J Sched (2008) 11: 263–277
DOI 10.1007/s10951-008-0066-8

# Graph colouring approaches for a satellite range scheduling problem

**Nicolas Zufferey · Patrick Amstutz · Philippe Giaccari**

Received: 16 May 2007 / Accepted: 25 March 2008 / Published online: 1 May 2008


**Abstract** A goal of this paper is to efficiently adapt the best ingredients of the graph colouring techniques to an NP-hard satellite range scheduling problem, called MuRRSP. We propose two new heuristics for the MuRRSP, where as many jobs as possible have to be scheduled on several resources, while respecting time and capacity constraints. In the permutation solution space, which is widely used by other researchers, a solution is represented by a permutation of the jobs, and a schedule builder is needed to generate and evaluate a feasible schedule from the permutation. On the contrary, our heuristics are based on the solution space which contains all the feasible schedules. Based on the similarities between the graph colouring problem and the MuRRSP, we show that the latter solution space has significant advantages. A tabu search and an adaptive memory algorithms are designed to tackle the MuRRSP. These heuristics are derived from efficient graph colouring methods. Numerical experiments, performed on large, realistic, and challenging instances, showed that our heuristics are very competitive, robust, and outperform algorithms based on the permutation solution space.

N. Zufferey (✉)
HEC, Faculté des Sciences Économiques et Sociales,
University of Geneva, Boulevard du Pont-d'Arve 40,
1211 Geneva 4, Switzerland
e-mail: Nicolas.Zufferey@hec.unige.ch

P. Amstutz
Department of Communication and Computer Science,
Swiss Federal Institute of Technology, Lausanne, Switzerland

P. Giaccari
Centre d'Optique, Photonique et Laser, Université Laval, Québec,
Canada

**Keywords** Oversubscribed satellite scheduling · Graph colouring heuristics

## 1 Introduction

As mentioned in (Marinelli et al. 2006), scheduling the daily communications between satellites and ground control stations is getting very difficult, since an increasing number of satellites must be controlled by a small set of ground stations. As a consequence, a large number of communication requests (also called jobs) are unserved. Scheduling requests (or jobs) on a satellite constellation is referred to as *Satellite Range Scheduling problem*. In this paper, we only focus on a specific subproblem called *Multi-Resource Range Scheduling Problem* (MuRRSP for short), which is NP-hard (Barbulescu et al. 2004a).

One goal of this paper is to show the similarities between the MuRRSP and the graph colouring problem with a fixed number $k$ of colours, which is denoted $k$-GCP (where GCP holds for Graph Colouring Problem). We will see that scheduling a job on a resource is equivalent to assigning a vertex to a colour. A straightforward idea is therefore to derive graph colouring heuristics to tackle the MuRRSP. Thus, our goal is not to focus on standard jobs scheduling techniques to tackle the MuRRSP. The reader interested in classical scheduling problems and techniques is referred to (Pinedo 2002), in which dynamic aspects are also considered. Currently, in the best heuristics for the MuRRSP (Barbulescu et al. 2004a), each solution is encoded as a permutation of the jobs. From each permutation, it is possible to build a feasible schedule by the use of a greedy constructive heuristic. From a permutation of the vertices, it is also possible to greedily build a feasible colouring of the considered graph. Such a strategy was already tested

within the context of graph colouring (e.g., Brélaz 1979; Culberson 1992; Culberson and Luo 1995; Hertz and Costa 1997; Davis 1991). However, the results are very poor when compared to the best colouring heuristics which are not based on the permutation solution space. Therefore, it seems to be meaningful to adapt some of the best graph colouring techniques (e.g., Bloechliger and Zufferey 2008; Galinier and Hao 1999; Galinier et al. 2008) to tackle the MuRRSP.

The paper is organized as follows. We formally introduce general satellite range scheduling problems and the MuRRSP in Sect. 2. In Sect. 3, we present the main lessons from the GCP literature that are useful when designing heuristics for the MuRRSP. In Sect. 4, we derive a tabu search for the MuRRSP from the tabu search for the $k$-GCP proposed in (Bloechliger and Zufferey 2008). In Sect. 5, we design an adaptive memory algorithm for the MuRRSP from the evolutionary algorithms for the $k$-GCP proposed in (Galinier and Hao 1999) and (Galinier et al. 2008). Results are reported and discussed in Sect. 6. Finally, general conclusions and research avenues are drawn in Sect. 7.

## 2 The MuRRS: formulation and related works

Consider a set of satellites and a set of ground stations. Ground stations are communication facilities (e.g., antennae). Depending on the mission, satellites may be geosynchronous or not. In the latter case, a ground station can communicate with a satellite only when the satellite lies within the transmitting horizon of the ground station. In general, this happens periodically within the planning horizon. Several operations must be performed on spacecrafts, related to satellite control or payload. These operations require ground-to-space communications, called jobs. Therefore, a job is associated with some information representing the corresponding on-board operation. In a satellite range scheduling problem, a set $J = \{1, \ldots, n\}$ of jobs have to be scheduled. Each job $j$ is characterized by the following parameters:

- $sat_j$, the (unique) satellite requested by $j$;
- $M_j$, the set of ground stations able to process $j$;
- $p_j$, the processing time, i.e. the duration of communication $j$;
- $r_j$, the job release time, i.e. the time at which $j$ becomes available for processing;
- $d_j$, the job due-date, i.e. the time by which $j$ must be completed;
- $w_j$, the revenue of $j$;
- $l_j$, the size of the data packets (in kilobytes) associated with $j$.

Note that $p_j, r_j$, and $d_j$ may depend on the considered ground station. Transmission and reception are regarded as being simultaneous and preemptions are not allowed. Moreover, both satellites and ground stations have unit capacity, that is, can process at most one job at a time. The following families of technological constraints have also to be taken into account: precedence constraints between jobs, ground station setup times (when a ground station processes two consecutive jobs with two distinct satellites), on-board storage constraints (at any time, the size of data packets uploaded on satellites cannot exceed a given limit).

A lot of work has already been done on satellite range scheduling problems (e.g., Bensana et al. 1999; Harrison et al. 1999; Pemberton 2000; Wolfe and Sorensen 2000; Verfaillie and Lemaitre 2001; Frank et al. 2001; Lemaitre et al. 2002; Gabrel and Murat 2003; Vasquez and Hao 2003; Globus et al. 2003, 2004; Cordeau and Laporte 2005). In this paper, we only focus on a subcase of the above general problem, where $w_j = 1$ for every job $j$, i.e. all the jobs have the same revenue. In addition, we ignore precedence, setup and on-board storage constraints. Such a resulting problem is called the *Multi-Resource Range Scheduling Problem* (MuRRSP) and is NP-hard (Barbulescu et al. 2004a). An important application is the one encountered by the U.S. Air Force Satellite Control Network (AFSCN), where more than 100 satellites and 16 antennae located at 9 ground stations are considered. Customers request an antenna at a ground station for a specific time window along with possible alternative slots. The problem is, in general, oversubscribed, i.e. all the jobs cannot be performed. Any job that is not scheduled on the requested day is said to be in *conflict*. Minimizing the number of conflicting jobs is of crucial importance in a practical standpoint, because human schedulers do not consider any conflicting job worse than any other conflicting job (e.g., Schlack 1993; Gooley 1993; Parish 1994). The human schedulers themselves state that minimizing the number of conflicts reduces (1) their workload, (2) communication with outside agencies, and (3) the time required to produce a conflict-free schedule (Barbulescu et al. 2004a). Anytime there is a conflict, human schedulers have to contact the customer associated with the conflicting job and have to discuss in order to try to schedule the job somewhere else in the planning horizon.

In summary, in the MuRRSP, for each job $j$, we know the set of resources $M_j$ on which $j$ can be scheduled. For each job $j$, we know its processing time $p_j$ and its time window $[r_j; d_j]$, where $p_j, r_j$, and $d_j$ can have different values depending on the considered resource of $M_j$. The goal is to schedule as many jobs as possible within its time window, such that the processing of two jobs cannot overlap on the same resource.

In the upper part of Fig. 1, we illustrate how the data associated with the MuRRSP can be represented, considering $k = 3$ resources. In such an example, jobs 1 and 5 can be

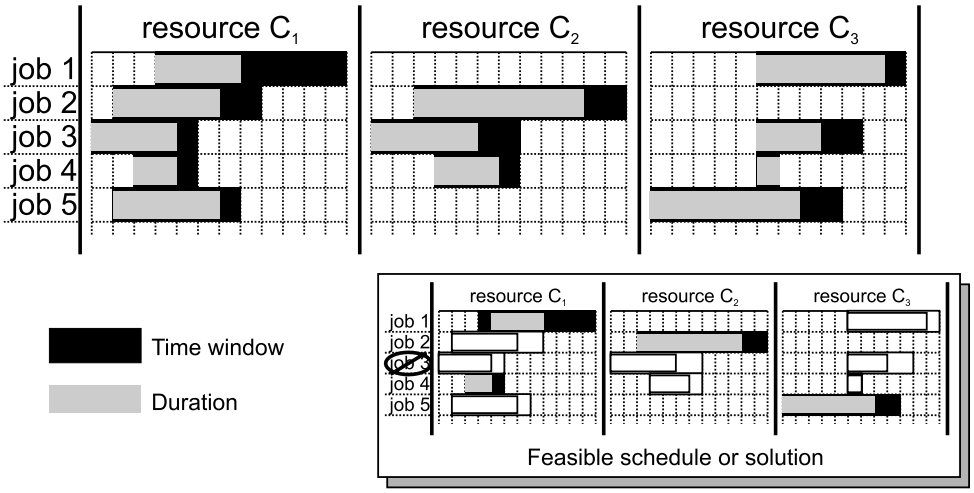


**Fig. 1** Representation of a feasible solution for the MuRRSP, with $k = 3$

scheduled on resources 1 and 3, job 2 can be scheduled on resources 1 and 2, and jobs 3 and 4 can be scheduled on all the resources. For each couple (job $j$, resource $C_i$), the associated time window $T_j^{(i)}$ is represented in black, while the duration $p_j^{(i)}$ is represented in grey. In the lower part of Fig. 1, we propose a way to represent a feasible solution. We can see that, for a given time, only one job can be scheduled on a resource (each resource has a unit capacity). In addition, each job is scheduled on only one resource. Finally, we can observe that job 3 is not scheduled in the proposed solution, which means that the value of the objective function is equal to one.

