Makespan Minimization for a Parallel Machine Scheduling Problem with Preemption and Job Incompatibility

Simon Thevenin
Nicolas Zufferey
Jean-Yves Potvin

April 2016

CIRRELT-2016-22

Makespan Minimization for a Parallel Machine Scheduling Problem with Preemption and Job Incompatibility

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

[1] Geneva School of Economics and Management, University of Geneva, UNIMAIL, 40 Boulevard du Pont-d'Arve, 1211 Geneva 4, Switzerland

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

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

Abstract. In this paper, an extension of the graph coloring problem is introduced to model a parallel machine scheduling problem with job incompatibility. To get closer to real-world applications, where the number of machines is limited and jobs have different processing times, each vertex of the graph requires multiple colors and the number of vertices with the same color is bounded. In addition, several objectives related to scheduling are considered: makespan, number of preemptions, and summation over the jobs' throughput times. Different solution methods are proposed, namely, two greedy heuristics, two tabu search methods and an adaptive memory algorithm. The latter uses multiple recombination operators, each one being designed for optimizing a subset of objectives. The most appropriate operator is selected dynamically at each iteration, depending on its past performance. Experiments show that the proposed algorithm is effective and robust, while providing high quality solutions on benchmark instances for the graph multi-coloring problem, a simplification of the considered problem.

Keywords: Graph coloring, scheduling, metaheuristics, tabu search, adaptive memory.

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

---

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

Dépôt légal – Bibliothèque et Archives nationales du Québec
Bibliothèque et Archives Canada, 2016

1 Introduction

The graph coloring problem (GCP) consists in coloring the vertices of a graph such that no pair of adjacent vertices has a common color, while minimizing the number of colors. This problem is NP-hard (Malaguti and Toth, 2010) and has attracted a lot of attention due to its simplicity and its numerous practical applications, for example, in scheduling (Marx, 2004), frequency assignment (Aardal et al., 2007) and exam timetabling (Lewis, 2007). This work examines its application to the scheduling of jobs on parallel machines. The GCP is used to account for pairs of incompatible jobs that cannot be processed simultaneously because they require a non-sharable resource (e.g., tools, operators, etc.). To this end, a so-called conflict graph is created where vertices represent jobs and edges connect incompatible pairs of jobs. Next, each color corresponds to a time slot and the total number of used colors corresponds to the makespan (i.e., completion time of all jobs). This is depicted in Figure 1 for a problem instance with five jobs. The picture on the left side shows the graph representation, whereas a production schedule is shown on the right side. The information inside (resp. next to) each vertex is the identifier of the job (resp. its color, represented as an integer number). Here, jobs a, b and c are mutually incompatible, as well as jobs c, d, and e.

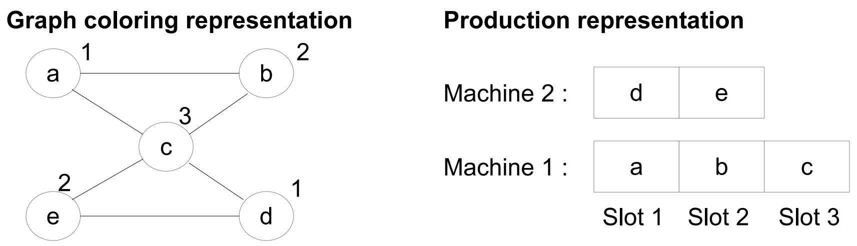

Figure 1: Graph coloring to model job scheduling on parallel machines

This model is however unrealistic for the following reasons. First, as each vertex requires a single color, the jobs must have identical processing times (i.e., a single time slot). Second, as there is no constraint on the number of vertices with the same color, the number of parallel machines is unlimited. To consider jobs with different processing times, the multi-coloring problem ($MGCP$) can be used. In this extension of the GCP, each vertex requires multiple colors. However, as the colors assigned to a vertex are not necessarily consecutive, preemption can occur (i.e., a job can be stopped and restarted later), which is usually undesirable in production systems. In addition, preemption increases the throughput time of the jobs (i.e., the total time spent by each job in the system). This is undesirable as it contributes to increase the inventory costs. In this work, we extend the multi-coloring problem by limiting the number of vertices with the same color and by considering the minimization of multiple objectives, corresponding to the makespan, number of preemptions and summation of the jobs' throughput times. A lexicographical ordering of these three objectives is considered (i.e., each objective is infinitely more important than any lower level one). This problem is denoted by P.

As P is an extension of the GCP, it is NP-hard too, and the use of heuristics and metaheuristics is indicated to obtain high quality solutions for instances of realistic size. In this work, we propose two greedy constructive heuristics, two tabu search methods and an adaptive memory algorithm. A mixed integer programming formulation is also proposed for solving small instances. The experiments show that the proposed methods are efficient for solving problem P. In addition, the adaptive memory algorithm provides high-quality solutions on $MCGP$ benchmark instances, even if this problem is a simplification of problem P. The development of metaheuristics for lexicographic multi-objective problems is a challenging task because it is difficult to efficiently optimize lower level objectives without degrading higher level ones. The proposed adaptive memory algorithm addresses this challenge by using different recombination operators and by dynamically selecting one operator at each iteration depending on its past performance.

The reader interested in additional information about metaheuristics is referred to (Gendreau and Potvin, 2010) and (Zufferey, 2012). A survey on the graph coloring problem and its extensions can be found in (Malaguti and Toth, 2010) and (Galinier et al., 2013).

The rest of the paper is organized as follows. Section 2 formally describes problem P, introduces a mixed integer programming model and discusses some practical applications. A literature review is then presented in Section 3. The greedy heuristics and tabu search methods are presented in Section 4, whereas our adaptive memory algorithm is described in Section 5. Finally, Section 6 reports computational results and a conclusion follows. From now on, the graph coloring terminology (e.g., vertex, color, edge) and the scheduling terminology (e.g., job, time slot, incompatibility) will be used interchangeably, depending on the context.

2 Formal description of problem P with scheduling applications

This section provides a formal description of the problem, introduces a mathematical programming formulation and motivates its study for scheduling applications.

2.1 Formal description

Given a graph $G(V, E)$, where V is the vertex set and E is the edge set, problem P consists in assigning a given number p_j of colors to each vertex j of V, where p_j is the processing time of job j. No two adjacent vertices (i, j) in E must have a common color. In addition, no more than m_t vertices can be assigned color t. That is, the size of each color class C_t (i.e., set of vertices with color t) must not exceed m_t. A color t such that $|C_t| = m_t$ is said to be saturated. Three objectives f_1, f_2 and f_3 must be minimized based on a lexicographic ordering, where each higher level objective is infinitely more important than any lower level objective: f_1 is the total number of used colors (makespan); f_2 sums, over all vertices, the number of interruptions in the sequence of colors assigned to a vertex (number of preemptions); f_3 sums, over all vertices, the range of colors assigned to a vertex, which is the difference between the largest and smallest colors (throughput time).

A mathematical programming formulation with five different types of variables is given below.

Its aim is to formally state problem P and to allow the computation of optimal solutions for small instances. Let α_j (resp. β_j) be the smallest (resp. largest) color assigned to vertex j, with k the largest color used over all vertices. Let $u_t = 1$ if color t is used and $u_t = 0$ otherwise. Finally, $x_{it} = 1$ if vertex i has color t and $x_{it} = 0$, otherwise. Equation (1) lists the three objectives to be minimized, where U is an upper bound on the required number of colors, typically set to $\sum_{j\in V} p_j$. Note that in objective f_2, there is an interruption when some color t is assigned to a vertex, but color $t-1$ is not (except for the smallest color assigned to each vertex, therefore n is subtracted at the end). Dummy variables x_{j0} are set to 0.

The problem is solved in three steps. First, f_1 is minimized while ignoring f_2 and f_3. Let us denote by $f_1^\star$ the optimal value of f_1. Next, the constraint $f_1 \le f_1^\star$ is added to the mathematical model and f_2 is minimized resulting in a value $f_2^\star$ for f_2. Finally, the constraint $f_2 \le f_2^\star$ is added and the model is solved for f_3, which leads to the optimal solution to problem P.

$$f_1 = k \qquad f_2 = \sum_{j\in V}\sum_{t=1}^{U}\big(x_{jt} \; - \; x_{j(t-1)}\big) - n \qquad f_3 = \sum_{j\in V}(\beta_j - \alpha_j) \tag{1}$$

The constraints in the model are the following. Constraint (2) states that each vertex must be colored with exactly p_j colors. Constraint (3) forbids the assignment of color t to more than m_t vertices. Constraints (4), (5) and (6) establish relationships between the variables x_{jt}, u_t, α_j, β_j and k. Constraint (7) computes the value of u_t and forbids two adjacent vertices to share the same color. Finally, the domain constraints are found in (8) and (9). Note that variables α_j, β_j and k will be integer in any optimal solution, so there is no need to force them to be integer. The resulting mixed integer program, which is denoted as $MILP$, was implemented and solved with CPLEX 12.5.

$$\sum_{t=1}^{U} x_{jt} \; = \; p_j \qquad j \in V \tag{2}$$

$$\sum_{j\in V} x_{jt} \; \le \; m_t \qquad 1 \le \; t \le U \tag{3}$$

$$t\cdot x_{jt} + U\cdot(1-x_{jt}) \ge \alpha_j \qquad 1 \le t \le U, \quad j \in V \tag{4}$$

$$t\cdot x_{jt} \; \le \; \beta_j \qquad 1 \le \; t \le U, \quad j \in V \tag{5}$$

$$k \; \ge \; t\cdot u_t \qquad 1 \le t \le U \tag{6}$$

$$x_{it} \; + \; x_{jt} \le u_t \qquad 1 \le t \le U, \;\; (i,j) \in E \tag{7}$$

$$x_{jt}, \; u_t \in \{0,1\} \qquad 1 \le t \le U, \quad j \in V \tag{8}$$

$$\alpha_j, \; \beta_j, \; k \; > \; 0 \qquad j \in V \tag{9}$$

2.2 Scheduling applications

The definition of problem P is motivated from scheduling applications (see (Pinedo, 2008) for a general reference on scheduling). We provide below some practical insights and motivations about the features of our problem: incompatible jobs, variable number of parallel machines, preemption penalties and job's throughput times.

As mentioned in (Thevenin, Zufferey, and Potvin, 2014), *incompatibilities* between jobs arise when some scarce resources are required for processing the jobs. Examples of such resources include expensive tools to equip the machines, employees with specific skills, etc. The tooling constraints are particularly relevant in flexible manufacturing systems (Hertz and Widmer, 1996). In addition, the authors in (Almeder and Almada-Lobo, 2011) mention three real-world examples of such scarce resources: syrup tanks in the bottling of soft drinks, testing heads in wafer testing and employees with specific skills in car production lines. When such resources come in a single exemplar, or if they have already been pre-assigned, two jobs requiring that common scarce resource are said to be incompatible. Note that incompatibility constraints have been studied in different types of production problems, like planning (Pesenti and Ukovich, 2003), assembly line balancing (Corominas, Pastor, and Plans, 2008) and lot-sizing and scheduling (Persi, Ukovich, and Pesenti, 2001).

Parallel machines is a typical setting in job scheduling. We assume here that the number m_t of available machines can be different in each time slot t, as one or more machines might be unavailable during a period of time due to preventive maintenance or to a variable workforce. The reader is referred to (Ma, Chu, and Zuo, 2010) for a comprehensive survey on scheduling with machine availability. Clearly, a special case of P is obtained when the number of machines is a constant (i.e., $m_t = m$ for all t).

