RAIRO-Oper. Res. 49 (2015) 99–121
DOI: 10.1051/ro/2014027

RAIRO Operations Research
www.rairo-ro.org

A GENERALIZED CONSISTENT NEIGHBORHOOD SEARCH FOR SATELLITE RANGE SCHEDULING PROBLEMS

Nicolas Zufferey[1] and Michel Vasquez[2]

Abstract. Many optimization problems require the use of a local search to find a satisfying solution in a reasonable amount of time, even if the optimality is not guaranteed. Usually, local search algorithms operate in a search space which contains complete solutions (feasible or not) to the problem. In contrast, in *Consistent Neighborhood Search* (CNS), after each variable assignment, the conflicting variables are deleted to keep the partial solution feasible, and the search can stop when all the variables have a value. In this paper, we propose a generalized version of CNS, discuss its performance according to various criteria, and present successful adaptations of CNS to three types of satellite range scheduling problems. Such problems are motivated by applications encountered by the *French National Space and Aeronautic Agencies* and the *US Air Force Satellite Control Network*. The described numerical experiments will demonstrate that CNS is a powerful and flexible method, which can be easily combined with efficient ingredients.

Keywords. Metaheuristics, combinatorial optimization, satellite scheduling, consistent neighborhood search.

Mathematics Subject Classification. 9008.

Received April 26, 2014. Accepted May 5, 2014.

[1] Geneva School of Economics and Management, GSEM – University of Geneva, Uni-Mail, 1211 Geneva 4, Switzerland. n.zufferey@unige.ch

[2] École des Mines d'Alès, LGI2P Research Center, Site EERIE, Parc scientifique Georges Besse, 30035 Nimes Cedex 01, France. michel.vasquez@mines-ales.fr

1. Introduction

As discussed in [23], an *exact method* guarantees the optimality of the provided solution. However, for a large number of applications and most real-life optimization problems, such methods need a prohibitive amount of time to find an optimal solution, because such problems are NP-hard [8]. For these difficult problems, one should prefer to quickly find a satisfying solution, which is the goal of *heuristic* and *metaheuristic* solution methods. There mainly exist three families of (meta)heuristics: constructive algorithms (a solution is built step by step from scratch, like the greedy algorithm), local search methods (a solution is iteratively modified: this will be discussed below), and evolutionary metaheuristics (a population of solutions is managed, like genetic algorithms and ant algorithms).

Only the context of local search methods will be considered in this work. A *local search* algorithm starts with an initial solution and tries to improve it iteratively. At each iteration, a modification, called *move*, of the current solution is performed in order to generate a neighbor solution. The definition of a move, *i.e.* the definition of the *neighborhood* structure, depends on the considered problem. Popular local search methods are simulated annealing, tabu search, threshold algorithms, variable neighborhood search, and guided local search. For a survey on these local search methods as well as on other metaheuristics, the reader is referred to [9].

Within a local search context, the usual approach consists in working with *complete* solutions, *i.e.* each variable has a value and the solution might be feasible or not. In the latter case, a penalty function is often used, which depends on the number of violated constraints. In contrast, in *Consistent Neighborhood Search* (CNS, which was first introduced in [24]), partial feasible solutions are used. Thus, not every variable has a value but there is no constraint violation. In such a case, the goal is to minimize the number of non assigned variables, and a move is performed in at least two phases: (1) give a value to an unassigned variable s_i; and (2) delete the value of the created conflicting variables (*i.e.* the variables different from s_i involved in a constraint violation). An intermediate phase might occur between these two phases, which consists in adjusting the value of conflicting variables under some specific conditions.

In this paper, we present a generalized version of the CNS methodology and the adaptation of tabu search within its framework, then we discuss, with a unified terminology, the great success of some existing solutions methods, which can however be considered as belonging to the CNS methodology, for three NP-hard satellite range scheduling problems (denoted SAT1, SAT2 and SAT3). Only a baseline study is given for each problem. More precisely, for SAT1 (resp. SAT2 and SAT3), the reader is referred to [22] (resp. [12, 27]) to have detailed information on the NP-hard state, the complexity issues, the literature review, the experimental conditions (computer, language, *etc.*), and the presentation of the results. These three references are again given at the beginning of Sections 3–5. For each considered problem, the CNS approach will always be compared with state-of-the-art methods, even if such methods are not very recent. The main numerical results will be

highlighted, which allows to conclude on the excellent performance of CNS. Note that the reader interested in having accurate ways to design and analyze numerical experimentations is referred to [2,14].

The performance of a metaheuristic can be evaluated according to several criteria [26]. The most relevant criteria are: (1) *quality*: value of the obtained results, according to a given objective function; (2) *speed*: time needed to get good results; (3) *robustness*: sensitivity to variations in problem characteristics and data quality; (4) *ease of adaptation*: the ability to organize the method so that it can appropriately apply to different specific classes of problems; (5) *ability to take advantage of problem structure* (considering that efficiency often depends on making effective use of properties that differentiate a given class of problems from other classes). It will be discussed that the proposed CNS methodology has a good behavior according to these criteria.

The contributions of this paper, which is an extension of [23,24], are the following: (1) the proposed CNS method is a generalization of the existing one, as the following elements are new: the design and management of *active* and *relaxed* constraints, the development and classification of *deleting rules*, and the management of *threshold values* triggering a reparation phase; (2) the repairing phase procedure is accurately formulated and illustrated for a linear program and a graph coloring instance; (3) the performance of CNS is discussed according to a larger set of criteria (namely, quality, speed, robustness, ease of adaptation, and ability to take advantage of the problem structure); (4) the focus is specifically put on satellite range scheduling (three problems in this field are considered, denoted SAT1, SAT2 and SAT3); (5) ingredients which can be successfully combined with CNS are clearly identified when discussing SAT1, SAT2 and SAT3 (*e.g.*, intensification/diversification procedures, dynamic management of the tabu tenures).

The paper is organized as follows. In Section 2, a generalized CNS methodology is introduced. In the next sections, metaheuristics for various satellite range scheduling problems are described within a CNS framework. Such problems are motivated by the *French National Space and Aeronautic Agencies* and the *US Air Force Satellite Control Network*, for which they have important implications. In Section 3 is discussed the daily photograph scheduling of an Earth observation satellite, which aims at maximizing the total gain associated with the selected pictures. When dealing with several satellites, the so-called *multi-resource satellite range scheduling problem* is presented in Section 4. In Section 5 is depicted a situation where partial acquisitions are possible and transition times are considered. In Section 6, we end up the paper with a discussion focused mainly on the performance of CNS.

2. Consistent Neighborhood Search

In this section, we describe the CNS methodology, which has a search behavior between exhaustive tree search and local search working with complete solutions [23].

Let (P) be the considered problem with n variables $s_1, \ldots, s_n$, let f be the objective function to minimize, and let C be the set of constraints to satisfy. It is assumed that the constraint set C can be partitioned in two disjoint subsets A (*active* constraints) and B (*relaxed* constraints). A solution satisfying all the active (resp. relaxed) constraints is said to be A-*feasible* (resp. B-*feasible*). A solution is *feasible* if it is both A-feasible and B-feasible. It is of course possible to set $A = C$ and $B = \emptyset$ in order to only deal with feasible solutions. Further, each variable s_i can only have a value in its value domain D_i. A solution of (P) is denoted $s = (s_1, \ldots, s_n)$, where $s_i \in D_i$.

In most local search methods, the search space contains *complete* solutions, *i.e.* each variable s_i has a value in D_i, and the solutions can be feasible or not. If the search space only contains feasible solutions, the goal is generally to directly minimize the given objective function f associated with (P); otherwise, the aim often consists in minimizing $f(s)+\alpha \cdot p(s)$, where $p(s)$ penalizes the constraint violations associated with s, and α is a parameter which gives more or less importance to the constraint violations.

