Ann Oper Res (2013) 211:83–101
DOI 10.1007/s10479-013-1397-1

Multi-coloring and job-scheduling with assignment and incompatibility costs

Ivo Blöchliger · Nicolas Zufferey

Published online: 29 May 2013

Abstract Consider a scheduling problem (P) which consists of a set of jobs to be performed within a limited number of time periods. For each job, we know its duration as an integer number of time periods, and preemptions are allowed. The goal is to assign the required number of time periods to each job while minimizing the assignment and incompatibility costs. When a job is performed within a time period, an assignment cost is encountered, which depends on the involved job and on the considered time period. In addition, for some pairs of jobs, incompatibility costs are encountered if they are performed within common time periods. (P) can be seen as an extension of the multi-coloring problem. We propose various solution methods for (P) (namely a greedy algorithm, a descent method, a tabu search and a genetic local search), as well as an exact approach. All these methods are compared on different types of instances.

Keywords Job-scheduling · Multi-coloring · Tabu search · Genetic algorithm

1 Introduction

In this paper, we consider a scheduling problem (P) where a set of jobs have to be performed within a limited number of time periods. For each job, we know its duration as an integer number of time periods. Preemptions are allowed (i.e. it is possible to perform a job within non consecutive time periods). Two types of costs are considered: assignment costs and incompatibility costs. When a time period is assigned to a job, an assignment cost is encountered. In addition, for some pairs of jobs, incompatibility costs are encountered if they are performed within common time periods (i.e. the realization of the two jobs overlap

I. Blöchliger
University of Fribourg, Fribourg, Switzerland
e-mail: ivo.bloechliger@unifr.ch

N. Zufferey (✉)
Faculty of Economics and Social Sciences, HEC—University of Geneva, Geneva, Switzerland
e-mail: nicolas.zufferey-hec@unige.ch

in time). The goal is to perform all the jobs at minimum cost. Such a problem can be seen as an extension of the problem studied in Zufferey et al. (2012), for which all the jobs have a duration of one time period. Problem (P) is a new scheduling problem and there is no literature on it. The reader desiring a review on scheduling models and algorithms is referred to Pinedo (2008). Problem (P) can also be seen as an extension of the multi-coloring problem (also known as the set-coloring problem, for which there is no cost). Relevant references for the multi-coloring problem with applications in scheduling and in frequency assignment are Dorne and Hao (1998), Aardal et al. (2003), Halldorsson (2004), Gandhi et al. (2005), and Chiarandini and Stuetzle (2007). Problem (P) can finally be seen as a project management problem or a project scheduling problem. The reader interested in a general project management book with applications to planning and scheduling is referred to Kerzner (2003). Finally, the reader interested in project scheduling is referred to Icmeli et al. (1993), Kolisch and Padman (2001), Demeulemeester and Herroelen (2002), and Lancaster and Ozbayrak (2007).

The paper is organized as follows. In Sect. 2, we formally present and position problem (P), and we show the links between (P) and the multi-coloring problem. In Sect. 3, we design two mathematical models for (P) to be solved by exact methods/solvers. In Sect. 4 are proposed for (P) a constructive greedy algorithm, local search methods (namely a descent algorithm and a tabu search), and a population based method (namely a genetic local search). Results are reported in Sect. 5, where we also compare the proposed methods with other existing algorithms. We end the paper with a conclusion in Sect. 6.

2 Presentation and positioning of problem (P)

In this section, we formally describe problem (P), then we present the links between (P) and the multi-coloring problem, and we finally position (P) according to its practical relevance and according to classical scheduling problems (based on the well-known notation $\alpha \mid \beta \mid \gamma$).

2.1 Formal description of (P)

Problem (P) consists in a set V of n jobs to be performed, at minimum cost, within a discrete horizon of k time periods, each time period having the same duration (e.g., a day, an hour, a minute, etc.). No precedence constraints are considered and preemptions are allowed (i.e. it is possible to perform a job within nonconsecutive time periods, and interruptions are only allowed at integer time points). For each job j, we know its integer duration p_j. The goal is to assign p_j (not necessarily consecutive) time periods to each job j while minimizing assignment and incompatibility costs. An *assignment* cost $a_{j,t}$ occurs if job j is performed (partially or totally) within time period t. It represents for example the cost of the resources (e.g., staff, machines) which have to perform job j at period t. In addition, let $c^m_{j,j'} > 0$ (with $m \in \mathbb{N}^\star$) denote an *incompatibility* cost between incompatible jobs j and j', which is to be paid if both jobs have m common time periods. Note that this cost may be ∞ for jobs which can in no way be executed at the same time period. For compatible jobs j and j', we assume that $c^m_{j,j'} = 0$ for all m. Further, for each job j, let I_j denote the set of jobs j' which are incompatible with job j. We assume that $c^m_{j,j'} = c^m_{j',j}$ for all $j, j' \in V$. Hence, $j \in I_{j'}$ implies $j' \in I_j$ for all $j, j' \in V$. The incompatibility costs $c^m_{j,j'}$ represent for example that the same resources have to perform jobs j and j', thus additional resources are requested in order to be able to perform both jobs within common time periods. Thus, it is reasonable to assume that $c^{m+1}_{j,j'} \geq c^m_{j,j'}$ for all m.

In order to represent a solution s using a maximum of k time periods, we associate with each time period $t \in \{1, \dots, k\}$ a set J_t containing the jobs which are performed at time period t. It means that a single job j has to belong to p_j sets of type J_t in order to be totally performed. Let $\delta^m_{j,j'}$ be equal to 1 if jobs j and j' are performed within m common time periods, and 0 otherwise. Thus, a solution s can be denoted $s = (J_1, \dots, J_k)$, and the associated objective function $f(s)$ to minimize is:

$$\begin{aligned} f(s) &= \sum_t \sum_{j \in J_t} a_{j,t} + \sum_{j<j',m} c^m_{j,j'} \cdot \delta^m_{j,j'} \\ &= \sum_{t=1}^{k} \sum_{j \in J_t} a_{jt} + \sum_{j=1}^{n-1} \sum_{j' \in \{j+1,\dots,n\} \cap I_j} \sum_{m=1}^{\min\{p_j, p_{j'}\}} c^m_{jj'} \cdot \delta^m_{jj'} \end{aligned} \tag{1}$$

2.2 Links between (P) and the multi-coloring problem

Problem (P) can be considered as an extension of the *k-multi-coloring* problem, which can be defined as follows from the k-coloring problem. First, the *k-coloring* problem of a graph G consists in assigning a single color in $\{1, \dots, k\}$ to each vertex such that adjacent vertices have different colors. The *graph coloring* problem consists in finding the smallest k for which a k-coloring exists. Such a coloring problem is NP-hard (Garey and Johnson 1979). In the *k-multi-coloring* problem, each vertex has to receive a predefined number of colors in $\{1, \dots, k\}$ such that adjacent vertices have no common color (a *conflict* occurs if two adjacent vertices have at least one color in common). The *graph multi-coloring* problem consists in finding the smallest k for which a k-multi-coloring exists.

As mentioned in Halldorsson (2004), the multi-coloring problem can be reduced to graph coloring by replacing each vertex j by a clique of size p_j (where a *clique* is a set of mutually adjacent vertices). Edges are then replaced with complete bipartite graphs between the corresponding cliques (a graph is *bipartite* if its vertices can be partitioned in two sets such that there is no edge in each of the two sets). Such a transformation both increases the size of the graph and embeds an unwanted symmetry into the problem. Thus, it is useful to develop specialized algorithms for the multi-coloring problem, and therefore it is also useful to develop specialized algorithms for problem (P).

The correspondence between problem (P) and the k-multi-coloring problem is obvious: a vertex represents a job, a color is a time period, and the required number p_j of colors to assign to vertex j is the duration of job j. Apart from the objective function, the main differences between these two problems are the following: in problem (P), conflicts are allowed but lead to incompatibility costs, and, in addition, assignment costs are also considered (while they are all equal in the k-multi-coloring problem and can thus be ignored). Therefore, problem (P) can be considered as an extension of the k-multi-coloring problem, which means that it is also NP-hard. From now on, we will indifferently use the *scheduling* terminology (e.g., jobs, time periods) and the *graph* terminology (e.g., vertices, colors).