Various methods already exists to tackle the MuRRSP. Gooley (1993) and Schlack (1993) developed mixed-integer algorithms and insertion heuristics. Parish (1994) adapt a genetic algorithm called GENITOR. A comparison of greedy constructive heuristics (derived from the job scheduling methods proposed in Dauzère-Pérès 1995 and Bar-Noy et al. 2002), local search algorithms (e.g., hill-climbing) and GENITOR is performed in (Barbulescu et al. 2004a). Several researchers (e.g., Globus et al. 2004; Barbulescu et al. 2004a) propose to encode a solution as a permutation $\pi$ of the $n$ jobs to schedule. Let $S_1$ be the solution space containing all the possible permutations, and let $S_2$ be the solution space containing all the possible feasible schedules to the original problem. From a permutation $\pi$ in $S_1$, it is possible to generate a schedule in $S_2$ by the use of a schedule builder *SB* (which is a greedy constructive heuristic). More precisely, *SB* considers jobs in the order that they appear in the permutation $\pi$. In order to build the schedule associated with $\pi$, which is denoted by $SB(\pi)$, each job is assigned to the first available resource, from its list of alternatives, and at the earliest possible starting time. If the job cannot be scheduled on any of the alternative resources, thus, it is a conflicting job, it is dropped from the schedule, i.e. bumped. The objective function to minimize is the total number of jobs bumped from the schedule. As mentioned in (Globus et al. 2004), the solution space $S_1$ has the following advantages.

1. *SB* can take any permutation as input.
2. *SB* always provides feasible solutions to the problem. Thus, there is no need to implement any repairing procedure, which can be time consuming.
3. If there are many possible times at which jobs can be scheduled, it is often the case that $S_1$ is significantly smaller than $S_2$.

The latter can, however, be also considered as a drawback, because some solutions of $S_2$ might not be reachable from a permutation of $S_1$. Note that one may allow at each step to nondeterministically assign an available resource to the considered job, instead of the first available resource. However, in such a case, it is not possible to assign a single schedule to a single permutation, which is another drawback. In this paper, we only focus on the above described deterministic builder *SB*.

In order to design a local search (e.g., hill climbing, tabu search, simulated annealing), it is necessary to define a neighborhood structure. In solution space $S_1$, in order to generate a neighbor solution $\pi'$ from a current solution $\pi$, one may simply move a single job from its current position in $\pi$ to another position. An alternative consists to swap two jobs in $\pi$ to get $\pi'$. The latter neighborhood structure was studied in (Globus et al. 2004) and (Barbulescu et al. 2004a) within a hill climbing method, a tabu search, and a simulated annealing algorithm. The following two main drawbacks were noticed.

1. There are too many neighbor candidates for any current solution ($n \cdot (n-1)$ possibilities). Thus, it is too much time consuming to evaluate all the neighbor candidates at each iteration. Even if only a sample of candidate neighbor solutions are considered at each iteration, it is still difficult to find rules to select a promising sample.

2. $BS(\pi')$ is often equal to $BS(\pi)$ (in 33% of the cases, as experimented in Barbulescu et al. 2004b). In other words, $\pi$ and $\pi'$ in $S_1$ such that $\pi \neq \pi'$ often lead to the same schedule in $S_2$. Thus, most of the candidate neighbor solutions in $S_1$ do not change the current schedule $BS(\pi)$ in $S_2$.

Avoiding the second drawback within the context of the MuRRSP, in (Barbulescu et al. 2004a), the authors used a genetic algorithm based on $S_1$, called GENITOR, which was first proposed in (Parish 1994). Each permutation $\pi$ in $S_1$ can be evaluated in two steps as follows: (1) apply $SB$ on $\pi$; (2) compute the number of bumped jobs in $SB(\pi)$. In GENITOR, at each generation, two parent permutations $\pi_1$ and $\pi_2$ are selected, using rank-based selection, in the population $P$ of permutations to generate an offspring permutation $\pi_{(\text{off})}$, which then replaces the worst permutation of the population $P$. Such algorithm is able to perform large changes on parent schedules at each generation. More precisely, $SB(\pi_{(\text{off})})$ is usually very different from $SB(\pi_1)$ and $SB(\pi_2)$. This is due to the use of Syswerda's crossover (see Syswerda and Palmucci 1991 for more details), which randomly selects about half of the positions in parent permutations and reorder them to produce offspring permutations. It was observed that such a crossover may change as much as 50% of the resulting offspring schedules when compared to parent schedules. In GENITOR for the MuRRSP, the population size is 200 and the algorithm stops as soon as 8000 applications of $SB$ have been performed. The algorithm does not perform better if one increases the number of evaluations from 8000 to 50,000, and the population size from 200 to 400 (Barbulescu et al. 2004a).

## 3 Lessons from the GCP literature

In this section, we present the ingredients from the GCP literature that may help to design heuristics for the MuRRSP.

The *Graph Colouring Problem* (GCP) is a well known NP-hard problem (Garey and Johnson 1979). Given a graph $G = (V, E)$ with vertex set $V$ and edge set $E$, and given an integer $k$, a *k-colouring* of $G$ is a function $c : V \rightarrow \{1, \ldots, k\}$. The value $c(x)$ of a vertex $x$ is called the *colour* of $x$. The vertices with colour $i$ $(1 \leq i \leq k)$ define a *colour class*, denoted $C_i$. If two adjacent vertices $x$ and $y$ have the same colour $i$, vertices $x$ and $y$, the edge $[x, y]$ and colour $i$ are said *conflicting*. A $k$-colouring without conflicting edges is said *legal* and its colour classes are called *stable sets*. The GCP is to determine the smallest integer $k$, called *chromatic number* of $G$ and denoted $\chi(G)$, such that there exists a legal $k$-colouring of $G$. Given a fixed integer $k$, the optimization problem $k$-GCP is to determine a $k$-colouring of $G$ that minimizes the number of conflicting edges. If the optimal value of the $k$-GCP is zero, this means that $G$ has a legal $k$-colouring. An algorithm that solves the $k$-GCP can be used to solve the GCP, by starting with an upper bound $k$ on $\chi(G)$, and then decreasing $k$ as long as a legal $k$-colouring can be found. The most efficient colouring heuristics are based on that strategy. For a recent review on graph coloring techniques, the reader may refer to (Galinier and Hertz 2006).

In the $k$-GCP, we have to assign a vertex $v \in \{1, \ldots, n\}$ to a colour class $C_i$, with $i \in \{1, \ldots, k\}$, while respecting the following constraint: two adjacent vertices should not have the same colour. In the MuRRSP, we have to assign a job $v \in \{1, \ldots, n\}$ to a resource $C_i$, with $i$ in a subset of $\{1, \ldots, k\}$, while respecting the following constraint: job $j$ of duration $p_j$ has to be processed within its time window $T_j = [r_j; d_j]$ defined by a release time $r_j$ and a due date $d_j$. We can observe that the GCP is simpler than the MuRRS in the sense that a vertex can be associated with any resource as long as the vertex is not in conflict. In addition, in the GCP, there is no need to order the vertices in each class or to consider processing times. In the situations encountered by the AFSCN (Air Force Satellite Control Network), the value of $k$ is set equal to 16.

The permutation solution space was also considered within the context of graph colouring. Any permutation $\pi$ of the vertices can be transformed in a legal colouring by the use of a colouring builder $CB$. Such a greedy constructive heuristic considers the vertices in the order they appear in permutation $\pi$. At each step, $CB$ gives the smallest colour which does not create any conflict to the considered vertex. Methods based on such a solution space are not competitive at all with the best colouring. Among such methods, we can mention: a greedy heuristic (Brélaz 1979), iterated greedy algorithms (e.g., Culberson 1992; Culberson and Luo 1995), an ant algorithm (Hertz and Costa 1997), and moreover, a genetic algorithm described in (Davis 1991) based on a crossover close to the one proposed by Syswerda in (Syswerda and Palmucci 1991), which is used in GENITOR for the MuRRSP in (Barbulescu et al. 2004a). To understand why methods based on the permutation solution space are not as efficient as methods directly dealing with $k$-colourings (legal or not, complete or partial), we can mention the following drawbacks associated with the permutation solution space, when designing local search heuristics. Let $\pi$ and $\pi'$ be two permutations of the vertices such that $\pi'$ can be obtained from $\pi$ by only moving the position of a single vertex in $\pi$, or by switching the positions of two vertices in $\pi$.