Preemption might occur when the setup time is negligible when compared to the length of a time slot. But, generally speaking, preemption is not desirable and should be minimized (Liu and Cheng, 2002). Another drawback of preemption is the increase in the *throughput time* of a job, which is defined as the difference between the end time and the start time of a job. As a partially-filled job cannot be shipped to a customer, the job must stay on the production floor until its completion, which increases the work-in-progress inventory costs.

Finally, the lexicographic multi-objective optimization approach proposed here is common practice, as reported in (Solnon et al., 2008) and (Prats et al., 2010). This approach is very convenient when an ordering can be established among the objectives, as it is the case for problem P.

3 Literature review

To the best of our knowledge, problem P has never been studied in the literature. Subsection 3.1 presents some related work, whereas subsection 3.2 reviews the graph multi-coloring and bounded coloring problems.

3.1 Scheduling with job incompatibility

Scheduling with job incompatibility was considered in different environments and under different constraints, like precedence constraints (Meuwly, Ries, and Zufferey, 2010), batch scheduling (Epstein et al., 2009; Halldórsson, 2004; de Werra et al., 2005), multiprocessor task systems (Giaro, Kubale, and Obszarski, 2009) and project scheduling (Zufferey, Labarthe, and Schindl, 2012). Here, the literature review will focus on some recent work in parallel machine scheduling problems, with a bounded number of machines, job incompatibility constraints and different job processing times (a comprehensive review on parallel machine scheduling can be found in (Cheng and Sin, 1990)).

In (Even et al., 2009), the authors consider the problem of minimizing the makespan for a parallel machine scheduling problem with job incompatibility. The authors propose an exact algorithm for two machines and jobs with a processing time of one or two time units. They show that the problem is NP-hard when the processing time of a job is allowed to be as large as four time units. In addition, approximation methods are proposed for some special cases, including dynamic job arrivals. (Bendraouche and Boudhar, 2012) extend this work by showing that the problem with two machines and processing times of 1, 2 and 3 time units is NP-hard. They also show that when release dates are considered, the problem is NP-hard for jobs with processing times of only 1 or 2 time units. A polynomial-time exact method is proposed for a problem defined on a bipartite agreement graph (which is the complement of a conflict graph) and the same processing time for all jobs. Finally, some lower bounds and a greedy heuristic are reported for a more general case. (Lushchakova and Strusevich, 2010) consider two different sets of jobs to be scheduled on two different machines with incompatibilities between the jobs in each set (bipartite conflict graph). They propose an exact method running in linear time. (Hong, Sun, and Jou, 2009) describe a genetic algorithm for a parallel machine scheduling problem with minimization of the makespan. As each job requires a single mold, the considered incompatibilities exhibit a special structure.

Some of the papers mentioned above, by focusing on very specific problem cases, unveiled interesting theoretical properties, although of limited applicability. This is to be opposed to our goal, which is to develop methodologies for a more complex problem which is closer to what is observed in the real-world.

3.2 Multi-coloring and bounded coloring problems

The $MGCP$, where multiple colors are assigned to each vertex, has applications in domains like scheduling (Marx, 2004) and frequency assignment (Aardal et al., 2007). The solution methods proposed in the literature include exact methods such as branch-and-bound (Xue, 1998) and branch-and-price with column generation (Mehrotra and Trick, 2007; Gualandi and Malucelli, 2012). The $MGCP$ being NP-hard, metaheuristics are relevant for tackling medium and large instances. For example, (Lim et al., 2005) introduce a greedy heuristic, a tabu search, and a squeaky wheel optimization approach; (Satratzemi, 2004) proposes four greedy heuristics; and (Prestwich, 2008) describes a hybrid method involving local search and constraint programming.

Some recent applications of this problem in production scheduling are the following. The authors in Halldórsson and Kortsarz (2002) consider parallel machine scheduling with and without preemption. They study the minimization of two different objectives: the number of used colors (makespan) and the sum, over all vertices, of the largest color assigned to each vertex (sum of job completion times). They propose an approximation algorithm and a number of exact methods for graphs with a particular structure. (Méndez-Díaz and Zabala, 2010) consider a graph multi-coloring problem where only a preset number of conflicts is allowed. This situation corresponds to a production scheduling environment where conflicting resources can be shared up to a certain limit. They propose a mathematical programming formulation and a branch-and-cut method to solve it. (Blöchliger and Zufferey, 2013) model the problem of scheduling jobs in a parallel machine environment with preemption, incompatibility penalties and assignment costs, and propose problem-solving methods based on exact methods and metaheuristics. It is worth noting that minimizing the sum of the completion times in a parallel machine environment with preemption and job incompatibility can be modeled with a variant of the $MGCP$, called the sum multi-coloring problem (Bar-Noy et al., 2000).

To the best of our knowledge, the $MGCP$ has never been considered in a context where a color cannot be assigned to more than a given number of vertices. However, such constraints were added to the GCP, resulting in the bounded coloring problem (Baker and Coffman, 1996). Lower and upper bounds, as well as exact methods running in polynomial time on graphs with a special structure, are proposed in (Hansen, Hertz, and Kuplinsky, 1993). In fact, most papers focus on particular classes of graphs, such as interval graphs (Gardi, 2009), permutation graphs (Bonomo, Mattia, and Oriolo, 2011), permutation and overlap graphs (Demange, Stefano, and Leroy-Beaulieu, 2008), permutation and comparability graphs (Jansen, 1998) and trees (Jarvis and Zhou, 2001),

To summarize, this paper extends the literature on multi-coloring by introducing new constraints and objectives typically found in scheduling applications, thus reducing the gap between the scheduling and graph coloring research communities.

4 Greedy heuristics and tabu search methods

In this section, two greedy heuristics, denoted as GR^F and GR^{NF}, are first presented in subsections 4.1 and 4.2. Two tabu search methods, denoted as TS^F and TS^{NF}, are then proposed in subsections 4.3 and 4.4. GR^F and TS^F are based on a well known strategy in graph coloring where a feasible solution with a fixed number k of colors is looked for. If successful, the search is applied again with $k-1$ colors. Starting from some upper bound on the number of colors, the value of k is progressively reduced until no feasible coloring can be found. This is to be opposed to the search strategy of GR^{NF} and TS^{NF} where the number of colors is free. Based on preliminary experiments, the best way to combine these methods is described in subsection 4.5.

4.1 Greedy heuristic with fixed k (GR^F)

The greedy construction heuristic GR^F is an extension of the *Dsatur* method (Brélaz, 1979). Here, the vertices are colored one by one with the required number of colors in $\{1, 2, \ldots, k\}$.

Two metrics are used to select the next vertex to be colored: (1) the saturation degree $Dsat(j)$ of vertex j, which is the number of colors that cannot be assigned to j (because these colors are saturated or assigned to adjacent vertices), and (2) the number $\delta(j)$ of non-colored vertices adjacent to j. The selected vertex is the one that maximizes the $Dsat$ metric, with ties broken by selecting the vertex with maximum δ. The colors are then assigned one by one to the selected vertex. At each step, the color t leading to the minimum increase $\Delta Dsat(t)$ in the sum of the saturation degrees over all vertices in the graph is chosen, with ties broken randomly. Two cases can occur: if there are $m_t - 1$ vertices with color t, this color becomes saturated when it is assigned to the selected vertex and $\Delta Dsat(t)$ is then equal to the number of non-colored vertices; otherwise, $\Delta Dsat(t)$ is equal to the number of vertices, adjacent to the selected vertex, for which color t was admissible but is not anymore when this color is assigned to the selected vertex.

The method stops when the graph is fully colored or when there is a non-colored vertex j with a saturation degree larger than $k - p_j$ (such a vertex cannot be colored). Note that random decisions take place in GR^F when ties occur in the selection of the next vertex and in the color assignment. As different runs of the method are likely to produce different solutions, a more robust method can be obtained by restarting the procedure until some time limit is reached and by returning the best solution found.

4.2 Greedy heuristic with free k (GR^{NF})

GR^{NF} differs from GR^F in the assignment of colors to each vertex. GR^{NF} simply assigns the p_j smallest colors that are neither used by an adjacent vertex, nor saturated. In addition, since there is no constraint on the number of colors, the method stops only when all vertices are colored.

4.3 Tabu search with fixed k (TS^F)

Tabu search (Glover, 1986) is a local search method where small modifications (called moves) are performed on the current solution s to obtain new solutions in the so-called neighborhood of s, denoted by $N(s)$. The best neighbor solution (either improving or not) then becomes the new current solution for the next iteration. Tabu status are used to prevent cycling by forbidding some recently performed moves to be undone.

The search space considered in TS^F consists of feasible partial colorings of the graph, using at most k colors. That is, non-colored vertices are allowed. As mentioned in (Malaguti and Toth, 2010), this search space usually leads to the best results for the GCP and its extensions. To comply with the proposed search space, the first objective had to be modified. Now, the number of non-colored vertices is minimized (instead of the number of colors). Therefore, the tabu search is aimed at finding a coloring of the graph with k colors with as few non-colored vertices as possible, while minimizing f_2 and f_3. Figure 2 shows a solution for a problem where each vertex requires two colors and $k = 4$. In this case, one vertex is not colored because no color in the set $\{1, 2, 3, 4\}$ can be assigned to vertex b without creating a conflict.

A *move* $M_s(j)$ consists in assigning a set C of p_j colors (as described in the next paragraph) to a non-colored vertex j in the current solution s. To be sure that two adjacent vertices do not share the same color and that the bound m_t on the number of vertices with color t (for every color t) is not exceeded, the solution is repaired as follows. First, all colors are removed from the vertices adjacent to j with a color in set C. These vertices are said to be uncolored. Second, if a color t is assigned to more than m_t vertices, all colors are removed from one of these

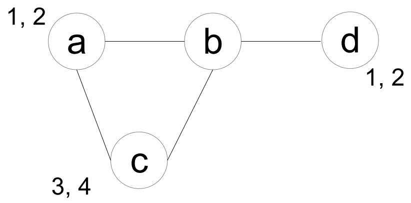

Figure 2: Possible solution in the TS^F search space

vertices (randomly selected but different from j). The *tabu status* forbids removing the colors of a recently colored vertex for tab^F iterations. This parameter was tested in the interval $[1, 100]$ and was finally set to 10.

The assignment of colors to vertex j depends on its saturation degree. If $Dsat(j) > k - p_j$ then the repair process is unavoidable. In this case, the p_j colors are chosen one by one and, at each step, the color leading to the minimum number of additional uncolored vertices is inserted in set C. If $Dsat(j) \leq k - p_j$ then it is possible to find p_j colors that are not saturated nor used by any adjacent vertex. These colors are added one by one to set C by selecting at each step the admissible color t that minimizes the increase of $\Delta Dsat(t)$.

All moves $M_s(j)$ are evaluated for every non-colored vertex j in the current solution s, and the best non-tabu move is performed at each iteration. A survey about this type of moves can be found in (Zufferey and Vasquez, 2014). In particular, this approach was successfully applied to the GCP in (Blöchliger and Zufferey, 2008).

4.4 Tabu search with free k (TS^{NF})

As opposed to TS^F, the search space considered in TS^{NF} consists of complete solutions without any restriction on the number of colors. As shown in Figure 3, a solution can be represented as a set of color classes, where all vertices in a given class share the same color. In this example, we have $k = 5$ with $|C_5| = 1$. The search strategy consists in minimizing the number of vertices with the largest color k (in the hope of removing this color). The objectives are modified accordingly: the first objective is to minimize k, the second is to minimize the size of color class C_k, whereas the third and fourth are the original objectives f_2 and f_3, respectively.

A *move* $M_s(j, t)$ consists in removing a vertex j from C_k and adding it to C_t with $t < k$ (i.e., j is given a smaller color). The solution is then repaired. Three situations can occur depending on $|C_t|$ and the set $\mathcal{J}$ of vertices in C_t adjacent to j.