In contrast, a specificity of CNS consists in working with A-feasible partial solutions, *i.e.* where some s_i's do not have a value but all the active constraints are satisfied. In such a case, the goal is to minimize the number $\hat{f}(s)$ of non assigned variables in s. Let $\hat{F}$ be a threshold value associated with $\hat{f}$. Then, if a neighbor solution $s^{(\text{neighbor})}$ of the current solution $s^{(\text{current})}$ is found such that $\hat{f}(s^{(\text{neighbor})}) < \hat{F}$, then a repairing process is triggered to transform $s^{(\text{neighbor})}$ into a feasible partial solution (by deleting the value of some of the associated s_i's in order to remove the violations of the B constraints). The process stops of course if the two following conditions are met: $\hat{f}(s^{(\text{neighbor})}) = 0$ and $s^{(\text{neighbor})}$ is a feasible solution. We propose two options to manage $\hat{F}$. On the one hand, let $\hat{f}^\star$ be the smallest value of $\hat{f}$ encountered so far during the search process, which is associated with a feasible partial solution. A first option consists in setting $\hat{F} = \hat{f}^\star$. On the other hand, another option simply consists in setting $\hat{F} = \hat{f}(s^{(\text{current})})$. Other options to determine $\hat{F}$ could of course be derived.

Therefore, three search spaces are possible: (1) the complete and feasible search space $S^{(\text{feasible})}$; (2) the complete and non necessarily feasible search space $S^{(\text{penalty})}$, where unfeasible solutions are penalized; and (3) the partial and A-feasible search space $S^{(\text{partial})}$. When working in $S^{(\text{feasible})}$, it can be very difficult to define a move which maintains the feasibility of the solution. When working in $S^{(\text{penalty})}$, it is challenging to: (1) define a move which does not augment too much $p(s)$; (2) tune the above mentioned parameter α; and (3) find a feasible solution because $S^{(\text{penalty})}$ is much larger than $S^{(\text{feasible})}$. We will see that such drawbacks are avoided when working in $S^{(\text{partial})}$.

An important feature of CNS is the definition of the neighborhood structure in $S^{(\text{partial})}$. In most local search methods, in order to generate a neighbor solution s' from the current solution s, a move m consists in changing the value of one (or more) variable(s) of s. The set of neighbor solutions of s is denoted $N(s)$. Let $d(s, s')$ be the *distance* between s and $s' \in N(s)$. Usually, $d(s, s')$ is proportional

to the number of modified variables when moving from s to s', thus $d(s, s')$ is a constant for all $s' \in N(s)$.

For CNS in contrast, as presented below, any move m is performed in at least two phases. The distance between s and a neighbor solution in $N(s)$ is thus usually not a constant.

(1) *Assignment phase.* A value of D_i is assigned to a non assigned variable s_i. Let $C(m)$ be the set of conflicting variables (excluding s_i) created by move m (a variable is in *conflict* if it is involved in at least an A-constraint violation).
(2) *Reassignment phase (optional).* Reduce the set $C(m)$ as follows: for each variable of $C(m)$, if it is possible to assign an A-feasible value to it without creating new constraint violations, do it.
(3) *Repairing phase.* In order to keep the partial solution A-feasible, remove the value of all the variables of $C(m)$.

Note that the above repairing process relies on the assumption that it is always possible to repair an A-unfeasible solution by deleting the value of some decision variables. To better capture this assumption, two problems are briefly discussed below (namely a linear program and a graph coloring instance) and show how to manage the repairing phase.

Firstly, consider the following linear program with variables s_1 and s_2. The objective function $f(s_1, s_2)$ to maximize is equal to $2s_1 + 3s_2$. In addition to the positivity constraint (*i.e.*, $s_1, s_2 \geq 0$), the constraint of type A to satisfy is $s_1 + s_2 \leq 5$. Consider solution $s = (s_1, s_2) = (3, *)$, where $*$ indicates that s_2 has no value. Such a solution s is assumed to be A-feasible, because constraint $s_1 + s_2 \leq 5$ is assumed to be not violated. In contrast, solution $s = (6, *)$ is not A-feasible, because $s_1 + s_2 \leq 5$ is assumed to be violated. Further, from $s = (3, *)$, consider the move m which consists in assigning value 4 to variable s_2. In such an assignment phase, $C(m) = \{s_1\}$ (set of conflicting variables created by move m, excluding s_2) because constraint $s_1 + s_2 \leq 5$ is violated. Then, if no reassignment phase is performed (*i.e.* assign a new value to s_1 such that the updated solution is A-feasible), the repairing process simply consists in removing the value of s_1, and the neighbor solution is $s' = (s'_1, s'_2) = (*, 4)$.

Secondly, consider the following graph coloring instance associated with the famous k-graph coloring problem. The graph consists in three vertices (denoted s_1, s_2 and s_3) and two edges (namely $[s_1, s_2]$ and $[s_2, s_3]$). The number of available colors is $k = 2$. The constraint of type A to satisfy is that two adjacent vertices cannot receive the same color. Consider solution $s = (s_1, s_2, s_3) = (1, *, 2)$, where $*$ indicates that s_2 has no color. Such a solution s is assumed to be A-feasible because there is no pair of adjacent vertices having the same color. In contrast, solution $s = (1, 1, 2)$ is not A-feasible. Further, from $s = (1, *, 2)$, consider the move m which consists in assigning color 2 to vertex s_2. In such an assignment phase, $C(m) = \{s_3\}$ because s_2 and s_3 are adjacent and get the same color. Then, if no reassignment phase is performed, the repairing process simply consists in removing the color of s_3, and the neighbor solution is $s' = (1, 2, *)$. Then, the next

neighbor solution is likely to be $(1, 2, 1)$, which is a complete and feasible solution. Note by the way that CNS was very efficiently adapted for the k-coloring problem in [4], and thus for the graph coloring problem (as the latter can be tackled by sequentially solving a series of k-coloring problems, starting with k equal to the number of vertices of the graph).

In most local search algorithms, the selected neighbor solution s' of the current solution s is usually (one of) the best (according to f or $f + \alpha \cdot p$) solution chosen among a *sample* of $N(s)$. Sampling is usually unavoidable because it is too much time consuming to evaluate all the neighbor solutions of s, either because $N(s)$ is too large or because it is cumbersome to evaluate a single move m. An important issue is thus to determine the sample (random or not) as well as the size of the sample.

In contrast, in CNS, *all* the neighbor solutions can be considered at each iteration. This is possible in a reasonable amount of time because of two reasons: (1) it is quick to evaluate a neighbor solution by incremental computation: it is simply $|C(m)|$; (2) the number of non assigned variables in the current solution s is in general small when compared for example with the size $|N(s)|$ of the neighbor solutions of s in a standard local search approach (working in $S^{(\text{feasible})}$ or in $S^{(\text{penalty})}$). By construction, several neighbor solutions are likely to have the same value in CNS. In such a case, the way ties are broken can become an important issue. It is thus relevant to define a set $\mathcal{R} = \{\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \ldots\}$ of rules to sequentially use in order to break ties. The last rule of this set consists in performing a random choice. More precisely, if two neighbor solutions s_1 (obtained with move m_1) and s_2 (obtained with move m_2) have the same value $|C(m_1)| = |C(m_2)|$, then use rule $\mathcal{R}_1$ to break ties. If $\mathcal{R}_1$ is not able to discriminate between s_1 and s_2, then use $\mathcal{R}_2$, and so on until a selection is made. The set $\mathcal{R}$ of rules is called the *deleting rules*, as its role is to determine the set of values to remove from a current solution s in order to transform it into an A-feasible solution (because at the first step of a move, one of the unassigned s_i's gets a value which creates conflicts).

In summary, CNS is an approach dealing with partial A-feasible solutions, which can explore the whole neighborhood of the current solution at each iteration because a straightforward incremental computation can be designed. Many local search methods (*e.g.*, tabu search, simulated annealing, random walk, threshold algorithms, *etc.*) can be adapted within the framework of CNS. We have now all the ingredients to formulate a pseudo-code of CNS in Algorithm 1 (note that at step 4(b), the test "if $\hat{f}(s) < \hat{f}^\star$" has to be performed again because of the transformation occurring at step 4(a), as such a transformation might decrease the quality of s).