1. The resulting colourings $s = CB(\pi)$ and $s' = CB(\pi')$ can have structures which are very close or very different. Therefore, it is difficult to have a good control on the search.

2. Let $f(s)$ be the number of colours used in a colouring $s$. It is difficult to quickly compute $f(s')$ from $f(s)$ and $s'$. Instead, one may apply $CB$ on $\pi'$ to know the number of colours associated with $\pi'$. In other words, incremental computation is difficult to implement.

The two above mentioned drawbacks, which occur in the local search algorithms proposed for the MuRRSP within the permutation solution space (e.g., Globus et al. 2004; Barbulescu et al. 2004a) can be easily avoided when dealing with $k$-colourings. For example, in the neighborhood structures proposed in (Hertz and de Werra 1987) and (Bloechliger and Zufferey 2008), which are tabu search methods, only a few number of vertices are involved in a colour change. Thus, two neighbor solutions have a very close structure, and incremental computation is easy to develop when minimizing the number of conflicts or the number of noncoloured vertices.

In the remaining part of the paper, and as suggested by Globus et al. in the conclusion of (Globus et al. 2004), we propose various heuristics working in the schedule solution space $S_2$, without using any schedule builder procedure.

## 4 Tabu search for the MuRRSP

In this section, we propose a tabu search heuristic for the MuRRSP. Tabu search is a well-known local search heuristic which was originally proposed in (Glover 1986) and in (Hansen 1986). Its crucial component is to prevent the search to cycle by forbidding the reverse of some moves during a certain number of iterations (we also speak about the duration of the tabu status). Many variants of tabu search can be found, for example, in (Glover and Laguna 1997).

Remember that, in order to design an efficient heuristic for the $k$-GCP, it is useful to deal with $k$-colourings or partial legal $k$-colourings. The most efficient local search and population based methods work in such solution spaces (e.g., Hertz and de Werra 1987; Morgenstern 1996; Galinier and Hao 1999; Galinier et al. 2008; Bloechliger and Zufferey 2008) and not in the permutation solution space. In this section, we derive a tabu search TS-MuRRSP for the MuRRSP from the tabu search TS-GCP colouring method proposed in (Bloechliger and Zufferey 2008).

In (Morgenstern 1996), the author proposed the following strategy to tackle the $k$-GCP. He considers the set of *partial legal $k$-colourings* which are defined as legal $k$-colourings of a subset of vertices of $G$. Such colourings can be represented by a partition of the vertex set into $k+1$ subsets $C_1, \ldots, C_{k+1}$, where $C_1, \ldots, C_k$ are $k$ disjoint stable sets (i.e. legal colour classes) and $C_{k+1}$ is the set of noncoloured vertices. In order to solve the $k$-GCP, the objective can thus be to minimize the number of vertices in $C_{k+1}$ (i.e. the number of noncoloured vertices). Similarly, in the MuRRSP, the goal is to minimize the number of non scheduled jobs.

Consider the solution space containing the set of partial legal $k$-colourings. The associated objective function is hence to minimize the number of noncoloured vertices. In TS-GCP (Bloechliger and Zufferey 2008), in order to generate a neighbor solution $s' = \{C'_1, \ldots, C'_{k+1}\}$ from $s = \{C_1, \ldots, C_{k+1}\}$, a vertex $v$ is moved from $C_{k+1}$ (the set of noncoloured vertices) to a colour class $C_i$, then, vertices in $C_i$ which are in conflict with $v$ are moved to $C_{k+1}$ (such vertices are removed from the partial colouring). Then, all the moves which will put $v$ back in $C_{k+1}$ are tabu for a certain number of iterations. At each iteration, the best nontabu move is performed, i.e. the move leading to the smallest resulting size of $C'_{k+1}$. Some extensions of the above neighbor structure were tested in (Bloechliger 2005). For example, suppose that we would like to move vertex $v$ from $C_{k+1}$ to $C_i$ (with $0 < i < k+1$), and that such a move implies that one vertex $x$ has to be removed from $C_i$ (because $x$ and $v$ are adjacent). Instead of immediately putting $x$ in $C_{k+1}$, it seems to be straightforward to try to put $x$ in another color class $C_j$ (with $0 < i \neq j < k+1$) if $C_j + \{x\}$ remains stable. However, it was showed that such an extension needs more computational effort without leading to better results.

It is now straightforward to derive TS-MuRRSP from TS-GCP. In TS-MuRRSP, $C_{k+1}$ denotes the set of nonscheduled jobs, and the goal is to minimize its size. In order to generate a neighbor solution $s' = \{C'_1, \ldots, C'_{k+1}\}$ from $s = \{C_1, \ldots, C_{k+1}\}$, a job $j$ is moved from $C_{k+1}$ to an admissible resource $C_i$, then, jobs in $C_i$ which are in conflict with $j$ are moved to $C_{k+1}$ (such jobs are removed from the schedule associated with resource $i$). Then, all the moves which will put $j$ back in $C_{k+1}$ are tabu for a certain number of iterations. As it is the case for TS-GCP (see the previous paragraph), we do not try to immediately reschedule the jobs of $C_i$ which are in conflict with $j$, and we simply put them in $C_{k+1}$.

At each iteration, for TS-GCP as well as for TS-MuRRSP, we have the following.

- The selected neighbor solution is chosen among all the possible neighbor (and nontabu) solutions.
- Incremental computation is easy to develop, i.e. it is easy to compute the value of a neighbor solution if we know the current solution and its value.
- The tabu duration $t(v)$ during which it is forbidden to put vertex $v$ back in $C_{k+1}$ is equal to $0.6 \cdot n_c + \text{RANDOM}(0; 9)$, where $n_c$ is the number of vertices or jobs in $C_{k+1}$ in the current solution, and $\text{RANDOM}(a; b)$ is a function which returns a number randomly generated in the set $\{a, a+1, \ldots, b-1, b\}$.

However, moving a job $j$ from $C_{k+1}$ to $C_i$ (with $i \in \{1, \ldots, k\}$ such that $i$ is admissible for $j$) is not as easy as moving a vertex $v$ from $C_{k+1}$ to $C_i$ (with $i \in \{1, \ldots, k\}$), because each job $j$ has to be performed within its time window

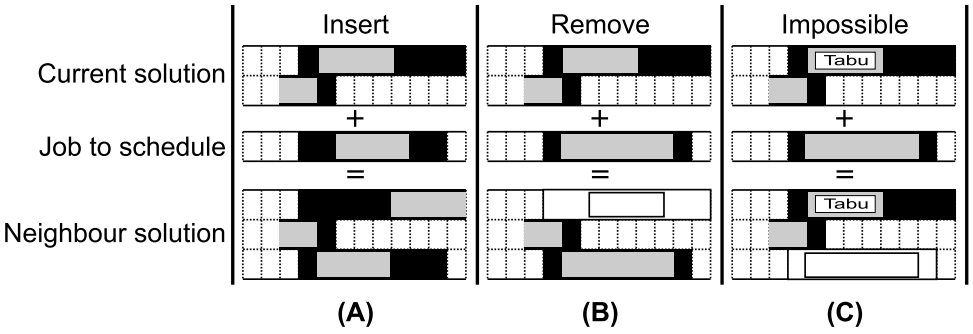


**Fig. 2** Three situations which can occur when generating neighbor solutions

$T_j = [r_j; d_j]$, and each resource cannot process two jobs at the same time (if such a situation may occur, there is a conflict). Suppose that jobs $j_1, \ldots, j_r$ are already scheduled (in that order) on resource $C_i$. In order to schedule job $j$ in $C_i$ within its time window, three situations may occur.

(A) In the best situation, it is possible to adjust the schedules of jobs $j_1, \ldots, j_r$ within their own time windows (while keeping the same relative order $j_1, \ldots, j_r$) in order to add job $j$ to $C_i$ without creating any conflict. Such a successful move is represented in part (A) of Fig. 2.
(B) If it is not possible, we have to remove some nontabu jobs from $C_i$. Such a move is represented in part (B) of Fig. 2, when one job is removed from the current schedule.
(C) If the only way to schedule the considered job consists in removing a tabu job from the current schedule, we do not consider such a move further (i.e. it is an impossible move, as represented in part (C) of Fig. 2).

Considering situation (B), the main question is now: which nontabu jobs should be removed? Before answering such a question, we define the *flexibility* $F_j$ of a job $j$ as the number of resources it can be scheduled on, and we define the *age* $A_j$ of job $j$ as the number of iterations, since job $j$ has the same position in the current solution (either a resource $C_i$ with $i \in \{1, \ldots, k\}$ or the set $C_{k+1}$). We test every possibility to insert job $j$ in $C_i$ (while keeping the same relative order $j_1, \ldots, j_r$).

We propose to apply the following rules (by priority order) to remove jobs from $C_i$ when it is not possible to insert $j$ in $C_i$ without creating any conflict.

1. Remove the smallest number of nontabu jobs.
2. If there are several possibilities, break ties by removing the set of jobs with the largest average flexibility.
3. If there are still several possibilities, break ties by removing the set of jobs with the largest average age.
4. If there are still several possibilities, break ties randomly.