Among the few existing methods for the multi-coloring problem, tabu search was shown to provide very competitive results (e.g., Dorne and Hao 1998, Chiarandini and Stuetzle 2007). Among the best methods for the standard graph coloring problem (many algorithms exist, see Malaguti and Toth 2010 for a survey), we mention two tabu search methods (Hertz and de Werra 1987; Bloechliger and Zufferey 2008), an adaptive memory algorithm (Malaguti et al. 2008), a genetic local search algorithm (Galinier and Hao 1999), an ant local search (Plumettaz et al. 2010), a variable space search (Hertz et al. 2008), and a memetic algorithm (Lu and Hao 2010). All these methods are metaheuristics which rely on tabu search.

For these reasons, it seems appropriate to propose tabu search algorithms and genetic local search approaches for problem (P).

2.3 Positioning and practical relevance of (P)

The classical three-fields notation $\alpha \mid \beta \mid \gamma$ is well-known in the scheduling community (Pinedo 2008): field α represents the configuration of the machines, field β represents the constraints and specific characteristics of the problem, and field γ represents the objective to optimize. With such a notation, problem (P) could be denoted $P_m \mid \text{int-}p_j, \text{int-}prmp \mid C(a_{j,t}, c^m_{j,j'})$, where P_m holds for m parallel identical machines (with m large enough to be able to simultaneously perform as many jobs as desired), p_j holds for integer processing times int-p_j, int-$prmp$ holds for preemptions which are only allowed at integer time points, and $C(a_{j,t}, c^m_{j,j'})$ represents a cost function involving the assignment costs $(a_{j,t})$ and the incompatibility costs $(c^m_{j,j'})$.

As the multi-coloring problem can be viewed as roots of problem (P), all the application domains of the former problems can be relevant for the latter problem, which include scheduling file transfers (Coffman et al. 1985) or frequency assignment (Aardal et al. 2003). More generally, all the application domains of graph coloring can be relevant for problem (P) (e.g., Gamst and Rave 1982, Leighton 1979, Stecke 1985, Zufferey et al. 2008, Burke et al. 2010, and Duives et al. 2011).

Problem (P) has obvious applications in industries where: (1) parallel machines are considered; (2) preemptions are allowed; (3) overlapping some processing times is costly because it requires additional resources to augment the production capacity; (4) the processing cost of a job depends on the time slot it is performed (and thus earliness and tardiness issues can also be considered). These four specific features of (P) are now briefly discussed according to the existing literature. Most of the remaining references of the current section pointed out the practical relevance of the associated problems.

Parallel machines Most of the scheduling research dealing with multiple objectives focuses on single machine scheduling, and not as much research has been done on parallel machine problems with multiple objectives (Pinedo 2008). However, as stressed in T'Kindt and Billaut (2001), parallel machines scheduling problems are important because in practice there are often multiple resources dedicated to the processing of some operations. In Mendes et al. (2002), some metaheuristics are compared for a parallel machine scheduling problem while minimizing the makespan. Despite its relevance in the chemical, paper and textile industries, only few algorithms have been proposed to tackle this problem.

Preemptions In scheduling problems, when preemptions are allowed, a job can be stopped and restarted later. Preemptions are particularly relevant when the considered production process has negligible setup times, which is for instance the case with automated production systems. Several papers have tackled scheduling problems with preemption possibilities. A few of them are now mentioned. Liu et al. (2002) study the single machine scheduling problem with preemptions, release dates, preemption penalties, and delivery times, where the objective is to minimize the delivery time of the last job. Schuurman and Woeginger (1999) studied the problem of scheduling n jobs on m machines with setup times. Shachnai et al. (2002) impose a maximum number of preemptions in a multiprocessor scheduling environment. In Mohammadi and 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 preemptions. In Sun et al. (2005), the authors study the problem of scheduling jobs on a single machine with availability constraints and preemption possibilities. The objective is to minimize total weighted job completion times.

Incompatibility costs and production capacity The consideration of incompatibility costs can have an important impact in a competitive make-to-order environment, where a manufacturer should tune the production capacity efficiently, satisfy the expectations of customers, and gain the maximum revenue from the incoming orders (jobs). Hence, the question is which orders to accept and in what sequence to process them to maximize the revenues. The practical relevance and importance of order acceptance in make-to-order systems was underlined in Rogers and Nandi (2007) and Zorzini et al. (2008). For a recent review on the trade-off "which orders to accept" and "how to schedule them", the reader is referred to Slotnick (2011). As mentioned in Yang and Geunes (2007), only a small amount of past research considers demand management decisions in a production scheduling context. In order acceptance problems, if a job j is rejected, a penalty cost in incurred (or the revenue associated with j is lost). The consideration of incompatibility costs can obviously enhance the production capacity in such production environment, as instead of encountering penalty costs for rejected jobs, incompatibility costs could be paid if additional resources are used at some specific time periods (the ones for which incompatible jobs are performed).

Incompatibility costs can also be defined according to additional machine activation costs. In Panwalkar and Liman (2002), the authors consider a number of identical machines that can be activated to perform the work. Each machine that is activated incurs a fixed machine activation cost. A component of the objective function is the sum of machine activation costs. In Dosa and He (2006), the authors consider a problem for which rejection cost (a penalty cost is associated with each non performed job) and machine cost (buying new machines is possible if a cost is paid) are taken into account. In Chen (2004), the author considers a scheduling problem on parallel machines with resource allocation costs.

Assignment costs and earliness/tardiness penalties The assignment costs can be fixed in order to account for earliness and tardiness costs. Earliness and tardiness costs are crucial elements of many scheduling problems in practice, but they are only beginning to be *simultaneously* considered in the literature (e.g., Beck and Refalo 2003, Yang et al. 2004, Feng and Lau 2008, Shabtay 2008, Toksari et al. 2010). Consider that with each job j are associated four information: (1) an available date R_j (it is not possible to start j before R_j), (2) a release date r_j (it is possible to start j before r_j if an earliness cost is paid), (3) a due date d_j (it is possible to complete j after d_j if a tardiness cost is encountered), (4) a deadline D_j (it is not possible to finish j after D_j). The following relationship is likely to occur: $R_j < r_j < d_j < D_j$. The earliness costs can represent urgent transportation cost to get the raw material from a supplier earlier than r_j (such costs are decreasing with time, that is from time period R_j to time period $r_j - 1$). The tardiness costs can represent penalty costs associated with late deliveries to the clients (such cost are increasing with time, that is from time period $d_j + 1$ to time period D_j). To account for the hard constraints (i.e. R_j and D_j), assignment cost $a_{j,t}$ could be set to an arbitrary large number M for each $t < R_j$ and each $t > D_j$. To represent the earliness costs, $a_{j,t}$ can be set to a decreasing function when t varies from R_j to $r_j - 1$. To represent the tardiness costs, $a_{j,t}$ can be set to an increasing function when t varies from $d_j + 1$ to D_j.

3 Exact methods for problem (P)

The incompatibility cost component of the objective function f is non decreasing and generally non linear with the number of conflicts. In this section, we propose two integer programs for (P). The first one is based on a linear incompatibility cost component, whereas the second is general. The decision variables are the following: $x_{j,t} = 1$ if color t is assigned to vertex j, and 0 otherwise.

