Comput Optim Appl (2012) 51:1231–1252
DOI 10.1007/s10589-010-9380-0

Heuristics for a project management problem with incompatibility and assignment costs

Nicolas Zufferey · Olivier Labarthe · David Schindl

Received: 13 January 2009 / Published online: 2 December 2010

Abstract Consider a project which consists of a set of jobs to be performed, assuming each job has a duration of at most one time period. We assume that the project manager provides a set of possible durations (in time periods) for the whole project. When a job is assigned to a specific time period, an assignment cost is encountered. In addition, for some pairs of jobs, an incompatibility cost is encountered if they are performed at the same time period. Both types of cost depend on the duration of the whole project, which also has to be determined. The goal is to assign a time period to each job while minimizing the costs. We propose a tabu search heuristic, as well as an adaptive memory algorithm, and compare them with other heuristics on large instances, and with an exact method on small instances. Variations of the problems are also discussed

Keywords Project management · Scheduling · Tabu search heuristic · Adaptive memory algorithm

N. Zufferey (✉)
Faculty of Economics and Social Sciences, HEC—University of Geneva, Uni-Mail, 1211 Geneva 4, Switzerland
e-mail: nicolas.zufferey-hec@unige.ch

O. Labarthe
Faculté des Sciences de l'Administration, Université Laval, Québec, QC, G1K 7P4, Canada
e-mail: olivier.labarthe@fsa.ulaval.ca

D. Schindl
Geneva School of Business Administration (HEG), Geneva, Switzerland
e-mail: david.schindl@hesge.ch

1 Introduction

In this paper, we consider a specific management problem (P) where a set of jobs have to be performed within k time periods, where different values for k are given by the project manager. Each job has a duration of at most one time period. Two types of costs are considered: assignment costs and incompatibility costs. When a job is assigned to a specific time period, an assignment cost is encountered. In addition, for some pairs of jobs, an incompatibility cost is encountered if they are performed at the same time period. Both costs depend on the duration k of the whole project. The goal is to assign a time period to each job while minimizing the total costs. The determination of the duration k of the whole project is also a key issue. Such a problem is new and there exists no literature on it. We can remark that it can be viewed as a project management problem as well as a scheduling problem. The reader interested in a general project management book with applications to planning and scheduling is referred to [23]. The reader desiring a review on scheduling models and algorithms is referred to [31]. Finally, the reader interested in project scheduling is referred to [5, 21, 25], and [26].

The paper is organized as follows. We formally express the considered management problem (P) in Sect. 2, we show that it is NP-hard and discuss some variations of the problem. In Sect. 3, we describe the tabu search method and its adaptation to problem (P). In Sect. 4, we describe the adaptive memory algorithm and its adaptation to problem (P). Results are reported in Sect. 5, where we compare the proposed methods with other heuristics as well as with an exact method. We end up the paper with a conclusion and possible extensions in Sect. 6.

2 Presentation of the problem

In this section, we formulate and illustrate the considered management problem (P), for which two types of costs are considered, namely assignment costs and incompatibility costs. Then, based on the similarities with the k-coloring problem, we show its NP-completeness, and thus the relevance of the use of heuristics for large instances. Finally, we end the section with the presentation of some possible variations of the problem, and with a discussion on the practical relevance of the proposed models.

2.1 Formal description of the problem

Consider a project which consists of a set V of n jobs to be performed. We consider the situation where the project manager does not only provide a single target number $\hat{k}$ of time periods within which the project has to be performed, but he provides a set $K = \{\hat{k} - r, \ldots, \hat{k}, \ldots, \hat{k} + r'\}$, where $\hat{k} - r$ is the smallest number of periods to consider, and $\hat{k} + r'$ is the largest. We can imagine that it is unfeasible to realize the project in less than $\hat{k} - r$ periods, and that the penalty cost is too large if the project has a duration larger than $\hat{k} + r'$. We assume that: (a) each time period has the same duration, typically a working day; (b) there is no precedence constraint between jobs; (c) all the jobs can have different durations; (d) each job has a duration of at most one

time period. Let $c^{(k)}(j;\, j') \geq 0$ denote an *incompatibility* cost between jobs j and j', which is to be paid if both jobs are performed at the same time period, where $k \in K$ is the duration of the whole project (in time periods). Further, for each job j, let $N(j)$ denote the set of jobs j' such that $c(j;\, j') > 0$. We assume that $c(j;\, j') = c(j';\, j)$ for all $j, j' \in V$ and all $k \in K$. Hence, $j \in N(j')$ implies $j' \in N(j)$ for all $j, j' \in V$. The incompatibility costs $c^{(k)}(j;\, j')$ represent for example that the same staff has to perform jobs j and j', thus additional human resources must be hired in order to be able to perform both jobs at the same period. In addition, for each duration k, each job j and each time period t, we know the cost $a^{(k)}(j;\, t)$ to pay if we perform job j at time period t if the duration of the project is k. Such a cost is referred to as an *assignment* cost. The assignment cost $a^{(k)}(j;\, t)$ represents for example the cost of the staff and machines which have to perform job j at period t if the duration of the project is k time periods. The goal is to assign each job j to a time period t while minimizing the total costs.

A solution using k periods can be generated by the use of a function $per : V \longrightarrow \{1, \ldots, k\}$. The value $per(j)$ of a job j is the time period assigned to job j. In order to represent a solution $s^{(k)}$ using k time periods, with each time period $t \in \{1, \ldots, k\}$, we associate a set C_t that contains the set of jobs which are performed at time period t. Thus, a solution $s^{(k)}$ can be denoted $s^{(k)} = (C_1; \ldots; C_k)$, and the associated encountered costs are:

$$f^{(k)}(s^{(k)}) = \sum_{t=1}^{k} \sum_{j \in C_t} a^{(k)}(j;\, t) + \sum_{j=1}^{n-1} \sum_{j' \in \{j+1;\ldots;n\} \cap C_{per(j)}} c^{(k)}(j;\, j') \qquad (1)$$

Problem $(P^{(k)})$ consists in finding a solution with k time periods which minimizes the above costs. An associated optimal solution is $s^{(k)}_{opt} = \arg\min_{s^{(k)}} f^{(k)}(s^{(k)})$. Problem (P) consists in finding a solution minimizing the costs when allowing any duration $k \in K$. Formally, it consists in finding a solution $s_{opt} = s^{(k^*)}_{opt}$, where $k^* = \arg\min_k f^{(k)}(s^{(k)}_{opt})$. Note that, based also on non quantitative considerations, the project manager can choose the most appropriate solution among the $|K|$ proposed ones, namely among solutions in $\{s^{(\hat{k}-r)}_{opt}, \ldots, s^{(\hat{k}+r')}_{opt}\}$, which is not necessarily solution s_{opt}.

To illustrate the above model $(P^{(k)})$, we propose the following example. A time period is one working day (typically 8 hours) and the planning horizon is $k = 2$ days. We have to perform jobs j_1, j_2, j_3 and j_4 with respective durations 0.75, 1, 0.25 and 0.50 working days. Note that the duration of a single job can not exceed one working day (assumption of the model). The assignment costs are: $a^{(2)}(j_1;\, 1) = 1$, $a^{(2)}(j_1;\, 2) = 2$, $a^{(2)}(j_2;\, 1) = 1$, $a^{(2)}(j_2;\, 2) = 2$, $a^{(2)}(j_3;\, 1) = 3$, $a^{(2)}(j_3;\, 2) = 1$, $a^{(2)}(j_4;\, 1) = 3$, $a^{(2)}(j_4;\, 2) = 1$. In addition, the incompatibility costs which are different from 0 are: $c^{(2)}(j_1;\, j_2) = 10$, $c^{(2)}(j_3;\, j_4) = 10$. According to the assignment costs, it seems better to perform jobs j_1 and j_2 in time period 1, and jobs j_3 and j_4 in time period 2. But according to the incompatibility costs, it seems better to perform jobs j_1 and j_2 in different time periods, and the same holds for jobs j_3 and j_4. The optimal solution can be easily found by hand: we should assign jobs j_1 and j_3 to time period 1, and jobs j_2 and j_4 to time period 2.

2.2 Difficulty of the problem

Given a graph $G = (V, E)$ with vertex set V and edge set E, the k-coloring problem (k-GCP) consists to assign an integer (called color) in $\{1, \ldots, k\}$ to every vertex such that two adjacent vertices have different colors. The graph coloring problem (GCP) consists in finding a k-coloring with the smallest possible value of k. Both problems are NP-hard [14] and many heuristics were proposed to solve them. For a recent survey, the reader is referred to [11].