1. If $\mathcal{J} = \emptyset$ and $|C_t| \leq m_t$ (after the addition), then vertex j is successfully added to color class C_t.
2. If $\mathcal{J} = \emptyset$ and $|C_t| = m_t + 1$ (after the addition), then a vertex in C_t is randomly chosen and moved to the smallest admissible color class. Note that a new color class can be created, which increases the value of k.

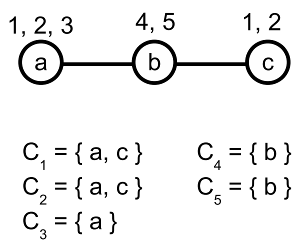

Figure 3: Representation of a solution as a set of color classes

3. If $\mathcal{J} \neq \emptyset$, the vertices in $\mathcal{J}$ are examined one by one and each vertex is moved from C_t to the smallest admissible color class (which can also create a new color class).

The *tabu status* associated with move $M_s(j,t)$ forbids removing vertex j from color class C_t for tab^{NF} iterations. This parameter was tested in the interval $[1, 100]$ and was finally set to 25. At each iteration, all non-tabu moves $M_s(j,t)$, $j \in C_k$, $t < k$, are considered and the best one is performed at the end.

4.5 A complete tabu search framework (TS)

During preliminary tests (reported in subsection 6.2), we observed that GR^F outperforms GR^{NF} whereas TS^{NF} outperforms TS^F. Accordingly, the following tabu search (TS) framework is proposed:

1. Apply GR^{NF} to find an initial upper bound k_0 on the required number of colors. Set $k = k_0 - 1$.
2. Try to find a feasible solution with k colors using TS^F. If a feasible solution is found, decrement k by one.
3. If 100 consecutive restarts of GR^F are performed without decreasing the value of k then go to step 4, otherwise return to step 2.
4. Apply TS^{NF} starting from the best feasible solution found in step 2.

5 Adaptive memory algorithms

Algorithm 1 shows the basic steps of an adaptive memory algorithm (AMA), as initially proposed in (Rochat and Taillard, 1995). Recent and successful adaptations of adaptive memory algorithms to the GCP can be found in (Galinier, Hertz, and Zufferey, 2008), (Lü and Hao, 2010a), (Lü and Hao, 2010b). These successes motivate the choice of AMA for our problem. In the following, subsection 5.1 describes the memory structure, the initialization step and the updating process.

Next, in subsection 5.2, three different recombination operators are proposed, where each one is effective on a subset of the three original objectives. Finally, an algorithm using the three recombination operators is presented in subsection 5.3.

Algorithm 1 Adaptive memory algorithm (AMA)

```
Initialize the memory $\mathcal{M}$ with maximum stable sets.
While a stopping condition is not met do
    1.1 Create a solution $s$ from $\mathcal{M}$ with a recombination operator.
    1.2 Improve $s$ with the tabu search $TS^{NF}$ during $I$ iterations.
    1.3 Update $\mathcal{M}$ with $s$.
```

5.1 Memory structure

As mentioned in subsection 4.4, a solution s is represented as a set of color classes $S = (C_1, C_2, \ldots)$, where C_t is the set of vertices with color t. Clearly, each color class in a feasible coloring corresponds to a stable set (i.e., a set of non-adjacent vertices). Thus, our adaptive memory $\mathcal{M}$ contains maximum stable sets, that is, stable sets that are not contained in any other stable set. The memory also records the number $h(W)$ of times that the maximum stable set W was obtained from solutions returned by the tabu search (see step 1.3 of Algorithm 1), as well as the average objective function values of these solutions, denoted by $f_1(W), f_2(W), f_3(W)$. We also have $f(W) < f(W')$ if the solution from which stable set W was obtained is better than the solution from which stable set W' was obtained, based on the lexicographic ordering of the three objectives f_1, f_2 and f_3. Note that a similar memory structure was used in (Galinier, Hertz, and Zufferey, 2008).

To start the algorithm, p randomly generated solutions are first created and improved with TS^{NF} during I iterations. Maximum stable sets are then extracted from these solutions to initialize the memory $\mathcal{M}$. Parameter p was tested in the interval $[1, 100]$ and set to 20, whereas parameter I was tested with different values between 50 and 100,000 and was finally set to $2,000$. Next, the basic loop creates a starting solution for the tabu search by combining stable sets in $\mathcal{M}$ in step 1.1, running the tabu search in step 1.2, and updating the memory with the solution returned by the tabu search in step 1.3.

Just before updating $\mathcal{M}$ in step 1.3 of Algorithm 1, every stable set C_t in a solution returned by the tabu search is transformed into a maximum stable set with the iterative procedure *Extend*. More precisely, for every stable set C_t, *Extend* performs the following steps, where Ω is the set of vertices that are not in the current stable set W (initially equal to C_t) and are not adjacent to any vertex of W:

1. set $W = C_t$;

2. while $\Omega \neq \emptyset$, do: (a) select a vertex $j \in \Omega$; (b) add j to W; remove j and its adjacent vertices from Ω.

In step 2(a), the vertex $j \in \Omega$ with the least number of adjacent vertices in Ω is selected to favor the creation of a large stable set W. In case of ties, the vertex j that maximizes p_j/κ_j is chosen, where κ_j is the number of stable sets in $\mathcal{M}$ that contains vertex j. Therefore, a vertex j with a large p_j value and only a few occurrences in $\mathcal{M}$ is favored.

If the stable set returned by the procedure *Extend* is already in $\mathcal{M}$, $h(W)$ is increased by one and $f(W)$ is updated accordingly. Otherwise, W is added to $\mathcal{M}$ and another stable set W' is removed from $\mathcal{M}$ to keep a constant memory size. If W' is the worst stable set in the memory and $f(W) < f(W')$, then W replaces W'. Otherwise, diversity is favored by removing the stable set W' that is the most similar to the other stable sets in $\mathcal{M}$. The similarity measure for W is $Sim_{\mathcal{M}}(W) = \sum_{i,j \in W} r_{ij}$, where r_{ij} is the number of stable sets in $\mathcal{M}$ that contain both vertices i and j. Clearly, W' is the stable set in $\mathcal{M}$ for which this measure is maximized.

The next subsection will now explain how different stable sets in the memory are combined to form an initial solution for the tabu search (step 1.1 of Algorithm 1).

5.2 Recombination operators

Three different recombination operators are proposed here. Subsection 5.2.1 describes operators $\mathcal{R}^{Deg}$ and $\mathcal{R}^{Cost}$ aimed at coloring the graph with as few stable sets as possible to minimize f_1. In contrast, the recombination operator $\mathcal{R}^{NoInt}$ presented in subsection 5.2.2 returns solutions with as few interruptions as possible, in order to minimize f_2 and f_3.

5.2.1 Recombination operators $\mathcal{R}^{Deg}$ and $\mathcal{R}^{Cost}$

$\mathcal{R}^{Deg}$ and $\mathcal{R}^{Cost}$ generate a solution from the memory in a similar way, therefore they are presented together in this subsection. Both operators proceed in two phases. The *construction phase* first uses the memory $\mathcal{M}$ to sequentially build a set S of stable sets (without assigning colors yet). The *color assignment phase* assigns then a color label in $\{1, \ldots, |S|\}$ to each stable set in S (each color label can only be used once) while minimizing f_2 and f_3. These two phases are presented below with an example at the end.

Construction phase. This phase tries to generate as few stable sets as possible in S to minimize f_1. To allow a feasible solution to be obtained in the following *color assignment phase*, the two conditions below must be satisfied.

(C1) Each vertex must appear in exactly p_j stable sets of S (otherwise vertex j cannot be fully colored).

(C2) There exists at least one color assignment to the stable sets in S such that each stable set

C_t contains m_t vertices or less (i.e., for each color t, the number of vertices with this color does not exceed the bound m_t).

The construction method, which is described in Algorithm 2, uses the following notation:

- k is the currently best known value for objective f_1.
- V is the set of vertices in the graph.
- S is the current set of stable sets (iteratively constructed).
- Z is the set of vertices j that appear p_j times in S (i.e., vertices that can be fully colored).
- Θ is the set that contains the different values of m_t, for $t \in \{1, \ldots, k\}$. For instance, with $k = 5$, if $m_1 = m_2 = 2$ and $m_3 = m_4 = m_5 = 1$, $\Theta = \{1, 2\}$ as m_t takes either value 1 or 2.
- For each $m \in \Theta$, a_m is the number of colors $t \in \{1, \ldots, k\}$ such that $m_t = m$ (i.e., the number of colors with bound m). Note that $\sum_{m\in\Theta} a_m = k$. For the above example, $a_1 = 3$ and $a_2 = 2$.
- For each $m \in \Theta$, Ψ_m is a set of stable sets that can be assigned to any color with bound m.

The general flow of Algorithm 2 works as follows. After initialization, k stable sets that satisfy condition (C2) are constructed in step 2.2. Indeed, at the end of this step, to obtain a color assignment respecting the bounds, it suffices to assign the a_m colors with bound m to the stable sets in Ψ_m, for each $m \in \Theta$. However, it might happen that condition (C1) is not satisfied at the end of step 2.2. In this case (i.e., if $Z \neq V$), a new stable set W is created in step 2.4 for time unit (color) $K = k + 1$. This is repeated with increasing values of K until all vertices can be fully colored. At the end, the set S of all generated stable sets is returned.

Step 2.2(a) needs further explanation. First, a set Γ_m of candidate stable sets (generated from memory $\mathcal{M}$) is built through a *candidate generation procedure*. A stable set $W \in \Gamma_m$ is then selected among these candidates and added to Ψ_m during a *candidate selection procedure*. This is explained below.

The *candidate generation procedure* produces the largest possible stable sets (in order to minimize f_1) but with no more vertices than the considered value m. Let L be the size of a largest stable set in $\mathcal{M}$. Two situations can occur when building Γ_m. If $L \leq m$, then all stable sets in $\mathcal{M}$ of cardinality L are added to Γ_m. Otherwise, all stable sets with more than m vertices are added to Γ_m, but are reduced to only m vertices thereafter. If we define $\delta(j)$ as the degree of vertex j in $V - Z$, the reduction process of a stable set $W \in \Gamma_m$ removes at each iteration the vertex j that minimizes $\delta(j)$ (as such vertices will be easier to color later) until the number of vertices is equal to m.