3.1 Linear incompatibility costs

If the incompatibility costs are linear with the number of conflicts, they can be rewritten as follows: $c^m_{j,j'} = m \cdot c_{j,j'}$ (with $c_{j,j'} \geq 0$). With those penalties, one obtains the following quadratic binary program:

$$\begin{aligned}
&\text{minimize} && f = \sum_{j=1}^{n} \left(\sum_{t=1}^{k} a_{j,t} \cdot x_{j,t} + \sum_{\substack{j' \in I_j \\ j < j'}} c_{j,j'} \cdot \sum_{t=1}^{k} x_{j,t} \cdot x_{j',t} \right) \\
&\text{constraints} && \sum_{t=1}^{k} x_{j,t} = p_j \quad \forall j \\
&\text{variables} && x_{j,t} \in \{0, 1\} \quad \forall j, t
\end{aligned} \tag{2}$$

When using this quadratic formulation with CPLEX 12.4, even the smallest problems cannot be solved within reasonable time and memory limits. However, this formulation can be linearized. Using the notation $e = \{j, j'\}$, the product $x_{j,t} \cdot x_{j',t}$ of two binary variables can be linearized using a standard technique as follows:

$$\begin{aligned}
h_{e,t} &\geq x_{j,t} + x_{j',t} - 1 \\
h_{e,t} &\leq x_{j,t} \\
h_{e,t} &\leq x_{j',t} \\
h_{e,t} &\in \{0, 1\}
\end{aligned} \tag{3}$$

With these conditions, we have $h_{e,t} = x_{j,t} \cdot x_{j,t'}$ for every possible conflict $e = \{j, j'\}$. Note that only the first inequality of (3) is necessary. The two others are implicit since the associated coefficients in the objective function are all positive. Using this formulation, the smallest problems can be efficiently solved by CPLEX 12.4.

3.2 Non-linear incompatibility costs

The general problem (P) has an arbitrary, albeit monotone, objective function. We now present a linear program to model this objective function. The basic idea is to encode the number of conflicts on each edge in unary notation.

For this, we introduce further helper variables. Let $H_{j,j'}$ be the total number of conflicts on an edge e. This number $H_{j,j'}$ is then encoded as a sum of binary variables $H_e = y^1_{j,j'} + y^2_e + y^3_e + \cdots + y_e^{M_{j,j'}}$ where $M_{j,j} = \min\{p_j, p_{j'}\}$ is the maximum number of conflicts possible on the edge $e = [j, j']$. We further impose that $y^i_e \geq y^{i+1}_e$ for $i \in \{1, \ldots, M_e - 1\}$. This implies that exactly the first H_e variables y^i_e will be set to one to represent H_e. The objective function can now be rewritten in a linear fashion and the problem formulation becomes the following (where E is the set of edges in the graph):

$$\begin{aligned}
\text{minimize} \quad & f = \sum_{j,t} a_{j,t} \cdot x_{j,t} + \sum_{[j,j'] \in E | j<j'} \left(c^1_{j,j'} \cdot y^1_{j,j'} + \sum_{i=2}^{M_{j,j'}} y^i_{j,j'} \cdot \left(c^i_{j,j'} - c^{i-1}_{j,j'}\right) \right) \\
\text{constraints} \quad & \sum_t x_{j,t} = p_j \quad \forall j \in V \\
& h_{j,j',t} \geq x_{j,t} + x_{j',t} - 1 \quad \forall [j,j'] \in E,\ j < j', t \\
& \sum_t h_{j,j',t} = \sum_{i=1}^{M_{j,j'}} y^i_{j,j'} \quad \forall [j,j'] \in E,\ j < j', t \qquad (4) \\
& y^i_{j,j'} \geq y^{i+1}_{j,j'} \quad \forall [j,j'] \in E,\ j < j',\ i = 1, \ldots, M_{j,j'} \\
\text{variables} \quad & x_{j,t} \in \{0,1\} \quad \forall j, t \\
& h_{j,j',t} \in \{0,1\} \quad \forall [j,j'] \in E,\ j < j', 1 \leq t \leq k \\
& y_{j,j',i} \in \{0,1\} \quad \forall [j,j'] \in E,\ j < j', i = 1, \ldots, M_{j,j'}
\end{aligned}$$

It is easy to check that the big parenthesis in the second sum of the objective function equals indeed $c^{H_{j,j'}}_{j,j'}$.

3.3 Extracting lower bounds

The models presented above can either be solved to optimality for smaller instances, or used to extract lower bounds for larger instances.

For the general problem and larger instances, the gaps are big (typically 50 % and more). In some cases, one obtains better lower bounds by relaxing the problem with a linear approximation of the objective function as follows. We replace the coefficients $c^m_{j,j'}$ by such that $c'^1_{j,j'} := \min_m \frac{c^m_{j,j'}}{m}$ and $c'^m_{j,j'} := m \cdot c'^1_{j,j'}$ for $m \geq 2$. In other words, the conflicts costs c' correspond to the steepest linear function which never overestimates the original non-linear conflict cost function (i.e. the least steep linear function which coincides with the original conflict cost function in at least one point).

The optimal solution of this relaxed problem is clearly inferior (or equal) to the optimal solution of the original problem. The relaxed problem however is much easier to solve, and therefore it might provide better lower bounds considering memory and CPU-time limits.

4 Heuristics for problem (P)

In this section, we propose four solution methods for problem (P): a greedy algorithm, a descent method, a tabu search and a genetic local search.

4.1 Greedy algorithm

The following constructive heuristic has been implemented. Starting with an empty solution (where no color is assigned to any vertex), the best assignment (vertex j, color t) is performed at each step, i.e. the assignment which leads to the smallest augmentation of f. If several vertices satisfy this condition, the vertex with the smallest number of available colors is selected (a color t is defined as available for a vertex j if t is not used in the vertices adjacent to j). Ties are then broken randomly. Note that such a way of selecting the vertices leads to much better results than a random selection. Such a *Greedy*(P) heuristic can be considered as a generalization of the DSATUR standard coloring algorithm proposed in Brélaz (1979), where at each step, the next vertex to color (with the smallest possible color which does not create any conflict) is the one with the largest number of different colors represented in the set of its adjacent colored vertices.

4.2 Local search methods

We propose below two local search algorithms for (P): a descent method and a tabu search, respectively denoted by *Descent*(P) and *Tabu*(P) .

A *local search* can be described as follows. Let f be an objective function which has to be minimized. At each step, a *neighbor* solution s' is generated from the current solution s by performing a specific modification on s, called a *move*. All solutions obtained from s by performing a move are called *neighbor solutions* of s. The set of all the neighbor solutions of s is denoted $N(s)$. First, a local search needs an initial solution s_0 as input. Then, the algorithm generates a sequence of solutions $s_1, s_2, \dots$ in the search space such that $s_{r+1} \in N(s_r)$. The process is stopped for example when an optimal solution is found (if it is known), or when a fixed number of iterations have been performed. Some famous local search algorithms are the descent method, simulated annealing, variable neighborhood search, and tabu search.

In a *descent method*, the best move (among all possible candidate moves) is performed at each iteration and the algorithms stops as soon as a local optimum is reached. In a *tabu search*, when a move is performed from s_r to s_{r+1}, it is forbidden (with some exceptions) to perform the inverse of that move during τ (parameter) iterations: such forbidden moves are called *tabu* moves. The solution s_{r+1} is computed as $s_{r+1} = \arg\min_{s \in N'(s_r)} f(s)$, where $N'(s)$ is a subset of $N(s)$ containing solutions which can be obtained from s by performing a non tabu move. Many variants and extensions of this basic tabu search algorithm can be found for example in Glover and Laguna (1997). More generally, the reader is referred to Gendreau and Potvin (2010) and Osman and Laporte (1996) for the metaheuristics literature, and to Zufferey (2012) for guidelines helping to efficiently design a metaheuristic.

In order to propose a local search for problem (P), we mainly have to define the search space, the way to generate an initial solution, the neighborhood structure (i.e. the nature of a move), the stopping criterion, and, in the case a tabu search is designed, the way to manage the tabu tenures.

Search space A feasible solution is any assignment of the correct number of colors to each vertex. The search space is the set of all the feasible solutions of (P), and the associated objective function is simply f as defined in (1). Note that feasible solutions may have an infinite objective value, as some incompatibility costs are set to ∞ in the considered instances. Such infinite costs represent the situation where it is logistically impossible to have a certain number of common time periods for the processing of some pairs of jobs.