We can now obviously remark the similarities between $(P^{(k)})$ and the k-GCP. From the input data of problem $(P^{(k)})$, we can build a graph $G = (V; E)$ as follows. We associate a vertex j with each job j, an edge $[j; j']$ each time $j' \in N(j)$ (but not more than one edge between two vertices). In addition, we can associate a color t with each time period t. Coloring G with k colors while trying to minimize the number of conflicting edges is equivalent to assign a time period $t \in \{1, \ldots, k\}$ to each job while trying to minimize the number of incompatibilities. $(P^{(k)})$ is actually an extension of the k-GCP, because the latter is a subcase of the former where $a^{(k)}(j; t) = 0$ for all j and all t, and $c^{(k)}(j; j') = 1$ for all j and all $j' \in N(j)$.

Therefore, we can deduce that problem $(P^{(k)})$ is NP-hard too, and the use of heuristic algorithms is unavoidable to tackle large instances of $(P^{(k)})$ and (P). Two main classes of heuristics are local search methods and population based algorithms. The most popular local search methods are simulated annealing [24], tabu search [16], variable neighborhood search [30], guided local search [35], threshold algorithms [3], and GRASP [8], while the most used population based methods are genetic algorithms [4], ant colonies [6], and adaptive memory algorithms [33].

Because of the similarities of problems $(P^{(k)})$ and k-GCP, an important point is to check if it is possible to select the best ingredients from the graph coloring methods and to adapt them for problem $(P^{(k)})$. Currently, no known exact solution method is able to solve all instances with up to 100 vertices (e.g. [29] and [17]). For larger instances, upper bounds on the chromatic number can be obtained by using heuristic algorithms. State-of-the-art local search heuristics for the k-GCP are the following: tabu search [1, 19], simulated annealing [2, 22], and variable space search [20]. Evolutionary heuristics were also successfully applied: genetic algorithms (e.g. [9, 10]), ant local search [32], and adaptive memory algorithms [12, 28].

All these heuristics are based on two strategies. The first approach consists in allowing conflicts (a conflict occurs if two adjacent vertices have the same color) while minimizing the number of conflicts. In such a context, a straightforward move is to change the color of a conflicting vertex, as proposed in [19]. Instead of relaxing the constraint that the endpoints of an edge should have different colors, one may relax the constraint imposing that all vertices should be colored. Thus the goal is to minimize the number of uncolored vertices. In such a situation, where partial but legal solutions are involved, a straightforward move consists in assigning a color to an uncolored vertex and to remove the color of the created conflicting vertices, as proposed in [1].

As it is difficult to define a partial solution for problem $(P^{(k)})$, it seems more appropriate to only consider the first above mentioned approach for further investigations. In this paper, one could consider the proposed tabu search for problem $(P^{(k)})$

as an extension of the quick and efficient tabu search coloring method proposed in [19], and could consider the adaptive memory algorithm for problem $(P^{(k)})$ as an extension of the very efficient adaptive memory algorithm for the k-GCP proposed in [12].

2.3 Variations of the problem

The above model can also cover the following specific situations. Let M be an arbitrarily large number. If the project manager does not want job j to be performed at time period t, he can set, for all $k \in K$, $a^{(k)}(j,t) = M$, and then, hopefully, the solution procedure will not assign j to time period t. More generally, it means that the proposed model can cover the situation where some forbidden time periods are associated with some jobs.

The model can also be adapted to cover the situation where the project manager prefers that job j is performed before job j'. Let k be the considered number of time periods. We create k artificial jobs denoted $j_1, j_2, \ldots, j_k$ associated with job j, as well as artificial jobs $j'_1, j'_2, \ldots, j'_k$ associated with j'. We also create an additional and artificial time period denoted k_0. All new assignment and incompatibility costs should be set to 0, except the following:

(A) $a^{(k)}(j_i,t) = a^{(k)}(j'_i,t) = M$ for $i \neq t$;
(B) $c^{(k)}(j_p; j_q) = c^{(k)}(j'_p; j'_q) = M$ if $p \neq q$;
(C) $a^{(k)}(l,k_0) = M$ for each non artificial job l;
(D) $c^{(k)}(j; j_i) = c^{(k)}(j'; j'_i) = M$ for $i \in \{1,\ldots,k\}$;
(E) $c^{(k)}(j_p; j'_q) = M$ if $p > q$.

With such a set of constraints, the solutions such that j is performed after j' will strongly be penalized. In order to better understand it, let us consider the above constraints step by step. Constraints (A), considered alone, favor solutions such that j_i and j'_i are assigned to time period i, for every $i \in \{1,\ldots,k\}$. The impact of constraints (B) is the following: at most one of the j_i's (say j_c) and at most one of the j'_i's (say j_d) should be assigned to time period k_0. Assigning more than one of the j_i's to k_0 would be strongly penalized with constraints (B), and the same holds for the j'_i's. Adding constraints (C) and (D) imply that jobs j and j' are assigned to periods different from k_0 (say c and d respectively), and that the corresponding jobs j_c and j'_d are both assigned to k_0. Finally constraints (E) avoid c to be larger than d, hence avoid job j to be performed after job j'. If we want job j to be performed strictly before job j', we only need to set $c^{(k)}(j,j') = M$.

The above technique can of course be generalized if we have to respect several precedence constraints. Even if each precedence constraint induces the creation of $2k$ artificial jobs, we believe that the model is still relevant in a practical standpoint, because the heuristics proposed in this paper are able to tackle instances up to 10,000 vertices in a reasonable amount of time (this was confirmed by preliminary experiments which will not be described here).

2.4 Practical relevance of the problem

As the GCP and the k-GCP can be viewed as roots of problems (P) and $(P^{(k)})$, all the application domains of the former problems might be relevant for the latter problems.

Relevant practical applications for graph coloring include the creation of timetables, frequency assignment, scheduling, design and operation of flexible manufacturing systems, bag rationalization for food manufacturers (e.g. [13, 15, 27, 34, 36]).

In addition, the problem considered in this paper is relevant when we have to assign jobs to time periods, which are typically working days. At the end of such a planning phase, for every day, we know the set of jobs that have to be performed during that day. In a second phase, given a day d and its associated jobs, one can imagine that we have to assign a resource (a human being or a machine) to each job. We can also consider the situation where a manager provides a set $K = \{\hat{k} - r, \ldots, \hat{k}, \ldots, \hat{k} + r'\}$ of possible resources available at day d. When k resources are available at day d, an incompatibility cost $c^{(k)}(j; j')$ between jobs j and j' has to be paid if both jobs are performed by the same resource, because of additional time that the concerned resource has to work. In addition, for each number k of resources, each job j and each resource t, we know the assignment cost $a^{(k)}(j; t)$ to pay if resource t performs job j, if k resources are available during that day. The goal is to assign each job j to a resource t while minimizing the total costs. Such a problem is equivalent to problem (P).

In contrast, another possible type of application associated with problem $(P^{(k)})$ is the following. Consider a set of technicians, with different profiles (i.e. qualification levels and capabilities), divided into k different working teams of various sizes. The planning horizon is one working day (typically 8 hours). A job represents a request of a client, with which is associated a set of required qualifications and a duration. The goal is to dispatch a set of n jobs to the k working teams while considering the qualifications of the working teams. The duration of each job is supposed to be much smaller than a working day. For any pair of jobs j and j', the incompatibility cost $c^{(k)}(j; j')$ is proportional to the difficulty encountered by the same team if it performs both j and j' within the same day. If it is impossible for the same team to perform j and j' within the same day, $c^{(k)}(j; j')$ is set to an arbitrarily large number M. In addition, for any job j and any working team t, the assignment cost $a^{(k)}(j; t)$ associated with the assignment of job j to working team t is proportional to the qualification level of team t to perform job j. If a team t is not qualified for a job j, $a^{(k)}(j; t)$ is set to an arbitrarily large number M. For a given solution of such a problem, if the manager feels that the amount of jobs associated with a team t is too large, he can create an additional team and tackle problem $(P^{(k+1)})$. On the contrary, if the jobs associated with a team t are negligible, the manager can remove such a team (and dispatch its members in other teams) and consider problem $(P^{(k-1)})$.

3 Tabu search for $(P^{(k)})$ and (P)

A basic version of tabu search can be described as follows. Let f be an objective function which has to be minimized over the solution space S. 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 $A(s)$. First, tabu search needs an initial solution $s_0 \in S$ as input. Then, the

algorithm generates a sequence of solutions $s_1, s_2, \ldots$ in the search space S such that $s_{r+1} \in A(s_r)$. When a move is performed from s_r to s_{r+1}, the inverse of that move is stored in a *tabu list* L. For the following t iterations, where t is the *tabu tenure* (also called *tabu list length*), a move stays *tabu* and cannot be used (with some exceptions) to generate a neighbor solution. The solution s_{r+1} is computed as $s_{r+1} = \arg\min_{s \in A'(s_r)} f(s)$, where $A'(s)$ is a subset of $A(s)$ containing all solutions s' which can be obtained from s either by performing a move that is not in L (i.e. not tabu) or such that $f(s') < f(s^*)$, where s^* is the best solution encountered along the search so far. The process is stopped for example when an optimal solution is found (when it is known), or when a fixed number of iterations have been performed. Many variants and extensions of this basic algorithm can be found for example in [16].