An important issue is that Algorithm 1 in its current state might cycle if a local optimum is reached. There are two ways to tackle this issue: (1) restart the algorithm when a local optimum is reached; (2) define a way to escape from local optima. As tabu search has an efficient mechanism to escape from local optima, its adaptation within the framework of CNS is relevant and now discussed. A generic and standard version of tabu search can be described as follows, assuming f

Algorithm 1 CNS

Initialization

(1) generate an initial B-feasible partial solution s;
(2) set $s^\star = s$ and $\hat{f}^\star = \hat{f}(s)$;

While a stopping time condition is not met and $\hat{f}^\star > 0$, do

(1) initialize the value of the best move: set $g = +\infty$;
(2) generate the best move: for each non assigned variable s_i and each value $d_j \in D_i$, test move $m = (s_i, d_j)$ on s as follows:
   (a) *assignment phase:* give value d_j to variable s_i and compute the associated set $C(m)$ of conflicting variables;
   (b) *reassignment phase (optional):* for each variable s_r of $C(m)$, if it is possible to assign an A-feasible value to s_r without augmenting $C(m)$, do it and remove s_r from $C(m)$;
   (c) let s^{cand} be the so obtained candidate neighbor solution (which might be non A-feasible at this stage);
   (d) update the best candidate move: if $|C(m)| \leq g$, do
      (i) if $|C(m)| = g$, use $\mathcal{R}$ to determine if $s' = s^{cand}$ should be set;
      (ii) if $|C(m)| < g$, set $s' = s^{cand}$ and $g = |C(m)|$;
(3) *repairing phase on the best move:* remove the value of the g conflicting variables of s' and let s be the resulting new current solution;
(4) update the record (among the B feasible solutions): if $\hat{f}(s) < \hat{f}^\star$, do
   (a) transform s into a B-feasible partial solution (by removing the value of some s_i's);
   (b) if $\hat{f}(s) < \hat{f}^\star$, set $s^\star = s$ and $\hat{f}^\star = \hat{f}(s)$

Output: solution $s^\star$ (which is a complete feasible solution if $\hat{f}^\star = 0$);

has to be minimized. First, tabu search needs an initial solution as input. Then, the algorithm generates a sequence of neighbor solutions. When a move is performed from s to s', the inverse of that move is forbidden during the following t (parameter) iterations (with some exceptions). The solution s' is computed as $s' = \arg \min_{s'' \in N'(s)} f(s'')$, where $N'(s)$ is a subset of $N(s)$ containing all solutions s' which can be obtained from s either by performing a move that is not tabu or such that $f(s') < f(s^\star)$, where $s^\star$ is the best solution encountered along the search so far. Usually, $N'(s)$ is too large and only a sample of neighbor solutions are selected from $N'(s)$ to be evaluated. The choice of the sample often has a strong impact on the final results. The process is stopped for example when an optimal solution is found (when it is known), or when a fixed number of iterations have been performed. Many variants and extensions of this basic algorithm can be found for example in [11].

Tabu search adapted within the framework of CNS has the following specificities: working in $S^{(\text{feasible})}$; minimizing $\hat{f}$ instead of f; exploring the whole neighborhood of the current solution; using an efficient and straightforward incremental computation; after each move when a value is given to a variable s_i and other values might be adjusted or deleted, it is then tabu to remove the value of s_i for a certain number of iterations.

In the next three sections are discussed satellite range scheduling problems which were tackled with CNS types of metaheuristics.

3. Basic satellite range scheduling

Scheduling requests (or jobs) on a satellite constellation is referred to as *satellite range scheduling problems*. The baseline study associated with this section is [22], in which the problem is referred to as the *daily photograph scheduling of an Earth observation satellite* (and denoted SAT1 in this paper). The authors proposed a tabu search approach working in $S^{(\text{partial})}$, denoted CNS-SAT1 below.

3.1. Description of the problem

The considered satellite range scheduling problem can be described as follows [3]. Let $P = \{p_1, \ldots, p_n\}$ be the set of n candidate photographs which can be scheduled to be taken on the next day. A set of possibilities is associated with each photograph p_i, corresponding to the different ways to take p_i: (1) for a *mono* p_i, there are three possibilities because a mono photograph can be taken by any of the three cameras (front, middle and rear) on the satellite; (2) for a *stereo* p_i, there is one single possibility because a stereo photograph requires simultaneously the front and the rear camera. With each mono photograph $p_i \in P$ are associated three pairs of elements $(p_i, \text{camera\_1}), (p_i, \text{camera\_2}), (p_i, \text{camera\_3})$. Similarly, with each stereo photograph $p_i \in P$ is associated one pair $(p_i, \text{camera\_13})$. Letting n_1 and n_2 be respectively the number of mono and stereo photographs in P (where $n = n_1 + n_2$), there are in total $m = 3 \cdot n_1 + n_2$ possible pairs of elements for the given set P of candidates. Now, associating a binary (decision) variable s_i with each such pair, a photograph schedule corresponds to a binary vector: $s = (s_1, s_2, \ldots, s_m)$, where $s_i = 1$ if the corresponding pair (photo, camera) is present in the schedule, and $s_i = 0$ otherwise. For example, if $P = \{p_1, p_2, p_3\}$ where p_1 and p_2 are mono photographs and p_3 is a stereo photograph, then $s = (1, 0, 0, 0, 0, 0, 1)$ represents a schedule in which p_1 is taken by camera 1, p_2 is rejected and p_3 is taken by cameras 1 and 3.

SAT1 consists in finding a subset P' of P which satisfies all the imperative constraints and maximizes the sum of the profits of the photographs in P'. The profit function reflects several criteria such as client importance, demand urgency or meteorological forecast. The objective function can be defined as follows. First, the profit of a pair (p, camera) (or its 0-1 variable) is defined as the profit of the photograph p. The total profit of all the pairs of the given set P is then represented by a vector: $g = (g_1, g_2, \ldots, g_m)$, where $g_i = g_j$ $(i \neq j)$ if g_i and g_j correspond to two different pairs of elements involving the same photograph p, *i.e.* (p, camera_x) and (p, camera_y). Then the total profit value of a schedule $s = (s_1, s_2, \ldots, s_m)$ is the sum of the profits of the photographs in s, *i.e.* $f(s) = \sum_{i=1}^{m} g_i \cdot s_i$.

A capacity constraint is the following. A size is associated with each photograph p_i, which represents the amount of memory required to record p_i when it is taken. The size of a pair (p, camera) (or its 0-1 variable) is defined as the size of the photograph p. The total size of all the pairs of the given set P is then represented by a vector: $c = (c_1, c_2, \ldots, c_m)$, where $c_i = c_j$ $(i \neq j)$ if c_i and c_j correspond to two different pairs of elements involving the same photograph p, *i.e.* (p, camera_x) and (p, camera_y). The capacity constraint states that the sum of the sizes of the photographs in a schedule $s = (s_1, s_2, \ldots, s_m)$ cannot exceed the maximal recording capacity on board, which is expressed as $\sum_{i=1}^{m} c_i \cdot s_i \leq Max\_capacity$.

Binary constraints involving the non overlapping of two trials and the minimal transition time between two successive trials of a camera, and also some constraints involving limitations on instantaneous data flow, are conveniently expressed by simple relations over two pairs (photo, camera). A binary constraint forbids the simultaneous presence of a pair (p_i, k_i) and another pair (p_j, k_j) in a schedule. If s_i and s_j are the corresponding decision variables of such two pairs, then a binary constraint is defined as follows: $s_i + s_j \leq 1$. Let C2 denote the set of all such pairs (s_i, s_j) which should verify the above binary constraint.

Some constraints involving limitations on instantaneous data flow cannot be expressed in the form of binary constraints as above. These remaining constraints may however be expressed by relations over three pairs (photo, camera). A ternary constraint forbids the simultaneous presence of three pairs $(p_i, k_i), (p_j, k_j)$, and (p_l, k_l). Letting s_i, s_j and s_l be the decision variables corresponding to these pairs, then such a ternary constraint is written: $s_i + s_j + s_l \leq 2$. Let C3_1 denote the set of all such triplets (s_i, s_j, s_l) which should verify this ternary constraint.