Initial solution The initial solution for *Descent*(P) and *Tabu*(P) is a random assignment of p_j colors to each vertex j.

Table 1 Time limit and parameters used for *Tabu*(P) on the multi-coloring instances

n	T	N	K	τ_1
≤30	300 s	0.5	0.5	20
50	600 s	0.5	0.5	30
100	1200 s	0.3	0.5	40
200	3600 s	0.2	0.5	50

Neighborhood structure A neighbor solution is produced by changing exactly one color on a vertex. In other words, a move consists in replacing, for a single job j, a time period t with another time period t'. Such a move can be denoted by (j,t,t'). For the current solution s, let C_j be the set of time periods associated with job j, and let $n_{j,j'}$ be the number of common time periods between jobs j and j', i.e. $n_{j,j'} = |C_j \cap C_{j'}|$. Knowing the current solution s and its value $f(s)$, it is now straightforward to develop an *incremental* computation to evaluate a candidate neighbor solution s' obtained from s by replacing t with t' for j:

$$f(s') = f(s) - a_{j,t} + a_{j,t'} - \sum_{j' \in I_j | t \in C_{j'}} \left(c_{j,j'}^{n_{j,j'}} - c_{j,j'}^{(n_{j,j'}-1)}\right) + \sum_{j' \in I_j | t' \in C_{j'}} \left(c_{j,j'}^{(n_{j,j'}+1)} - c_{j,j'}^{n_{j,j'}}\right) \quad (5)$$

In *Descent*(P), starting with a random initial solution, the best move is selected at each iteration (i.e. an *exhaustive* search is performed), by using the above incremental computation to evaluate all the possible candidate moves. In *Tabu*(P), we avoid the exhaustive search strategy at each iteration. Instead, we propose to control the size of the evaluated set of candidate moves by two parameters N and K, which respectively indicate the considered proportion of jobs and time periods. Such parameters seem to be critical to the search process. For larger instances, it pays off to choose N smaller, down to 0.1 for very large instances.

Stopping criterion The stopping condition of all (meta)heuristics is a time limit of T seconds, where T depends on the number of vertices of the graph. T was chosen such that (in preliminary experiments) the improvement after this amount of time was negligible. In order to apply *Greedy*(P) for T seconds, we restart is as soon as a complete solution is found (as long as the time limit is not reached). At each iteration of *Greedy*(P), several equivalent options usually occur. As such ties are randomly broken, two runs of *Greedy*(P) are likely to provide two different solutions. Similarly, in order to perform *Descent*(P) for T seconds, we restart it with another random initial solution as soon as a local optimum is reached. At the end, the best encountered solution within T seconds is given as output.

Tabu tenures Assume that move (j,t,t') has just been performed. The moves (j,t,t') and (j,t',t) are then forbidden for a given number τ_1 of iterations (parameter, which will be tuned depending on the size of the instance). Preliminary experiments on random instances showed that $\tau_1 \in [1, 50]$ is suitable, with small tenures for small instances, and large tenures for large instances.

Parameter settings The parameters N, K and τ_1, as well as the time limit T (in seconds, which will be the same for all the methods), are given in Table 1 and depend on the number n of vertices.

Algorithm 1 Genetic local search

```
Generate an initial population $\mathcal{M}$ of solutions.

While a stopping condition is not met, do:

1. create offspring solutions from $\mathcal{M}$ by using a recombination operator;
2. improve the offspring solutions by the use of a local search operator;
3. update $\mathcal{M}$ by the use of the improved offspring solutions.
```

4.3 Genetic local search

In this subsection, we first briefly present the main ingredients of a *genetic local search* algorithm. Then we adapt such algorithm to problem (P), and the resulting method is denoted $GLS(P)$. A basic version of a genetic local search (which is close to the *adaptive memory algorithm* proposed in Rochat and Taillard 1995) is summarized in Algorithm 1, where performing steps (1), (2) and (3) is called a *generation*. Therefore, in order to design a genetic local search for a problem, we have to define: the way to initialize the population $\mathcal{M}$ of solutions, the recombination operator, the intensification (or local search) operator, and the population update operator. The reader interested in more detail about genetic (local search) algorithms is referred to Gendreau and Potvin (2010).

Initialization of $\mathcal{M}$ Based on preliminary experiments, we propose to work with a population $\mathcal{M}$ of size 6. Each initial solution is randomly generated and then improved by $Tabu(P)$ performed during 10,000 iterations without improvement of the best visited solution.

Recombination operator The recombination operator consists in two phases. In the first phase, a set $\mathcal{E}(\mathcal{M})$ of *elite* solutions is generated from the solutions in $\mathcal{M}$ as follows. First, the p (parameter tuned to 3) best *different* solutions of $\mathcal{M}$ are put in $\mathcal{E}(\mathcal{M})$. Two solutions are considered equal if they have the same (32-bit) hash value (as defined below). If there are no p different solutions but only $p-q$, then q random solutions are created to complete $\mathcal{E}(\mathcal{M})$. In addition, if the very best solution $s^\star$ found so far is not included in $\mathcal{E}(\mathcal{M})$, its worst solution is replaced by $s^\star$. In the second phase, $|\mathcal{M}|$ new solutions are generated by *crossing* two randomly selected solutions of $\mathcal{E}(\mathcal{M})$. The crossing of two solutions is performed by assigning to each vertex the colors of the corresponding vertex in either the first or the second solution (random choice).

Intensification operator It is a procedure close to $Tabu(P)$, which will be applied on any offspring solution. The difference relies in the fact that a second way of managing the tabu tenure is also used, which works as follows. For each solution, we compute a 20-bits *hash* value. Then we forbid to visit solutions with the same hash value for τ_2 (parameter) iterations. The hash value is computed as follows: for each vertex j and each color t, a 32-bit random value $r_{j,t}$ is generated. The hash value of a solution is then computed by *xor-ing* (bit-wise addition in $\mathbb{F}_2$) the corresponding values as follows:

$$hash = \bigoplus_{j \text{ has color } t} r_{j,t} \quad \oplus \text{ denotes bit-wise xor} \tag{6}$$

This hash function is very easy to update after a move (two xor operations are in fact sufficient). The used 20-bit hash value is simply the lower 20 bits of the 32-bit hash value. Preliminary experiments showed on the one hand that $GLS(P)$ performs better when both tabu

Table 2 Parameters used for *Tabu*(P) within *GLS*(P)

Tabu scheme	Tenure τ	Vertex fraction N	Color fraction K
1	$\tau_1 = 10 + \frac{n}{2} + 20 \cdot u$	$0.1 + 0.2 \cdot u + \frac{5}{n}$	$0.4 + 0.2 \cdot u$
2	$\tau_2 = 2 \cdot n + 400 \cdot u$		

schemes are used as intensification mechanisms, and on the other hand that $\tau_2 \in [10, 800]$ was found to be suitable. Note however that with small values of τ_2, the algorithm often fails to find a solution with a finite objective function value (as some costs are set to ∞ in the used instances), but finds good solutions if non infinite values are reached. For large values of τ_2, the algorithm usually finds a solution with a finite objective function value, but the quality of those solutions however vary a lot, and are generally not as good as with the first tabu scheme (i.e. based on τ_1). For this reason, the τ_2 tabu scheme is only used as an intensification operator within *GLS*(P).

At each iteration, the used tabu scheme is randomly chosen with equal probability. The tabu tenures τ_1 and τ_2, as well as the parameters fixing the explored neighborhood fraction, namely N and K, are first randomly chosen in sensible intervals according to Table 2, where u denotes a random generated value in [0, 1]. Then, the tabu search parameters associated with an offspring solution are copied from the parameters of one of the parent solutions. With a certain probability (tuned to 20 %), the scheme and its associated parameters are again randomly chosen according to Table 2.

As we would like to perform a significant number of generations, we have to perform *Tabu*(P) for a short time during each generation. Such a strategy will balance the roles associated with the recombination operator and the intensification operator (*Tabu*(P)). Let i_t (parameter) be the number of iterations without improvement performed by *Tabu*(P) at each generation of *GLS*(P). Preliminary experiments showed that $i_t = 100{,}000$ is a reasonable choice, which corresponds on small instances to a fraction of a second, and up to several minutes on large instances.