The *candidate selection procedure* differs for $\mathcal{R}^{Deg}$ and $\mathcal{R}^{Cost}$. $\mathcal{R}^{Deg}$ is degree-oriented and, consequently, it selects the stable set $W \in \Gamma_m$ that maximizes $\sum_{j\in W} \delta(j)$. On the other hand, $\mathcal{R}^{Cost}$ is cost-oriented and selects the stable set $W \in \Gamma_m$ based on the probability distribution

Algorithm 2 *Construction phase* of the recombination operators $\mathcal{R}^{Deg}$ and $\mathcal{R}^{Cost}$

```
2.1 Initialization: set $S = \emptyset$, $\Psi_m = \emptyset$ (for each $m \in \Theta$ ) and $Z = \emptyset$.

2.2 For each $m \in \Theta$ (in decreasing order), create $a_m$ stable sets by repeating $a_m$ times the
    following steps.
    (a) Generate a stable set $W$ from $\mathcal{M}$ (with maximum cardinality $m$) and add it to the set
        $\Psi_m$.
    (b) Compute $Z$ and for each vertex $j \in Z$, remove $j$ from the stable sets of $\mathcal{M}$ that contain
        it (i.e., vertex $j$ can be ignored, as it is now possible to fully color it).
    (c) Augment the stable sets of $\mathcal{M}$ involved in step 2.2(b) by applying the procedure *Extend*
        to each one of them, using the subgraph induced by the vertices in $V - Z$ (this step is
        likely to favor larger stable sets in $S$, which helps to minimize $f_1$).
    Set $S = S \cup \Psi_m$.

2.3 If $Z = V$, go to step 2.5 (all vertices can be fully colored). Otherwise, set $K = k$.

2.4 While $Z \neq V$, do: set $K = K + 1$ and create a stable set with at most $m = m_K$ vertices by
    performing step 2.2.

2.5 Return $S$.
```

$Pr(W, \Gamma_m)$ shown in Equation (10). The probability $Pr(W, \Gamma_m)$ involves a function $g(W)$ which is made of two components: the number of solutions $h(W)$ returned by the tabu search that produced the maximum stable set W stored in memory $\mathcal{M}$ (quantity) and the average objective function values $f_i(W)$ (with $i \in \{1, 2, 3\}$) of the solutions returned by the tabu search that produced W (quality). Three different $g(W)$ functions were tested, as shown in Equation (11), where $\bar{f}_i$ is the normalized value of objective f_i. The best results were obtained with $\frac{h}{f_1}$, thus indicating that the focus should be on the quality of the first objective. Therefore, from now on, $g(W) = \frac{h}{f_1}$.

$$Pr(W, \Gamma_m) = \frac{g(W)}{\sum_{W' \in \Gamma_m} g(W')} \tag{10}$$

$$g(W) = \frac{h(W)}{f_1(W)} \quad , \quad g(W) = \frac{h(W)}{f_2(W)} \quad , \quad g(W) = \frac{h(W)}{\bar{f}_1(W) + \bar{f}_2(W) + \bar{f}_3(W)} \tag{11}$$

Color assignment phase The *color assignment phase* must assign a color to each stable set in S, while minimizing f_2 and f_3. Let us denote $\Delta_2^t(W)$ and $\Delta_3^t(W)$, respectively, the increase in the objectives f_2 and f_3 if color t is assigned to a stable set W. The color assignment phase then takes place as follows. The color $t = 1$ is first assigned to a randomly selected stable set $W \in \Psi_{m_1} \subseteq S$ and set W is then excluded from further consideration. Then, for each color t in increasing order, the stable set that minimizes $\Delta_2^t(W) \in \Psi_{m_t} \subseteq S$ is assigned color t and excluded from further consideration, with ties broken by selecting the stable set that minimizes $\Delta_3^t(W)$. This procedure is repeated until all stable sets in S are done.

The recombination operators $\mathcal{R}^{Deg}$ and $\mathcal{R}^{Cost}$ are now illustrated on a small example, based on the graph of Figure 2. We assume the following: the vertices a, b, c and d require 2, 1, 2 and 2 colors, respectively; also, 2 (resp. 1) machines are available during time units 1 and 3 (resp. 2, 4, 5, 6); $k = 5$; $\mathcal{M} = \{\{a, d\}, \{c, d\}, \{b\}\}$. Therefore, $\Theta = \{1, 2\}$ (set of different m_t values), $a_1 = 3$ (for colors 2, 4 and 5) and $a_2 = 2$ (for colors 1 and 3). $\mathcal{R}^{Deg}$ and $\mathcal{R}^{Cost}$ first create the set Ψ_2. In step 2.2(a) of Algorithm 2, there are two stable sets of cardinality 2, namely $\{a, d\}$ and $\{c, d\}$. $\mathcal{R}^{Deg}$ selects a stable set randomly, as the sum of the degrees in both candidate stable sets is equal to 3. $\mathcal{R}^{Cost}$ selects a stable set based on Equation (10). Let us assume that $\{a, d\}$ is selected. A second stable set of size 2 is then required in Ψ_2 because $a_2 = 2$. Therefore, the same procedure is repeated, leading to $\Psi_2 = \{\{a, d\}, \{a, d\}\}$. Now $Z = \{a, d\}$, as these vertices can be fully colored. Afterward, the central memory is updated to $\mathcal{M} = \{\{c\}, \{b\}\}$. Step 2.2 is then performed again to build $\Psi_1 = \{\{c\}, \{c\}, \{b\}\}$. In the *color assignment phase*, the stable set $\{a, d\}$ from Ψ_2 is assigned color 1 because $m_1 = 2$. The stable set assigned to color 2 is chosen randomly among $\{b\}$ and $\{c\}$, as both would increase f_2 and f_3 by 2 units (i.e., vertices a and d would be interrupted and their completion times would increase by one time unit). Assuming that $\{b\}$ is selected, the next steps assign color 3 to the stable set $\{a, d\}$, and colors 4 and 5 to the stable set $\{c\}$. Thus, the solution s obtained at the end corresponds to $S = (C_1, \ldots, C_5) = (\{a, d\}, \{b\}, \{a, d\}, \{c\}, \{c\})$, with $f_1 = 5$, $f_2 = 2$ and $f_3 = 5$.

5.2.2 Recombination operator $\mathcal{R}^{NoInt}$

The recombination operator $\mathcal{R}^{NoInt}$ creates a solution with stable sets that are similar to those in $\mathcal{M}$, but with no interruption, in order to minimize f_2 and f_3. The tabu search in step 1.2 of Algorithm 1 will then tend to minimize f_1, while trying not to increase too much f_2 and f_3.

The recombination operator is described in Algorithm 3, where all color classes are initially empty. The process starts with $t = 1$ (where t denotes the current color) and $Z = \emptyset$. The operator colors the vertices one by one. At each iteration, two main steps are performed. First, a vertex j is selected from a set Q of candidates generated in step 3.1. Then (assuming $Q \neq \emptyset$), the consecutive colors $\{t, t+1, \ldots, t+p_j-1\}$ are assigned to vertex j. It should be noted that no conflict is created when assigning these consecutive colors to j, because the color classes C_t' with $t' > t$ will necessarily be subsets of C_t. However, some colors (larger than t) might become saturated (where a color class $C_{t'}$ is saturated if $|C_{t'}| = m_{t'}$). Therefore, the selected vertex j in Q is the one with the largest p_j which can be given colors $\{t, t+1, \ldots, t+p_j-1\}$ without over-saturating a color class. Vertices with large p_j values are considered first, as they are more difficult to fully color. If no such vertex exists, the current color is set to $t+1$ and the procedure is repeated until all vertices are fully colored.

Algorithm 3 Recombination operator $\mathcal{R}^{NoInt}$

```
Initialization: set $t = 1$ (current color) and $Z = \emptyset$.
While $Z \neq V$, do
  3.1 Create a set $Q$ of candidate vertices as follows.
      (a) If $C_t$ is a maximum stable set or if $|C_t| = m_t$, set $Q = \emptyset$ and go to step 3.3.
      (b) Compute $\Phi = \{W \in \mathcal{M} \mid C_t \subseteq W\}$.
      (c) If $\Phi \neq \emptyset$, a stable set $W'$ is selected in $\Phi$ based on the probability $Pr(W, \Phi)$ in Equation (10).
      (d) If $\Phi = \emptyset$, the procedure Extend is used to generate a maximum stable set $W'$ from $C_t$.
      (e) Set $Q = W' - C_t$.
  3.2 Remove from $Q$ every vertex $j$ which is either in $Z$ or would over-saturate a color class if colored with $p_j$ consecutive colors (starting from color $t$).
  3.3 If $Q = \emptyset$, set $t = t + 1$ (no vertex can be colored with color $t$). Otherwise, select the vertex $j \in Q$ with the largest $p_j$ (break ties randomly), and put $j$ in the consecutive color classes $C_t, C_{t+1}, \dots, C_{t+p_j-1}$.
  3.4 Compute $Z$ and for each vertex $j \in Z$, remove $j$ from all stable sets of $\mathcal{M}$ that contain it.
```

This operator is illustrated below, based on the graph and memory $\mathcal{M}$ used in the example at the end of Section 5.2.1. The method starts by building C_1 which is initially empty. Thus, for the first iteration, Φ corresponds to all stable sets in memory (i.e., $\Phi = \{\{a,d\},\{c,d\},\{b\}\}$) and one of them is selected using Equation (10). If $W' = \{a,d\}$ then $Q = \{a,d\}$, as C_1 is empty. Now, let us assume that vertex a is selected (randomly, because $p_a = p_d = 2$) and that colors 1 and 2 are assigned to it. Then $C_1 = \{a\}$, $Z = \{a\}$ and Φ is computed again in step 3.1(b), leading to $\Phi = \{\{a,d\}\}$. Therefore, Q is empty as a is in Z and d requires two colors that would over-saturate the color class associated with time unit 2. The current set under investigation becomes C_2. As it already contains the vertex a, it is saturated. The method then considers C_3. At this time, $\Phi = \{\{c\},\{b\},\{d\}\}$, and one of those sets is selected using Equation (10). If $\{c\}$ is chosen, it is assigned colors 3 and 4, and those color classes are now saturated ($C_3 = C_4 = \{c\}$). After two additional iterations, $C_5 = \{b\}$, $C_6 = \{d\}$ $C_7 = \{d\}$. Thus, the solution s obtained at the end corresponds to $S = (C_1, \dots, C_7) = (\{a\},\{a\},\{c\},\{c\},\{b\},\{d\},\{d\})$, with $f_1 = 7$, $f_2 = 0$ and $f_3 = 3$.

5.3 Resulting adaptive memory algorithm (AMA)

Some experiments reported in subsection 6.3 show that R^{Cost} and R^{Deg} are well adapted to f_1 and R^{NoInt} to f_2 and f_3. To get a robust algorithm, the following AMA using the three recombination operators is proposed. At each iteration, a recombination operator R is selected in the set $\mathcal{R} = \{R^{Cost}, R^{Deg}, R^{NoInt}\}$ based on its past performance. More precisely, a value $score(R)$ is associated with each operator R. The selection probability of R is then $\frac{score(R)}{\sum_{R \in \mathcal{R}} score(R)}$. After running the tabu search on the initial solution obtained from the selected operator R, its score is updated as follows: if the resulting solution is a new best solution then $score(R) = score(R) + 2$; otherwise, if it is better than the solution obtained at the previous iteration then $score(R) = score(R) + 1$.

6 Numerical results

We first describe in subsection 6.1 how we generated test instances for our problem. Computational results for the greedy heuristics and the tabu search methods are then reported in subsection 6.2. A comparison of the various recombination operators used in the adaptive memory algorithm is found in subsection 6.3. Finally, subsection 6.4 describes the results obtained with the best algorithmic configurations.

The C++ language was used to implement the algorithms and the tests were performed on a processor *Intel(R) Xeon(R) CPU E5-2660 2.20GHz.* Every method was run five times on each instance, with a time limit of n minutes for each run (where n is the number of vertices in the considered instance).

6.1 Generation of test instances

The instance generator is inspired from the one used in (Galinier, Hertz, and Zufferey, 2008). First, the number of vertices n is chosen in the set $\{10, 20, 30, 40, 50, 100, 150, 200\}$. Second, the graph density d, which is the probability for an edge to be present between two vertices, is chosen in the set $\{0.2, 0.5, 0.8\}$. The last parameter β is related to the color bounds. A subset of instances was created with a constant bound $m_t = \beta$ for all t, with β belonging to $\{3, 5, 10\}$. In the remaining instances, the bound is not necessarily the same for each color. The bounds are generated with three different distributions called *small*, *medium* and *large*. A distribution is characterized by a mean μ, a standard deviation σ, a maximum value max, and a minimum value min, as listed in Table 1. The m_t values are cyclic, as it is observed in practice, and the size of a cycle is set to 10 (i.e., $m_1 = m_{11} = m_{21},\ m_2 = m_{12} = m_{22},\ m_3 = m_{13} = m_{23},\ \ldots$). The number p_j of colors required for each vertex j is chosen uniformly in the interval $[1, 10]$. Three instances are generated for each triplet (n, d, β). This leads to a total of 432 instances (since there are 8 values for n, 3 values for d and 6 values for β).