Below, we first propose a tabu search Tabu-$(P^{(k)})$ for problem $(P^{(k)})$ and then a general heuristic for problem (P). In order to design Tabu-$(P^{(k)})$, we mainly have to define the search space, the neighborhood structure (i.e. the nature of a move), and the way to manage the tabu tenures.

Search space The search space is the set of k-partitions of V and the objective function to minimize is the total cost $f^{(k)}$.

Neighborhood structure A move consists in changing the period assigned to a job. But in order to avoid testing every possible move at each iteration, we propose the following. For each job j, its contribution $cost^{(k)}(j)$ to $f^{(k)}$ is:

$$cost^{(k)}(j) = a^{(k)}(j; per(j)) + \frac{1}{2} \sum_{j' \in N(j) \cap C_{per(j)}} c^{(k)}(j; j') \tag{2}$$

Note that the fraction $\frac{1}{2}$ is used to consider that jobs j and j' equivalently contribute to $c^{(k)}(j; j')$. Thus, the other part of $c^{(k)}(j; j')$ is taken into account in $cost^{(k)}(j')$. At each iteration, we propose to only consider to change the period of a job j if $cost^{(k)}(j)$ is in the $q\%$ more costly jobs, where q is a parameter. Preliminary experiments showed that $q = 40\%$ is a reasonable value.

Tabu tenures When the period of a job j is modified from t to t', period t is declared tabu for job j for a certain number of iterations, and all solutions where j has period t are tabu solutions. At each iteration, we determine the best neighbor s' of the current solution s (ties are broken randomly) such that either s' is a non-tabu solution, or $f^{(k)}(s') < f^*$, where f^* is the value of the best solution s^* encountered so far during the search. If job j moves from C_t to $C_{t'}$ when going from the current solution s to the neighbor solution s', it is forbidden to put j back in C_t during $tab(j; C_t)$ iterations, where

$$tab(j; C_t) = \max \left\{ 1; \text{UNIFORM}(u; v) + \alpha \cdot \frac{f^{(k)}(s) - f^{(k)}(s')}{f^{(k)}(s)} \right\} \tag{3}$$

where u, v and α are parameters such that $0 < u < v$ and $\alpha > 0$. The maximum is used to enforce $tab(j; C_t)$ to be positive. The last term of (3) represents the improvement of the objective function from s to s'. If s' is better than s, the improvement

is positive and the reverse of the performed move will be forbidden for a larger time than if the improvement is negative, which is straightforward. Preliminary experiments showed that $(u; v; \alpha) = (10; 20; 15)$ is a reasonable parameter setting.

Incremental computation To save CPU time at each iteration, a key issue is the use of incremental computation. Suppose that we move j from C_t to $C_{t'}$ in order to generate a neighbor solution s' from the current solution s. Then, instead of computing $f^{(k)}(s')$ from scratch by the use of (1), we propose to only compute the variation $\Delta f^{(k)}(s; s') = f^{(k)}(s') - f^{(k)}(s)$ of $f^{(k)}$. It is easy to see that:

$$\Delta f^{(k)}(s; s') = a^{(k)}(j; t') + \sum_{j' \in N(j) \cap C_{t'}} c^{(k)}(j; j') - a^{(k)}(j; t) - \sum_{j' \in N(j) \cap C_t} c^{(k)}(j; j') \tag{4}$$

In other words, $\Delta f^{(k)}(s; s')$ is the difference between the costs induced by the move and the costs removed by the move. Thus, at each iteration, the performed move s' is the one minimizing $\Delta f^{(k)}$, which is computed for the non tabu moves and for the tabu moves leading to an improvement of f^*.

We have now all the ingredients to formulate Tabu-($P^{(k)}$) in Algorithm 1, where T^{max} is a parameter defining the stopping condition of the method, which is a maximum CPU time. Note that in order to generate an initial solution, we randomly assign a time period to each job.

Algorithm 1 Tabu-($P^{(k)}$)

Input: set of jobs, assignment and incompatibility costs, T^{max};

Initialization

1. generate an initial solution s (randomly);
2. set $s^* = s$ and $f^* = f^{(k)}(s)$;
3. set $Iter = 0$;

While T^{max} is not reached, **do**:

1. update the iteration counter: set $Iter = Iter + 1$;
2. compute the set C containing the $q\%$ most costly jobs;
3. generate the set B of all non tabu candidate neighbor solutions obtained from s by modifying the current time period associated with a job $j \in C$ (exception: B can contain tabu solutions if such solutions have values smaller than f^*);
4. set $s' = \arg\min_{s'' \in B} f^{(k)}(s'')$;
   assume we generate s' from s by moving job j from C_t to $C_{t'}$;
5. update the best solution: if $f^{(k)}(s') < f^*$, set $f^* = f^{(k)}(s')$ and $s^* = s'$;
6. update the tabu status: do not put j in C_t until iteration $Iter + tab(j; C_t)$;
7. update the current solution: set $s = s'$;

Output: solution s^* with value f^*;

Using Tabu-($P^{(k)}$) proposed in Algorithm 1, we propose to tackle problem (P) with the heuristic proposed in Algorithm 2.

Algorithm 2 Tabu-(P)

Input: set of jobs, assignment and incompatibility costs, T^{max};

For $k = \hat{k} - r$ **to** $\hat{k} + r$, **do**:

apply Tabu-($P^{(k)}$) during T^{max} time units;
let $s_{best}^{(k)}$ be the provided solution;

Output: set of solutions $\{s_{best}^{(\hat{k}-r)}, \ldots, s_{best}^{(\hat{k})}, \ldots, s_{best}^{(\hat{k}+r')}\}$;

Advanced tabu tenures We now propose a more refined way of updating the tabu tenures, based on the following straightforward idea: if the diversity of the visited solutions is below a predetermined threshold, the tabu durations $tab(j, C_t)$ should be augmented from that time (for all j and t), and, on the contrary, if the diversity becomes above the threshold, the tabu durations can be reduced from that time. Therefore, two crucial questions have to be answered: (1) how to determine the diversity of the visited solutions, (2) how to determine the threshold associated with the diversity. Answers to these questions are proposed below.

The diversity of the visited solutions depends on a distance function, which is discussed now. The similarity $sim(s, s')$ between two solutions $s = (C_1, C_2, \ldots, C_k)$ and $s' = (C'_1, C'_2, \ldots, C'_k)$ can be defined as follows (with the following convention: if $\frac{|C_i \cap C'_i|}{|C_i \cup C'_i|} = \frac{0}{0}$, we set $\frac{|C_i \cap C'_i|}{|C_i \cup C'_i|} = 1$, because in such a case $C_i = C'_i = \emptyset$):

$$sim(s, s') = \sum_{i=1}^{k} \frac{|C_i \cap C'_i|}{|C_i \cup C'_i|} \tag{5}$$

Thus, the distance $d(s, s')$ between two solutions s and s' can be defined as $d(s, s') = k - sim(s, s')$. In addition the distance $d(s, Z)$ between a solution s and a set Z of solutions can be defined as:

$$d(s, Z) = \frac{\sum_{s' \in Z} d(s, s')}{|Z|} \tag{6}$$

Finally, the diversity $d(Z)$ of a set Z of solutions can be defined as the average distance between two solutions of Z:

$$d(Z) = \frac{\sum_{s \in Z} d(s, Z - \{s\})}{|Z|} \tag{7}$$

Let δ be a threshold associated with the diversity of the visited solutions of tabu search. We propose to empirically determine δ at the beginning of the search, where

the diversity of the visited solutions is likely to be large enough, as follows. Starting with $Z = \emptyset$, from the first time h (parameter) iterations are spent without improvement of the best encountered solution, every h iterations, put in Z the best visited solution during these h iterations. When $|Z| = z$ (parameter), set $\delta = d(Z)$. Then, during every $z \cdot h$ iterations, a new set Z of solutions is collected the same way and its diversity $d(Z)$ is calculated. If $d(Z) < \delta$, the tabu duration of every new move is set to $T \cdot (1 + \varepsilon)$ (i.e. augmented because the diversity is not large enough), where T is the previous tabu duration and $\varepsilon > 0$ is a parameter. If $d(Z) \geq \delta$, the tabu duration of every new move is set to $\frac{T}{1+\epsilon}$ (i.e. reduced because the diversity is large enough). Preliminary experiments showed that $h = 50$, $z = 10$ and $\varepsilon = 4$ is a reasonable parameter setting. When such tabu tenures are used within Tabu-($P^{(k)}$), we use the notation AdvTabu-($P^{(k)}$) (for advanced tabu).