Population update operator As $|\mathcal{M}|$ solutions are produced at each generation by the recombination operator, all the population is renewed (except for the very best solution, which is always part of the population).

5 Results

In this section, we first present the considered instances, and then discuss the obtained results of *Greedy*(P), *Descent*(P), *Tabu*(P), *GLS*(P), as well as the proposed exact methods.

5.1 Generation of instances

We use a file format which is inspired by the well-known DIMACS graph format. Three types of instances are generated: 32 *random* instances, 10 *trivial* instances (which are constructed such that they can be optimally solved by a greedy algorithm), and 10 *linear* instances (for which the incompatibility costs are linear with the number of conflicts). In addition, existing instances from the literature (see Zufferey et al. 2012) are also tackled, but only one color per vertex is allowed for such instances, which are divided into 14 large instances (with $n \geq 300$) and 90 small instances (with $n \leq 100$).

5.1.1 Random *instances*

The 32 *random* instances are generated giving the following information: the number n of vertices, the density d (probability that an edge exists between two arbitrary vertices), the minimal number $p^{\min}$ of colors per vertex, the maximal number $p^{\max}$ of colors per vertex, and the total number k of colors. The costs $a_{j,t}$ are chosen uniformly at random in the interval [0, 1]. The number of colors p_j for vertex j is chosen uniformly between $p^{\min}$ and $p^{\max}$ (inclusively). The penalties $c^m_{j,j'}$ are generated sequentially for $m = 1, 2, \ldots$ as follows: $c^0_{j,j'} = 0$, then $c^m_{j,j'}$ is set to ∞ with probability $\frac{m}{\hat{p}}$, where $\hat{p} = \min\{p_j, p_{j'}\}$. Otherwise $c^m_{j,j'}$ is set to $c^{m-1}_{j,j'}$ plus a real uniform random value in [1, 11] (which implies that if $c^{m-1}_{j,j'} = \infty$, then $c^m_{j,j'} = \infty$). For $m \geq \min\{p_j, p_{j'}\}$, all the $c^m_{j,j'}$ penalties are set to ∞. 32 random instances of different sizes and densities have been generated. They have been selected by briefly running $GLS(P)$ such that a solution having a finite objective function could be found, but not instantly.

5.1.2 Trivial *instances*

The 10 *trivial* instances are constructed such that the optimum is known. They are called trivial because they can be optimally solved in a greedy fashion. The following values are again given: $n, d, p^{\min}, p^{\max}, k$. The p_j's and the $a_{j,t}$'s are generated as in the random instances. Then, for each vertex j, the p_j "cheapest" colors are assigned (i.e. those for which $a_{j,t}$ is the smallest). Let d_j denote the difference between the last color assigned to j and the next cheapest, non-assigned color. Then edges are randomly assigned with the constraint that at most two conflicts per color are allowed. This process is repeated until either the desired density is obtained, or $5 \cdot n^2$ attempts to add an edge have been performed. Penalties are set to ∞, except on conflicting edges $[j, j']$. Let $\hat{m}$ be the number of conflicts for vertex j. Then the penalties are set to $\frac{m \cdot \min\{d_j, d_{j'}\}}{\hat{m}}$ for $m \leq \hat{m}$. This guarantees that the conflicts are cheaper than any other choice of colors. For larger m, the penalties are also set to ∞. As a consequence the assignment costs $a_{j,t}$ are considerably larger than the conflict costs $c^m_{j,j'}$ which become negligible.

5.1.3 Linear *instances*

The 10 *linear* instances have linear penalties, so they can be solved by linear programming more easily. Again are given $n, d, p^{\min}, p^{\max}, k$. The p_j's and the $a_{j,t}$'s are generated as in the *random* instances. Edges are generated randomly, up to the desired density. The penalty $c^1_{j,j'}$ is set to a uniform random number in [1, 11]. Then $c^m_{j,j'} = m \cdot c^1_{j,j'}$ for $m \geq 2$.

5.1.4 Names of the instances

The file-names starts with the name of the generating algorithm (e.g., *random*, *trivial* or *linear*), followed by n, d, k, $p^{\min}$, $p^{\max}$, and the random seed used to generate the instance. For example `random-n10-d50-k12-m2-M5-r2121` is a *random* instance with $n = 10$ vertices, a density d of 0.5, allowing a total of $k = 12$ different colors, with $p^{\min} = 2$ to $p^{\max} = 5$ colors per vertex. The random seed used to generate this instance was 2121.

5.1.5 *Existing instances from the literature*

We are not aware of existing instances for our multi-coloring problem. However, Zufferey et al. (2012) have described a simplified problem with the difference that only one color per vertex is allowed. Since their problem setting is just a special case of (P), we can apply our methods without any change to their instances (which can be converted in a straightforward manner into the format used for our experiments). There are 14 large (from 300 to 1000 vertices) instances well-known from plain graph coloring with the additional costs associated. These instances have the prefix *dsjc*, *flat* and *leighton*. There are another 90 generated smaller instances (from 10 to 100 vertices, to be colored with 2 to 10 colors). Their prefix is *generated*. They are in fact random graphs with density 0.5. Each cost $a_{j,t}$ is an integer randomly generated in $[0, 200]$, and each incompatibility cost $c_{j,j'}$ is an integer randomly generated in [0, 1000] (in a uniform way for both types of costs).

5.2 Test design

All tests presented here were executed on an Intel® Core™ i7-2620M CPU @ 2.70 GHz with 4 GB of RAM (DDR3). The processes are single threaded, so up to 4 tests could be conducted simultaneously on this 2-core 4-threads processor. The stopping criterion was the effectively consumed CPU-time reported by the system. For each instance, we report the very best objective function value $f^\star$ ever found by our algorithms, which includes all preliminary tests, final tests and other test (over 100,000 runs), and some of them allowing for more CPU-time than for the tests reported here. We indicate the results of our algorithms as the average over 10 runs of the percentage gap above $f^\star$.

5.2.1 *Exact methods*

We used CPLEX 12.4 on the same machine to attempt to solve the instances where possible. Otherwise we indicate the best lower bound found by either the exact model (indicated by † in the table) or the linear approximation (indicated by ‡). Note that the time limit allowed for the CPLEX solver will however be 4 hours (at 4 threads, so roughly 16 hours of CPU-time), and not T seconds, as it is the case for the solution methods. Note that for many instances, the CPLEX solver runs out of memory before the 4 hours are used. The solvable instances can be found in Tables 3 and 4 (marked by opt). Optimal $f^\star$-values are underlined. All *trivial* instances can be optimally solved by CPLEX very quickly (240 seconds for the largest instance trivial-n200-d50-k22-m2-M6-r643). With the exception of the random instance n20-d50-k15-m2-M5-r43 (solved to optimality by CPLEX), the upper bounds found by CPLEX before running out of memory or time were considerably higher than the bounds obtained by our heuristics. Note that with one exception, our proposed metaheuristics (i.e. *Tabu*(P) and *GLS*(P)) find an optimal solution where CPLEX does.

5.2.2 *Multi-coloring instances*

For the multi-coloring instances (namely *random*, *linear* and *trivial*), 10 runs were performed with each of the following methods: *Greedy*(P), *Descent*(P), *Tabu*(P) and *GLS*(P). Remind that the time limit T (in seconds) is 300 for $n \le 30$, 600 for $n = 50$, 1200 for $n = 100$, and 3600 for $n = 200$ (see Table 1 in Sect. 4.2).