β	min	max	μ	σ
small	1	5	3	0.5
medium	2	10	6	2
large	5	15	10	3

Table 1: Parameters of the β distribution used to generate the m_t values

6.2 Experiments with the greedy heuristics and tabu search

This subsection analyzes the performance of the four methods proposed in Section 4, using the instances of size $n = 100$. Remember that GR^F and TS^F are the greedy and tabu search with a fixed number of colors, whereas the number of colors is free in GR^{NF} and TS^{NF}. Table 2 shows the average value of each objective (f_1, f_2, f_3) obtained with GR^{NF} and TS^{NF}. For GR^F and TS^F, four values are given: the minimum value of f_1 (number of colors) with all vertices fully colored, the rate of successful runs (over five runs) with that f_1 value, and the average values of f_2 and f_3 over the successful runs. Each line in this table corresponds to the average obtained over five runs on the three instances associated with the triplet (n, d, β). The entries in bold indicate that the corresponding method performed the best.

d	β	GR^F	GR^{NF}	TS^F	TS^{NF}
0.2	3	(175.3, 5/5, 48, 871.5)	(175.3, 42.3, 923.3)	(175.3, 5/5, 372.2, 8777.3)	**(175.3, 13.5, 810.9)**
	5	(103.3, 5/5, 67, 966.8)	(105.3, 74.3, 1135.7)	(103.3, 5/5, 353.1, 5498.5)	**(103.3, 18, 774.8)**
	10	(55.3, 3.7/5, 153, 1505)	(62.3, 148.3, 1577.3)	(55, 3.3/5, 367.2, 3267.4)	**(55, 28.7, 819)**
	small	**(176.7, 5/5, 105.1, 1195.3)**	(177.3, 113, 1414.3)	(176.7, 5/5, 407.7, 9688.6)	(177.3, 44, 1616.3)
	medium	(94, 4.7/5, 129.7, 1488.7)	(95.7, 133, 1589)	(94, 5/5, 383.1, 5247.1)	**(94, 41.1, 1157)**
	large	(54.7, 4/5, 143.9, 1524.8)	(59.7, 142.7, 1711.3)	(54.3, 5/5, 316.3, 2751.8)	**(54, 37.1, 875.9)**
0.5	3	(189.7, 5/5, 83.5, 1833.1)	(192, 69.7, 1936.3)	(189.7, 5/5, 430.8, 11050.1)	**(189.7, 28.5, 1405.6)**
	5	(110, 3.7/5, 133.3, 2471.1)	(117.3, 108, 2459.7)	(109.3, 5/5, 397.9, 6442.5)	**(108, 82.8, 2180.7)**
	10	(94.3, 2.3/5, 169.4, 2820.1)	(102, 124.7, 2877.3)	(84, 2.3/5, 294.5, 3859.2)	**(81.7, 250.2, 3971.7)**
	small	(191, 4.7/5, 166.6, 2735.4)	(194, 150, 2946)	(191, 5/5, 427, 11198.4)	**(191, 81.9, 2783.7)**
	medium	(110.3, 3.3/5, 168.9, 3037.4)	(117.3, 135, 3127)	(104.7, 3.7/5, 389.2, 5742.1)	**(98.6, 141.6, 3043.3)**
	large	(95.3, 2.3/5, 147, 2808.2)	(102, 121.3, 2722.7)	(85.7, 3.7/5, 281.4, 3706.6)	**(82.4, 257.4, 4011.5)**
0.8	3	(188, 2/5, 114.7, 3867)	(200.3, 69.7, 3492.3)	(186.3, 3.3/5, 424.3, 11273.6)	**(182.3, 82.2, 3460.3)**
	5	(169.7, 3.3/5, 114.7, 3754.5)	(180, 75, 3694.4)	(159, 3/5, 294.1, 5764.4)	**(152.1, 212.3, 6231.1)**
	10	(184.7, 2/5, 117.5, 4379.1)	(199.7, 79, 4251.3)	(170.3, 1/5, 292, 6245)	**(164.9, 244.3, 7233.6)**
	small	(186.7, 2/5, 144.8, 4479.2)	(192, 89.3, 4279.8)	(178.3, 2.7/5, 414.4, 10373.9)	**(168.7, 152.4, 4970.6)**
	medium	(178, 3.3/5, 114.4, 3851.1)	(188.3, 71.7, 3731.5)	(165.7, 2.7/5, 345.2, 7600.5)	**(159.1, 228.9, 6693)**
	large	(177, 2.7/5, 122, 3807.5)	(188.7, 67, 3529.2)	(167, 2.7/5, 313.7, 6759.3)	**(160.2, 231.4, 6870.8)**

Table 2: Comparison of the greedy and tabu search algorithms ($n = 100$)

These results indicate that GR^F outperforms GR^{NF}. With regard to f_1, the minimum number of colors produced by GR^F with all vertices fully colored is smaller than the average value of GR^{NF} on 17 (n, d, β) triplets, whereas the two heuristics are the same on the other one. Conversely, and in contrast with the state-of-the-art graph coloring methods which are based on a fixed number of colors, TS^{NF} clearly outperforms TS^F. First, the average value of objective f_1 obtained by TS^{NF} is strictly smaller than the minimum number of colors identified by TS^F on 11 triplets. Otherwise, the two methods are the same on 6 triplets and TS^F is the best on the remaining one. Furthermore, TS^{NF} finds better values for f_2 and f_3 when the two methods are the same with regard to f_1.

As expected, the tabu search methods outperform the greedy heuristics. With regard to f_1, the instances with $d = 0.2$ and $\beta = small$ are the only ones for which a greedy heuristic outperforms a tabu search, namely TS^{NF}. Furthermore, when there is a tie, TS^{NF} is always better on f_2 and f_3. TS^F is always better than or equal to the two greedy heuristics on f_1. But when there is a tie, GR^F outperforms TS^F with regard to f_2 and f_3. These results indicate that TS^F is well adapted to f_1, but not to f_2 and f_3.

6.3 Experiments with adaptive memory algorithms

This subsection compares the various recombination operators proposed for the adaptive memory algorithm. Remember that $\mathcal{R}^{Deg}$ (resp. $\mathcal{R}^{Cost}$) creates an initial solution for the tabu search by selecting stable sets in the memory with vertices of large degree (resp. stable sets that were part of good solutions). On the other hand, $\mathcal{R}^{NoInt}$ builds a solution without any interruption in the sequence of colors assigned to a vertex. AMA^{Deg}, AMA^{Cost} and AMA^{NoInt} are the adaptive memory algorithms using the recombination operators $\mathcal{R}^{Deg}$, $\mathcal{R}^{Cost}$ and $\mathcal{R}^{NoInt}$, respectively. They were tested on the instances of size $n = 100$. Table 3 reports the average values of each objective using the format (f_1, f_2, f_3).

With regard to objective f_1 only, AMA^{Cost} is strictly better than AMA^{NoInt} on 6 triplets out of 18 (12 ties). On the other hand, AMA^{Deg} is better than AMA^{NoInt} on 6 triplets, but worse on 2 (10 ties). Finally, AMA^{Cost} is strictly better than AMA^{Deg} on 3 triplets and worse on 1 (14 ties). When there is a tie between the three methods, we might conjecture that the minimum number of colors has been found. In these cases, AMA^{NoInt} obtains better average values on f_2 and f_3. Therefore, we conclude that AMA^{Cost} is slightly better than AMA^{Deg} with regard to f_1. However AMA^{NoInt} performs much better than the two other methods on f_2 and f_3. As a consequence, AMA^{Cost} or AMA^{Deg} should first be used, followed by AMA^{NoInt} when f_1 cannot be improved further. This is exactly the goal of the dynamic operator selection proposed in subsection 5.3.

6.4 Experiments with the best algorithms

This subsection reports results obtained with the tabu search framework TS proposed in subsection 4.5 which takes advantage of the strengths of the previous greedy and tabu search methods. In addition, results are reported with the AMA of subsection 5.3 using the three recombination operators. First, a comparison with optimal solutions is reported for small instances with up to 30

n	d	β	AMA^{Deg}	AMA^{Cost}	AMA^{NoInt}
100	0.2	3	(175.3, 7.7, 583.7)	(175.3, 7.4, 614.3)	**(175.3, 0, 524.7)**
		5	(103.3, 22.7, 810.9)	(103.3, 24, 836.1)	**(103.3, 0.4, 516.9)**
		10	(55, 41.3, 965.7)	(55, 61.2, 1250.5)	**(55, 28.6, 928.5)**
		small	(177.3, 52.5, 1888.7)	(177.3, 52.3, 1898.9)	**(177.3, 32.1, 1428.5)**
		medium	(94, 58.8, 1283.3)	(94, 58.7, 1301.8)	**(94, 27.9, 1040.3)**
		large	(54, 72.5, 1283.3)	(54, 72.5, 1300.3)	**(54, 46.4, 1042.5)**
	0.5	3	(189.7, 29.7, 1523.5)	(189.7, 36.3, 1418.9)	**(189.7, 0.3, 573.7)**
		5	(108, 62.5, 2001.9)	(108, 88.8, 2489.7)	**(108, 36.9, 1583.1)**
		10	**(81.3, 210.6, 3532.9)**	(81.4, 184.7, 3421)	(81.5, 228.2, 3568.7)
		small	(191, 100.2, 2400)	(191, 100.1, 2396.6)	**(191, 65.9, 2248.6)**
		medium	(97.1, 242.6, 4222.7)	**(97.1, 218.9, 4032.1)**	(99.3, 147.3, 3038.7)
		large	(82.5, 237.5, 3884.9)	(82.4, 239.3, 3982.9)	**(82.4, 232.2, 3869.7)**
	0.8	3	(182.3, 44.3, 2596.9)	(182.3, 72.2, 3355.9)	**(182.3, 42.2, 2315.5)**
		5	(152.1, 194.2, 5772.3)	**(152.1, 149.8, 4828.3)**	(152.2, 189.1, 5558.5)
		10	(165, 202.5, 6394.3)	**(164.6, 165.7, 5588.3)**	(165.1, 201.2, 6414.6)
		small	(167, 201.1, 4517.4)	**(167, 192.7, 4942.2)**	(170.3, 141.7, 4718.5)
		medium	(159.1, 223.5, 6509.3)	**(159.1, 219.4, 6334)**	(159.2, 198.5, 5859.5)
		large	(160.4, 217.2, 6578.5)	(160.3, 217.8, 6261.5)	**(160.3, 191.4, 6060.1)**

Table 3: Comparison of the recombination operators

vertices in subsection 6.4.1. These optimal solutions were obtained by solving the mixed integer program of Section 2, which is denoted as $MILP$. Next, tests were performed for larger instances with up to $n = 200$ vertices in subsection 6.4.2. The results indicate that TS and AMA can solve problems of realistic size with hundreds of jobs, as exemplified by the manufacturing test bed in (Le Pape, 2007)). Finally, AMA is compared with state-of-art methods on the $MGCP$ in subsection 6.4.3.