Diversification strategy Assume $a^{(k)}(j,t) \in [a_{min}, a_{max}]$ and $c^{(k)}(j,j') \in [c_{min}, c_{max}]$, for all j, j', t. If a_{max} and c_{max} are not in the same range (it will be the case in our test instances, where $a_{max} = 200$ and $c_{max} = 1000$), the following problem can occur during the search of our tabu algorithm: a job j might not visit a time period t' because it is better to pay the assignment cost associated with job j in its current time period t rather than to pay additional incompatibility costs if j is performed during period t' (because some jobs of $N(j)$ are currently in $C_{t'}$). In order to give a better chance for job j to visit $C_{t'}$, an idea consists in artificially increasing all the assignment costs. As the opposite situation might occur (i.e. assignment costs dominate incompatibility costs), decreasing the assignment costs could also be a good idea. Therefore, the following general process is proposed, where $\hat{T}$, T_{div} and ρ are parameters with $T_{div} < \hat{T}$. Within the considered tabu search (i.e. Tabu-($P^{(k)}$) or AdvTabu-($P^{(k)}$)), every $\hat{T}$ seconds, apply Tabu-($P^{(k)}$) during T_{div} seconds (diversification phase), starting with the best visited solution during the previous $\hat{T}$ seconds of the considered tabu search, but with all the assignment costs multiplied or divided by ρ (we multiply when the last decision was to divide, and vice-versa). Then we continue with the considered tabu search, but from the best solution visited during the diversification phase. Preliminary experiments showed that $(\hat{T}, T_{div}, \rho) = (300, 30, 10)$ is a reasonable parameter setting. When the above diversification strategy is used within Tabu-($P^{(k)}$), we use the notation DivTabu-($P^{(k)}$) (for diversification tabu). When it is used within AdvTabu-($P^{(k)}$), we use the notation DivAdvTabu-($P^{(k)}$).

4 Adaptive memory algorithm for ($P^{(k)}$) and (P)

In this section, we first briefly present the main ingredients of an adaptive memory algorithm. Then we design an adaptive memory algorithm from the evolutionary coloring methods presented in [10] and [12].
A basic version of the adaptive memory algorithm [33] is summarized in Algorithm 3, where performing steps (1), (2) and (3) is called a *generation*.

Therefore, in order to design an adaptive memory algorithm for problem ($P^{(k)}$), 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 memory update operator.

Algorithm 3 Adaptive memory algorithm

Initialize the central memory $\mathcal{M}$ with solutions.

While a stopping condition is not met, **do**:

1. create an offspring solution s from $\mathcal{M}$ by using a recombination operator;
2. apply a local search operator on s and let s' denote the resulting solution;
3. update $\mathcal{M}$ with the use of s'.

Initialization of $\mathcal{M}$ We propose to work with a memory $\mathcal{M}$ of size 10 as used in [10, 12, 36]. In order to initialize $\mathcal{M}$, we randomly generate 10 solutions and improve such solutions by performing Tabu-($P^{(k)}$) during 1000 iterations.

Recombination operator It is very similar to the one proposed in [36] for a satellite range scheduling problem, which is derived from the recombination operator proposed in [10] for the k-GCP. At each generation, an offspring solution $s_{(off)} = \{C_1^{(off)}, \ldots, C_k^{(off)}\}$ is built time period by time period from $\mathcal{M}$. Suppose that sets $C_1^{(off)}, \ldots, C_{t-1}^{(off)}$ are already built from parent solutions, and that parent solution $s_{r'}$ (with r' in $\{1, \ldots, 10\}$) provided the set $C_{t-1}^{(off)}$. In addition, let A be the set of already scheduled jobs (i.e. the jobs in $C_1^{(off)}, \ldots, C_{t-1}^{(off)}$). At that moment, we choose the next non considered time period t to deal with. Then, the set $C_t^{(off)}$ is provided by the solution $s_r = \{C_1^{(r)}, \ldots, C_k^{(r)}\}$ in $\mathcal{M}$ (with $r \neq r'$, i.e. the same solution of $\mathcal{M}$ cannot consecutively provide two sets of the offspring solution) such that $|C_t^{(r)} - A|$ is maximum (we break ties randomly). Thus we set $C_t^{(off)} = C_t^{(r)} - A$. At the end of such a process, we successively schedule each non scheduled job (considered in a random order) in a greedy fashion.

Of course, different recombination operators could be developed and tested for problem ($P^{(k)}$). We however decide to only focus on the one proposed in [36] (which provides the best results for the considered scheduling problem) that was derived from the best recombination operator for the k-GCP proposed in [10].

Intensification operator It is simply Tabu-($P^{(k)}$) described in the previous section. As we would like to perform a significant number of generations in our adaptive memory algorithm, at each generation, we have to perform Tabu-($P^{(k)}$) for a short time. Such a strategy will balance the roles associated with the recombination operator and the intensification operator (tabu search). Let I (parameter) be the number of iterations performed by Tabu-($P^{(k)}$) at each generation of AMA-($P^{(k)}$) (which denotes the adaptive memory algorithm for ($P^{(k)}$)). Preliminary experiments showed that $I = 10{,}000$ is a reasonable choice, which corresponds on most instances to a few seconds of CPU time on the computer used for the experiments and described in the next section. For these reasons, it is not relevant to use the diversification phase within the intensification operator.

Memory update operator It is based on the efficient strategy proposed in [36]. Let s be the solution provided by Tabu-$(P^{(k)})$ at each generation, let s_{worst} be the worst solution of $\mathcal{M}$, and let s_{old} be the oldest solution of $\mathcal{M}$. We propose to update $\mathcal{M}$ with s as follows. If s is better or equal to s_{worst}, we replace s_{worst} with s in $\mathcal{M}$ and update s_{worst}. Otherwise, we replace s_{old} with s in $\mathcal{M}$ and update s_{old}. In the latter case, even if s is not able to improve the average quality of $\mathcal{M}$, it is at least able to bring "new blood" in $\mathcal{M}$.

As proposed in the previous section, we propose to tackle problem (P) with the heuristic proposed in Algorithm 2, but of course by replacing Tabu-$(P^{(k)})$ with AMA-$(P^{(k)})$. Similarly, AMA-$(P^{(k)})$ is stopped when a CPU time limit T^{max} is reached.

5 Obtained results

In this section, we first propose other methods to be compared with the above proposed heuristics. Then we describe the way we generated instances. Finally, we present and discuss the obtained results. Note that all the instances associated with this paper can be provided by the authors upon request.

5.1 Comparison with other methods

Because no other method exists to tackle problem (P), an important issue is to be able to compare our heuristics with an appropriate set of other methods. We propose the following strategy: Tabu-(P) and AMA-(P) will be compared with other heuristics (a greedy heuristic and a descent algorithm) on large instances, and with an exact method (using CPLEX 10.0) on small instances.

Below, we propose three methods to tackle problem $(P^{(k)})$: a greedy heuristic, a local search heuristic, and an exact method, respectively denoted by Greedy-$(P^{(k)})$, Descent-$(P^{(k)})$ and Exact-$(P^{(k)})$. Each of these methods can then be used to tackle problem (P) by applying it for all $k \in K$. The provided solution for (P) is then the best one among the $|K|$ generated ones. The usefulness of Exact-(P) is to show that Tabu-(P) is able to find optimal solutions on small instances, whereas the usefulness of Descent-(P) is to show that all the ingredients added to Descent-(P) to derive Tabu-(P) are relevant (namely the tabu tenures and the restricted neighborhood which is examined at each iteration). Greedy-(P) is considered to measure the gap between local search and constructive algorithms.

5.1.1 A greedy constructive heuristic for (P)

We first propose a greedy method for $(P^{(k)})$, denoted by Greedy-$(P^{(k)})$, which builds a solution step by step as follows. Let J be the set of jobs which are placed in a time period. We start with an empty solution $s^{(k)}$, i.e. with $J = \emptyset$. Then, at each step: (1) a job $j \notin J$ is randomly chosen; (2) a time period $t \in \{1, \ldots, k\}$ is assigned to j, such that the augmentation of the costs is as small as possible (ties are broken randomly); (3) we add j in J. The process stops when $J = V$, i.e. when a time period is associated with every job.

5.1.2 A descent local search heuristic for (P)

We first propose a descent method for $(P^{(k)})$, denoted by Descent-$(P^{(k)})$, which starts with a random solution $s^{(k)}$ and tries to improve it iteratively. At each iteration, we perform the best possible move, where a move consists in changing the time period associated with a job. In other words, $(k-1)\cdot|V|$ neighbor solutions are evaluated at each iteration. The process stops when there is no neighbor solution which is strictly better than the current solution. In contrast with Tabu-$(P^{(k)})$, we can remark that in Descent-$(P^{(k)})$, at each iteration, all the neighbor solutions of the current solution are evaluated and there is no forbidden move. In addition, the stopping conditions are different.