Table 3 Results on the random instances

Instances *random*	$f^\star$	Lb / Ub	*Tabu*(P)	*GLS*(P)	*Descent*(P)
n10-d50-k12-m2-M5-r42	$\underline{41.5}$	opt	**0.0** %	**0.0** %	1.3 %
n10-d50-k12-m2-M5-r2121	$\underline{39.63}$	opt	**0.0** %	**0.0** %	0.8 %
n10-d50-k13-m2-M5-r2323	$\underline{20.12}$	opt	**0.0** %	0.1 %	2.4 %
n20-d20-k9-m2-M5-r44	$\underline{120.41}$	opt	1.8 %	**0.3** %	1.4 %
n20-d20-k12-m3-M5-r42	$\underline{59.04}$	opt	2.9 %	**0.2** %	4.7 %
n20-d50-k15-m2-M5-r43	59.31	$\underline{58.84}$ (opt)	1.3 %	**0.5** %	4.8 %
n20-d50-k15-m3-M5-r42	108.43	83.01‡ / 110.96	1.8 %	**0.9** %	7.8 %
n20-d80-k20-m2-M5-r42	312.27	118.61‡ / 322.30	1.4 %	**0.9** %	∞
n20-d80-k21-m3-M6-r42	410.25	130.1646‡ / 486.78	1.1 %	**0.7** %	3.1 %
n30-d20-k12-m3-M5-r4242	183.19	150.56† / 181.95	3.7 %	**2.1** %	19.0 %
n30-d20-k13-m1-M6-r123456	$\underline{51.09}$	opt	5.7 %	**0.9** %	5.0 %
n30-d50-k17-m1-M6-r21	249.5	127.37† / 260.75	1.1 %	**0.8** %	18.8 %
n30-d50-k17-m2-M5-r42	221.24	74.72† / 310.14	2.1 %	**1.0** %	20.6 %
n30-d80-k22-m2-M5-r314159	625.06	73.82† / 781.23	1.0 %	**0.8** %	13.9 %
n30-d80-k24-m2-M6-r55555	677.98	84.09‡ / 911.32	**0.5** %	0.6 %	13.3 %
n50-d20-k16-m1-M6-r1024	242.83	155.28† / 284.90	7.5 %	**2.7** %	43.4 %
n50-d20-k24-m3-M8-r77	133.52	94.52‡ / 161.92	17.1 %	**4.5** %	64.2 %
n50-d50-k16-m2-M4-r4321	860.54	70.79‡ / ∞	**2.0** %	3.0 %	∞
n50-d50-k20-m2-M6-r1024	1828.71	225.90‡ / ∞	**0.9** %	2.6 %	∞
n50-d80-k23-m1-M4-r123	608.38	39.51‡ / ∞	1.1 %	**1.0** %	∞
n50-d80-k28-m2-M6-r888	1674.88	92.22† / 2410.44	**0.8** %	1.5 %	∞
n100-d20-k25-m3-M6-r9876	406.81	133.22† / 1242.84	5.5 %	**4.8** %	116.3 %
n100-d20-k28-m2-M6-r42	162.47	97.87† / ∞	25.1 %	**11.8** %	127.8 %
n100-d50-k23-m2-M4-r54321	2264.0	44.97† / ∞	**2.7** %	4.6 %	∞
n100-d50-k33-m3-M7-r12345	5026.92	81.56† / ∞	**2.0** %	3.9 %	∞
n200-d20-k23-m1-M5-r6245	2108.69	106.53† / ∞	**3.7** %	5.4 %	∞
n200-d20-k31-m2-M7-r98761	3610.31	152.71† / ∞	4.8 %	**4.4** %	∞

Random *instances* Test results on the *random* instances are reported in Table 3. For each method, we report the percentage gap relative to $f^\star$ (where the best results are indicated in bold face). On instances up to 30 vertices, *GLS*(P) clearly outperforms *Tabu*(P). On larger instances, the two methods perform comparably. Note that *Greedy*(P) never manages to find a solution with a finite objective function value. *Descent*(P) finds solutions with a finite objective function value on most smaller instances, but is always outperformed by both *Tabu*(P) and *GLS*(P).

Linear *instances* Results on the *linear* instances are reported in Table 4. We have very similar results as with the *random* instances. Since there is no infinite incompatibility cost, every solution has a finite objective function value. $Greedy(P)$ performs very poorly, and so does *Descent*(P) on instances with 50 or more vertices.

Trivial *instances* Remind that the *trivial* instances are built such that they can be optimally solved by a greedy algorithm (thus, $f^\star$ denotes here the optimal value of f). The results are reported in Table 5. As expected, *Greedy*(P) solves all the instances by their very construc-

Table 4 Results on the linear instances

Instances *linear*	$f^\star$	Lb / Ub	*Tabu*(P)	*GLS*(P)	*Greedy*(P)	*Descent*(P)
n10-d50-k9-m2-M5-r42	<u>62.37</u>	opt	**0.0** %	**0.0** %	84.6 %	**0.0** %
n10-d80-k14-m2-M5-r43	<u>44.66</u>	opt	0.2 %	**0.0** %	181.4 %	3.9 %
n20-d50-k14-m2-M5-r113	81.99	67.68 / 82.49	1.4 %	**0.5** %	217.6 %	2.9 %
n20-d80-k17-m2-M5-r13	181.12	95.2 / 199.17	1.3 %	**0.7** %	138.9 %	2.4 %
n30-d50-k20-m2-M6-r422	146.02	68.77 / 171.37	**1.6** %	1.9 %	348.2 %	14.4 %
n30-d80-k30-m3-M6-r745	438.28	73.53 / 640.02	**0.6** %	1.0 %	166.3 %	8.8 %
n50-d20-k17-m2-M6-r5432	134.16	81.73 / 162.46	16.8 %	**3.5** %	531.9 %	29.3 %
n50-d50-k22-m1-M4-r123	85.94	38.73 / 183.59	**2.4** %	3.0 %	1075.2 %	43.2 %
n100-d20-k20-m1-M5-r53412	170.84	85.18 / 449.86	7.7 %	**7.5** %	1096.5 %	66.3 %
n200-d20-k25-m1-M5-r54321	883.17	100.9121 / ∞	**1.8** %	3.7 %	696.1 %	65.3 %
Average	222.85		3.38	2.18	453.67	23.65

Table 5 Results on the trivial instances

Instances *trivial*	$f^\star$	*Tabu*(P)	*GLS*(P)	*Greedy*(P)	*Descent*(P)
n10-d50-k10-m3-M7-r42	16.2	**0.0** %	**0.0** %	**0.0** %	**0.0** %
n10-d80-k12-m3-M7-r123	9.92	**0.0** %	**0.0** %	**0.0** %	**0.0** %
n20-d50-k15-m3-M7-r532	25.88	**0.0** %	**0.0** %	**0.0** %	**0.0** %
n30-d50-k15-m2-M5-r32	19.2	**0.0** %	**0.0** %	**0.0** %	**0.0** %
n30-d50-k20-m3-M8-r543	26.45	**0.0** %	**0.0** %	**0.0** %	**0.0** %
n50-d50-k20-m3-M8-r312	45.39	15.1 %	**0.0** %	**0.0** %	**0.0** %
n100-d20-k18-m2-M5-r532	50.3	1.8 %	**0.0** %	**0.0** %	**0.0** %
n100-d50-k25-m2-M6-r432	40.84	254.7 %	58.7 %	**0.0** %	**0.0** %
n200-d20-k18-m2-M6-r432	112.31	46.8 %	2.8 %	**0.0** %	**0.0** %
n200-d50-k22-m2-M6-r643	89.55	293.6 %	176.3 %	**0.0** %	**0.0** %
Average	43.6	61.2	23.78	0.0	0.0

tion. Surprisingly, *Descent*(P) solves all the instances as well. Not all the trivial instances are solved by *Tabu*(P) and *GLS*(P) . This is due to the fact that those methods start with a completely random solution. *GLS*(P) outperforms *Tabu*(P), probably because *GLS*(P) can be seen as a kind of tabu search with restarts (and with much more diversification potential). For those particular instances, *Tabu*(P) and *GLS*(P) will of course find optimal solutions if they start with at least a greedily constructed solution. However, especially for *GLS*(P), random starting points in the solution space are crucial ingredients for the method (particularly from a diversification standpoint).