6.4.1 Results for small instances (with up to 30 vertices)

Our mixed integer program was solved with CPLEX 12.5 with a time limit of 10 hours for each objective (for a total of 30 hours). The results are shown in Table 4. There is a $*$ besides a given objective when the optimum is found on the three instances of the corresponding (n, d, β) triplet. The average value of each objective (f_1, f_2, f_3), along with the average time (in seconds) to reach the best solutions, are indicated for both TS and AMA. $MILP$ finds the optimal solutions for the three objectives on 3 triplets with $n = 10$. It also finds the optimal value of f_1 for all but one triplet with $n = 10$, for all but 3 triplets with $n = 20$, for all triplets with $n = 30$, $d = 0.2$ and for one triplet with $n = 30$, $d = 0.5$. As $MILP$ already struggles for instances of size 30, this is is a clear indication that it is not suitable for larger instances.

TS does a good job on objective f_1, but the gap between $MILP$ and TS can be substantial on f_2 and f_3. When compared to $MILP$, TS finds a better or equal f_1 value on 47 triplets (out of 54). However, there is a tie on 40 triplets (out of 47) and TS is not very good with regard to the second and third objectives. For example, TS finds solutions with more than 5 additional interruptions on 10 triplets. AMA leads to much better results. In fact, AMA is better than or equal to $MILP$ on 46 triplets (out of 54), when considering the three objectives in lexicographic order. In the few cases where $MILP$ is better, the gap is very small. More precisely, AMA requires 0.3 additional color on average on 2 triplets, whereas on the 6 remaining triplets the f_1 value is the same and AMA leads to a slightly larger number of interruptions (which never exceeds 3). It should be noted that when the f_1 and f_2 values produced by $MILP$ and AMA are the same, the f_3 value is also the same. Finally, note that the average computation times required by TS and AMA to reach their best solutions are comparable (350 vs 334 seconds, respectively).

6.4.2 Results for larger instances (with up to 200 vertices)

We performed some preliminary experiments and observed that AMA often produces the best solution for instances with 100 vertices, when compared to AMA^{Deg}, AMA^{Cost} and AMA^{NoInt}. Furthermore, its performance is not sensitive to the characteristics of the test instances, as opposed to the other adaptive memory algorithms based on only one recombination operator. Therefore, we decided to compare AMA with TS for instances of size $n \in \{100, 150, 200\}$. These results are reported in Table 5, using the format of the previous table. We can see that AMA produces better solutions than TS on 45 triplets (out of 54). On average, AMA outperforms TS by 0.1%, 57.2% and 45.0% on objectives f_1, f_2 and f_3, respectively. It should be noted that the instances defined on sparse graphs with $d = 0.2$ and with small color bounds seem to be relatively easy to solve with regard to objective f_1, given that AMA and TS often produce the same value. On graphs with $d = 0.2$, AMA reaches its best solutions faster than TS (4057 vs 6309 seconds). On the other hand, for graphs with $d = 0.8$, the convergence of AMA is slower than TS (7352 vs 4914 seconds) although it produces better results. Thus, it seems that TS has some difficulty to overcome local optima.

Unsurprisingly, the numbers of colors decrease when the color bounds increase, because more vertices can share the same color. At the same time, the number of interruptions and color ranges increase, which indicates that objective f_1 is in conflict with the two other objectives f_2 and f_3. If we consider the group of instances with $n = 100$ and $d = 0.5$, the value of f_1 decreases from 189.7 to 81.4 when β increases from 3 to 10, while f_2 (resp., f_3) increases from 26.4 to 245.9 (resp., from 1320.3 to 3798.4). No impact on f_1 is observed when constant or variable color bounds are used. However, it impacts f_2, in particular when the bounds are small. On the same instances with $n = 100$ and $d = 0.5$, f_2 increases from 26.4 to 81.5 when β changes from the constant value 3 to the distribution *small* (in both cases, the average color bound is 3).

n	d	β	$MILP$	TS	AMA	Time TS	Time AMA
10	0.2	3	**(17.7*, 0*, 49.7)**	(17.7, 0.6, 52.9)	**(17.7, 0, 49.7)**	141.7	1.5
		5	**(18*, 0.3, 56.3)**	(18, 1.2, 59.1)	**(18, 0.3, 56.3)**	0.1	2.7
		10	**(16*, 0*, 52.3*)**	**(16, 0, 52.3)**	**(16, 0, 52.3)**	0.1	0.1
		small	**(21.7*, 5.3, 74.7*)**	(22, 8.5, 95)	(22, 7.1, 84.3)	262.6	162.1
		medium	**(21*, 0*, 60)**	(21, 0.4, 60.4)	**(21, 0, 60)**	73.2	2.0
		large	**(18.7*, 0*, 60.3)**	(18.7, 0.1, 61)	**(18.7, 0, 60.3)**	87.8	0.5
	0.5	3	**(24.3*, 0*, 63*)**	(24.4, 3.3, 80.2)	**(24.3, 0, 63)**	222.2	26.3
		5	**(24*, 0*, 64)**	(24, 0.7, 67.9)	**(24, 0, 64)**	22.2	0.4
		10	**(26*, 0*, 51.3)**	(26, 1.2, 58.1)	**(26, 0, 51.3)**	46.6	1.2
		small	**(26*, 1.3, 56.7)**	(26, 3.9, 74.6)	(26, 2, 60.1)	116.4	68.9
		medium	**(23.3*, 0*, 56.3)**	(23.3, 2.3, 68)	**(23.3, 0, 56.3)**	88.3	4.3
		large	**(26*, 0*, 57)**	(26, 0.1, 57.1)	**(26, 0, 57)**	85.3	3.4
	0.8	3	(32, 1.3, 68)	(32, 2.5, 76.1)	**(32, 0.3, 58.3)**	170	70.1
		5	**(37*, 0*, 50.3)**	(37, 0.6, 54)	**(37, 0, 50.3)**	77.5	1.5
		10	**(47.7*, 0*, 65.7)**	(47.7, 1.3, 80.3)	**(47.7, 0, 65.7)**	0.1	5.1
		small	**(30.3*, 0*, 48.3)**	(30.3, 0.2, 49.5)	**(30.3, 0, 48.3)**	32.5	1.3
		medium	**(37.3*, 0*, 51.3)**	(37.3, 0.5, 56.2)	**(37.3, 0, 51.3)**	0.1	0.8
		large	**(36.7*, 0*, 51.7)**	(36.7, 0.5, 56.1)	**(36.7, 0, 51.7)**	34.5	1.0
20	0.2	3	(35*, 0.7, 104.7)	(35.2, 5.3, 135.1)	**(35, 0, 103.7)**	514.1	33.5
		5	**(25.3*, 0*, 92.3)**	(25.6, 7.7, 117.5)	(25.3, 0.3, 93.9)	71.9	152.6
		10	**(24.7*, 0*, 112)**	(24.7, 7.8, 156.2)	(24.7, 0.5, 114.8)	366.9	106.9
		small	**(38.7*, 7.3, 140.3)**	(38.7, 10.5, 167.2)	(38.7, 9.5, 162.2)	415.1	682.3
		medium	**(23*, 6.3, 121.3)**	(23, 14.8, 163.7)	(23, 9.6, 143.1)	473.7	522.0
		large	**(25.3*, 0*, 112)**	(25.3, 7.9, 149.1)	**(25.3, 0, 112)**	349.3	14.5
	0.5	3	(40*, 10.7, 128)	(40, 12.7, 232.9)	**(40, 2.3, 138.9)**	477.0	411.1
		5	(34*, 4.3, 139.7)	(34, 13.3, 207.3)	**(34, 3.1, 144)**	406.4	308.8
		10	(36.7*, 2, 128.7)	(36.7, 10.4, 210.5)	**(36.7, 0, 117)**	109.2	45.7
		small	(37.3*, 36.7, 244.3)	(37.3, 25.4, 276.8)	**(37.3, 18, 210.9)**	459.2	768.4
		medium	(39.3, 12, 195)	(39.3, 12.1, 231.2)	**(39.3, 3.9, 173.1)**	186.0	311.7
		large	**(33.3*, 0.3, 112)**	(33.3, 10.7, 185.3)	(33.3, 1.5, 124.7)	185.0	135.8
	0.8	3	(63.3*, 6, 121)	(63.3, 4.1, 178)	**(63.3, 0, 110.7)**	0.0	33.5
		5	(60.7*, 1.3, 138.3)	(60.7, 6.9, 211.2)	**(60.7, 0.5, 121.9)**	97.9	190.7
		10	(59, 16.3, 236)	(59, 7.1, 203.7)	**(59, 0.7, 114.2)**	234.0	257.2
		small	**(56, 47, 219.7)**	(56.3, 16.7, 280.8)	(56.3, 12.3, 238.7)	657.3	716.2
		medium	(50*, 16.7, 109.7)	(50, 4.7, 163.7)	**(50, 0, 102.7)**	153.3	35.8
		large	(56.3*, 1.3, 115.3)	(56.3, 5.5, 185.9)	**(56.3, 0, 106.3)**	0.1	27.7
30	0.2	3	(53.3*, 4.3, 190.7)	(53.5, 7.4, 207.5)	**(53.3, 0, 159)**	997.0	44.9
		5	(35.3*, 6, 197.3)	(35.3, 5.4, 197.9)	**(35.3, 1.2, 185.8)**	711.6	953.6
		10	(25.7*, 30.7, 218)	(25.7, 20.3, 263.7)	**(25.7, 4.6, 196.7)**	618.4	359.4
		small	(58.7*, 19, 296.3)	(58.7, 19.9, 338.7)	**(58.7, 15.5, 324.9)**	934.0	808.0
		medium	(28*, 20.3, 226)	(28, 14.5, 237.1)	**(28, 10.5, 219.7)**	716.5	1062.2
		large	**(26.7*, 5.3, 193.7)**	(26.7, 23.3, 285.8)	(26.7, 8.6, 219.6)	808.2	530.6
	0.5	3	(51*, 37.7, 2964)	(51, 17.8, 346.9)	**(51, 2.9, 201.7)**	625.9	858.9
		5	(134.3, 36, 348)	(51.7, 22.5, 407)	**(51.7, 0, 172.3)**	332.3	436.0
		10	(43, 20.3, 272)	(43, 24.2, 380.1)	**(43, 7.1, 238.9)**	513.3	607
		small	(49.7*, 107.3, 389.7)	(49.7, 36.7, 464.5)	**(49.7, 29.3, 404.7)**	1088.4	1016.9
		medium	(40, 46.3, 463)	(40, 33.9, 431.1)	**(40, 16.7, 291.9)**	596.7	659.7
		large	(44.7, 79.3, 527.7)	(43.3, 39.1, 519.1)	**(43.3, 23.1, 415.5)**	729.9	724.6
	0.8	3	(152.7, 982, 397)	(76.7, 18, 494.9)	**(76.7, 6.3, 256.5)**	196.3	651.4
		5	(142, 40.3, 444)	(67.7, 30.6, 559.3)	**(67.7, 6.3, 271.1)**	898.5	600.8
		10	(145.7, 67, 413)	(70.7, 17.7, 379.3)	**(70.7, 3, 195.8)**	463.1	571.6
		small	(224.7, 28, 2495.7)	(69.4, 35.7, 621.4)	**(69.3, 22.3, 455.9)**	720.9	1056.8
		medium	(145.3, 55.3, 418)	(71.7, 31.1, 625.6)	**(71.7, 10.5, 320.8)**	772.9	1218.4
		large	(167.3, 5.3, 271.3)	(68.7, 27.9, 562)	**(68.7, 5.4, 267.8)**	604.3	776.1