5.1.3 An exact method using CPLEX for (P)

We first propose the following integer program to model problem $(P^{(k)})$. Let $x_{j,t}=1$ if time period t is assigned to job j, and $x_{j,t}=0$ otherwise. Then, we can formulate $(P^{(k)})$ as follows.

$$\text{Obj. fct.} \quad \min \quad \sum_{t=1}^{k}\sum_{j=1}^{n} a^{(k)}(j;t)\cdot x_{j,t} + \sum_{t=1}^{k}\sum_{j=1}^{n-1}\sum_{j'=j+1}^{n} c^{(k)}(j;j')\cdot x_{j,t}\cdot x_{j',t} \tag{8}$$

$$\text{Constraints} \quad \sum_{t=1}^{k} x_{j,t} = 1, \quad \forall j \in \{1,\ldots,n\} \tag{9}$$

$$x_{j,t} \in \{0,1\}, \quad \forall j \in \{1,\ldots,n\},\ \forall t \in \{1,\ldots,k\} \tag{10}$$

Equation (9) imposes to assign exactly one time period to each job. In order to tackle such a problem with the CPLEX MIP Solver, we should formulate it as an integer linear program. In order to have a linear objective, we propose to set $y_{j,j'} = \sum_{t=1}^{k} x_{j,t}x_{j',t}$. Because of constraint (9), we have $y_{j,j'}=1$ if jobs j and j' are performed at the same time period, and $y_{j,j'}=0$ otherwise. The former implication is expressed in our formulation by the linking constraint (13). The resulting linear model is described below. The associated exact method based on the use of the CPLEX MIP Solver is denoted Exact-$(P^{(k)})$.

$$\text{Obj. fct.} \quad \min \quad \sum_{t=1}^{k}\sum_{j=1}^{n} a^{(k)}(j;t)\cdot x_{j,t} + \sum_{j=1}^{n-1}\sum_{j'=j+1}^{n} c^{(k)}(j;j')\cdot y_{j,j'} \tag{11}$$

$$\text{Constraints} \quad \sum_{t=1}^{k} x_{j,t} = 1, \quad \forall j \in \{1,\ldots,n\} \tag{12}$$

$$x_{j,t} + x_{j',t} - 1 \le y_{j,j'}, \quad \forall j < j' \in \{1,\ldots,n\},\ \forall t \in \{1,\ldots,k\} \tag{13}$$

$$x_{j,t}, y_{j,j'} \in \{0,1\}, \quad \forall j < j' \in \{1,\ldots,n\},\ \forall t \in \{1,\ldots,k\} \tag{14}$$

Notice that in an admissible solution of the above formulation, $y_{j,j'} = 1$ may occur even if $\sum_{t=1}^{k} x_{j,t} x_{j',t} = 0$, but as the objective is to be minimized, this cannot happen in any optimal solution as long as $c^{(k)}(j; j') > 0$, since setting $y_{j,j'}$ to 0 would provide an admissible solution with lower cost. Hence the optimal solution value of our formulation is equal to $f^{(k)}(s^{(k)}_{opt})$.

5.2 Generation of the instances

We build instances from 13 graphs from the DIMACS Challenge (see ftp://dimacs.rutgers.edu/pub/challenge/graph/). We selected those graphs because they are the most challenging ones in the graph coloring community [1, 12]. This means that the incompatibility costs are difficult to minimize, and the difficulty is increased because of the assignment costs. In addition, three types of graphs are considered and described below. For these reasons, we strongly believe that the proposed set of benchmark instances is consistent.

- Five DSJC$n.d$ graphs: the DSJC's are random graphs with n vertices and a density of $\frac{d}{10}$. It means that each pair of vertices has a probability of $\frac{d}{10}$ to form an edge. We use the DSJC's graphs with $n \in \{500, 1000\}$ and $d \in \{1, 5, 9\}$.
- Four flatn_χ_0 graphs: the flat graphs are constructed graphs with n vertices and a chromatic number χ. The end number '0' means that all vertices are adjacent to the same number of vertices.
- Four le$n_\chi x$ graphs: the Leighton graphs have n vertices and a chromatic number χ equal to the size of the largest *clique* (i.e., the largest number of pairwise adjacent vertices). The end letter 'x' stands for different graphs with similar settings.

In addition, in order to compare Tabu-($P^{(k)}$) with Exact-($P^{(k)}$) on smaller instances (up to 100 vertices), we generated random graphs with a density of 0.5. For these graphs, we selected $n \in \{10; 20; 30; \ldots; 100\}$ and $k \in \{2; 3; 4; \ldots; 10\}$. Such graphs are labeled *random*$(n; k)$.

5.3 Generation of the costs

Remember that we consider the situation where the project manager does not only provide a single number $\hat{k}$ of time periods within which the project has to be performed, but he provides a set $K = \{\hat{k} - r, \ldots, \hat{k}, \ldots, \hat{k} + r'\}$ of possible durations. Without loss of generality, we assume that $r = r'$. We propose now a way to generate instances for problem $P^{(\hat{k})}$, and a way to adjust such costs for generating instances of problem (P). We suggest the following methodology for $P^{(\hat{k})}$. We randomly generate the costs by the use of the uniform function UNIFORM$(u; v)$, which returns an integer in the set $\{u, u + 1, \ldots, v - 1, v\}$ (assuming $u < v$). We set $a^{(\hat{k})}(j; t) = \text{UNIFORM}(1; 200)$, for all j and t, and $c^{(\hat{k})}(j; j') = \text{UNIFORM}(1; 1000)$, for all j and j'. A key issue is the way to adjust the costs for other project's durations. The idea is the following: if the horizon is smaller than $\hat{k}$ days, the incompatibility costs will augment (we will have more conflicts to deal with) but the assignment costs will be reduced (the staff and the machines will work a smaller

number of days). On the contrary, if the horizon is larger than $\hat{k}$ days, the incompatibility costs will be reduced but the assignment costs will augment. We propose the following. For $1 \leq i \leq r$, and all j and t, we set: $a^{(\hat{k}+i)}(j;t) = 200 + 50 \cdot i$ and $a^{(\hat{k}-i)}(j;t) = \max\{1; (1 - \frac{i}{10}) \cdot a^{(\hat{k})}(j;t)\}$. In addition, as the number of conflicts is naturally decreasing with the length of the horizon, we can simply set $c^{(k)}(j;j') = c^{(\hat{k})}(j;j')$, for all $k \neq \hat{k}$, j and j'.

5.4 Presentation of the results

Our algorithms were implemented in C++ and run on a computer Intel Pentium 4 (4.00 GHz, RAM 1024 Mo DDR2), and the exact method running CPLEX 10.0 was performed on a computer AMD Opteron (Processor 250 2.4 GHz, 16 GB). The stopping condition for Tabu-($P^{(k)}$) and AMA-($P^{(k)}$) is a maximum time limit T^{max}. As Greedy-($P^{(k)}$) needs less than one minute to provide a solution, it is restarted as long as T^{max} minutes are not reached. The provided solution is then the best among all the restarts occurring within T^{max} minutes. The same holds for Descent-($P^{(k)}$). Preliminary experiments showed that $T^{max} = 60$ minutes is a reasonable value for the DIMACS instances. For the *random*$(n;k)$ instances, we set $T^{max} = 15$ minutes because such instances are of relatively small size ($n \leq 100$).