The priority of the above rules are easy to understand. The first rule focuses on the objective function to minimize (the number of non scheduled jobs). The second rule proposes to remove jobs which would be easier to schedule on another resource later. The third rule has a diversification role because it proposes to remove the jobs which were scheduled on $C_i$ a long time ago. Several preliminary numerical experiments showed that the above proposed list of rules is the best combination of such rules, even if the proposed strategy might cause the search to miss some better solutions.

Note that in order to generate an initial solution in TS-MuRRSP, we apply *SB* on a random permutation $\pi$ of the $n$ jobs. In addition, we mention that some of the ingredients of the proposed neighborhood and tabu list structures were also efficiently and independently used to tackle the selecting and scheduling satellite photographs problem in (Habet and Vasquez 2004), where only one resource (i.e. using $k = 1$ in our notation) is considered. In (Bianchessi et al. 2007), the authors considered a satellite scheduling problem involving two satellites and multiple orbits. Their tabu search method works in the solution space of nonnecessarily feasible schedules, and constraints violations are penalized. Note that the less time one have to build a schedule, the less it is appropriate to spend time dealing with nonfeasible schedules. For this reason, we think that working within $S_2$ (i.e. the set of feasible schedules) is a better option for real-life problems.

In order to better diversify the search of TS-MuRRSP, we propose two diversification mechanisms denoted DIV1 and DIV2. The goal of DIV1 is to renew the set of nonscheduled jobs by forcing their insertion in the current schedule, and the goal of DIV2 is to renew the set of scheduled jobs by removing jobs which are in the current solution for a long time. After a given number $p$ (parameter) of iterations without improvement of the best solution $s^*$ encountered so far, we perform DIV1 or DIV2 (we perform DIV1 if DIV2 was the previous choice, and vice-versa).

The procedure DIV1 consists in scheduling as many jobs as possible from $C_{k+1}$ of the current solution $s = \{C_1, \ldots, C_{k+1}\}$ (jobs are considered in a random order). More precisely, at each step of DIV1, we randomly select a job in $C_{k+1}$ (each job of $C_{k+1}$ can only be selected once during DIV1) and insert it at the best position in the schedule, even if we need to remove one ore more non tabu jobs

from the current schedule $s$ to do it (see the situations (A), (B), and (C) above). Therefore, at the end of DIV1, it might happen that the resulting set of non scheduled jobs is completely different from the set $C_{k+1}$ of $s$. Note that anytime a job is inserted in the schedule during DIV1, then, in TS-MuRRSP, it is tabu to remove it from the schedule during $t_{\text{DIV1}}$ (parameter) iterations.

The procedure DIV2 consists in removing the oldest job from every $C_i$, for all $i \in \{1, \ldots, k\}$, in order to create additional and new room in every $C_i$. It is then tabu to insert such removed old jobs at their original position during $t_{\text{DIV2}}$ iterations. Preliminary experiments showed that $p = 50{,}000$, $t_{\text{DIV1}} = t_{\text{DIV2}} = 50$ are good values for these parameters. In addition, outside the context of the two diversification mechanisms, when a job $j$ is inserted in the current schedule, it is then forbidden to remove it from the schedule during $t = \text{RANDOM}(30; 50)$ iterations.

## 5 Adaptive memory algorithm for the MuRRSP

In this section, we first briefly present the main ingredients of the adaptive memory algorithm. Second, we describe the recombination operators of the best evolutionary colouring heuristics. Then we design an adaptive memory algorithm from the evolutionary colouring methods presented in (Galinier and Hao 1999) and (Galinier et al. 2008).

A recent hybrid evolutionary heuristic is the *adaptive memory algorithm* (Rochat and Taillard 1995) that may store pieces of solutions or complete solutions in a central memory $\mathcal{M}$. While two parent solutions are combined to create an offspring in a standard genetic local search, all pieces of solutions in the central memory can contribute to the creation of an offspring in an adaptive memory algorithm, which is summarized below.

1. Initialize the central memory $\mathcal{M}$ with (pieces of) solutions.
2. Repeat until a stopping criterion is met:
   (a) create an offspring solution $s$ by using a recombination operator;
   (b) apply a local search operator on $s$ and let $s'$ denote the resulting solution;
   (c) use (pieces of) $s'$ in order to update $\mathcal{M}$.

The process consisting of creating an offspring solution $s$, applying the local search operator on $s'$, and updating the central memory with the resulting solution is called a *generation*. In order to create an offspring at step 2(a), the recombination operator first chooses pieces of solutions in $\mathcal{M}$.

Consider the $k$-GCP when minimizing the number of conflicts in the $k$-colouring solution space. In the genetic hybrid method proposed in (Galinier and Hao 1999), as well as in the adaptive memory algorithm proposed in (Galinier et al. 2008), at each generation, an offspring solution $s_{(\text{off})} = \{C_1^{(\text{off})}, \ldots, C_k^{(\text{off})}\}$ is build class by class from the population of (pieces of) solutions. In (Galinier and Hao 1999), if colour classes $C_1^{(\text{off})}, \ldots, C_{i-1}^{(\text{off})}$ are already built from parent solutions $s_1 = \{C_1^{(1)}, \ldots, C_k^{(1)}\}$ and $s_2 = \{C_1^{(2)}, \ldots, C_k^{(2)}\}$, the next colour class $C_i^{(\text{off})}$ is build as follows. Let $A$ be the set of already coloured vertices (i.e. the vertices in $C_1^{(\text{off})}, \ldots, C_{i-1}^{(\text{off})}$). If $i$ is an odd number, then choose the set $C_j^{(1)}$ in $s_1$ that contains a maximum number of noncoloured vertices and set $C_i^{(\text{off})} = C_j^{(1)} - A$; else choose the set $C_j^{(2)}$ in $s_2$ that contains a maximum number of noncoloured vertices and set $C_i^{(\text{off})} = C_j^{(2)} - A$. At the end of such a process, the remaining noncoloured vertices are randomly coloured.

If we compare the crossover proposed for graph colouring in (Davis 1991) (which is close to the one used in GENITOR for the MuRRSP, Barbulescu et al. 2004a) with the two recombination operators proposed for graph colouring in (Galinier and Hao 1999) and (Galinier et al. 2008), we can observe that in the two latter ones, the structure of the graph colouring problem is efficiently exploited (whole colour classes are transmitted from parent solutions to offspring solutions). This is not the case in the crossover proposed in (Davis 1991), which can be considered more as a diversification mechanism rather than a recombination operator exploiting the structure of the considered problem.

In order to design an adaptive memory algorithm for the MuRRSP (named hereafter AMA-MuRRSP), we have to define: the way to initialize the population $P$ of (pieces of) solutions, the recombination operator, the intensification (or local search) operator, and the memory update operator.

We propose to work with partial legal schedules (which correspond to partial legal $k$-colouring). In addition, we choose $|P| = 10$ as in (Galinier and Hao 1999) and (Galinier et al. 2008). In order to initialize $P$, we randomly generate 10 solutions in $S_2$ (by performing $SB$ on 10 random permutations generated in $S_1$) and improve such solutions by performing TS-MuRRSP during 1000 iterations.

Our recombination operator is very similar to the one proposed in (Galinier and Hao 1999) described above. A difference holds in the fact that more than two solutions may contribute to build and offspring solution. Note that the idea of not fixing the number of parents of the recombination operator was very efficiently experimented in (Zufferey 2002) for the $k$-colouring problem. This will be confirmed for the MuRRSP in Sect. 6. At each generation, an offspring solution $s_{(\text{off})} = \{C_1^{(\text{off})}, \ldots, C_k^{(\text{off})}\}$ is build set by set from $P$. Suppose that sets $C_1^{(\text{off})}, \ldots, C_{i-1}^{(\text{off})}$ are already built from parent solutions, and that parent solution $s_{r'}$ (with $r'$ in $\{1, \ldots, 10\}$) provided the set $C_{i-1}^{(\text{off})}$. In addition, let $A$ be the set of already scheduled jobs (i.e. the jobs in $C_1^{(\text{off})}, \ldots, C_{i-1}^{(\text{off})}$). At that moment, we randomly choose the next non considered resource $i$ to deal with. Then, the