Finally, we need to be sure that a schedule contains no more than one pair from $\{(p, k_i), (p, k_j), (p, k_l)\}$ for any (mono) photograph p. Letting s_i, s_j and s_l be the decision variables corresponding to these pairs, then this ternary constraint is expressed as: $s_i + s_j + s_l \leq 1$. Clearly there are exactly n_1 ternary constraints of this type. Let C3_2 denote the set of all such triplets (s_i, s_j, s_l) which verify this second type of ternary constraints. C3 denotes the union of C3_1 and C3_2, *i.e.* C3 = C3_1 $\cup$ C3_2.

3.2. Description of the method

First, the set B of relaxed constraints only contains the capacity constraint. Thus, the binary and ternary constraints belong to the set A of active constraints. The relaxation of the capacity constraint clearly helps to obtain better results, to accelerate the search, and to have a more robust method. This was experimentally confirmed in [22].

Let $s = (s_1, s_2, \ldots, s_m) \in$ C and $s' = (s'_1, s'_2, \ldots, s'_m)$, then s' is a neighbor of s, *i.e.* $s' \in N(s)$, if and only if the following conditions are verified:

(1) there is only one i such that $s_i = 0$ and $s'_i = 1$, for $1 \leq i \leq m$;
(2) for the above i, $\forall (s_i, s_j) \in$ C2, $s'_j = 0$ for $1 \leq j \leq m$;
(3) for the above i, $\forall (s_i, s_j, s_k) \in$ C3_1, $s'_j + s'_k \leq 1$ for $1 \leq j, k \leq m$.

Thus, a neighbor of s can be obtained by adding a pair (photo, camera) (*i.e.* flipping a variable s_i from 0 to 1) in the current schedule and then dropping some pairs (photo, camera) (*i.e.* flipping some s_j's from 1 to 0) to repair binary and ternary constraint violations. A binary constraint violation is easy to repair since there is a single way to do this. However, the case for a ternary constraint violation is more complicated since there are different ways to achieve such a repair. Suppose that $s_j = s_k = 1$ and the move sets s_i to 1. The constraint $s_i + s_j + s_k \leq 2$ is thus violated and needs to be repaired. Experiments leads to the following *deleting rules*, which can be sequentially applied.

$\mathcal{R}_1$ Look ahead in an exhaustive way to determine the best choice in terms of lost profit.
$\mathcal{R}_2$ Set to 0 the element which has a smaller profit.
$\mathcal{R}_3$ Randomly set either s_j or s_k to 0.
$\mathcal{R}_4$ Set both s_j and s_k to 0.

For the instances used in this study, experiments showed that no more than five elements need to be reset to repair a ternary constraint violation after a move, confirming the relevance of $\mathcal{R}_1$. For the cases where more elements need to be forward-checked, $\mathcal{R}_2$ should be the best choice.

As B contains the capacity constraint, it may be violated by the current solution $s = (s_1, s_2, \ldots, s_m)$ (*i.e.*, the total size of s may exceed the maximal allowed capacity). To satisfy the capacity constraint, the following mechanism is used. Each time the current solution is improved, the capacity constraint is checked. If the constraint is violated, the solution is immediately repaired by suppressing the s_i's which have the worst ratio g_i/c_i until the capacity constraint is satisfied. In other words, the option $\hat{F} = \hat{f}(s^{(\text{current})})$ proposed in Section 2 is used.

Each time a move is carried out, a single variable s_i flips from 0 to 1, and several s_j's flip from 1 to 0. It is then tabu to flip again these s_j values from 0 to 1 during $tabu(j)$ iterations, where $tabu(j) = C(j) + \alpha \cdot freq(j)$, where $C(j)$ is the number of binary and ternary constraints involving the element s_j, $freq(j)$ the number of times s_j is flipped from 1 to 0 from the beginning of the search, and α is an instance-dependent coefficient which defines a penalty factor for each move. To explain this, a variable involved in a large number of constraints has naturally more risk to be flipped during a move than a variable having few constraints on it. It is thus logic to give a longer tabu tenure for a move whose variable has many constraints on it. The second part of the function aims to penalize a move which repeats too often.

Intensification and diversification procedures were also used to enhance the efficiency and the robustness of CNS-SAT1. The intensification process is based on the following idea: if an element is present in a large number of good solutions, then it is highly possible that this element is part of an optimal solution. In contrast, the diversification process focuses on the jobs which a small flipping frequency (because such jobs are likely to be almost ignored by the regular search process).

TABLE 1. Results on the SAT1 instances.

Instance	n	m	$f^{\star}_{TS}$	$Time_{TS}$	$\overline{f}_{CNS}$	$Time_{CNS}$
1401	488	914	174 058	846	176 055	120
1403	665	1317	174 137	1324	176 134	332
1405	855	1815	174 174	1574	176 175	1314
1407	1057	2355	174 238	2197	176 241	2422
1504	605	1253	124 238	1011	124 241	405
1506	940	2060	165 244	1945	168 224	1423
Average			164 348.17	1272.86	166 178.33	859.57

3.3. COMPARISONS WITH OTHER METHODS

Experiments were carried out on a set of 20 realistic instances provided by the CNES (*French National Space Agency*) and described in details in [3]. These instances belong to two different sets: without capacity constraint (13 instances) and with capacity constraint (7 instances). The instances without capacity constraints, as well as one instance with capacity constraint, will not be commented: it is very easy to solve them, either with an exact method or with the above described CNS-SAT1 algorithm. The other six instances have from 488 to 1057 candidate photographs, giving up to 2355 binary variables and 35 933 constraints.

The best known non exact algorithm was a tabu search TS-SAT1 proposed by the CNES. The main differences with CNS-SAT1 algorithm are the following. (1) TS-SAT1 uses a different (integer) formulation of the problem: (2) it manipulates only feasible solutions (the search space is thus $S^{(\text{feasible})}$); (3) it uses a different neighborhood structure; (4) it considers only a sample of neighbor solutions to make a move; (5) the tabu tenure for each move is randomly taken from predefined (very small) ranges.

To solve an instance, CNS-SAT1 is allowed to run 9 million iterations on a PC (200 MHz, 32 MB of RAM), which is considered as reasonable by the CNES. CNS-SAT1 was run 100 times on each instance with different random seeds and the average value is returned for each instance. The first three columns of Table 1 give the name of the instance, the number of candidate photographs n, and the number of 0-1 variables m. Columns 4 and 5 respectively show the best profit $f^{\star}_{TS}$ and the associated computing time $time_{TS}$ (in seconds) obtained with TS-SAT1. Columns 6-7 give the average profit value $\overline{f}_{CNS}$ and the average time $time_{CNS}$ needed by CNS-SAT1 to find such a solution. Given that we compare average values for CNS-SAT1 *versus* best values for TS-SAT1, we can observe that CNS-SAT1 appears to be more efficient and significantly quicker than TS-SAT1 (see also the line labeled *Average*).

At the time of publication of CNS-SAT1 (*i.e.* in 2001), existing exact algorithms were unable to solve optimally these instances (*i.e.* an ILP formulation relying on CPLEX, and a non-standard Branch and Bound algorithm using a Valued Constraint Satisfaction Problem formulation [25]). Nine years later, it was showed

in [19] that CNS-SAT1 provides very good solutions, and for the difficult instances (*i.e.* the instances with capacity constraint considered here), CNS-SAT1 found all optimal solutions, except for instance 1403 (for which the gap is close to 1%).

4. Multi-resource satellite range scheduling

The baseline study associated with this section is [27], where the *multi-resource satellite range scheduling problem* is tackled (denoted here by SAT2 for sake of simplicity). In contrast with SAT1, several satellites are considered.

