8 ; -20.1 ; -17.9)	(1.5 ; -29.4 ; -27.9)	**(0.9 ; -3.8 ; -4.4)**	(3 ; -35.9 ; -43.5)	(2.1 ; -27.3 ; -39)
300_5_d	(15387 ; 1096 ; 20177)	(7.7 ; -73.8 ; -81.8)	(8.6 ; -72.3 ; -79.8)	(6.1 ; -58.6 ; -62.4)	(3.6 ; -51.6 ; -50.8)	**(0.2 ; -5.3 ; -3.7)**	(3.6 ; -47.1 ; -58.9)	(3.4 ; -45.2 ; -55.9)
300_5_e	(17285 ; 952 ; 19521)	(6.1 ; -63.9 ; -75.6)	(7 ; -65.2 ; -76.8)	(5.5 ; -54.9 ; -66.9)	(5.8 ; -57.9 ; -67)	**(0.6 ; -6.9 ; -8.4)**	(3 ; -34.5 ; -53.6)	(3.5 ; -41.8 ; -59.6)
300_8_a	(13072 ; 887 ; 24207)	(7.8 ; -75 ; -84.5)	(9 ; -74.7 ; -84.7)	(4.3 ; -49.2 ; -45.9)	(2.9 ; -45.5 ; -45.3)	**(0.7 ; -7.1 ; -2.8)**	(3.8 ; -39.8 ; -52.8)	(2.7 ; -36.2 ; -47.3)
300_8_b	(13960 ; 878 ; 25511)	(8.7 ; -73.6 ; -86.9)	(9.9 ; -73.4 ; -85.4)	(5.1 ; -38.8 ; -37.9)	(3.7 ; -42.6 ; -39.7)	**(0.6 ; -0.4 ; -0.1)**	(4.2 ; -40.4 ; -53.8)	(4.4 ; -25.8 ; -44.9)
300_8_c	(14800 ; 849 ; 27722)	(8.8 ; -70.9 ; -85.5)	(10.2 ; -72.3 ; -86.2)	(2.9 ; -25.6 ; -30)	(2.8 ; -35.3 ; -41.1)	**(0.7 ; -6.8 ; -10.3)**	(4.5 ; -29.8 ; -53.9)	(2.5 ; -33.2 ; -53.9)
300_8_d	(17772 ; 855 ; 29837)	(3.9 ; -59 ; -76.3)	(3.8 ; -57.8 ; -73.3)	(3.6 ; -52.3 ; -66)	(3.3 ; -49.9 ; -65.7)	**(0.5 ; -14.4 ; -12.6)**	(2.6 ; -44.7 ; -60.8)	(2.3 ; -45.6 ; -66.3)
300_8_e	(15456 ; 729 ; 25436)	(6.2 ; -60.5 ; -82.4)	(6.3 ; -56 ; -75.7)	(4.4 ; -44.7 ; -61.2)	(4.4 ; -45 ; -62.5)	**(1.1 ; 1.9 ; -0.9)**	(3.7 ; -26.5 ; -40)	(3.4 ; -22.4 ; -44.5)
500_2_a	(21608 ; 1068 ; 11274)	(10.1 ; -66.6 ; -71)	(11.2 ; -66.1 ; -70.9)	(1 ; -1.5 ; -1.8)	(3.1 ; -14.8 ; -15.6)	**(0.8 ; 18.2 ; 11.7)**	(4.8 ; -16.8 ; -35.5)	(4.4 ; -14 ; -34.7)
500_2_b	(27414 ; 599 ; 5694)	(0.8 ; 11.7 ; 21.7)	(1.3 ; 0.6 ; 4.5)	(1.7 ; 1.7 ; 8.8)	(1.2 ; 3.1 ; 9.5)	(1.7 ; 2.5 ; 11.4)	(1.7 ; -2.4 ; -7.1)	**(0.7 ; -2.8 ; -11)**
500_2_c	(25331 ; 1664 ; 19339)	(7.5 ; -70.6 ; -78.6)	(8.6 ; -69.9 ; -73.6)	(8.4 ; -66.2 ; -69.7)	(8.1 ; -64.5 ; -67.7)	**(0.7 ; -3.7 ; -1.3)**	(4.4 ; -35.2 ; -42.3)	(4.4 ; -46.4 ; -55.6)
500_2_d	(26026 ; 1524 ; 19716)	(6.3 ; -65.6 ; -71.6)	(7.2 ; -63.8 ; -71.9)	(7.3 ; -63 ; -69)	(7 ; -62.8 ; -68.6)	**(0.8 ; -6.1 ; -2.8)**	(5.8 ; -56.7 ; -64.9)	(6.1 ; -61.3 ; -71.2)