First of all, we have to tune the proposed tabu search algorithm with a fixed value of k. Table 1 reports the average results (over 5 runs) obtained with Tabu-($P^{(k)}$), AdvTabu-($P^{(k)}$), DivTabu-($P^{(k)}$) and DivAdvTabu-($P^{(k)}$) on the DIMACS instances. Let f^* be the best solution encountered by one run of Tabu-($P^{(k)}$) within the considered time limit T^{max}, and f^{Tabu} be the average value of f^* (rounded to the nearest integer) over the considered number of runs. We similarly define $f^{AdvTabu}$

Table 1 Comparison of Tabu-($P^{(k)}$), AdvTabu-($P^{(k)}$), DivTabu-($P^{(k)}$) and DivAdvTabu-($P^{(k)}$) on the DIMACS instances with a time limit of one hour

Graph G	n	d	$\chi(G)$	$k^\star$	$\hat{k}$	f^{Tab}	G^{Adv}	G^{Div}	G^{DivAdv}
DSJC1000.1	1000	0.1	?	20	13	241601	−14.07%	−12.62%	−17.10%
DSJC1000.5	1000	0.5	?	83	55	250977	−2.56%	−3.40%	−6.20%
DSJC1000.9	1000	0.9	?	224	149	166102	−0.59%	−1.76%	−0.93%
DSJC500.5	500	0.5	?	48	32	98102	−6.84%	−5.34%	−8.09%
DSJC500.9	500	0.9	?	126	84	64224	−10.05%	−8.68%	−10.75%
flat1000_50_0	1000	0.49	50	50	33	665449	−2.01%	−2.06%	−2.51%
flat1000_60_0	1000	0.49	60	60	40	462612	−2.48%	−1.23%	−2.86%
flat1000_76_0	1000	0.49	76	82	55	246157	−3.84%	−4.36%	−5.87%
flat300_28_0	300	0.48	28	28	19	62862	−0.77%	−2.43%	−1.54%
le450_15c	450	0.16	15	15	10	149041	−13.36%	−7.98%	−13.04%
le450_15d	450	0.17	15	15	10	146696	−11.87%	−6.20%	−12.56%
le450_25c	450	0.17	25	25	17	72974	−11.41%	−23.73%	−25.09%
le450_25d	450	0.17	25	25	17	70852	−8.97%	−21.12%	−24.70%
Average						207511	−6.83%	−7.76%	−10.10%

Table 2 Comparison of DivAdvTabu-($P^{(k)}$), AMA-($P^{(k)}$), Descent-($P^{(k)}$) and Greedy-($P^{(k)}$) on the DIMACS instances with a time limit of one hour

Graph G	$\hat{k}$	$f^{DivAdvTabu}$	G^{Greedy}	$G^{Descent}$	G^{AMA}
DSJC1000.1	13	200281	57.23%	28.49%	5.42%
DSJC1000.5	55	235409	33.30%	18.43%	−0.28%
DSJC1000.9	149	164550	10.30%	11.35%	−4.98%
DSJC500.5	32	90163	55.03%	34.76%	3.50%
DSJC500.9	84	57320	43.69%	42.71%	2.69%
flat1000_50_0	33	648770	24.97%	10.82%	−0.82%
flat1000_60_0	40	449389	28.63%	14.16%	−1.18%
flat1000_76_0	55	231710	32.15%	18.23%	−1.92%
flat300_28_0	19	61891	51.20%	29.19%	1.50%
le450_15c	10	129605	40.75%	20.45%	2.86%
le450_15d	10	128273	42.89%	22.49%	5.19%
le450_25c	17	54667	39.99%	27.11%	21.32%
le450_25d	17	53351	43.40%	29.40%	23.28%
Average			38.73%	23.66%	4.35%

(associated with AdvTabu-($P^{(k)}$)), $f^{DivTabu}$ (associated with DivTabu-($P^{(k)}$)) and $f^{DivAdvTabu}$ (associated with DivAdvTabu-($P^{(k)}$)). From left to right, the columns indicate: the name of the DIMACS graph G, the number n of vertices of G, the density d of graph G, the chromatic number $\chi(G)$ of G (we put a "?" when it is not known), the best upper bound $k^\star$ on $\chi(G)$ ever found by a heuristic, the considered value $\hat{k}$ of k with which we performed the heuristics, f^{Tabu}, the percentage gap $G^{Adv} = \frac{f^{AdvTabu} - f^{Tabu}}{f^{Tabu}}$ between f^{Tabu} and $f^{AdvTabu}$, the percentage gap G^{Div} between f^{Tabu} and $f^{DivTabu}$, and the percentage gap G^{DivAdv} between f^{Tabu} and $f^{DivAdvTabu}$. Average results are indicated in the last line. On the one hand, AdvTabu-($P^{(k)}$) is better than Tabu-($P^{(k)}$), which means that the use of the advanced tabu tenures is relevant. On the other hand, DivTabu-($P^{(k)}$) provides better results than Tabu-($P^{(k)}$), which indicates that the proposed diversification strategy is useful. Finally, the joint use of the advanced tabu tenures and the diversification strategy leads to the best results. Therefore, only DivAdvTabu-($P^{(k)}$) will be considered for farther comparisons.

For a fixed value of k, Table 2 reports the average results (over 10 runs) obtained with DivAdvTabu-($P^{(k)}$), Greedy-($P^{(k)}$), Descent-($P^{(k)}$), and AMA-($P^{(k)}$) on the DIMACS instances. Let f^* be the best solution encountered by one run of DivAdvTabu-($P^{(k)}$) within the considered time limit T^{max}, and $f^{DivAdvTabu}$ be the average value of f^* (rounded to the nearest integer) over the considered number of runs. We similarly define f^{AMA}, $f^{Descent}$ and f^{Greedy}. From left to right, the columns indicate: the name of the DIMACS graph G, the considered value $\hat{k}$ of k with which we performed the heuristics, $f^{DivAdvTabu}$, the percentage gap $G^{Greedy} = \frac{f^{Greedy} - f^{DivAdvTabu}}{f^{DivAdvTabu}}$ between $f^{DivAdvTabu}$ and f^{Greedy}, the percentage gap $G^{Descent}$ between $f^{DivAdvTabu}$

and $f^{Descent}$, and the percentage gap G^{AMA} between $f^{DivAdvTabu}$ and f^{AMA}. We indicate average gaps in the last line. Note that in order to encounter a relevant number of incompatibility costs, we have to choose $\hat{k}$ smaller than k^{best}. Otherwise, the number of pairs $\{j, j'\}$ of jobs linked with an incompatibility cost and performed within the same time period will be too small. Thus, we propose to choose $\hat{k}$ close to $\frac{2}{3}k^{best}$. If $\hat{k}$ is much smaller than k^{best}, it means that the incompatibility costs will be much larger.

When comparing DivAdvTabu-($P^{(k)}$) with simpler methods, i.e. Descent-($P^{(k)}$) and Greedy-($P^{(k)}$), we can observe that DivAdvTabu-($P^{(k)}$) obtained, by far, the best results on each instance. In average, DivAdvTabu-($P^{(k)}$) is about 24% better than Descent-($P^{(k)}$) and 39% better than Greedy-($P^{(k)}$). Note that no general conclusion can be drawn according to the density variation of the graphs. Such a remark holds for the whole paper.

When comparing DivAdvTabu-($P^{(k)}$) with a more refined method, i.e. AMA-($P^{(k)}$), we can remark that the former is slightly better (in average 4.35%) than the latter. However, if we have a closer look at Table 2, we can see that AMA-($P^{(k)}$) is better on five instances and worse on 8 instances. Both methods have a similar performance on the DSJC's graphs, AMA-($P^{(k)}$) is slightly better on the flat graphs but has a bad behavior on the Leighton's graphs. Note that if we only allow a time limit of 10 minutes, the gap between DivAdvTabu-($P^{(k)}$) and AMA-($P^{(k)}$) is in average 8.35% instead of 4.35% in favor of tabu search. This means that if the scheduler has to react in a few minutes (for example in the case he has to quickly reschedule some jobs after a non planned event), tabu search is obviously the best option. Therefore, DivAdvTabu-($P^{(k)}$) seems to be the best overall compromise between quality and simplicity. For this reason, only tabu search will be considered further in this paper.

Table 3 Variation of f^{Tabu} when k ranges from $\hat{k}-8$ to $\hat{k}-4$

Graph G	$G(\hat{k}-8)$	$G(\hat{k}-7)$	$G(\hat{k}-6)$	$G(\hat{k}-5)$	$G(\hat{k}-4)$
DSJC1000.1	583.27%	400.12%	276.54%	192.86%	130.44%
DSJC1000.5	42.07%	34.77%	29.17%	24.35%	21.79%
DSJC1000.9	8.74%	9.90%	5.87%	4.44%	3.78%
DSJC500.5	97.24%	78.63%	66.60%	52.27%	36.71%
DSJC500.9	27.47%	23.91%	20.57%	15.80%	14.07%
flat1000_50_0	63.34%	53.45%	43.25%	34.80%	25.43%
flat1000_60_0	52.98%	45.46%	36.77%	29.60%	22.00%
flat1000_76_0	41.50%	34.42%	30.99%	21.89%	17.02%
flat300_28_0	222.15%	171.30%	133.54%	99.17%	72.91%
le450_15c	1287.70%	697.92%	422.82%	266.99%	167.89%
le450_15d	1297.13%	704.12%	423.93%	268.33%	171.36%
le450_25c	256.84%	194.47%	142.36%	103.83%	75.22%
le450_25d	263.61%	199.37%	147.40%	107.85%	76.36%
Average	326.46%	203.68%	136.91%	94.01%	64.23%

In addition, for any instance, let f_{min}^{Tabu} (resp. f_{max}^{Tabu}) be the smallest (resp. largest) value of f^* (rounded to the nearest integer) encountered by DivAdvTabu-($P^{(k)}$) among the considered number of runs. We observed that the average gap (over the 13 instances) between f_{min}^{Tabu} and f^{Tabu} is 1.82%, and the average gap between f_{max}^{Tabu} and f^{Tabu} is 1.98%. This means that the proposed tabu search is robust.