**Fig. 3** Recombination operator for the MuRRSP

| | resource $C_1$ | resource $C_2$ | resource $C_3$ | Bumped jobs |
|---|---|---|---|---|
| solution $s_1$ | $C_1^{(1)} = (j_1, j_2)$ | **$C_2^{(1)} = (j_8)$** **3** | $C_3^{(1)} = (j_3, j_4, j_6)$ | $C_4^{(1)} = (j_5, j_7)$ |
| solution $s_2$ | **$C_1^{(2)} = (j_1, j_3, j_4)$** **1** | $C_2^{(2)} = (j_6, j_7)$ | $C_3^{(2)} = (j_2, j_5)$ | $C_4^{(2)} = (j_8)$ |
| solution $s_3$ | $C_1^{(3)} = (j_3, j_6)$ | $C_2^{(3)} = (j_2, j_8)$ | **$C_3^{(3)} = (j_1, j_5, j_7)$** **2** | $C_4^{(3)} = (j_4)$ |
| temporary offspring solution | $C_1^{(off,tmp)} = (j_1, j_3, j_4)$ | $C_2^{(off,tmp)} = (j_8)$ | $C_3^{(off,tmp)} = (j_5, j_7)$ | $C_4^{(off,tmp)} = (j_2, j_6)$ |
| offspring solution | $C_1^{(off)} = (j_1, j_3, j_4)$ | $C_2^{(off)} = (j_6, j_8)$ | $C_3^{(off)} = (j_5, j_7)$ | $C_4^{(off)} = (j_2)$ |

set $C_i^{(\text{off})}$ is provided by the solution $s_r = \{C_1^{(r)}, \ldots, C_{k+1}^{(r)}\}$ in $P$ (with $r \neq r'$, i.e. the same solution of $P$ can not consecutively provide two sets of the offspring solution) which contains the maximum number of nonscheduled jobs (we break ties randomly). Thus we set $C_i^{(\text{off})} = C_i^{(r)} - A$. At the end of such a process, we try to successively schedule each nonscheduled job (considered in a random order) in a feasible resource without removing other jobs (ties are broken randomly). Therefore, in contrast with Syswerda's crossover used in GENITOR (Barbulescu et al. 2004a), our recombination operator exploits much of the structure of the MuRRSP. Our recombination operator is illustrated in Fig. 3 with 8 jobs $j_1, \ldots, j_8$ and $k = 3$. For sake of simplicity, we do not represent the time windows and the durations, and we assume that the central memory consists in three solutions denoted $s_1, s_2$, and $s_3$, and that every job can be scheduled on every resource. $C_i^{(r)}$ is the ordered set of scheduled jobs in solution $s_r$ on the $i$th resource. For example, jobs $j_1$, $j_3$, and $j_4$ are scheduled (in that order) on resource $C_1$ in solution $s_2$. In order to build the offspring solution $s_{(\text{off})} = (\{C_1^{(\text{off})}, C_2^{(\text{off})}, C_3^{(\text{off})}\})$ from $s_1, s_2$, and $s_3$, we proceed as follows. First, we can either set $C_1^{(\text{off})} = C_1^{(2)}$ or $C_3^{(\text{off})} = C_3^{(3)}$ or $C_3^{(\text{off})} = C_3^{(1)}$. Assume we randomly choose $C_1^{(\text{off})} = C_1^{(2)} = (j_1, j_3, j_4)$ (this step 1 is drawn in bold face in the second column of Fig. 3). Now, solution $s_2$ cannot be considered for step 2. Then, we can either set $C_2^{(\text{off})} = C_2^{(3)}$ or $C_3^{(\text{off})} = C_3^{(3)} - \{j_1\}$ (but not $C_3^{(\text{off})} = C_3^{(1)} - \{j_3, j_4\}$). Assume we randomly choose $C_3^{(\text{off})} = C_3^{(3)} - \{j_1\} = (j_5, j_7)$ (this step 2 is drawn in bold face in the fourth column of Fig. 3). Now, solutions $s_1$ and $s_2$ can be considered for step 3. Then, we can either set $C_2^{(\text{off})} = C_2^{(1)}$ or $C_2^{(\text{off})} = C_2^{(2)} - \{j_7\}$. Assume we randomly choose $C_2^{(\text{off})} = C_2^{(1)} = (j_8)$ (this step 3 is drawn in bold face in the third column of Fig. 3). Therefore, the resulting temporary offspring solution is $\{(j_1, j_3, j_4), (j_8), (j_1, j_5, j_7)\}$ (see the line labeled "temporary offspring solution" in Fig. 3). Finally, it may be possible to schedule $j_6$ before $j_8$ on resource $C_2$, which leads to the resulting offspring solution $\{(j_1, j_3, j_4), (j_6, j_8), (j_1, j_5, j_7)\}$.

Of course, different recombination operators could be developed and tested for the MuRRSP. We, however, decided to only focus on the one that we derived from the best recombination operator for the $k$-GCP proposed in (Galinier and Hao 1999).

The intensification operator is simply TS-MuRRSP 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 TS-MuRRSP 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 TS-MuRRSP at each generation of AMA-MuRRSP. Preliminary experiments showed that $I = 100{,}000$ is a reasonable choice, which corresponds to a few seconds of CPU time on the computer used for the experiments and described in the next section.

Finally, let $s$ be the solution provided by TS-MuRRSP at each generation, let $s_{\text{worst}}$ be the worst solution of $P$, and let $s_{\text{old}}$ be the oldest solution of $P$. We propose to update $P$ with $s$ as follows. If $s$ is better or equal to $s_{\text{worst}}$, we replace $s_{\text{worst}}$ with $s$ in $P$ and update $s_{\text{worst}}$. Otherwise, we replace $s_{\text{old}}$ with $s$ in $P$ and update $s_{\text{old}}$. In the latter case, even if $s$ is not able to improve the average quality of $P$, it is at least able to bring "new blood" in $P$.

We have now all the ingredients to build AMA-MuRRSP. At the beginning, we initialize the population $P$ with 10 random solutions improved by TS-MuRRSP during 1000 iterations. Then, as long as a time condition is not met, we perform a generation. At each generation, we first apply the recombination operator to build an offspring solution (resource by resource), which is then improved by TS-MuRRSP during $I = 100{,}000$ iterations. The resulting solution is finally used to replace the worst (in this case, the average quality of $P$ is improved) or the oldest (in this case, new blood is provided to $P$) solution of $P$.

The stopping conditions of TS-MuRRSP and AMA-MuRRSP will be discussed in the next section.

## 6 Results

In this section, we give details on the considered instances and then compare three heuristics on these instances: TS-

**Table 1** Comparison of GENITOR, TS-MuRRSP and AMA-MuRRSP on the instances $E$ (time limit = 5 minutes)

| Instances | GENITOR | | TS-MuRRSP | | AMA-MuRRSP | |
|---|---|---|---|---|---|---|
| | Mean | Min | Mean | Min | Mean | Min |
| 1 | 2.5 | 2 | 2 | 2 | 2 | 2 |
| 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 3 | 5.5 | 5 | 5 | 5 | 5 | 5 |
| 4 | 7 | 6 | 6 | 6 | 6 | 6 |
| 5 | 5.5 | 5 | 5 | 5 | 5 | 5 |
| 6 | 7.75 | 7 | 7 | 7 | 7 | 7 |
| 7 | 1 | 1 | 1 | 1 | 1 | 1 |
| 8 | 3 | 3 | 3 | 3 | 3 | 3 |
| 9 | 2 | 2 | 2 | 2 | 2 | 2 |
| 10 | 3 | 3 | 3 | 3 | 3 | 3 |
| MEAN | 3.93 | 3.6 | 3.6 | 3.6 | 3.6 | 3.6 |

MuRRSP, AMA-MuRRSP, and GENITOR. We choose to compare our two heuristics with GENITOR because the latter usually provides the best solutions for the MuRRSP in the permutation solution space (Barbulescu et al. 2004a). We end the section with a discussion on the number of parents involved in the recombination operator of AMA-MuRRSP.

### 6.1 Description of the performed experiments

We perform experiments on instances of size 500 (i.e. $n = 500$), because it corresponds to the most difficult problems encountered in practice. Each instance is built by a well-known generator which can be provided by the Colorado State University (see http://www.cs.colostate.edu/sched/people.html). The generator is well described and used in (Barbulescu et al. 2004a). It produces instances of the MuRRSP by modeling realistic features. Different types of jobs are represented, such as downloading data from a satellite, transmitting information or commands from a ground station to a satellite, and checking the health and status of a satellite. In addition, the generator takes into account some customer behaviors. The duration and window size for each job are determined using the parameters associated with the job type. For each job, one or several resource(s) is (are) specified.

Using the provided generator, we generated two types of instances: type $E$, which stands for "Easy", and type $D$, which stands for "Difficult". The type $E$ contains 10 instances with $k = 16$ resources (such instances are exactly of the same type as the ones considered in Barbulescu et al. 2004a), and the type $D$ contains the same 10 instances but with only $k = 9$ resources, and 10 other instances with $k = 9$. Because of the different numbers of available resources, it is now straightforward to understand why instances of type $E$ are easier than the ones of type $D$. Note that all the considered instances can be provided by the corresponding author of this paper upon request.

On each instance, we performed 30 runs, and we consider three stopping time conditions for every method, which are 5, 15, and 30 minutes on a *2 GHz Pentium 4 with 512 MB of RAM*. Such stopping conditions are consistent with the ones used in (Barbulescu et al. 2004a), and are appropriate in a practical standpoint. Considering larger CPU times is usually too much in practice, where people usually try to find schedules as quickly as possible.

### 6.2 Comparison of the considered heuristics

With a time limit of 5 minutes, we compare GENITOR, TS-MuRRSP, and AMA-MuRRSP on the instances $E$ (the easier ones) in Table 1. For each method and each instance, we give the average (see label "Mean") and the minimum (see label "Min") numbers of bumped jobs (over 10 runs). In addition, we also give the average number of bumped jobs over the 10 instances (see the line labeled "MEAN"). Note that in every table of the paper, the numbers are always rounded. We can observe that the considered methods are comparable, as only a few number of jobs are bumped. However, GENITOR seems to be slightly less robust than TS-MuRRSP and AMA-MuRRSP, because not every run leads to the same solution (namely, on instances 1, 3, 4, 5, and 6). We would like to mention that extending the time limit to 30 minutes does not affect the results.