Table 4: Results of $MILP$, TS and AMA for small instances ($n \leq 30$)

n	d	β	TS	AMA	Time TS	Time AMA
100	0.2	3	(175.3, 14.3, 780.7)	**(175.3, 0, 524.7)**	2771.7	378.3
		5	(103.3, 17, 729.3)	**(103.3, 3.1, 542.7)**	4086.0	2420.2
		10	**(55, 27.8, 806.9)**	(55, 31.1, 950.6)	3082.7	3525.8
		small	(177.3, 44.1, 1692.6)	**(177.3, 33, 1345.9)**	3720.3	2070.9
		medium	(94, 42.1, 1143.9)	**(94, 29.5, 1063.5)**	2756.7	3856.2
		large	**(54, 38.6, 888.3)**	(54, 45.5, 1037.3)	3374.0	4286.6
	0.5	3	(189.7, 26.4, 1320.3)	**(189.7, 0.9, 587.9)**	3691.8	3745.9
		5	(108, 90.3, 2342.7)	**(108, 35.5, 1509.2)**	2528.7	2813.6
		10	(81.4, 245.9, 3798.4)	**(81.3, 194.3, 3405.3)**	1839	4046.1
		small	(191, 81.5, 2703.5)	**(191, 65.9, 2137.5)**	3944.4	2817.9
		medium	(98.3, 148.7, 3202.5)	**(97.1, 231.7, 4131.5)**	3627.1	2317.4
		large	**(82.4, 243.5, 3885.9)**	(82.5, 232.2, 3655.7)	2074.7	4220.0
	0.8	3	(182.3, 84.9, 3728.3)	**(182.3, 45.1, 2467.2)**	2791.5	3786.8
		5	(152.1, 218.3, 6283.3)	**(152, 163.6, 5147.7)**	2968.1	360.3
		10	(164.9, 245.4, 7395.7)	**(164.7, 170.8, 5682.1)**	3293.8	4150.4
		small	(168.7, 144.7, 4859)	**(167, 195.1, 5106.4)**	3603.5	3929.6
		medium	**(158.9, 232.6, 6722.5)**	(159.2, 208.9, 6288.5)	2551.7	3759.5
		large	(160.3, 226.4, 6830)	**(160.3, 197.1, 5939.9)**	2780.1	3357.3
150	0.2	3	(271.7, 34.5, 2243.9)	**(271.7, 0, 814.3)**	7062.5	1129.8
		5	(173.7, 31.3, 1760.6)	**(173.7, 2.4, 890.9)**	6919.4	3792.5
		10	(85.3, 44.7, 1503.7)	**(85.3, 30.3, 1537.7)**	5418.3	5881.8
		small	(242.7, 62.1, 2797.5)	**(242.7, 36.8, 1693.4)**	6653.0	2180.5
		medium	(136.3, 48.7, 1769.9)	**(136.3, 40.9, 1687)**	6257.3	3818.0
		large	(84.3, 46.9, 1445.9)	**(84.3, 46.1, 1613.7)**	6229.8	6191.8
	0.5	3	(268.3, 40.1, 2487.4)	**(268.3, 1, 818)**	5546.8	5458.5
		5	(164.3, 65.3, 2739.5)	**(164.3, 33.3, 2212.7)**	5331.7	5874.2
		10	(105.9, 364.1, 7233.7)	**(105.7, 250.1, 6251.7)**	3756.9	3595.5
		small	(256, 87.9, 4367.1)	**(256, 34.4, 2569.5)**	5466.7	6672.5
		medium	**(150.3, 143.7, 4688.3)**	(150.3, 175.4, 5249.7)	5226.0	3653.2
		large	**(108.7, 370, 7494.5)**	(108.9, 389.9, 7533.7)	3194.7	4724.9
	0.8	3	(278.3, 73, 4671.1)	**(278.3, 44.1, 3903.5)**	5870.6	7257.4
		5	(204.9, 334.5, 12339.3)	**(204.5, 261.7, 10611.1)**	4655.2	7344.5
		10	(213.5, 361.3, 13360.7)	**(213.1, 271.9, 11488.8)**	3516.5	7735.5
		small	(268.8, 209.6, 10709.3)	**(266, 289.7, 11181.5)**	6813.5	8066.0
		medium	**(217, 363.7, 13527.7)**	(217.5, 345.3, 12680.5)	4743.5	7098.5
		large	**(221.7, 350.2, 13903.5)**	(221.9, 333.1, 13230.3)	3762.3	5662.3
200	0.2	3	(351.7, 46.9, 3740.5)	**(351.7, 0, 1053.7)**	10099.3	1968.3
		5	(219, 44.1, 2646)	**(219, 4.9, 1231.9)**	9225.6	2487.9
		10	(111.3, 126.2, 3681.1)	**(111.3, 34.1, 2124.3)**	7300.9	8657.1
		small	(334, 81.3, 5012)	**(334, 43.7, 2336.5)**	10535.2	3109.5
		medium	(178.3, 73.4, 2912.3)	**(178.3, 55.7, 2640.3)**	9003.2	7123.3
		large	(113.3, 112.4, 3425.9)	**(113.3, 84.1, 3061.8)**	9073.6	7123.3
	0.5	3	(379.7, 64.7, 5084.9)	**(379.7, 1.1, 1184.9)**	9735.4	6955.3
		5	(216.3, 80.9, 4328.3)	**(216.3, 33.9, 2844.4)**	9102.8	7088.7
		10	(131, 508.9, 11972.4)	**(130.6, 373.3, 10755.2)**	6491.7	6291.4
		small	(359.7, 108.7, 6881.1)	**(359.7, 65.3, 4037.7)**	10491.2	6614.5
		medium	(195.6, 168.9, 7457.1)	**(195.3, 220.7, 8217.1)**	9762.4	7245.3
		large	(138.2, 507, 12692.8)	**(137.1, 659.1, 13919.7)**	5422.8	6070.2
	0.8	3	(372.7, 216, 14884)	**(372.7, 42.5, 5146.3)**	8103.2	11010.9
		5	(267.1, 517.7, 22773.4)	**(266.4, 362.7, 18436.5)**	6208.3	11055.7
		10	(270.9, 505.3, 23080.9)	**(270.3, 376.1, 19127.5)**	6435.1	11851.8
		small	(365.3, 262.4, 16911)	**(363.3, 238.2, 15186.9)**	9605.8	7947.6
		medium	(258.7, 472.7, 20908.3)	**(257.9, 535.7, 20608)**	5346.8	111065.1
		large	**(258.9, 495.8, 21603.4)**	(259.3, 532.7, 21480.3)	5412.4	11560.5

Table 5: Results of TS and AMA for large instances ($100 \leq n \leq 200$)

6.4.3 Results on the $MGCP$ benchmark instances

AMA was designed to solve a specific multi-coloring problem with applications in job scheduling. But, if the problem is simplified by removing some of its features, we can use AMA to solve the classical $MGCP$. Table 6 compares a single run of AMA with state-of-the art methods reported in (Mehrotra and Trick, 2007), (Gualandi and Malucelli, 2012), (Prestwich, 2008), and (Lim et al., 2005) on well-known $MGCP$ benchmark instances (see *http://mat.gsia.cmu.edu/COLOR04/*). In the following, these methods are denoted by MT, GM, Pr and $LZLR$, respectively. For each method, the smallest number of colors is indicated.

On the GEOM instances, AMA produces the same number of colors as the other methods. On the remaining 40 instances, AMA is competitive with Pr and GM: AMA found 8 better solutions versus 6 better solutions for Pr (26 ties) whereas it found 12 better solutions versus 14 for MT (14 ties). However, AMA is outperformed by GM with only one better solution for AMA and 12 for GM (17 ties). Nonetheless, these results show that AMA can provide high quality solutions on the $MGCP$, even if it was not specifically designed for this problem.

An adequate comparison with regard to the computation times is not possible for different reasons. First, the results presented in Table 6 were obtained in different, sometimes unspecified, computational environments and under different experimental protocols (e.g., time limit). Second, for some methods, only the time required to get a k-coloring (i.e., a solution using exactly k colors) is specified, and not the time needed to minimize the number of used colors. Third, the design of AMA was not optimized for the $MGCP$. Overall, the computation times of all methods are rather erratic and change a lot from one instance to another. Only a few seconds are required by all algorithms for the easiest instances (i.e., those for which all methods get the same result), but the computation times can grow up to thousands of seconds for the most difficult instances.

Instance	AMA	MT	GM	Pr	$LZLR$	Instance	AMA	MT	GM	Pr
GEOM20	**28**	**28**	**28**	**28**	**28**	R50-9ga	**64**	**64**	**64**	**64**
GEOM20a	**30**	**30**	**30**	**30**	**30**	R50-9gba	**228**	**228**	**228**	**228**
GEOM20b	**8**	**8**	**8**	**8**	**8**	R75-1ga	**14**	15	**14**	**14**
GEOM30	**26**	**26**	**26**	**26**	**26**	R75-1gba	**53**	54	**53**	**53**
GEOM30a	**40**	**40**	**40**	**40**	**40**	R75-5ga	39	**38**	**38**	39
GEOM30b	**11**	**11**	**11**	**11**	**11**	R75-5gba	133	**131**	**131**	132
GEOM40	**31**	**31**	**31**	**31**	**31**	R75-9ga	**94**	**94**	**94**	**94**
GEOM40a	**46**	**46**	**46**	**46**	**46**	R75-9gba	**328**	**328**	**328**	329
GEOM40b	**14**	**14**	**14**	**14**	**14**	R100-1ga	16	17	**15**	16
GEOM50	**35**	**35**	**35**	**35**	**35**	R100-1gba	**56**	57	**56**	**56**
GEOM50a	**61**	**61**	**61**	**61**	**61**	R100-5ga	45	**43**	**43**	**43**
GEOM50b	**17**	18	**17**	**17**	**17**	R100-5gba	157	**153**	**153**	154
GEOM60	**36**	**36**	**36**	**36**	**36**	R100-9ga	**118**	**118**	**118**	**118**
GEOM60a	**65**	**65**	**65**	**65**	**65**	R100-9gba	424	**422**	**422**	426
GEOM60b	**22**	**22**	**22**	**22**	**22**	myciel5	**14**	**14**	**14**	**14**
GEOM70	**44**	**44**	**44**	**44**	**44**	myciel5b	**45**	**45**	**45**	46
GEOM70a	**71**	**71**	**71**	**71**	**71**	myciel6	**16**	**16**	**16**	17
GEOM70b	**22**	23	**22**	**22**	**22**	myciel6b	**58**	**58**	**58**	60
GEOM80	**63**	**63**	**63**	**63**	**63**	myciel7	18	**17**	18	19
GEOM80a	**68**	**68**	**68**	**68**	**68**	myciel7b	64	**61**	63	67
GEOM80b	**25**	26	**25**	**25**	**25**	queen8-8	**28**	29	**28**	**28**
GEOM90	**51**	52	**51**	**51**	**51**	queen8-8b	**113**	**113**	**113**	**113**
GEOM90a	**65**	66	**65**	**65**	**65**	queen9-9	**35**	36	**35**	**35**
GEOM90b	**28**	29	**28**	**28**	**28**	queen9-9b	**135**	**135**	**135**	**135**
GEOM100	**60**	**60**	**60**	**60**	**60**	queen10-10	**38**	40	**38**	**38**
GEOM100a	**81**	**81**	**81**	**81**	**81**	queen10-10b	**136**	**136**	**136**	**136**
GEOM100b	**30**	31	**30**	**30**	**30**	queen11-11	**41**	44	**41**	**41**
GEOM110	**62**	63	**62**	**62**	**62**	queen11-11b	**140**	142	**140**	**140**
GEOM110a	**91**	92	**91**	**91**	**91**	queen12-12	**42**	47	44	**42**
GEOM110b	**37**	**37**	**37**	**37**	**37**	queen12-12b	**163**	165	**163**	**163**
GEOM120	**64**	**64**	**64**	**64**	**64**	DSJC125.1	**19**	21	**19**	**19**
GEOM120a	**93**	94	**93**	**93**	**93**	DSJC125.1b	**67**	68	**67**	**67**
GEOM120b	**34**	35	**34**	**34**	**34**	DSJC125.5	57	55	**54**	55
R50-1ga	**12**	**12**	**12**	**12**	-	DSJC125.5b	168	164	**163**	164
R50-1gba	**45**	**45**	**45**	**45**	-	DSJC125.9	140	140	**139**	140
R50-5ga	**29**	**29**	**29**	**29**	-	DSJC125.9b	501	**497**	**497**	502
R50-5gba	101	**100**	**100**	**100**	-					

Table 6: Comparison of AMA with state-of-the-art multi-coloring methods

7 Conclusion

In this paper, a mixed integer program is introduced and various methodologies are proposed to solve an extension of the graph multi-coloring problem. This extension is aimed at modeling in a more realistic way production scheduling environments with parallel machines and job incompatibility. In particular, three different objectives are considered in lexicographic order. The best method is an adaptive memory algorithm coupled with a dynamic management of three recombination operators (each one being effective on specific objectives). This feature allows the method to efficiently optimize lower level objectives, without degrading the performance of higher level objectives. Future research avenues include the adaptation of AMA to scheduling problems in other production environments, like flow-shop and job-shop, and the study of scheduling problems with setup incompatibility (instead of job incompatibility).