4.1. Description of the problem

Consider a set of satellites and a set $\{R_1, R_2, \ldots, R_k\}$ of ground stations. Ground stations are communication facilities (*e.g.* antennae). 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. SAT2 is a NP-hard problem [1] for which a set $J = \{1, \ldots, n\}$ of jobs have to be scheduled. Each job j is characterized by the following parameters: the (unique) satellite sat_j requested by j; the set M_j of ground stations able to process j; the duration p_j of communication j; the time r_j at which j becomes available for processing; the time d_j by which j must be completed. Note that p_j, r_j and d_j may depend on the considered ground station. An important application is the one encountered by the *US Air Force Satellite Control Network*, where more than 100 satellites and 16 antennae located at 9 ground stations are considered. In such an context, 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 can not be performed. The goal is to schedule as many jobs as possible within its time window, such that the processing of two jobs can not overlap on the same resource (unit capacity): in such a case, there is a conflict and one of the conflicting jobs has to be removed from the schedule. Minimizing the number of conflicting jobs (more precisely, the number of bumped jobs) is of crucial importance in a practical standpoint, because human schedulers do not consider any conflicting job worse than any other conflicting job [18]. 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 [1].

4.2. Description of the method

First, the set B of relaxed constraints is set to $\emptyset$. The component s_j of solution s indicates the resource on which job j is scheduled if j is scheduled (otherwise, an artificial value 0 can be given). For each solution s, there is an associated vector g where component g_j indicates the starting time of job j performed on resource s_j. As constraint violation are forbidden, for each resource R_i (with $i \in \{1, \ldots, k\}$),

all the jobs are scheduled within their time windows and there are no overlapping jobs.

In order to generate a neighbor solution s' from s, a bumped job j is first scheduled within its time window on a resource $R_i \in M_j$. More precisely, the assignment phase is the following. Assume that jobs $j_1, \ldots, j_r$ are already scheduled (in that order) on resource R_i. In order to schedule job j in R_i within its time window, two main situations may occur. In the best situation, it is possible to successively 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 R_i without creating any conflict (no overlapping). Otherwise, in the repairing phase, some non tabu jobs have to be removed from R_i. The goal is to remove the smallest number of non tabu jobs. If there are several possibilities, the following *deleting rules* are sequentially applied.

$\mathcal{R}_1$ Remove the set of jobs with the largest average flexibility (where the *flexibility* of a job is number of resources it can be scheduled on). The larger is the flexibility of a job, the easier it would be to re-schedule it on another resource.

$\mathcal{R}_2$ Remove the set of jobs with the largest average age (where the *age* of a job is the number of iterations from which it is in the solution). The removal of old jobs will help to diversify the search.

$\mathcal{R}_3$ Remove a random set of jobs.

When a job j is inserted into a resource R_i, it is forbidden to remove it for $tab(j)$ iterations, with $tab(j) = \gamma \cdot n_c + U(a, b)$, where n_c is the number of bumped jobs in the current solution, and $U(a, b)$ is a function which returns a number randomly generated in the set $\{a, a+1, \ldots, b-1, b\}$. Parameters γ, a, b have been respectively tuned to 0.6, 0 and 9. The component $U(0, 9)$ allows to add some useful randomness in the process, which helps to diversify the search. Such a dynamic way of managing the tabu tenures is likely to escape from a region of the solution space containing solutions with a large set of bumped jobs.

In order to better diversify the search, two diversification mechanisms, denoted *DIV1* and *DIV2*, are used. The goal of *DIV1* is to totally renew the set of non scheduled jobs by forcing their insertion in the current schedule, and the goal of *DIV2* is to renew the set of scheduled jobs by removing old jobs from the current solution. After a given number (parameter) of iterations without improvement of the best solution encountered so far, *DIV1* or *DIV2* is performed (not both, but in turns).

4.3. Comparisons with other methods

It is relevant to compare the three following heuristics: (1) CNS-SAT2 (which also uses diversification procedures); (2) AMA, which is an adaptive memory algorithm using CNS-SAT2 as intensification procedure (note that the AMA generic methodology was introduced in [20]); (3) GENITOR [1], which is a genetic algorithm that usually provides the best solutions for SAT2 up to 2007. GENITOR is based on the permutation search space, where, as proposed by several researchers

Table 2. Results on the SAT2 instances.

Instance	$\overline{f}_{GEN}$	$\overline{f}_{CNS}$	$\overline{f}_{AMA}$	Instance	$\overline{f}_{GEN}$	$\overline{f}_{CNS}$	$\overline{f}_{AMA}$
1	115.75	55.75	55.5	11	106	51.75	52
2	101.5	54	54	12	106.25	46.5	46.75
3	125.75	65.75	63.5	13	111	56.75	56.75
4	117	49.5	50.25	14	112	58	57.5
5	113.75	52.25	51.75	15	108.75	42.5	42.75
6	121	59	58.75	16	119.75	58	58.25
7	120.5	53.25	52.5	17	107.25	50.25	50.5
8	103.5	49.5	48.75	18	112.25	53.25	52.5
9	97	38.25	38.5	19	100.25	47.5	47
10	104	50.75	51.5	20	96.75	41	41

(*e.g.* [1, 10]), a solution is encoded as a permutation π of the n jobs to schedule. Let $S^{(\text{permutation})}$ be the search space containing all the possible permutations. From a permutation π in $S^{(\text{permutation})}$, it is possible to generate a schedule in $S^{(\text{feasible})}$ by the use of a schedule builder, which is a greedy constructive heuristic.

The most difficult benchmark instances are of size 500 (*i.e.* $n = 500$) with $k = 9$ resources (as described in [27]). Each instance was built by a well-known generator described and used in [1]. It produces instances of SAT2 by modeling realistic features. A maximum time limit of 15 min Pentium 4 (2 GHz, 512 MB of RAM) is considered in Table 2, which is consistent with the ones used in [1], and is appropriate in a practical standpoint. In Table 2, we compare the average number of bumped jobs of GENITOR, CNS-SAT2 and AMA, respectively denoted $\overline{f}_{GEN}$, $\overline{f}_{CNS}$ and $\overline{f}_{AMA}$. For each method and each instance, 30 runs were performed. We can easily compute that in average, there are 110 bumped jobs by GENITOR, 51.68 bumped jobs by CNS-SAT2, and 51.5 bumped jobs by AMA. Such huge differences between the previous best existing algorithm for SAT2 (namely GENITOR) and CNS-based methods (namely CNS-SAT2 and AMA) obviously show the outstanding performance of CNS approaches for this problem.

5. Satellite scheduling with partial acquisition and transition times

The baseline study associated with this section is [12], in which the considered problem (denoted SAT3) consists in scheduling the photographs of an agile Earth observing satellite.

5.1. Description of the problem

The *satellite range scheduling with partial acquisition and transition times problem* (SAT3) consists again in selecting and scheduling a subset of requests yielding

a maximal gain, subject to operational constraints, while considering a single orbit of one satellite. In contrast with SAT1 and SAT2, a request can either be a target or a polygon. In the latter case, several photographs have to be taken in order to capture the polygon, and a non linear profit function is associated with the partial acquisition of a polygon, which makes the problem harder. Another difference is the consideration of transition times between the realization of two shots.

More precisely, a *target* consists of a single strip (rectangular shape), whereas a *polygon* may cover a wide geographical area. Because of their size, polygons cannot usually be photographed in a single shot. For this reason, they are partitioned into strips of equal width but possibly unequal lengths and the time required to acquire a strip is proportional to its length. In the course of a single orbit, the satellite may be able to photograph several strips of the same polygon. Because of its agility, it is also able to acquire a strip using two opposite azimuths (direct and indirect) according to the satellite rotation sense. Hence, two shots (or images) per strip are possible. The starting date of the acquisition of each shot must be in accordance with its visibility time window. Moreover, some requests are mono while others are stereo. A *mono* request consists of a single shot of each strip in the polygon. A *stereo* request consists of two shots of each strip at different angles but in the same direction. A strip from a stereo request is considered to have been acquired only if its twin strip has also been acquired. Finally, for each pair of shots, the satellite requires a minimum transition time to maneuver the camera from the end of the first strip to the start of the second one.

Below are formally presented the given data, the constraints and the objective function.

5.1.1. Given data

A problem with n strips involves $2n$ possible acquisitions since for each strip i two shooting directions are possible. These shots are numbered $2i-1$ (an odd number for a shot acquired in the direct azimuth) and $2i$ (an even number for a shot acquired in the indirect azimuth).