Table 4 Variation of f^{Tabu} when k ranges from $\hat{k}-3$ to $\hat{k}+2$ ($\hat{k}$ non included)

Graph G	$G(\hat{k}-3)$	$G(\hat{k}-2)$	$G(\hat{k}-1)$	$G(\hat{k}+1)$	$G(\hat{k}+2)$
DSJC1000.1	82.63%	46.71%	21.14%	−12.77%	−29.51%
DSJC1000.5	16.16%	8.21%	6.69%	−2.14%	−5.96%
DSJC1000.9	6.00%	0.89%	−0.65%	−3.01%	−2.71%
DSJC500.5	27.94%	19.27%	8.30%	−6.39%	−14.90%
DSJC500.9	10.26%	5.87%	4.46%	−3.01%	−4.43%
flat1000_50_0	18.83%	12.03%	5.01%	−5.29%	−10.86%
flat1000_60_0	16.72%	11.42%	6.00%	−4.35%	−9.59%
flat1000_76_0	12.67%	8.21%	5.42%	−3.85%	−6.03%
flat300_28_0	48.92%	30.10%	14.04%	−11.86%	−22.01%
le450_15c	103.51%	56.84%	25.80%	−19.45%	−34.08%
le450_15d	105.10%	58.12%	25.60%	−17.09%	−33.98%
le450_25c	51.31%	30.60%	13.62%	−13.00%	−24.34%
le450_25d	52.51%	34.12%	12.73%	−12.59%	−25.98%
Average	42.50%	24.80%	11.40%	−8.83%	−17.26%

Table 5 Variation of f^{Tabu} when k ranges from $\hat{k}+3$ to $\hat{k}+8$

Graph G	$G(k^t+3)$	$G(k^t+4)$	$G(k^t+5)$	$G(k^t+6)$	$G(k^t+7)$	$G(k^t+8)$
DSJC1000.1	−38.07%	−47.45%	−53.25%	−59.35%	−63.83%	−68.80%
DSJC1000.5	−9.66%	−13.28%	−16.24%	−18.03%	−22.00%	−24.83%
DSJC1000.9	−5.25%	−6.15%	−7.80%	−9.26%	−11.15%	−11.25%
DSJC500.5	−19.97%	−26.28%	−30.91%	−35.53%	−40.44%	−44.01%
DSJC500.9	−8.23%	−10.66%	−13.26%	−15.30%	−17.10%	−19.97%
flat1000_50_0	−14.77%	−19.73%	−24.32%	−27.08%	−30.95%	−34.58%
flat1000_60_0	−14.30%	−17.59%	−21.15%	−24.69%	−28.06%	−31.28%
flat1000_76_0	−11.45%	−13.53%	−16.30%	−19.84%	−20.98%	−25.19%
flat300_28_0	−31.07%	−37.82%	−44.48%	−49.56%	−54.76%	−59.38%
le450_15c	−45.88%	−54.65%	−62.46%	−67.62%	−71.98%	−75.83%
le450_15d	−44.37%	−54.04%	−60.50%	−65.80%	−70.76%	−75.75%
le450_25c	−35.69%	−42.09%	−48.51%	−54.26%	−59.40%	−64.48%
le450_25d	−33.04%	−41.47%	−48.87%	−54.12%	−59.51%	−62.75%
Average	−23.98%	−29.60%	−34.47%	−38.50%	−42.38%	−46.01%

Based on the use of DivAdvTabu-($P^{(k)}$), it would be now interesting to search for a k^* with lower associated costs than the costs associated with $\hat{k}$. For different values of k, in Tables 3, 4 and 5, we report the average results (over 5 runs) obtained with DivAdvTabu-($P^{(k)}$) on the DIMACS instances. The columns indicate: the name of the DIMACS graph G, and the percentage gap $G(\hat{k}+r)$ between the costs associated with the targeted number of time periods and the costs associated with $\hat{k}+r$ time periods, with $r \in \{-8, \ldots, 8\} - \{0\}$. We give average gaps in the last lines. We can remark that the larger k is, the smaller the costs are. If we have a closer look at the average variation of the costs with respect to k (see also Fig. 1, where the horizontal axis indicates the number of periods, and the vertical axis gives the associated costs (average over 5 runs)), we can observe that the cost function is convex, with an almost

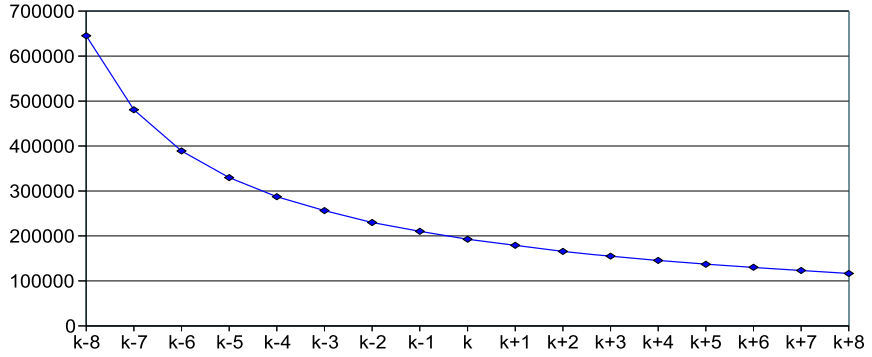

Fig. 1 Average variation of the costs according to the number k of periods

Table 6 Comparison of Tabu-($P^{(k)}$) and Exact-($P^{(k)}$) on the *random*(n; k) instances when optimal values are known

Value of k:		2	3	4	5	6	7	8	9	10
$n = 10$	f^{opt}	1727	719	496	396	254	254	263	185	174
	t^{opt}	0	0	0	0	0	0	0	0	0
	t^{Tabu}	0	0	0	0	0	0	0	0	0
$n = 20$	f^{opt}	8475	4429	2498	1526	1123	946	656	546	444
	t^{opt}	0	1	1	0	0	0	0	0	0
	t^{Tabu}	0	0	0	0	0	0	0	0	0
$n = 30$	f^{opt}	21183	10612	6127	3731	2556	1828	1543	1224	1026
	t^{opt}	4	408	752	303	98	11	2	1	0
	t^{Tabu}	0	0	0	0	10	24	0	2	0
$n = 40$	f^{opt}	35330	–	–	–	–	2606	1957	1535	1428
	t^{opt}	177	–	–	–	–	12311	373	59	34
	t^{Tabu}	0	0	0	4	34	11	12	3	29
$n = 50$	f^{opt}	59821	–	–	–	–	–	–	–	–
	t^{opt}	17306	–	–	–	–	–	–	–	–
	t^{Tabu}	0	0	1	6	48	32	41	414	10

Table 7 Comparison of Tabu-($P^{(k)}$) and Exact-($P^{(k)}$) on the *random*(n; k) instances when optimal values are not known

Value of k:		2	3	4	5	6	7	8	9	10
$n = 40$	*LB*	–	16089	8326	4800	3536	–	–	–	–
	UB	–	17734	9571	6108	4044	–	–	–	–
	f^{Tabu}	35330	17717	9360	5979	4039	2606	1957	1535	1428
	t^{Tabu}	0	0	0	4	143	11	12	3	29
$n = 50$	*LB*	–	18452	8710	5139	2666	2989	2379	2311	2029
	UB	–	32844	20666	12296	8632	6322	4519	2817	2148
	f^{Tabu}	59821	30474	17649	10713	7268	5004	3680	2738	2148
	t^{Tabu}	0	0	1	6	48	32	41	414	10
$n = 60$	*LB*	70206	21817	8955	5499	4229	3438	2987	2583	2431
	UB	88047	50330	30998	19139	14045	9073	6280	5167	3460
	f^{Tabu}	86457	44355	26723	16688	10944	7273	5635	4083	3249
	t^{Tabu}	0	1	7	1	24	105	475	376	16
$n = 70$	*LB*	82687	21788	9439	5894	4799	3811	3182	2898	2611
	UB	122888	75272	46783	31536	21120	16568	11448	9459	7421
	f^{Tabu}	121072	65576	40015	25863	17552	12304	8794	6428	4860
	t^{Tabu}	0	2	9	123	154	36	6	720	146
$n = 80$	*LB*	89364	21164	8604	6363	5007	4240	3771	3252	2908
	UB	162689	95846	64983	42757	31632	21915	16220	13156	10012
	f^{Tabu}	157858	85365	52166	34294	23597	16692	11595	8273	6693
	t^{Tabu}	1	8	9	117	452	245	797	832	516
$n = 90$	*LB*	96816	22264	9889	6700	5441	4777	4108	3688	3182
	UB	209894	126620	81835	61229	44075	30432	24502	20001	16265
	f^{Tabu}	199230	107039	67012	44092	29635	20712	15374	11816	8599
	t^{Tabu}	0	144	28	30	171	337	77	878	475
$n = 100$	*LB*	105529	23668	9600	6834	5384	4745	4106	3529	3305
	UB	271573	170754	112305	80882	58494	42405	36536	28319	16160
	f^{Tabu}	256040	142070	90039	60588	42417	30544	22554	16552	12423
	t^{Tabu}	0	15	26	161	240	819	573	147	442

asymptotical behavior when k tends to infinity (due to the fact that the contribution of the incompatibility costs tends to zero). Remember that for $\hat{k}$, each incompatibility cost is in $[1, 1000]$, and each assignment cost is in $[1, 200]$. With a small number of time periods, there are many conflicts and the associated incompatibility costs compose the main part of the total costs, whereas the assignment costs are small; on the contrary, with a large number of time periods, most conflicts can be removed and the assignment costs increase, but not enough to increase the total costs.