5.2.3 Single color instances

Large instances For the *dsjc*, *flat* and *leighton* instances, we have set the time limit to 60 minutes (which corresponds to the one used in Zufferey et al. 2012). The parameters for *Tabu*(P) and *GLS*(P) have been set as described in Table 6. Results can be found in Table 7. We report the name of the instance, the number n of vertices, the density d, the

Table 6 Parameters used for the large single color instances

Instances	*Tabu*(P)	*GLS*(P)
dscj, flat, leighton	$(\tau_1, N, K) = (50, 0.2, 0.5)$	$i_t = 2 \cdot 10^5$, $\lvert\mathcal{M}\rvert = 6$, $\mathcal{E}(\mathcal{M}) = 3$
Generated	$(\tau, N, K) = (15, 0.5, 0.5)$	$i_t = 2 \cdot 10^5$, $\lvert\mathcal{M}\rvert = 6$, $\mathcal{E}(\mathcal{M}) = 3$

Table 7 Results on the large instances with a single color per vertex

Instance	n	d	k	$f^\star$	*Tabu*(P)	*GLS*(P)	*TabuDiv*
DSJC1000.1	1000	10	13	183051	**1.5** %	4.2 %	9.4 %
DSJC1000.5	1000	50	55	193379	1.8 %	**1.3** %	21.7 %
DSJC1000.9	1000	90	149	117118	7.0 %	**1.8** %	40.5 %
DSJC500.5	500	50	32	81257	**2.3** %	4.6 %	11.0 %
DSJC500.9	500	90	84	51873	**2.0** %	3.2 %	10.5 %
flat1000_50_0	1000	49	33	600245	**0.6** %	0.9 %	8.1 %
flat1000_60_0	1000	49	40	401674	**1.1** %	1.2 %	11.9 %
flat1000_76_0	1000	49	55	187837	2.2 %	**1.5** %	23.4 %
flat300_28_0	300	48	19	57372	6.6 %	**5.5** %	7.9 %
le450_15c	450	17	10	123022	**1.7** %	3.2 %	5.4 %
le450_15d	450	17	10	123131	**1.6** %	3.3 %	4.2 %
le450_25c	450	17	17	45927	13.5 %	**2.4** %	19.0 %
le450_25d	450	17	17	44902	14.7 %	**4.4** %	18.8 %
Average					**4.4** %	2.9 %	14.8 %

allowed number k of colors, $f^\star$ as usually defined, followed by the average result (over 10 runs) for *Tabu*(P) and *GLS*(P). Finally, *TabuDiv* indicates the average values (over 5 runs) obtained by the tabu search proposed in Zufferey et al. (2012). Both our methods clearly outperform *TabuDiv*. This is remarkable insofar that our methods were designed to solve multi-coloring problem but seems to handle single coloring instances just as well. Both *Tabu*(P) and *GLS*(P) perform about equally, except on the two leighton450_25 instances, where *GLS*(P) outperforms *Tabu*(P).

Small instances For such *generated* instances, the time limit is set to $\frac{n \cdot k}{2}$ seconds, which roughly corresponds to half the CPU-times reported in Zufferey et al. (2012). As an example, for the instance with $n = 30$ vertices to be colored with $k = 8$ colors, 120 seconds of CPU-time were used. We used a tabu tenure of $\tau_1 = 50$ and we limit the neighborhood exploration using $N = K = 0.5$. The average results (over 10 runs) can be found in Table 8. Note that we only report the cases where one of our two metaheuristics (namely *Tabu*(P) and *GLS*(P)) did not produce the same objective function value as the basic tabu search proposed in Zufferey et al. (2012), which we will name here *TabuBasic*. Results marked with † mean that the median of all runs is the same as the result of *TabuBasic* (i.e. the same value was obtained in at least 60 % of the runs). Results marked with ‡ mean that the same value as the result of *TabuBasic* was obtained at least once. **Bold** results are better than those obtained with *TabuBasic*. *Italic* results mark the best result among *Tabu*(P) and *GLS*(P). Results with a $\star$ have been shown to be optimal by means of a commercial MIP-solver. We can observe that for many of the larger generated instances, our methods found new best upper

Table 8 Results on the small instances with a single color per vertex

k		2	3	4	5	6	7	8	9	10
$n=50$	*Tabu*(P)					‡7282	†5005	‡3734	‡2774	‡2184
	GLS(P)					†*7268*	†*5004*	†*3689*	†*2738*	†*2148*
	$f^\star$					7268	5004	3680	2738	2148
	TabuBasic					7268	5004	3680	2738	2148
$n=60$	*Tabu*(P)			†26729		†10952	†7294	5754	‡4181	‡3273
	GLS(P)			†*26723*		†*10949*	†*7289*	‡*5672*	†*4083*	†**3233**
	$f^\star$			26723		10944	7273	5635	4083	**3194**
	TabuBasic			26723		10944	7273	5635	4083	3249
$n=70$	*Tabu*(P)	†121075		†*40015*		‡17579	‡*12325*	‡8886	‡6479	5042
	GLS(P)	†*121072*		†40017		‡*17577*	‡12337	†*8794*	**6416**	‡*4942*
	$f^\star$	121072		40015		17552	12304	8794	**6405**	4860
	TabuBasic	121072		40015		17552	12304	8794	6428	4860
$n=80$	*Tabu*(P)	†157895	†*85365*		†*34294*	‡23711	**16668**	11675	‡8317	6851
	GLS(P)	†*157858*	†85387		†34365	†*23665*	**16677**	**11555**	†*8273*	*6714*
	$f^\star$	157858	85365		34294	23597	**16597**	**11546**	8273	**6648**
	TabuBasic	157858	85365		34294	23597	16692	11595	8273	6693
$n=90$	*Tabu*(P)		†*107039*	†*67018*	†*44092*	†*29635*	†*20712*	15481	**11771**	8610
	GLS(P)		†107077	‡67131	†44175	†29695	‡20911	*15436*	11844	**8597**
	$f^\star$		107039	67012	44092	29635	20712	**15325**	**11522**	**8533**
	TabuBasic		107039	67012	44092	29635	20712	15374	11816	8599
$n=100$	*Tabu*(P)	†256048	†*142070*	†*90039*	†*60588*	**42158**	**30494**	**22494**	*16612*	**12423**
	GLS(P)	†*256040*	†142300	†90157	‡60842	42594	30690	**22491**	16755	12459
	$f^\star$	256040	142070	90039	60588	**42072**	**30428**	**22312**	**16467**	**12235**
	TabuBasic	256040	142070	90039	60588	42417	30544	22554	16552	12423

bounds and outperformed *TabuBasic*. On almost all other instances, our methods confirmed in at least half of all runs the best upper bounds. On small instances, *GLS*(P) obtained better results than *Tabu*(P). However, on larger instances, the situation is mostly reversed.

6 Conclusion

In this paper, we consider a job scheduling problem (P) where for each job, we have to assign its requested number of time periods, and preemptions are allowed. Assignment and incompatibility costs have to be minimized over a given time horizon. This problem is difficult and is strongly connected with the multi-coloring problem. Two exact approaches are first proposed: one designed for linear incompatibility costs, and one for the general case. The exact approaches are mostly limited to 20 jobs, which indicates that (meta)heuristics are necessary for larger instances. Various solutions methods are thus proposed to tackle (P): a greedy heuristic, a descent method, a tabu search and a genetic local search. The obtained results showed that the genetic local search is the best method on the smaller instances (up to 50 jobs), and tabu search is the best on the larger instances (from 100 jobs). In addition, these two methods outperform the ones proposed in Zufferey et al. (2012), where a single time period has to be assigned to each job. Among the possible extensions, one can mention the consideration of precedence constraints and other types of costs. There are also many avenues of research in job scheduling under uncertainty.

References

Aardal, K. I., van Hoesel, S. P. M., Koster, A. M. C. A., Mannino, C., & Sassano, A. (2003). Models and solution techniques for frequency assignment problems. *4OR*, *1*(4), 261–317.