References

Aardal, Karen, Stan van Hoesel, Arie Koster, Carlo Mannino, and Antonio Sassano. 2007. “Models and solution techniques for frequency assignment problems.” *Annals of Operations Research* 153: 79–129.

Almeder, Christian, and Bernardo Almada-Lobo. 2011. “Synchronisation of scarce resources for a parallel machine lotsizing problem.” *International Journal of Production Research* 49 (24): 7315–7335.

Baker, Brenda S, and Edward G Jr. Coffman. 1996. “Mutual exclusion scheduling.” *Theoretical Computer Science* 162 (2): 225 – 243.

Bar-Noy, Amotz, Magnús M. Halldórsson, Guy Kortsarz, Ravit Salman, and Hadas Shachnai. 2000. “Sum Multicoloring of Graphs.” *Journal of Algorithms* 37 (2): 422 – 450.

Bendraouche, Mohamed, and Mourad Boudhar. 2012. “Scheduling jobs on identical machines with agreement graph.” *Computers & Operations Research* 39 (2): 382 – 390.

Blöchliger, Ivo, and Nicolas Zufferey. 2008. “A graph coloring heuristic using partial solutions and a reactive tabu scheme.” *Computers & Operations Research* 35 (3): 960 – 975.

Blöchliger, Ivo, and Nicolas Zufferey. 2013. “Multi-coloring and job-scheduling with assignment and incompatibility costs.” *Annals of Operations Research* 211: 83–101.

Bonomo, Flavia, Sara Mattia, and Gianpaolo Oriolo. 2011. “Bounded coloring of co-comparability graphs and the pickup and delivery tour combination problem.” *Theoretical Computer Science* 412 (45): 6261 – 6268.

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

Cheng, T.C.E., and Sin, C.C.S. 1990. "A state-of-the-art review of parallel-machine scheduling research." *European Journal of Operational Research* 47 (3), 271 – 292.

Corominas, Albert, Rafael Pastor, and Joan Plans. 2008. "Balancing assembly line with skilled and unskilled workers." *Omega* 36 (6): 1126 – 1132.

de Werra, D., M. Demange, J. Monnot, and V. Th. Paschos. 2005. "A hypocoloring model for batch scheduling." *Discrete Applied Mathematics* 146 (1): 3–26.

Demange, Marc, Gabriele Di Stefano, and Benjamin Leroy-Beaulieu. 2008. "Online Bounded Coloring of Permutation and Overlap Graphs." *Electronic Notes in Discrete Mathematics* 30: 213 – 218.

Epstein, Leah, Magns M. Halldórsson, Asaf Levin, and Hadas Shachnai. 2009. "Weighted Sum Coloring in Batch Scheduling of Conflicting Jobs." *Algorithmica* 55: 643–665.

Even, Guy, M. M. Halldórsson, Lotem Kaplan, and Dana Ron. 2009. "Scheduling with conflicts: online and offline algorithms." *Journal of Scheduling* 12 (2): 199–224.

Galinier, Philippe, Jean-Philippe Hamiez, Jin-Kao Hao, and Daniel Porumbel. 2013. "Recent advances in graph vertex coloring." In *Handbook of Optimization,* 505–528. Springer.

Galinier, P., A. Hertz, and N. Zufferey. 2008. "An Adaptive Memory Algorithm for the k-Coloring Problem." *Discrete Applied Mathematics* 156 (2): 267 – 279.

Gardi, Frédéric. 2009. "Mutual exclusion scheduling with interval graphs or related classes, Part I." *Discrete Applied Mathematics* 157 (1): 19 – 35.

Gendreau, Michel, and Jean -Yves Potvin, eds . 2010. *Handbook of Metaheuristics.* Springer. 2nd Edition.

Giaro, K., M. Kubale, and P. Obszarski. 2009. "A graph coloring approach to scheduling of multiprocessor tasks on dedicated machines with availability constraints." *Discrete Applied Mathematics* 157 (17): 3625–3630.

Glover, F. 1986. "Future paths for integer programming and linkage to artificial intelligence." *Computers & Operations Research* 13 (5): 533–549.

Gualandi, Stefano, and Federico Malucelli. 2012. "Exact solution of graph coloring problems via constraint programming and column generation." *INFORMS Journal on Computing* 24 (1): 81–100.

Halldórsson, M. M. 2004. "Multicoloring: Problems and Techniques." *Lecture Notes in Computer Science* 3153.

Halldórsson, Magnus M., and Guy Kortsarz. 2002. "Tools for Multicoloring with Applications to Planar Graphs and Partial k-Trees." *Journal of Algorithms* 42 (2): 334 – 366.

Hansen, Pierre, Alain Hertz, and Julio Kuplinsky. 1993. "Bounded vertex colorings of graphs." *Discrete Mathematics* 111 (13): 305 – 312.

Hertz, Alain, and Marino Widmer. 1996. “An improved tabu search approach for solving the job shop scheduling problem with tooling constraints.” *Discrete Applied Mathematics* 65 (1-3): 319 – 345.

Hong, Tzung-Pei, Pei-Chen Sun, and Sheng-Shin Jou. 2009. “Evolutionary Computation for Minimizing Makespan on Identical Machines with Mold Constraints.” *WSEAS Transaction on System and Control* 4 (7): 339–348.

Jansen, Klaus. 1998. “The mutual exclusion scheduling problem for permutation and comparability graphs.” *Lecture Notes in Computer Science* 1373: 287–297.

Jarvis, Mark, and Bing Zhou. 2001. “Bounded vertex coloring of trees.” *Discrete Mathematics* 232 (1-3): 145 – 151.

Le Pape, Claude. 2007. “A Test Bed for Manufacturing Planning and Scheduling Discussion of Design Principles.” In *Proceedings of the International Workshop on Scheduling a Scheduling Competition,* Providence, Rhode Island USA.

Lewis, Rhyd. 2007. “A survey of metaheuristic-based techniques for university timetabling problems.” *OR Spectrum* 30 (1): 167–190.

Lim, A., Y. Zhu, Q. Lou, and B. Rodrigues. 2005. “Heuristic methods for graph coloring problems.” In *Proceedings of the 2005 ACM Symposium on Applied Computing,* Santa Fe, New Mexico. SAC ’05. 933–939. New York, USA: ACM.

Liu, Zhaohui, and T. C. Edwin Cheng. 2002. “Scheduling with job release dates, delivery times and preemption penalties.” *Information Processing Letters* 82 (2): 107 – 111.

Lü, Zhipeng, and Jin-Kao Hao. 2010a. “Adaptive tabu search for course timetabling.” *European Journal of Operational Research* 200 (1): 235–244.

Lü, Zhipeng, and Jin-Kao Hao. 2010b. “A memetic algorithm for graph coloring.” *European Journal of Operational Research* 203 (1): 241–250.

Lushchakova, Irina N., and Vitaly A. Strusevich. 2010. “Scheduling incompatible tasks on two machines.” *European Journal of Operational Research* 200 (2): 334 – 346.

Ma, Ying, Chengbin Chu, and Chunrong Zuo. 2010. “A survey of scheduling with deterministic machine availability constraints.” *Computers & Industrial Engineering* 58 (2): 199 – 211.

Malaguti, E., and P. Toth. 2010. “A Survey on Vertex Coloring Problems.” *International Transactions in Operational Research* 17 (1): 1 – 34.

Marx, Dniel. 2004. “Graph Coloring Problems and Their Applications in Scheduling.” *Electrical Engineering* 48 (1-2): 11 – 16.

Mehrotra, Anuj, and MichaelA. Trick. 2007. “A Branch-And-Price Approach for Graph Multi-Coloring.” In *Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies,* Vol. 3715–29. Springer US.

Méndez-Díaz, Isabel, and Paula Zabala. 2010. “Solving a multicoloring problem with overlaps using integer programming.” *Discrete Applied Mathematics* 158 (4): 349 – 354.

Meuwly, F.-X., B. Ries, and N. Zufferey. 2010. “Solution Methods for a Scheduling Problem with Incompatibility and Precedence Constraints.” *Algorithmic Operations Research* 5 (2): 75 – 85.

Persi, Piero, Walter Ukovich, and Raffaele Pesenti. 2001. “Two Job Cyclic Scheduling With Incompatibility Constraints.” *International Transactions in Operational Research* 8 (2): 167 – 181.

Pesenti, Raffaele, and Walter Ukovich. 2003. “Economic lot scheduling on multiple production lines with resource constraints.” *International Journal of Production Economics* 81-82 (1): 469–481.

Pinedo, Michael. 2008. *Scheduling: Theory, Algorithms, and Systems*. Springer. Third Edition.

Prats, Xavier, Vicenç Puig, Joseba Quevedo, and Fatiha Nejjari. 2010. “Lexicographic optimisation for optimal departure aircraft trajectories.” *Aerospace Science and Technology* 14 (1): 26–37.

Prestwich, Steven. 2008. “Generalised graph colouring by a hybrid of local search and constraint programming.” *Discrete Applied Mathematics* 156 (2): 148 – 158.

Rochat, Y., and E. Taillard. 1995. “Probabilistic Diversification and Intensification in Local Search for Vehicle Routing.” *Journal of Heuristics* 1 (1): 147–167.

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

Solnon, Christine, Van Dat Cung, Alain Nguyen, and Christian Artigues. 2008. “The car sequencing problem: overview of state-of-the-art methods and industrial case-study of the ROADEF2005 challenge problem.” *European Journal of Operational Research* 191 (3): 912–927.

Thevenin, Simon, Nicolas Zufferey, and Jean-Yves Potvin. 2014. “Multi-objective parallel machine scheduling with incompatible jobs.” In *Proceedings of the 15th Annual Congress of the French Operations Research Society, ROADEF 2014,* Bordeaux, France.

Xue, Jue. 1998. “Solving the Minimum Weighted Integer Coloring Problem.” *Computational Optimization and Applications* 11 (1): 53–64.

Zufferey, Nicolas. 2012. “Metaheuristics: Some Principles for an Efficient Design.” *Computer Technology and Applications* 3: 446 – 462.

Zufferey, Nicolas, Olivier Labarthe, and David Schindl. 2012. “Heuristics for a project management problem with incompatibility and assignment costs.” *Computational Optimization and Applications* 51 (3): 1231–1252.

Zufferey, Nicolas, and Michel Vasquez. 2014. “A Generalized Consistent Neighborhood Search for Satellite Range Scheduling Problems.” *RAIRO Operations Research* 49 (1): 99 – 121.