In Tables 6 and 7, we compare Tabu-($P^{(k)}$) (with a time limit of 15 minutes) with Exact-($P^{(k)}$) (with a time limit of 10 hours) on the *random*(n; k) instances. In Table 6, we present results on the instances for which Exact-($P^{(k)}$) was able to find an optimal solution. On all these instances, Tabu-($P^{(k)}$) was also able to find an optimal solution. We indicate the value f^{opt} (which is hence equal to f^{Tabu}) of the optimal

solution, the times (in seconds) t^{opt} and t^{Tabu} respectively needed by Exact-($P^{(k)}$) and Tabu-($P^{(k)}$) to find an optimal solution. We observe that t^{Tabu} is much smaller than t^{opt}, which is very encouraging for our heuristic. In Table 7, we present results on the instances for which Exact-($P^{(k)}$) was not able to find an optimal solution. On such instances, CPLEX provided however two values: a lower bound *LB* and an upper bound *UB*. Such bounds are indicated in Table 7, as well as the value f^{Tabu}. Such a solution was obtained in t^{Tabu} seconds. We can remark that the upper bound provided by f^{Tabu} is never worse than *UB*. On the basis of its accuracy as shown in Table 6, we can expect it to be close to the optimal value.

6 Conclusion

In this paper, we tackle a new project management problem (P) for which assignment costs and incompatibility costs are taken into account. The goal is to assign a time period to each job while minimizing the total costs, considering that the duration of the project is also a decision variable. The methods we propose (namely tabu search and adaptive memory algorithm) are shown to provide good solutions in a reasonable amount of time, hence to be an excellent compromise between simplicity and efficiency.

We also propose some variations of the problem, such as the considerations of forbidden time periods for some jobs, as well as precedence constraints. We can remark that our heuristics are still appropriate for such extensions, because we were able to adapt the model (instead of the methods) to such situations. It is important to notice that the consideration of precedence constraints within the methods is a very different (and interesting) avenue of research. For example, the neighborhood structure which consists in changing the period assigned to a job would not be relevant anymore. In such a case, it seems better to deal with partial but legal solutions, as proposed in [7].

Another relevant avenue of research would be to consider other type of costs or a specific duration for each job. Finally, as showed in [18], there is still a lot of research possibilities in project scheduling under uncertainty.

References

1. Bloechliger, I., Zufferey, N.: A graph coloring heuristic using partial solutions and a reactive tabu scheme. Comput. Oper. Res. **35**, 960–975 (2008)
2. Chams, D., Hertz, A., de Werra, D.: Some experiments with simulated annealing for coloring graphs. Eur. J. Oper. Res. **32**, 260–266 (1987)
3. Charon, I., Hudry, O.: The noising method: a new method for combinatorial optimization. Oper. Res. Lett. **14**, 133–137 (1993)
4. Davis, L.: Handbook of Genetic Algorithms. Reinhold, New York (1991)
5. Demeulemeester, E.L., Herroelen, W.S.: Project Scheduling: A Research Handbook. Kluwer Academic, Norwell (2002)
6. Dorigo, M., Blum, C.: Ant colony optimization theory: a survey. Theor. Comput. Sci. **344**(2–3), 243–278 (2005)
7. Dupond, A., Vasquez, M., Habet, D.: Consistent neighbourhood in a tabu search. In: Metaheuristics: Progress as Real Problem Solvers, vol. 17, pp. 367–386. Springer, Berlin (2005)
8. Feo, T.A., Resende, M.G.: Greedy randomized adaptive search procedures. J. Glob. Optim. **6**, 109–133 (1995)

9. Fleurent, C., Ferland, J.A.: Genetic and hybrid algorithms for graph coloring. Ann. Oper. Res. **63**(3), 437–461 (1996)
10. Galinier, P., Hao, J.K.: Hybrid evolutionary algorithms for graph coloring. J. Comb. Optim. **3**(4), 379–397 (1999)
11. Galinier, P., Hertz, A.: A survey of local search methods for graph coloring. Comput. Oper. Res. **33**, 2547–2562 (2006)
12. Galinier, P., Hertz, A., Zufferey, N.: An adaptive memory algorithm for the graph coloring problem. Discrete Appl. Math. **156**, 267–279 (2008)
13. Gamst, A., Rave, W.: On the frequency assignment in mobile automatic telephone systems. In: Proceedings of GLOBECOM IEEE, Miami, United States, 29 November–2 December 1982, pp. 309–315 (1982)
14. Garey, M., Johnson, D.S.: Computer and Intractability: A Guide to the Theory of NP-Completeness. Freeman, San Francisco (1979)
15. Glass, C.: Bag rationalisation for a food manufacturer. J. Oper. Res. Soc. **53**, 544–551 (2002)
16. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic, Boston (1997)
17. Herrmann, F., Hertz, A.: Finding the chromatic number by means of critical graphs. ACM J. Exp. Algorithmics **7**(10), 1–9 (2002)
18. Herroelen, W., Leus, R.: Project scheduling under uncertainty: survey and research potentials. Eur. J. Oper. Res. **165**(2), 289–306 (2005)
19. Hertz, A., de Werra, D.: Using tabu search techniques for graph coloring. Computing **39**, 345–351 (1987)
20. Hertz, A., Plumettaz, M., Zufferey, N.: Variable space search for graph coloring. Discrete Appl. Math. **156**, 2551–2560 (2008)
21. Icmeli, O., Erenguc, S.S., Zappe, C.J.: Project scheduling problems: a survey. Int. J. Oper. Prod. Manag. **13**(11), 80–91 (1993)
22. Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C.: Optimization by simulated annealing: an experimental evaluation, Part II: Graph coloring and number partitioning. Oper. Res. **39**, 378–406 (1991)
23. Kerzner, H.: Project Management: A Systems Approach to Planning, Scheduling, and Controlling. Wiley, New York (2003)
24. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.: Optimization by simulated annealing. Science **220**(5498), 671–680 (1983)
25. Kolisch, R., Padman, R.: An integrated survey of deterministic project scheduling. Omega **29**(3), 249–272 (2001)
26. Lancaster, J., Ozbayrak, M.: Evolutionary algorithms applied to project scheduling problems—a survey of the state-of-the-art. Int. J. Prod. Res. **45**(2), 425–450 (2007)
27. Leighton, F.T.: A graph coloring algorithm for large scheduling problems. J. Res. Nat. Bur. Stand. **84**, 489–505 (1979)
28. Malaguti, E., Monaci, M., Toth, P.: A metaheuristic approach for the vertex coloring problem. INFORMS J. Comput. **20**(2), 302–316 (2008)
29. Mehrotra, A., Trick, M.: A column generation approach for graph coloring. INFORMS J. Comput. **8**(4), 344–354 (1996)
30. Mladenovic, N., Hansen, P.: Variable neighborhood search. Comput. Oper. Res. **24**, 1097–1100 (1997)
31. Pinedo, M.: Scheduling: Theory, Algorithms, and Systems. Prentice Hall, New York (2002)
32. Plumettaz, M., Schindl, D., Zufferey, N.: Ant local search and its efficient adaptation to graph colouring. J. Oper. Res. Soc. **61**, 819–826 (2010)
33. Rochat, Y., Taillard, E.: Probabilistic diversification and intensification in local search for vehicle routing. J. Heuristics **1**, 147–167 (1995)
34. Stecke, K.: Design planning, scheduling and control problems of flexible manufacturing. Ann. Oper. Res. **3**, 3–12 (1985)
35. Voudouris, C., Tsang, E.: Guided local search. Eur. J. Oper. Res. **113**(2), 469–499 (1999)
36. Zufferey, N., Amstutz, P., Giaccari, P.: Graph colouring approaches for a satellite range scheduling problem. J. Sched. **11**(4), 263–277 (2008)