Beck, J. C., & Refalo, P. (2003). A hybrid approach to scheduling with earliness and tardiness costs. *Annals of Operations Research*, *118*, 49–71.

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

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

Burke, E. K., Marecek, J., Parkes, A. J., & Rudova, H. (2010). A supernodal formulation of vertex colouring with applications in course timetabling. *Annals of Operations Research*, *179*, 105–130.

Chen, Z. L. (2004). Simultaneous job scheduling and resource allocation on parallel machines. *Annals of Operations Research*, *129*, 135–153.

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

Coffman, E. G., Garey, M. R., Johnson, D. S., & Lapaugh, A. S. (1985). Scheduling file transfers. *SIAM Journal on Computing*, *14*(4), 743–780.

Demeulemeester, E. L., & Herroelen, W. S. (2002). *Project scheduling: a research handbook*. Dordrecht: Kluwer Academic.

Dorne, R., & Hao, J.-K. (1998). Tabu search for graph coloring, T-colorings and set T-colorings. In *Metaheuristics: advances and trends in local search paradigms for optimization* (pp. 77–92). Norwell: Kluwer Academic.

Dosa, G., & He, Y. (2006). Scheduling with machine cost and rejection. *Journal of Combinatorial Optimization*, *12*, 337–350.

Duives, J., Lodi, A., & Malaguti, E. (2011). Test-assignment: a quadratic coloring problem. *Journal of Heuristics*, *1*, 1–16.

Feng, G., & Lau, H. C. (2008). Efficient algorithms for machine scheduling problems with earliness and tardiness penalties. *Annals of Operations Research*, *159*, 83–95.

Galinier, P., & Hao, J. K. (1999). Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, *3*(4), 379–397.

Gamst, A., & Rave, W. (1982). On the frequency assignment in mobile automatic telephone systems. In *Proceedings of GLOBECOM IEEE*, 29 November–2 December (pp. 309–315).

Gandhi, R., Halldorsson, M. M., Kortsarz, G., & Shachnai, H. (2005). Improved bounds for sum multicoloring and scheduling dependent jobs with minsum criteria. In *Lecture notes in computer science* (Vol. 3351, pp. 68–82).

Garey, M., & Johnson, D. S. (1979). *Computer and intractability: a guide to the theory of NP-completeness*. San Francisco: Freeman.

Gendreau, M., & Potvin, J.-Y. (2010). *International series in operations research & management science: Vol. 146. Handbook of metaheuristics*. Berlin: Springer.

Glover, F., & Laguna, M. (1997). *Tabu search*. Boston: Kluwer Academic.

Halldorsson, M. M. (2004). Multicoloring: problems and techniques. In *Lecture notes in computer science* (Vol. 3153).

Hertz, A., & de Werra, D. (1987). Using tabu search techniques for graph coloring. *Computing*, *39*, 345–351.

Hertz, A., Plumettaz, M., & Zufferey, N. (2008). Variable space search for graph coloring. *Discrete Applied Mathematics*, *156*, 2551–2560.

Icmeli, O., Erenguc, S. S., & Zappe, C. J. (1993). Project scheduling problems: a survey. *International Journal of Operations & Production Management*, *13*(11), 80–91.

Kerzner, H. (2003). *Project management: a systems approach to planning, scheduling, and controlling*. New York: Wiley.

Kolisch, R., & Padman, R. (2001). An integrated survey of deterministic project scheduling. *Omega*, *29*(3), 249–272.

Lancaster, J., & Ozbayrak, M. (2007). Evolutionary algorithms applied to project scheduling problems—a survey of the state-of-the-art. *International Journal of Production Research*, *45*(2), 425–450.

Leighton, F. T. (1979). A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, *84*, 489–505.

Liu, Z. & Cheng, T. C. E., & (2002). Scheduling with job release dates, delivery times and preemption penalties. *Information Processing Letters*, *82*(2), 107–111.

Lu, Z., & Hao, J.-K. (2010). A memetic algorithm for graph coloring. *European Journal of Operational Research*, *203*, 241–250.

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

Malaguti, E., Monaci, M., & Toth, P. (2008). A metaheuristic approach for the vertex coloring problem. *INFORMS Journal on Computing*, *20*(2), 302–316.

Mendes, A. S., Muller, F. M., França, P. M., & Moscato, P. (2002). Comparing meta-heuristic approaches for parallel machine scheduling problems. *Production Planning & Control*, *13*(2), 143–154.

Mohammadi, E., & Heydari, M. (2011). Single machine scheduling problem with minimax criteria and preemption penalties. *Computer Science and Automation Engineering (CSAE)* 440–444. doi:10.1109/CSAE.2011.5953257

Osman, I. H., & Laporte, G. (1996). Metaheuristics: a bibliography. *Annals of Operations Research*, *63*, 513–623.

Panwalkar, S. S., & Liman, S. D. (2002). Single operation earliness-tardiness scheduling with machine activation costs. *IIE Transactions*, *34*, 509–513.

Pinedo, M. (2008). *Scheduling: theory, algorithms, and systems multi-coloring*. New York: Prentice Hall.

Plumettaz, M., Schindl, D., & Zufferey, N. (2010). Ant local search and its efficient adaptation to graph colouring. *Journal of the Operational Research Society*, *61*, 819–826.

Rochat, Y., & Taillard, E. (1995). Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, *1*, 147–167.

Rogers, P., & Nandi, A. (2007). Judicious order acceptance and order release in make-to-order manufacturing systems. *Production Planning & Control*, *18*(7), 610–625.

Schuurman, P., & Woeginger, G. J. (1999). Preemptive scheduling with job-dependent setup times. In *Proceedings of the 10th annual acm-siam symposium on discrete algorithms soda'99* (pp. 759–767).

Shabtay, D. (2008). Due date assignments and scheduling a single machine with a general earliness/tardiness cost function. *Computers & Operations Research*, *35*, 1539–1545.

Shachnai, H., Tami, T., & Woeginger, G. (2002). Minimizing makespan and preemption costs on a system of uniform machines. In R. Mring & R. Raman (Eds.), *Lecture notes in computer science: Vol. 2461. Algorithms esa 2002* (pp. 203–212). Berlin: Springer.

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

Stecke, K. (1985). Design planning, scheduling and control problems of flexible manufacturing. *Annals of Operations Research*, *3*, 3–12.

Sun, H., Wang, G., & Chu, C. (2005). Preemptive scheduling with availability constraints to minimize total weighted completion times. *Annals of Operations Research*, *133*, 183–192.

T'Kindt, V., & Billaut, J.-C. (2001). Multicriteria scheduling problem: a survey. *RAIRO, Operations Research*, *35*, 143–163.

Toksari, M. D. & Guner, E. (2010). Parallel machine scheduling problem to minimize the earliness/tardiness costs with learning effect and deteriorating jobs. *Journal of Intelligent Manufacturing*, *21*, 843–851.

Yang, B., & Geunes, J. (2007). A single resource scheduling problem with job-selection flexibility, tardiness costs and controllable processing times. *Computers & Industrial Engineering*, *53*, 420–432.

Yang, B., Geunes, J., & O'Brien, W. J. (2004). A heuristic approach for minimizing weighted tardiness and overtime costs in single resource scheduling. *Computers & Operations Research*, *31*, 1273–1301.

Zorzini, M., Corti, D., & Pozzetti, A. (2008). Due date (DD) quotation and capacity planning in make-to-order companies: results from an empirical analysis. *International Journal of Production Economics*, *112*, 919–933.

Zufferey, N. (2012). Metaheuristics: some principles for an efficient design. *Computer Technology and Applications*, *3*(6), 446–462.

Zufferey, N., Amstutz, P., & Giaccari, P. (2008). Graph colouring approaches for a satellite range scheduling problem. *Journal of Scheduling*, *11*(4), 263–277.

Zufferey, N., Labarthe, O., & Schindl, D. (2012). Heuristics for a project management problem with incompatibility and assignment costs. *Computational Optimization and Applications*, *51*, 1231–1252.