We consider now the instances of type $D$ (the difficult ones) with three time limits (5, 15, and 30 minutes). The instances 1 to 10 are the same as the ones of type $E$ but with $k = 9$, and the instances 11 to 20 are other instances with $k = 9$. In Tables 2, 3, and 4, we respectively give the results obtained by GENITOR, TS-MuRRSP, and AMA-MuRRSP.

**Table 2** Results obtained with GENITOR on the instances *D*

| | 5 min | | | 15 min | | | 30 min | | |
|---|---|---|---|---|---|---|---|---|---|
| Inst. | Min | Mean | Max | Min | Mean | Max | Min | Mean | Max |
| 1 | 112 | 115.75 | 121 | 112 | 115.75 | 121 | 112 | 115.75 | 121 |
| 2 | 96 | 102 | 111 | 96 | 101.5 | 110 | 96 | 101.5 | 110 |
| 3 | 122 | 125.75 | 132 | 122 | 125.75 | 132 | 122 | 125.75 | 132 |
| 4 | 111 | 117.25 | 123 | 111 | 117 | 122 | 111 | 117 | 122 |
| 5 | 111 | 113.75 | 116 | 111 | 113.75 | 116 | 111 | 113.75 | 116 |
| 6 | 114 | 121 | 130 | 114 | 121 | 130 | 114 | 121 | 130 |
| 7 | 113 | 120.5 | 126 | 113 | 120.5 | 126 | 113 | 120.5 | 126 |
| 8 | 100 | 103.5 | 106 | 100 | 103.5 | 106 | 100 | 103.5 | 106 |
| 9 | 96 | 97 | 99 | 96 | 97 | 99 | 96 | 97 | 99 |
| 10 | 101 | 104.25 | 110 | 101 | 104 | 109 | 100 | 103.5 | 109 |
| 11 | 99 | 106.25 | 111 | 99 | 106 | 111 | 99 | 106 | 111 |
| 12 | 104 | 106.25 | 109 | 104 | 106.25 | 109 | 104 | 106.25 | 109 |
| 13 | 104 | 111 | 114 | 104 | 111 | 114 | 104 | 110.75 | 114 |
| 14 | 102 | 112.25 | 118 | 102 | 112 | 117 | 101 | 111.75 | 117 |
| 15 | 101 | 108.75 | 116 | 101 | 108.75 | 116 | 101 | 108.75 | 116 |
| 16 | 111 | 120.25 | 133 | 111 | 119.75 | 131 | 109 | 119.25 | 131 |
| 17 | 105 | 107.75 | 112 | 105 | 107.25 | 112 | 105 | 107.25 | 112 |
| 18 | 106 | 112.25 | 115 | 106 | 112.25 | 115 | 106 | 112.25 | 115 |
| 19 | 95 | 101 | 107 | 95 | 100.25 | 107 | 95 | 100.25 | 107 |
| 20 | 93 | 96.75 | 100 | 93 | 96.75 | 100 | 93 | 96.5 | 100 |
| MEAN | 104.8 | 110.16 | 115.45 | 104.8 | 110 | 115.15 | 104.6 | 109.91 | 115.15 |

**Table 3** Results obtained with TS-MuRRSP on the instances *D*

| | 5 min | | | 15 min | | | 30 min | | |
|---|---|---|---|---|---|---|---|---|---|
| Inst. | Min | Mean | Max | Min | Mean | Max | Min | Mean | Max |
| 1 | 56 | 56.75 | 57 | 55 | 55.75 | 57 | 55 | 55.25 | 56 |
| 2 | 54 | 54 | 54 | 54 | 54 | 54 | 54 | 54 | 54 |
| 3 | 66 | 66.5 | 67 | 65 | 65.75 | 66 | 65 | 65 | 65 |
| 4 | 49 | 50 | 51 | 49 | 49.5 | 50 | 49 | 49.25 | 50 |
| 5 | 52 | 52.75 | 54 | 51 | 52.25 | 53 | 51 | 51.75 | 52 |
| 6 | 59 | 59.75 | 60 | 59 | 59 | 59 | 59 | 59 | 59 |
| 7 | 54 | 55 | 56 | 52 | 53.25 | 55 | 52 | 53 | 54 |
| 8 | 49 | 49.5 | 50 | 49 | 49.5 | 50 | 48 | 49 | 50 |
| 9 | 38 | 38.75 | 39 | 38 | 38.25 | 39 | 38 | 38 | 38 |
| 10 | 51 | 51.5 | 52 | 50 | 50.75 | 51 | 50 | 50.5 | 51 |
| 11 | 52 | 52 | 52 | 51 | 51.75 | 52 | 51 | 51.5 | 52 |
| 12 | 47 | 47.5 | 48 | 46 | 46.5 | 47 | 45 | 45.5 | 46 |
| 13 | 57 | 58.5 | 60 | 56 | 56.75 | 57 | 55 | 56 | 57 |
| 14 | 58 | 59.75 | 62 | 56 | 58 | 60 | 56 | 57.5 | 59 |
| 15 | 43 | 43.75 | 44 | 42 | 42.5 | 43 | 42 | 42.5 | 43 |
| 16 | 58 | 58.5 | 59 | 57 | 58 | 59 | 56 | 57.25 | 58 |
| 17 | 49 | 50.25 | 52 | 49 | 50.25 | 52 | 49 | 50 | 52 |
| 18 | 53 | 54.5 | 55 | 53 | 53.25 | 54 | 52 | 52.75 | 53 |
| 19 | 47 | 48 | 49 | 47 | 47.5 | 48 | 47 | 47.25 | 48 |
| 20 | 41 | 41.25 | 42 | 41 | 41 | 41 | 41 | 41 | 41 |
| MEAN | 51.65 | 52.43 | 53.15 | 51 | 51.68 | 52.35 | 50.75 | 51.3 | 51.9 |
| GEN-TS | 53.15 | 57.73 | 62.3 | 53.8 | 58.32 | 62.8 | 53.85 | 58.61 | 63.25 |

**Table 4** Results obtained with AMA-MuRRSP on the instances *D*

| | 5 min | | | 15 min | | | 30 min | | |
|---|---|---|---|---|---|---|---|---|---|
| Inst. | Min | Mean | Max | Min | Mean | Max | Min | Mean | Max |
| 1 | 56 | 56.5 | 57 | 55 | 55.5 | 56 | 54 | 54.75 | 55 |
| 2 | 54 | 54 | 54 | 54 | 54 | 54 | 54 | 54 | 54 |
| 3 | 66 | 66.25 | 67 | 63 | 63.5 | 64 | 63 | 63.25 | 64 |
| 4 | 51 | 51 | 51 | 50 | 50.25 | 51 | 49 | 49.25 | 50 |
| 5 | 52 | 52.25 | 53 | 51 | 51.75 | 52 | 51 | 51.75 | 52 |
| 6 | 59 | 59.25 | 60 | 58 | 58.75 | 59 | 57 | 57.75 | 59 |
| 7 | 53 | 53.5 | 54 | 52 | 52.5 | 53 | 51 | 52 | 53 |
| 8 | 49 | 49.5 | 50 | 48 | 48.75 | 49 | 48 | 48.5 | 49 |
| 9 | 38 | 38.75 | 39 | 38 | 38.5 | 39 | 38 | 38.5 | 39 |
| 10 | 52 | 52 | 52 | 51 | 51.5 | 52 | 51 | 51 | 51 |
| 11 | 52 | 52.5 | 53 | 52 | 52 | 52 | 52 | 52 | 52 |
| 12 | 47 | 47.5 | 48 | 46 | 46.75 | 48 | 46 | 46 | 46 |
| 13 | 56 | 57.75 | 59 | 56 | 56.75 | 58 | 55 | 55.75 | 56 |
| 14 | 59 | 59 | 60 | 57 | 57.5 | 58 | 56 | 56.75 | 57 |
| 15 | 43 | 43.5 | 44 | 42 | 42.75 | 43 | 42 | 42.5 | 43 |
| 16 | 58 | 59 | 60 | 58 | 58.25 | 59 | 56 | 57.75 | 59 |
| 17 | 50 | 50.5 | 51 | 50 | 50.5 | 51 | 49 | 50 | 51 |
| 18 | 54 | 55.5 | 56 | 52 | 52.5 | 53 | 52 | 52.5 | 53 |
| 19 | 47 | 48.5 | 49 | 47 | 47 | 47 | 46 | 46.75 | 47 |
| 20 | 41 | 41.5 | 42 | 41 | 41 | 41 | 41 | 41 | 41 |
| MEAN | 51.85 | 52.41 | 52.95 | 51.05 | 51.5 | 51.95 | 50.55 | 51.09 | 51.55 |
| TS-AMA | −0.2 | 0.02 | 0.2 | −0.05 | 0.18 | 0.4 | 0.2 | 0.21 | 0.35 |

Such Tables have a structure which is very close to the structure of Table 1. In Table 3, the line labeled "GEN-TS" indicates the average difference of bumped jobs between GENITOR and TS-MuRRSP. For example, 53.15 (the first number of the last line of Table 3) is the difference between 104.8 (the first number of line labeled "MEAN" of Table 2) and 51.65 (the first number of line labeled "MEAN" of Table 3). Similarly, in Table 4, the line labeled "TS-AMA" indicates the average difference of bumped jobs between TS-MuRRSP and AMA-MuRRSP.