500_2_e	(26064 ; 1650 ; 20079)	(8.2 ; -71 ; -78.4)	(8.8 ; -69.8 ; -74.4)	(9.5 ; -67.1 ; -73.1)	(8.9 ; -66.1 ; -70.8)	**(0.8 ; -3.6 ; -5.9)**	(6.4 ; -41.9 ; -52.7)	(5.7 ; -48 ; -57.6)
500_5_a	(23875 ; 1074 ; 31257)	(5.8 ; -54.8 ; -76.2)	(6.7 ; -57 ; -78.1)	(6.1 ; -47.7 ; -67.3)	(6.9 ; -50 ; -70.1)	**(1.6 ; -2 ; -10.5)**	(4.9 ; -39.2 ; -60)	(5.3 ; -42.7 ; -64.1)
500_5_b	(13589 ; 887 ; 12623)	(6 ; -73.5 ; -78.5)	(7.1 ; -74 ; -80.4)	(4.4 ; -50.2 ; -51.7)	(4.7 ; -49.9 ; -52.8)	**(1.2 ; -11.9 ; -13.4)**	(4.2 ; -42.9 ; -53.3)	(3.4 ; -30 ; -38.4)
500_5_c	(20456 ; 1131 ; 24217)	(6.6 ; -69.2 ; -77.5)	(8.2 ; -67.4 ; -77.2)	(4.7 ; -45.5 ; -49.9)	(4.5 ; -47.2 ; -51)	**(1.4 ; -8.4 ; -8.3)**	(4.5 ; -40.9 ; -52.3)	(4.7 ; -36.9 ; -51.3)
500_5_d	(26030 ; 988 ; 29372)	(3.2 ; -42.3 ; -65.4)	(3.9 ; -43.5 ; -65.7)	(4.2 ; -41.2 ; -61.8)	(3.6 ; -39.7 ; -60.3)	**(1.4 ; -8.2 ; -14.1)**	(3.3 ; -32.7 ; -54.9)	(2.8 ; -25.4 ; -43.9)
500_5_e	(26781 ; 931 ; 23974)	(1.5 ; -33 ; -53)	(1.8 ; -33.1 ; -53.5)	(1.8 ; -31.1 ; -50.5)	(1.8 ; -30.4 ; -50.1)	**(1.3 ; -18.5 ; -26.8)**	(1.8 ; -33 ; -58)	(1.4 ; -32.9 ; -60.2)
500_8_a	(18937 ; 615 ; 20056)	(6.4 ; -49.1 ; -69.3)	(7.8 ; -52.1 ; -70.5)	(2.9 ; -14.6 ; -21.3)	**(1.4 ; -2.9 ; -1.2)**	(1.6 ; 4.4 ; 7.4)	(3.2 ; -3.6 ; -18.2)	(3.2 ; -9.2 ; -26.1)
500_8_b	(25953 ; 752 ; 23198)	(2.2 ; -26 ; -43)	(1.7 ; -18.2 ; -29)	(1.4 ; -13.9 ; -15.9)	(1.4 ; -11.7 ; -20.5)	**(1 ; -8 ; -13.4)**	(1.4 ; -19.4 ; -30.2)	(1 ; -17.3 ; -34.3)
500_8_c	(24253 ; 823 ; 34679)	(1.9 ; -36.4 ; -63.8)	(2.5 ; -33.8 ; -55.6)	(2.1 ; -31.3 ; -52.7)	(2.1 ; -32.5 ; -52.2)	(2.2 ; -25.9 ; -44.9)	(2 ; -29.6 ; -54.3)	**(1.4 ; -32.4 ; -57.3)**
500_8_d	(23701 ; 710 ; 27318)	**(2.1 ; -31.5 ; -60.2)**	(3.1 ; -31 ; -53.1)	(3.8 ; -29.4 ; -51.7)	(3.9 ; -28.6 ; -54.5)	(2.1 ; -12.4 ; -20)	(3.5 ; -26 ; -47.4)	(3.1 ; -24.7 ; -49.6)
500_8_e	(27087 ; 618 ; 15833)	(2.4 ; -6.3 ; -10.2)	(1.4 ; -1.6 ; 13.7)	(0.8 ; 6 ; 39.3)	(1 ; 2.3 ; 24)	(1.2 ; 3 ; 27.8)	(1.1 ; 4 ; 20.3)	**(0.5 ; 1.5 ; 10)**
Avg.		(6.9 ; -59.6 ; -62.7)	(7.7 ; -59.8 ; -62.7)	(4.5 ; -17.9 ; -21.9)	(3.6 ; -22.2 ; -24.8)	**(1.2 ; 5.8 ; 3.4)**	(3.1 ; -29 ; -38.1)	(2.8 ; -24.5 ; -34.9)

Table 5: Detailed results for large instances