For each strip $i \in [1,n]$ let: $tw(i)$ be the index of its stereo twin strip, 0 if i is mono; $d(i)$ be its shooting duration; $su(i)$ be its corresponding surface.

For each shot (image) $j \in [1,2n]$ let: $es(j)$ and $ls(j)$ be its earliest and latest start dates, respectively; $ee(j)$ and $le(j)$ be its earliest and latest end dates, respectively. For each shot pair (i,j), $i \neq j \in [1,2n]$, $t(i \rightarrow j)$ denotes the minimum transition time between the end of i to the beginning of j. It is assumed that $t(i \rightarrow j) \geq es(j) - le(i)$ (otherwise it is obviously underestimated).

Four variables are associated with each shot $i \in [1,2n]$: (1) $x_i \in \{0,1\}$ equals 1 if and only if shot i is selected; (2) $y_{0 \rightarrow i} \in \{0,1\}$ equals 1 if and only if shot i is the first of the selection; (3) $y_{i \rightarrow 2n+1} \in \{0,1\}$ equals 1 if and only if shot i is the last of the selection; (4) t_i is the shooting start date of i. The value of this variable is irrelevant when $x_i = 0$. The t_i values allow to order and to schedule the shots in time.

Let m be the number of polygons. The kth polygon is characterized by: its total area surface $s(k)$; its gain g_k when fully acquired; the set $p(k) \subset [1, 2n]$ of shots that it contains. Remind that each polygon is divided into a set of strips. Moreover, two shots are possible per strip according to the sense of its acquisition. Two continuous variables are associated with each polygon $k \in [1, m]$: $S_k \in [0, 1]$ is the percentage of the surface covered by the selected strips; $G_k \in [0, 1]$ is the corresponding percentage of the polygon gain. Finally, one binary variable is defined for each pair of acquisitions $i \neq j \in [1, 2n]$: $y_{i \to j} \in \{0, 1\}$ equals 1 if and only if the shot j is immediately acquired after the shot i.

5.1.2. Constraints

The equations constraining the variables are listed below. The acquisition of the same strip in both directions is forbidden by (1). Equation (2) states the stereo constraints: simultaneous selection with identical direction. The time window for the starting time of a shot is imposed by (3). Shooting dates of consecutive images must respect minimum transition times (4). These last two equations correspond to the time constraints.

(1) $\forall j \in [1, n]$, $x_{2j-1} + x_{2j} \leq 1$.

(2) $\forall j \in [1, n]$, if $tw(j) \neq 0$, then $(x_{2j-1} = x_{2tw(j)-1}$ and $x_{2j} = x_{2tw(j)})$.

(3) $\forall i \in [1, 2n]$, if $x_i = 1$, then $t_i \in [es(i), ls(i)]$.

(4) $\forall i \neq j \in [1, 2n]$, $t_j - t_i \geq (d(i) + t(i \to j)) \cdot y_{i \to j} + (es(j) - ls(j)) \cdot (1 - y_{i \to j})$. $C4$ denotes the set of shot pairs $(i,\ j)$ that satisfy the condition (4).

Furthermore, there is at most one first shot and one last shot (5), and each selected acquisition has exactly one predecessor and one successor (6).

(5) $\displaystyle\sum_{i \in [1, 2n]} y_{0 \to i} \leq 1$ and $\displaystyle\sum_{i \in [1, 2n]} y_{i \to 2n+1} \leq 1$.

(6) $\forall i \in [1, 2n]$, $\displaystyle\sum_{j \in [0, 2n+1] | j \neq i} y_{j \to i} = x_i = \sum_{j \in [0, 2n+1] | j \neq i} y_{i \to j}$.

In the remaining part of this section, constraints (5) and (6) are assumed to be always satisfied.

5.1.3. Objective function

The criterion to maximize is a global gain G defined by the sum of the gains associated with the complete or partial acquisition of each polygon k and formulated by: $G = \sum_{k=1}^{m} g_k \cdot G_k$ such that:

- $\forall k \in [1, m]$, $S_k = \dfrac{1}{s(k)} \displaystyle\sum_{i \in p(k)} su(i) \cdot x_i$.
- $G_k = f(S_k)$.

- $f : [0,1] \to [0,1]$ is a non linear function, piecewise linear and defined by the points $\{(0,0),(0.4,0.1),(0.7,0.4),(1,1)\}$.

5.2. Description of the method

CNS-SAT3 was combined with a secondary optimization problem, the latter being tackled by a "classical" tabu search algorithm. The set B of relaxed constraints is set to $\emptyset$.

5.2.1. Solution representation and neighborhood structure

A solution s can be modeled as $((x_1,t_1),(x_2,t_2),\ldots,(x_{2n},t_{2n}))$, with $i \in \{1,\ldots,2n\}$ such that $x_i \in \{0,1\}$ and t_i is a real, and such that constraints (1) to (4) are satisfied. In this formulation, i is a shot number among the $2 \cdot n$ possible ones issued from the n strips (two shots per strip), and t_i is the (unconstrained) beginning time acquisition of the shot i. Each vector s is evaluated by its corresponding gain $G(s) = \sum_{k=1}^{m} g_k \cdot f(\frac{1}{s(k)} \sum_{i \in p(k)} su(i) \cdot x_i)$. The number of shots that are selected in s is $|s| = \sum_{i=1}^{2n} x_i$.

The neighborhood function $N : X \to (2^X - \emptyset)$ is defined over the totally constrained search space X as follows (*i.e.* the set B of relaxed constraints is set to $\emptyset$). The solution $s' = ((x'_1,\ t'_1),(x'_2,\ t'_2),\ldots,(x'_{2n},\ t'_{2n}))$ is a neighbor of solution $s = ((x_1,\ t_1),(x_2,\ t_2),\ldots,(x_{2n},\ t_{2n}))$ (*i.e.* $s' \in N(s)$), if and only if the following conditions are checked.

(1) $\exists!\ i \in [1,2n]$ such that $x_i = 0$ and $x'_i = 1$. Moreover, if $tw(i) \neq 0$, then $x_{tw(i)} = 0$ and $x'_{tw(i)} = 1$ (try to insert exactly one shot in s, and its twin if it exists).

(2) For any shot i that satisfies the condition (1) and a shot $j \in [1,2n]$ such as i and j are issued from the same strip, we have $x_j = x'_j = 0$. Moreover, if $tw(i) \neq 0$, then $x_k = x'_k = 0$ such as the shots $tw(i)$ and k are also issued from the same twin strip. This condition forbids the acquisition of a strip (and its twin if it exists) in two directions.

(3) For any shot i that satisfies the condition (1), we have $t_i \in [es(i),\ ls(i)]$. Moreover, if $tw(i) \neq 0$, then $t_{tw(i)} \in [es(tw(i)),\ ls(tw(i))]$. This condition deals with the time window visibility constraint.

(4) For each shot i that satisfies the condition (1), $\forall k \in [1,\ 2n]$ such that $(i,\ k) \notin C4$ and $x_k = 1$, we have $x'_k = 0$ and if $tw(k) \neq 0$, then $x'_{tw(k)} = 0$. Moreover, if $tw(i) \neq 0$, then $\forall l \in [1,\ 2n]$ such that $(tw(i),\ l) \notin C4$ and $x_l = 1$, then $x'_l = 0$. In addition, if $tw(l) \neq 0$, then $x'_{tw(l)} = 0$.

(5) For any shot i that satisfies the condition (1), $-3 \leq |s'| - |s| \leq 1$. Also, if $tw(i) \neq 0$, then $-6 \leq |s'| - |s| \leq 2$.

(6) For any shot i that satisfies the condition (1) and such that its insertion in s requires at most the removal of two shots $j,\ k \in [1,\ 2n]$ with their twin shots

if they exist ($-3 \leq |s'| - |s| \leq -1$, $x_j = x_k = 1$, $x'_j = x'_k = 0$ and $j \neq tw(k)$), then even $y_{j \to k} = 1$ or $y_{k \to j} = 1$ in s (k and j are acquired one behind the other in s).