Considering Tables 2, 3, and 4, the following comments can be made.

- TS-MuRRSP is much better than GENITOR, as it is able to schedule about 58 more jobs (approximate average value of the line labeled "GEN-TS").
- AMA-MuRRSP is slightly better than TS-MuRRSP, because the former method is able to schedule, on average, more jobs than the latter one (see line labeled "TS-AMA"). More precisely, we can observe that the more time is allowed, the better is AMA-MuRRSP. AMA-MuRRSP is able to schedule 0.02 more jobs than TS-MuRRSP if 5 minutes are allowed, 0.18 more jobs if 15 minutes are allowed, and 0.21 more jobs if 30 minutes are allowed.
- If we consider the time limits of 5 and 30 minutes: GENITOR is able to schedule in average $110.16 - 109.91 = 0.25$ additional jobs, TS-MuRRSP is able to schedule in average $52.43 - 51.3 = 1.13$ additional jobs, and AMA-MuRRSP is able to schedule in average $52.41 - 51.09 = 1.32$ additional jobs. Hence, as time goes by, our two proposed methods seems to have more capability to improve their solutions.
- If, for each method, we consider the difference between columns "Mean", "Max", and "Min", we can see that the larger differences are obviously associated with GENITOR, which indicates that GENITOR is less robust than the two other methods. Notice also that AMA-MuRRSP seems to be slightly more robust than TS-MuRRSP.

In Fig. 4, based on Tables 2, 3, and 4, we compare the results of GENITOR, TS-MuRRSP, and AMA-MuRRSP when considering a time limit of 5 minutes. For each method and each difficult instance, we indicate the value of the best (see label "Min") and worst (see label "Max") solutions, as well as the average value over 30 runs (see label "Mean"). We can again observe that TS-MuRRSP and AMA-MuRRSP have similar behaviors and clearly outperform GENITOR according to the objective function and the robustness.

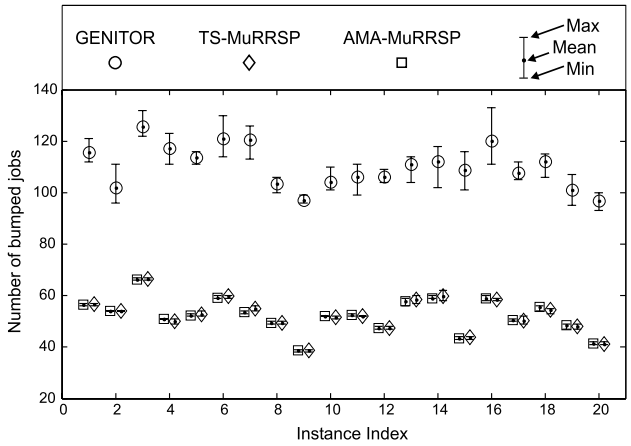


**Fig. 4** Number of bumped jobs with a time limit of 5 minutes

**Table 5** Average results of each method depending on the time

| Time (in minutes) | GENITOR | TS-MuRRSP | AMA-MuRRSP |
|---|---|---|---|
| 5 | 110.16 | 52.43 | 52.41 |
| 10 | 110.03 | 51.89 | 51.85 |
| 15 | 110 | 51.68 | 51.5 |
| 20 | 109.98 | 51.45 | 51.34 |
| 25 | 109.95 | 51.36 | 51.16 |
| 30 | 109.91 | 51.3 | 51.09 |

In Table 5, we compare the average number of bumped jobs of GENITOR, TS-MuRRSP, and AMA-MuRRSP. Each value is an average over the 20 instances of type $D$ (and 30 runs on each instance). As already mentioned above, the more time is allowed to tackle an instance, the larger is the gap between AMA-MuRRSP and the other heuristics. In addition, in order to have a better estimate on the lower bounds, we also ran AMA-MuRRSP with a larger time limit (120 minutes). We can mention that it was only able to schedule one more job on 8 instances (namely, instances 3, 5, 8, 10, 11, 12, 15, and 18). This confirms that a time limit of 30 minutes is relevant for our methods.

The improvement capabilities of the three considered methods presented in Fig. 5 are estimated from the results of Tables 2, 3, and 4. The number of placed request after 15 and 30 minutes are compared with those after 5 minutes (see label "Min" for the best solutions, label "Mean" for the average values, and label "Max" for the worst solutions). The number of additional placed requests are afterwards averaged over the 20 difficult instances. For example, the worst value of AMA-MuRRSP is improved by approximately one unit from minute 5 to minute 15.

Before concluding the paper, we would like to put forward that the idea of not fixing the number of parents in the recombination operator of AMA-MuRRSP is important, as it was showed for the $k$-colouring problem in (Zufferey

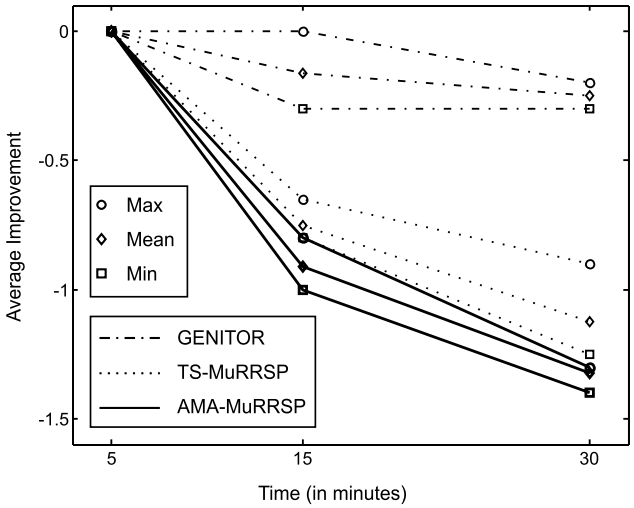


**Fig. 5** Improvements of the results from minute 5 to minute 30

2002). Let $q$ be the number of parents that are involved in the construction of the offspring solution. For example, $q = 2$ means that two parents are alternately involved in the building of the offspring solution. In Table 6, considering a time limit of 30 minutes, we compare the used version of AMA-MuRRSP where the number of parents is not fixed (second column) with different versions of AMA-MuRRSP with $q \in \{2, 3, 4, 5\}$ (four last columns). As usual, we performed 30 runs on each of the 20 instances of type $D$. For AMA-MuRRSP (where $q$ is not fixed), we indicate in column 2 the best number of bumped jobs. If the number of parents if fixed to 2 (third column), we indicate the number of additional bumped jobs that appears in the best found solution of that version of the method. For example, on instance 3, AMA-MuRRSP is able to schedule 63 jobs if the number of parents is not fixed, but 65 jobs if $q = 2$; thus, 2 additional jobs are not scheduled in the latter case (which is indicated in column 3). We can observe that it is always better to leave the number of parents unfixed, because it leads to equivalent or better results on 18 instances (the results are worse only on instances 10 and 11). In the last line of Table 6, we indicate the average number of bumped jobs in the best solutions found by each method.

## 7 Conclusion and future works

In this paper, we propose a tabu search heuristic and an adaptive memory algorithm for the MuRRSP. Both heuristics work in the solution space $S_2$ which contains all the feasible schedules. On the contrary, other methods (e.g., Barbulescu et al. 2004a; Globus et al. 2004) work in the permutation solution space $S_1$. We discuss the main advantages and drawbacks of $S_1$ and show that $S_2$ is a promising solution space, mainly because two neighbor solutions have

**Table 6** Results of AMA-MuRRSP depending on the number $q$ of parents

| Instance | $q$ not fixed | $q=2$ | $q=3$ | $q=4$ | $q=5$ |
|---|---|---|---|---|---|
| 1 | 54 | 1 | 2 | 3 | 1 |
| 2 | 54 | 0 | 0 | 0 | 0 |
| 3 | 63 | 2 | 1 | 2 | 1 |
| 4 | 49 | 1 | 1 | 1 | 1 |
| 5 | 51 | 0 | 0 | 1 | 0 |
| 6 | 57 | 1 | 0 | 1 | 1 |
| 7 | 51 | 1 | 1 | 2 | 1 |
| 8 | 48 | 1 | 1 | 1 | 1 |
| 9 | 38 | 0 | 0 | 0 | 1 |
| 10 | 51 | −1 | 0 | −1 | 0 |
| 11 | 52 | −1 | −1 | 0 | −2 |
| 12 | 46 | 0 | 1 | 0 | 0 |
| 13 | 55 | 1 | 2 | 2 | 1 |
| 14 | 56 | 1 | 0 | 0 | 2 |
| 15 | 42 | 0 | 1 | 0 | 0 |
| 16 | 56 | 1 | 2 | 1 | 1 |
| 17 | 49 | 2 | 0 | 2 | 2 |
| 18 | 52 | 0 | 1 | 1 | 2 |
| 19 | 46 | 2 | 2 | 1 | 1 |
| 20 | 41 | 0 | 0 | 0 | 0 |
| MEAN | 50.55 | 51.15 | 51.25 | 51.4 | 51.25 |

close structures and incremental computation is possible. These two elements allow a quick and well controlled trajectory in the search space $S_2$. This analysis was confirmed by our experiments, which showed that the proposed heuristics working in $S_2$ for the MuRRSP are much more efficient and robust than GENITOR, which is considered as one of the best heuristic working in $S_1$ (Barbulescu et al. 2004a).