Thus, the neighborhood of s is obtained by adding a free shot i (not yet scheduled, $x_i = 0$) by flipping x_i from 0 to 1 (condition 1), then removing some shots k (by flipping x_k from 1 to 0) to repair the violated constraints (condition 4). In fact, inserting a new shot may require to drop a certain number of the already fixed ones to maintain the consistency of the constraints (1) to (4). However, the condition (5) enforces a maximum of two shot removals if all the dropped shots are mono, and four if they are stereo (in order to maintain the consistency of the stereo constraint, if a stereo shot is removed then its twin is removed too).

According to the conditions mentioned above, the best case is the insertion of a stereo shot and its twin without any removal ($|s'| - |s| = 2$). Oppositely to this case, the worst one corresponds to two insertions (stereo shot plus it twin: +2), such as each of these two insertions needs the removal of two stereo shots plus their twins: −4 ($|s'| - |s| = +2 - 4 - 4 = -6$). Moreover, this choice heuristic is also restricted to the removal of successive shots as described in the condition (6).

5.2.2. Enumerations for the neighborhood evaluation

$N(s)$ is evaluated according to the gain criterion. For this purpose, consider a solution $s = ((x_1, t_1), (x_2, t_2), \ldots, (x_{2n}, t_{2n}))$ where $|s|$ images are selected and an image j such that $x_j = 0$ (*i.e.* j is not yet selected). The shot j can be inserted in s through $|s| + 1$ positions: before the first shot, after the last shot, or between two successive shots on the schedule s. Hence, as it was similarly done in CNS-SAT2, the insertion of the shot j in each of the $|s| + 1$ positions is tested by allowing successive image removals (as explained above). If a position is tested positively, then a neighbor solution s' is reached by inserting j at this position in s (and dropping some others images, if necessary). Solution s' is evaluated by computing its corresponding gain value $G(s')$. Among all the feasible insertions of j, the one that maximizes the gain value is selected. Consequently, at each step of CNS-SAT3, the decision problem of finding the best insertion position for each free shot in s according to the gain function is solved. If several moves lead to the same objective function value (such a situation often occurs), the following rules are sequentially used.

$\mathcal{R}_1$ Minimize the sum of the transition times.
$\mathcal{R}_2$ Perform a random choice.

5.2.3. Other successful ingredients

The tabu tenure is dynamically formulated by: $tabu(i) = iter + \alpha \cdot freq(i)$, where: (1) *iter* is the number of the current iteration of the tabu algorithm, (2) $freq(i)$ counts the number of times that the shot i has been selected by the tabu algorithm (note that a shot can be inserted at a given iteration and be dropped some iterations later, then reselected after and so on), and (3) α is a variable parameter used

to weight $tabu(i)$ according to $freq(i)$. Moreover, a sequence of shots is tabu if all its shots are tabu, and not tabu if it contains at least one non tabu shot. Then, a candidate neighbor schedule obtained from the current schedule by removing a tabu sequence is also tabu, otherwise it is not tabu.

The intensification procedure is based on a second objective function: the sum of transition durations that separate the acquisition of the shots. Such a problem is also tackled with a tabu search, but in its more usual form. For a given solution s, the sum of the transition times between the strips shooting is $TdT(s) = \sum_{i=1}^{2n} \sum_{j=1, j\neq i}^{2n} x_i \cdot x_j \cdot y_{i\rightarrow j} \cdot d(i \rightarrow j)$. The reduction of this sum can generate visibility time windows sufficiently broad and usable by the satellite to acquire new shots and without removing those that are already selected. The minimization of TdT is based on two operations: the exchange of the order of the shots and the inversion of the acquisition direction of the strips. The order exchange is inspired by the *2-opt* operation [15] which is widely used in solving the Traveling Salesman Problem.

The diversification procedure mainly consists in restarting CNS-SAT3 from a different initial solution.

5.3. Comparisons with other methods

The SAT3 was the subject of the Challenge ROADEF 2003. The problem was proposed by CNES and ONERA (the *French National Space and Aeronautic Agencies*). The two first ranks were obtained by the following teams (see http://www.roadef.org/content/roadef/challenge.htm for more details on the instances and the other competitors).

The Cordeau and Laporte team proposed a tabu search algorithm which borrows from the *Unified Tabu Search Algorithm* [5] developed for the vehicle routing problem with time windows. An important feature of their algorithm is the possibility of exploring infeasible solutions during the search by allowing the violation of the time window constraints. Moreover, the algorithm uses two powerful diversification mechanisms.

E.J. Kuipers proposed a local search algorithm based on two stages. In the first one, the most promising solutions are constructed then further optimized in the second stage. These two parts use simulated annealing algorithms. The neighborhood is constructed in two steps. In the first one, from one to four requests (or parts of requests) are removed from the current solution, and in the second step, from one to four requests (or parts of requests) are added to the solution resulting from the first one.

The provided instances are artificially generated, with the number of requests (m value) ranging from 2 to 375, and the number of strips (n value) varying from 2 to 534 with a maximum of 113 stereo strips (n_{stereo} value). Table 3 gives the properties of each of the 20 used instances. All the experiments were carried

Table 3. Results on the SAT3 instances.

Instance	m	n	n_{stereo}	$G^\star$	$\overline{Gap}_{CNS}$	$Gap^\star_{CNS}$	$time_{CNS}$
2 9 36	2	2	0	10 423 440	0	0	1
2 9 66	4	7	0	115 710 959	0	0	1
2 13 111	68	106	12	563 597 071	0	0	90
2 15 170	218	295	39	719 417 220	0	0	561
2 26 96	336	483	63	1 005 301 900	0	–1.55	443
2 27 22	375	534	67	967 910 750	–0.13	–0.13	57
2 9 170	12	25	4	191 358 231	0	0	1
3 25 22	150	342	113	425 983 220	0	0	95
3 8 155	12	28	10	121 680 360	0	0	1
4 17 186	77	147	48	185 406 200	0	0	1
2 21 140	284	420	58	1 029 892 360	0.02	–0.06	467
2 21 155	311	472	55	1 150 632 847	0	–1.57	762
2 21 170	294	450	71	891 060 370	2.68	2.37	1151
2 21 22	306	455	54	1 160 366 840	0.02	0.02	831
2 21 37	315	477	62	954 965 580	0	–0.02	602
2 21 7	289	410	49	842 378 700	0	0	116
2 21 81	297	436	59	986 679 410	0	–0.08	778
2 21 96	291	437	49	1 133 044 250	0.13	–0.25	932
3 21 155	135	295	105	460 196 570	0	0	3
3 21 81	135	283	88	373 553 350	0	0	16
Average				664 477 981	0.18	–0.14	345

on a Pentium 4 (1.9 GHz, 512 MB of RAM) with a imposed maximum time limit of 1800 s.

In Table 3, the following information is respectively given. The name of the instance, the m, n and n_{stereo} values, the value $G^\star$ of the best known solution, the average percentage gap $\overline{Gap}_{CNS}$ (over 10 runs) between CNS-SAT3 and $G^\star$, the smallest percentage gap $Gap^\star_{CNS}$ between CNS-SAT3 and $G^\star$, and the average computing time $time_{CNS}$ (in seconds) needed by CNS-SAT3 to provide a solution. CNS-SAT3 got the third rank in the Challenge (involving more than 50 research teams from all over the world). We can see that the obtained results are in average very close to the best ones (even better on two instances). In addition, the best results of CNS-SAT3 are usually better than the best known results. Such elements allow to conclude that CNS is very appropriate and competitive for SAT3.

6. Discussion and conclusions

In this paper, we propose an extended version of CNS, which is a generic method appropriate for hard combinatorial optimization problems. It was showed that CNS was efficiently adapted to various satellite scheduling problems. It is important to mention that CNS approaches were also successfully adapted in other fields: graph coloring (*e.g.*, [4,16,17]), telecommunications (*e.g.*, [6,7,21]), and the management of a fleet of vehicles (*e.g.*, [13]).

As mentioned in the introduction, the performance of a metaheuristic can be evaluated according to several criteria, such as quality, speed, robustness, ease of adaptation, and ability to take advantage of the problem structure. Relying on the guidelines given in [26], the following discussion aims at positioning CNS according to these criteria.

First of all, the design of CNS is not cumbersome, which means that CNS ranks highly according to the ease of adaptation criterion. More precisely, to adapt CNS to a problem, the following strategic decisions have to be made: (1) choose the nature of the local search (*e.g.*, descent algorithm, tabu search, simulated annealing); (2) determine the set A of active constraints (A is usually well adapted for the constraints involving a few number of variables, like binary or ternary constraints); (3) find a way to model a solution of the problem; (4) determine the deleting rules associated with the repairing process; (5) determine the external ingredients which can be efficiently combined with CNS (*e.g.*, intensification and diversification procedures).

The presented CNS deal with various ways of representing a solution, making it able to incorporate the properties of the problem. The component i of solution s, denoted s_i, can for example involve only one information (*e.g.*, a binary decision value associated with a selection of a photograph or not), or several data of different nature (*e.g.*, a binary decision value for the selection of a photograph and the associated camera). This also allows to better manage the repairing phase.

The generation of a single solution in the selected solution space is not time consuming. This is due on the one hand to the use of efficient incremental computations (*i.e.* the ability to evaluate a neighbor solution from the current solution, its value and the considered move). Incremental computation strongly contributes to speed. On the other hand, quick repairing procedures are performed, which are able to remove at least a drawback of the current solution when generating the neighbor solution (even if several new drawbacks are created in the latter). This specifically contributes to robustness (it also helps to escape from local optima) and to the consideration of the properties of the problem. Incremental computation and quick repairing procedures make the CNS metaheuristic very aggressive.

The use of deleting rules helps in avoiding plateaus of the solution space. A *plateau* occurs in solution space S associated with neighborhood structure N and objective function f if there are several neighbor solutions with the same value. The deleting rules have obviously the ability to take advantage of the problem structure and to guide the search in S.

To conclude, we would like to mention that CNS is a very flexible method for at least four reasons.

- CNS can consider various types of constraints. It is well adapted for constraints linking two or three variables together, because the repairing phase is usually straightforward in such situations. Such constraints usually belong to the set A of active constraints. In contrast, if a specific constraint involves several variables, it can be relaxed (at least to save CPU time) and put in the B set, as it was the case for the capacity constraint associated with SAT1.

- Flexible moves are always used in the existing CNS adaptations. More precisely, from any solution s, we conjecture that it is possible to reach any other solution s' of the solution space by performing a sequence of moves. This favors connectivity in the neighborhood graph and specifically contributes to robustness.
- Various ingredients can be easily added to the CNS framework to enhance its performance, such as intensification procedures (contributing to efficiency), diversification procedures (contributing to robustness), dynamic tabu tenures (helping in both efficiency and robustness).
- CNS can be hybridized with evolutionary metaheuristics, like genetic or adaptive memory algorithms. Such combinations were already successfully performed for graph coloring [16] and the SAT2 satellite range scheduling problem [27].

References

[1] L. Barbulescu, J.-P. Watson, L.D. Whitley and A.E. Howe, Scheduling space-ground communications for the air force satellite control network. *J. Sched.* **7** (2004) 7–34.
[2] R.S. Barr, B.L. Golden, J.P. Kelly, M.G.C. Resende and W.R. Stewart, Designing and reporting on computational experiments with heuristic methods. *J. Heuristics* **1** (1995) 9–32.
[3] E. Bensana, M. Lemaitre and G. Verfaillie, Earth observation satellite management. *Constraints* **4** (1999) 293–299.
[4] I. Bloechliger and N. Zufferey, A graph coloring heuristic using partial solutions and a reactive tabu scheme. *Comput. Oper. Res.* **35** (2008) 960–975.
[5] J.-F. Cordeau, G. Laporte and A. Mercier, A unified tabu search heuristic for vehicle routing problems with time windows. *J. Oper. Res. Soc.* **52** (2001) 928–936.
[6] A. Dupont, E. Alverhne and M. Vasquez, Efficient filtering and tabu search on a consistent neighbourhood for the frequency assignment problem with polarisation. *Ann. Oper. Res.* **130** (2004) 179–198.
[7] A. Dupont, A. Carneiro–Linhares, Ch. Artigues, D. Feillet, Ph. Michelon and M. Vasquez, The dynamic frequency assignment problem. *Eur. J. Oper. Res.* **195** (2009) 75–88.
[8] M. Garey and D.S. Johnson, *Computer and intractability: a guide to the theory of NP-completeness*. Freeman, San Francisco (1979).
[9] M. Gendreau and J.-Y. Potvin, *Handbook of metaheuristics*, International Series in Operations Research & Management Science, vol. 146. Springer (2010).
[10] A. Globus, J. Crawford, J. Lohn and A. Pryor, A comparison of techniques for scheduling earth observing satellites. In *Proceedings of the Sixteenth Innovative Applications of Artificial Intelligence Conference (IAAI-04)*, San Jose (2004).
[11] F. Glover and M. Laguna, *Tabu search*. Kluwer Academic Publishers, Boston (1997).
[12] D. Habet, M. Vasquez and Y. Vimont, Bounding the optimum for the problem of scheduling the photographs of an agile Earth observing satellite. *Comput. Opt. Appl.* **47** (2010) 307–333.
[13] A. Hertz, D. Schindl and N. Zufferey, A solution method for a car fleet management problem with maintenance constraints. *J. Heuristics* **15** (2009) 425–450.
[14] D.S. Johnson, A theoretician's guide to the experimental analysis of algorithms. *DIMACS Ser. Discr. Math. Theor. Comput. Sci.* **59** (2002) 215–259.
[15] S. Lin, Computer solutions of the traveling salesman problem. *Bell System Technical Journal* **44** (1965) 2245–2269.

[16] E. Malaguti, M. Monaci and P. Toth, A metaheuristic approach for the vertex coloring problem. *INFORMS J. Comput.* **20** (2008) 302–316.
[17] C. Morgenstern, Distributed coloration neighborhood search. *DIMACS Series in Discrete Math. Theor. Comput. Sci.* **26** (1996) 335–357.
[18] D.A. Parish, *A genetic algorithm approach to automating satellite range scheduling.* Master's thesis, Air Force Institute of Technology, USA (1994).
[19] G.M. Ribeiro, M.F. Constantino and L.A.N. Lorena, Strong formulation for the spot 5 daily photograph scheduling problem. *J. Combin. Opt.* **20** (2010) 385–398.
[20] Y. Rochat and E. Taillard, Probabilistic diversification and intensification in local search for vehicle routing. *J. Heuristics* **1** (1995) 147–167.
[21] M. Vasquez and J.-K. Hao, A heuristic approach for antenna positioning in cellular networks. *J. Heuristics* **7** (2001) 443–472.
[22] M. Vasquez and J.-K. Hao, A logic-constrained knapsack formulation and a tabu algorithm for the daily photograph scheduling of an earth observation satellite. *Comput. Opt. Appl.* **20** (2001) 137–157.
[23] M. Vasquez and N. Zufferey, Consistent neighborhood search for combinatorial optimization. *ISRN Comput. Math.* (2012) 671423.
[24] M. Vasquez and N. Zufferey, Consistent neighborhood search for constrained assignment problems. In *Proceedings of the 9th International Conference on Modeling, Opt. Simul. (MOSIM 2012)*, Bordeaux, France (2012).
[25] G. Verfaillie, M. Lemaitre and T. Schiex, Russian doll search for solving constraint optimization problems. In *13th National Conference on Artificial Intelligence (AAAI-96)*, Portland, USA (1996) 181–187.
[26] N. Zufferey, Metaheuristics: some Principles for an efficient design. *Comput. Technol. Appl.* **3** (2012) 446–462.
[27] N. Zufferey, P. Amstutz and P. Giaccari, Graph colouring approaches for a satellite range scheduling problem. *J. Sched.* **11** (2008) 263–277.