Even if the gap between GENITOR and our heuristics is very significant, other techniques could be added to TS-MuRRSP and AMA-MuRRSP. On the one hand, TS-MuRRSP might be improved with the use of: a long-term frequency memory, a compound neighborhood structure (e.g., ejection chain), and strategic oscillations. On the other hand, AMA-MuRRSP might be improved with the use of more flexible recombination operators and advanced diversification mechanisms. The goal of this paper is, however, reached as we proposed bridges between the graph colouring community and the satellite range scheduling community by: (1) adapting the best ingredients from the graph colouring techniques to the MuRRSP, and (2) showing the great potential of these ingredients. We would like to mention that some advanced techniques are not appropriate in situations when only a small amount of CPU time is allowed to the user.

Among the future works that can be done in the context of our paper, a research avenue consists to use other evaluation functions. For example, one can minimize the sum of overlap in the schedule while scheduling the entire list of jobs, as proposed in (Barbulescu et al. 2006). One may also take the priorities of the jobs into account, as mentioned in (Globus et al. 2004). Another research avenue is to consider precedence constraints between jobs. In this area, there also exists efficient graph colouring heuristics to colour mixed graphs (in a mixed graph, some edges are oriented, and the additional constraint for each oriented edge $x \rightarrow y$ is to give a strictly smaller colour to $x$ than to $y$) (Meuwly et al. 2007). Such heuristics may be efficiently adapted within the context of satellite range scheduling problems.

Finally, one may try to propose a general heuristic based on the joint use of the two studied solution spaces $S_1$ and $S_2$. For example, the use of a *Variable Space Search* (VSS, for short) can be appropriate, and such a method was already efficiently applied to the GCP (Hertz et al. 2007). VSS is an extension of the well known *Variable Neighborhood Search* (VNS, for short) (Mladenović and Hansen 1997) and of the *Reformulation Descent* (Mladenović et al. 2005).

**Acknowledgements** The authors would like to thank Darrell L. Whitley, Mark Roberts, Mark Rogers, and Laura Barbulescu for kindly answering the questions we had on the instance generator and solutions they provided to us, the MuRRS problem, and the GENITOR heuristic.

## References


Bar-Noy, A., Guha, S., Naor, J. S., & Schieber, B. (2002). Approximating the throughput of multiple machines in real-time scheduling. *SIAM Journal on Computing*, *31*(2), 331–352.

Barbulescu, L., Watson, J.-P., Whitley, L. D., & Howe, A. E. (2004a). Scheduling space-ground communications for the air force satellite control network. *Journal of Scheduling*, *7*(1), 7–34.

Barbulescu, L., Howe, A. E., Whitley, L. D., & Roberts, M. (2004b). Trading places: how to schedule more in a multi-resource oversubscribed scheduling problem. In *International conference on automated planning and scheduling*.

Barbulescu, L., Howe, A. E., & Whitley, L. D. (2006). AFSCN scheduling: how the problem and solution have evolved. *Mathematical and Computer Modelling*, *43*(9–10), 1023–1037

Bensana, E., Lemaitre, M., & Verfaillie, G. (1999). Earth observation satellite management. *Constraints*, *4*, 293–299.

Bianchessi, N., Cordeau, J.-F., Desrosiers, J., Laporte, G., & Raymond, V. (2007). A heuristic for the multi-satellite, multi-orbit and multi-user management of earth observation satellites. *European Journal of Operational Research*, *177*, 750–762.

Bloechliger, I. (2005). *Suboptimal colorings and solution of large chromatic scheduling problems*. PhD thesis, École Polytechnique Fédérale de Lausanne, Switzerland.

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

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

Cordeau, J.-F., & Laporte, G. (2005). Maximizing the value of an earth observation satellite orbit. *Journal of the Operational Research Society*, *56*, 962–968.

Culberson, J. (1992). *Iterated greedy graph coloring and the difficulty landscape* (Tech. rept. TR 92-07). Department of Computer Science, University of Alberta, Edmonton.

Culberson, J. C., & Luo, F. (1995). Exploring the $k$-colorable landscape with iterated greedy. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*.

Dauzère-Pérès, S. (1995). Minimizing late jobs in the general one machine scheduling problem. *European Journal of Operational Research*, *81*, 131–142.

Davis, L. (1991). *Handbook of genetic algorithms*. New York: Van Nostrand Reinhold.

Frank, J., Jonsson, A., Morris, R., & Smith, D. (2001). Planning and scheduling for fleets of earth observing satellites. In *Proceedings of the sixth international symposium on artificial intelligence, robotics, automation and space*.

Gabrel, V., & Murat, C. (2003). Operations research in space and air. In *Mathematical programming for Earth observation satellite mission planning*. Boston: Kluwer Academic (Chap. 7).

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

Galinier, P., & Hertz, A. (2006). A survey of local search methods for graph coloring. *Computers & Operations Research*, *33*, 2547–2562.

Galinier, P., Hertz, A., & Zufferey, N. (2008). An adaptive memory algorithm for the graph coloring problem. *Discrete Applied Mathematics*, *156*, 267–279

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

Globus, A., Crawford, J., Lohn, J., & Pryor, A. (2003). Scheduling earth observing satellites with evolutionary algorithms. In *International conference on space mission challenges for information technology*, Pasadena, CA.

Globus, A., Crawford, J., Lohn, J., & Pryor, A. (2004). A comparison of techniques for scheduling earth observing satellites. In *Proceedings of the sixteenth innovative applications of artificial intelligence conference (IAAI-04)*.

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

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

Gooley, T. D. (1993). *Automating the satellite range scheduling process*. M.Phil. thesis, Air Force Institute of Technology, USA.

Habet, D., & Vasquez, M. (2004). Solving the selecting and scheduling satellite photographs problem with a consistent neighborhood heuristic. In *Sixteenth IEEE international conference on tools with artificial intelligence*.

Hansen, P. (1986). The steepest ascent mildest descent heuristic for combinatorial programming. In *Congress on numerical methods in combinatorial optimization*, Capri, Italy.

Harrison, S. A., Philpott, M. S., & Price, M. E. (1999). Task scheduling for satellite based imagery. In *Proceedings of the 18th workshop of the UK planning and scheduling special interest group* (pp. 64–78), University of Salfort, UK.

Hertz, A., & Costa, D. (1997). Ants can color graphs. *The Journal of the Operational Research Society*, *48*(3), 295–305.

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

Hertz, A., Plumettaz, M., & Zufferey, N. (2007, accepted). Variable space search for graph coloring. *Discrete Applied Mathematics*.

Lemaitre, M., Verfaillie, G., Jouhaud, F., Lachiver, J.-M., & Bataille, N. (2002). Selecting and scheduling observations of agile satellites. *Aerospace Science and Technology*, *6*, 367–381.

Marinelli, F., Nocella, S., Rossi, F., & Smriglio, S. (2006). *A Lagrangian heuristic for satellite range scheduling with resource constraints* (Tech. rept. TRCS 004/2005). Dip. di Informatica, Università L'Aquila, Italy.

Meuwly, F.-X., Ries, B., & Zufferey, N. (2007). *Heuristiques pour la coloration de graphes mixtes*. Master thesis. Swiss Federal Institute of Technology, EPFL, Lausanne, Switzerland.

Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, *24*, 1097–1100.

Mladenović, N., Plastria, F., & Urošević, D. (2005). Reformulation descent applied to circle packing problems. *Computers & Operations Research*, *32*, 2419–2434.

Morgenstern, C. (1996). Distributed coloration neighborhood search. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, *26*, 335–357.

Parish, D. A. (1994). *A genetic algorithm approach to automating satellite range scheduling*. M.Phil. thesis, Air Force Institute of Technology, USA.

Pemberton, J. C. (2000). Toward scheduling over-constrained remote-sensing satellites. In *Proceedings of the second NASA international workshop on planning and scheduling for space*.

Pinedo, M. (2002). *Scheduling: theory, algorithms, and systems*. New York: Prentice Hall.

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

Schlack, S. M. (1993). *Automating satellite range scheduling*. M.Phil. thesis, Air Force Institute of Technology, USA.

Syswerda, G., & Palmucci, J. (1991). The application of genetic algorithms to resource scheduling. In *Proceedings of the 4th international conference on genetic algorithms* (pp. 502–508). San Mateo: Morgan Kaufmann.

Vasquez, M., & Hao, J.-K. (2003). A "logic-constrained" knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite. *Computational Optimization and Applications*, *7*, 87–103.

Verfaillie, G., & Lemaitre, M. (2001). Selecting and scheduling observations for agile satellites: some lessons from the contraint programming community point of view. In T. Walsh (Ed.), *Principles and practice of constraint programming* (pp. 670–684).

Wolfe, W. J., & Sorensen, S. E. (2000). Three scheduling algorithms applied to the earth observing systems domain. *Management Science*, *46*, 148–168.

Zufferey, N. (2002). *Heuristiques pour les problèmes de la coloration des sommets d'un graphe et d'affectation de fréquences avec polarités*. Ph.D. thesis, Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland.