Consistent Neighborhood Search for Combinatorial Optimization.

Michel Vasquez, Nicolas Zufferey

International Scholarly Research Network
ISRN Computational Mathematics
Volume 2012, Article ID 671423, 12 pages
doi:10.5402/2012/671423

Research Article

Consistent Neighborhood Search for Combinatorial Optimization

Michel Vasquez[1] and Nicolas Zufferey[2]

[1] *École des Mines d'Alès, LGI2P Research Center, Parc Scientifique Georges Besse, 30035 Nimes Cedex 01, France*
[2] *Faculty of Economics and Social Sciences, HEC-University of Geneva, Uni-Mail, 1211 Geneva 4, Switzerland*

Correspondence should be addressed to Nicolas Zufferey, nicolas.zufferey-hec@unige.ch

Received 11 May 2012; Accepted 28 June 2012

Academic Editors: D. S. Corti, R. K. Upadhyay, and E. Weber

Many optimization problems (from academia or industry) 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 formally propose a new heuristic solution method, CNS, which has a search behavior between exhaustive tree search and local search working with complete solutions. We then discuss, with a unified view, the great success of some existing heuristics, which can however be considered within the CNS framework, in various fields: graph coloring, frequency assignment in telecommunication networks, vehicle fleet management with maintenance constraints, and satellite range scheduling. Moreover, some lessons are given in order to have guidelines for the adaptation of CNS to other problems.

1. Introduction

An *exact method* (e.g., branch-and-bound, dynamic programming, Lagrangian relaxation-based methods) 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 [1]. For these difficult problems, one should prefer to quickly find a satisfying solution, which is the goal of *heuristic* solution methods. There mainly exist three families of heuristics: constructive heuristics (a solution is built step by step from scratch, like the greedy algorithm), local search heuristics (a solution is iteratively modified: this will be discussed below), and evolutionary heuristics (a population of solutions is managed, like genetic algorithms and ant algorithms). In this paper, only the context of local search methods will be considered.

A *local search* heuristic 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, that is, the definition of the *neighborhood* structure, depends on the considered problem. The most popular local search methods are simulated annealing [2], tabu search [3], threshold algorithms [4], variable neighborhood search [5], and guided local search [6].

Within a local search context, the usual approach consists in working with complete solutions, that is, 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), 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 nonassigned 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, which is an extension of [7] and [8], we formally introduce 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 heuristics, which can however be considered as belonging to the CNS methodology, for various NP-hard constrained combinatorial problems. For each problem, the reader is referred to the associated paper to have references on the NP-hard aspect, the literature review, and the detailed experimental conditions (computer, language, etc.). For each problem, comparisons have to be done carefully because the conditions of experimentation were not always the same. Remember however that a heuristic is generally performed until the potential to improve the best encountered solution becomes poor. In addition, 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 paper is organized as follows. In Section 2, the CNS methodology is proposed. Then, heuristics for various problems are presented within a CNS framework: graph coloring (Section 3), frequency assignment with polarization (Section 4), car fleet management with maintenance constraint (Section 5), and satellite range scheduling (Section 6). We end up the paper with a conclusion.

2. Consistent Neighborhood Search

In this section, we introduce the CNS methodology and situate it within the optimization methods.

2.1. Presentation of the Method. 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. 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 $s_i \in D_i$. Solution s is *feasible* if it satisfies all the constraints in C. In most local search methods, the search space contains *complete* solutions; that is, 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 *partial* and feasible solutions, that is, where some s_i's do not have a value but all the constraints are satisfied. In such a case, the goal is to minimize the number $\hat{f}(s)$ of nonassigned variables in s, and the process stops of course if $\hat{f}(s) = 0$.

Therefore, three search spaces are possible: (1) the complete and feasible search space $S^{(\text{feasible})}$, (2) the complete and unecessarily feasible search space $S^{(\text{penalty})}$, where unfeasible solutions are penalized, and (3) the partial and 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: define a move which does not augment too much $p(s)$, tune the above-mentioned parameter α, and 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 $s' \in N(s)$.

In contrast, any move m is performed in at least two phases in CNS.

(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 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 a new admissible value to it without creating new conflicts, do it.

(3) *Repairing phase*: in order to keep the partial solution feasible, remove the value of all the variables of $C(m)$.

Therefore, the distance between s and a neighbor solution in $N(s)$ is usually not a constant.

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 nonassigned 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 $S^{(\text{penalty})}$).

We have now all the ingredients to formulate a pseudocode of CNS in Algorithm 1.

In summary, CNS is an approach dealing with partial 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.

Initialization: generate an initial solution s, set $s^* = s$ and $\widehat{f}^* = \widehat{f}(s)$;
While a stopping time condition is not met and $\widehat{f}^* > 0$**, 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 another admissible value to s_r without augmenting the number of violations, do it and remove s_r from $C(m)$;
   (c) let s^{cand} be the so obtained candidate neighbor solution (which might be non feasible at this stage);
   (d) update the best candidate move: if $|C(m)| < g$, set $s' = s^{\text{cand}}$ 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: if $\widehat{f}(s) < \widehat{f}^*$, set $s^* = s$ and $\widehat{f}^* = \widehat{f}(s)$

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

Algorithm 1: CNS.

The adaptation of tabu search within the framework of CNS is now discussed. A generic and standard version of tabu search can be described as follows, assuming that f 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 $f(s') < f(s^*)$, where s^* 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 [9].

Tabu search adapted within the framework of CNS has the following specificities: working in $S^{(\text{feasible})}$, minimizing $\widehat{f}$ 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, and it is then tabu to remove the value of s_i for a certain number of iterations.

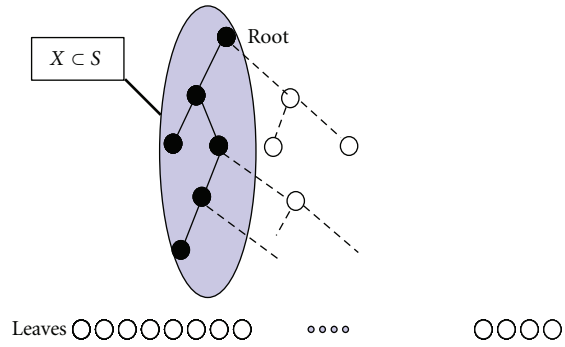

Figure 1: X for depth-first search.

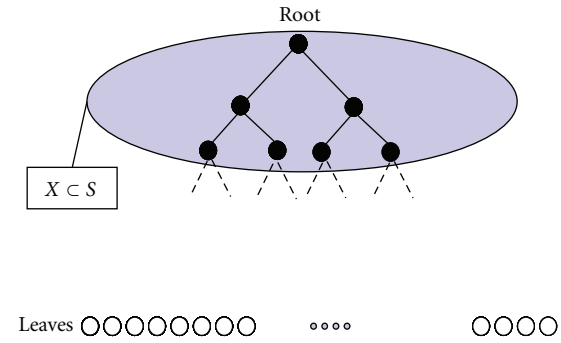

Figure 2: X for breadth-first search.

2.2. Search Characteristics of CNS. We now compare the general strategy of three kinds of optimization methods: tree search, standard local search, and CNS. These methods have a very different way to visit the search tree, where the *root* (the top node in Figures 1 to 4) is the empty solution (no variable is assigned), and the *leaves* (the bottom nodes in Figures 1 to 4) are complete solutions (all the variables have a value), which can be feasible or not. In such four figures, an empty node is not visited by the considered method; on the contrary a black node is visited, and a black arrow indicates a performed move from one node to another. Let S be the set of all the possible nodes in the search tree, and let X be subset of S which is mainly visited by a specific algorithm. We will see that X differs drastically from one method to the other.

Tree search algorithms visit neighbor nodes in the search tree. The visited subtree X is likely to be vertical for Depth-First Search as illustrated in Figure 1, where only a few leaves might be visited. In contrast, Breadth-First Search usually focuses on the top of the search tree because it often needs a prohibitive amount of time to go down, as illustrated in Figure 2 where no leaves are visited.

A very different strategy characterizes standard local search methods: as illustrated in Figure 3, only leaves are

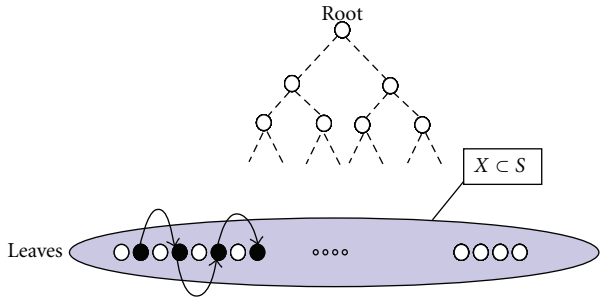

Figure 3: X for a standard local search.

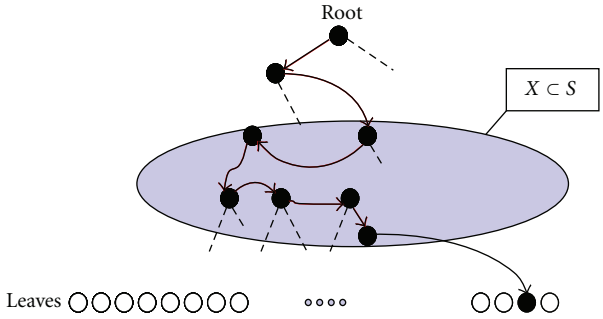

Figure 4: X for CNS.

visited, which is a major advantage when compared to tree search methods. However, standard local search algorithms usually have the following drawbacks. On the one hand, if constraint violation is forbidden, the search space X is not necessarily connected; that is, it is not always possible to join two leaves with a sequence of moves. In such a case, the search might be trapped in a connected component of S which does not contain good solutions. On the other hand, if constraint violation is allowed but penalized during the search, the number of leaves is drastically augmented, and the search might mainly focus on nonfeasible leaves, as it can be challenging to define the penalty function p and to tune its associated parameter α.

Even if CNS can be considered as a local search method, it mainly explores nodes which are close to the leaves, as illustrated in Figure 4. Because all the leaves correspond to complete and feasible solutions, CNS stops as soon as a leaf is reached.

CNS can start its search from the root, that is, no variable has a value. In such a case, its first iterations would basically consist in greedily assigning a value to a variable until the current solution becomes *saturated*, that is, when the repairing phase becomes unavoidable. CNS can also start its search from a node located below the root if an external procedure is used to generate the initial solution of CNS. The more efficient is such external procedure, the closer will be the first explored node to the bottom of S. Last but not least, CNS can perform jumps in S. Thus, the search space X is likely to be connected.

Therefore, CNS does not encounter the above-described drawbacks associated with tree search and standard local search methods. Notice however that there exists an implicit enumeration method able to perform jumps over S, called *Resolution Search* and proposed in [10].

3. Graph Coloring

The main reference associated with this section is [11]. The authors proposed a tabu search heuristic for the graph coloring problem, which we denote CNS-GCP for the sake of simplicity.

3.1. Description of the Problem. Given a graph $G = (V, E)$ with vertex set V and edge set E, the k-coloring problem (k-GCP) consists in assigning an integer (called *color*) in $\{1, \ldots, k\}$ to every vertex such that two adjacent vertices have different colors. The Graph Coloring Problem (GCP) consists in finding a k-coloring with the smallest possible value of k (called the chromatic number and denoted χ). Both problems are NP-hard [1], and many heuristics were proposed to solve them. For a recent survey, the reader is referred to [12]. Starting at most with $k = |V|$, an upper bound on the *chromatic* number of G can be determined by solving a series of k-GCPs with decreasing values of k until no feasible k-coloring can be obtained. Only such a strategy, which leads to the best results, will be considered below.

3.2. Description of the Method within a CNS Framework. The best k-coloring heuristics are based on two approaches. In $S^{(\text{penalty})}$, the constraint that the endpoints of an edge should have different colors is relaxed. Thus, the strategy consists in allowing conflicts (a *conflict* occurs if two adjacent vertices have the same color) while minimizing the number of conflicts. In a local search context, a straightforward move is thus to change the color of a conflicting vertex, as proposed in [13].

In contrast, in $S^{(\text{partial})}$, the constraint imposing that all vertices should be colored is relaxed, but conflicts are forbidden. We have $D_i = \{1, \ldots, k\}$ for each s_i. In such a case, the value s_i of a solution $s = (s_1, \ldots, s_n)$ in $S^{(\text{partial})}$ indicates the color of vertex i, which is in the set $\{1, \ldots, k\}$, and there is no value (or an artificial value 0) if vertex i is not colored. The goal is to minimize the number of uncolored vertices. A move $m = (s_i; c)$ consists in first assigning a color c to a uncolored vertex i (assignment phase), and then (repairing phase) to remove the color of the created conflicting vertices (i.e., all the vertices adjacent to i which have color c). Then, all the moves which will remove the color c of vertex i are tabu for a certain number of iterations. This number is dynamically managed and is proportional to the variation of the objective function $\widehat{f}(s) = |s| = |\{s_i | s_i > 0\}|$. At each iteration, the best nontabu move is performed (ties are randomly broken).

3.3. Numerical Comparison with Other Methods. It is shown in [14] and in [11] that the most difficult benchmark instances from the DIMACS Challenge (see ftp://dimacs.rutgers.edu/pub/challenge/graph/) are the ones presented in Table 1. Below, CNS-GCP is compared with

Table 1: Comparisons between CNS-GCP and state-of-the-art coloring algorithms.

Graph	n	χ, k^*	CNS-GCP	Tabucol	MMT	GH	MOR
DSJC1000.1	1000	?, 20	21	20	20	20	21
DSJC1000.5	1000	?, 83	89	89	83	83	88
DSJC1000.9	1000	?, 224	226	227	226	224	226
DSJC500.1	500	?, 12	12	12	12	12	12
DSJC500.5	500	?, 48	49	49	48	48	49
DSJC500.9	500	?, 126	127	127	127	126	128
DSJR500.1c	500	?, 85	85	85	85	—	85
DSJR500.5	500	?, 122	126	126	122	—	123
flat1000_50_0	1000	50, 50	50	50	50	50	50
flat1000_60_0	1000	60, 60	60	60	60	60	60
flat1000_76_0	1000	76, 82	88	88	82	83	89
flat300_28_0	300	28, 28	28	31	31	31	31
le450_15c	450	15, 15	15	16	15	15	15
le450_15d	450	15, 15	15	15	15	15	15
le450_25c	450	25, 25	27	26	25	26	25
le450_25d	450	25, 25	27	26	25	26	25

other state-of-the-art coloring heuristics, which are Tabucol [13], GH [15], MOR [16], and MMT [17]. Tabucol is a standard tabu search working in $S^{(\text{penalty})}$. GH, MOR, and MMT are all population-based methods which use local search procedures. GH uses Tabucol to improve offspring solutions, whereas MMT uses a procedure close to CNS-GCP. MOR works in the same search space as CNS-GCP but uses simulated annealing instead of tabu search, and much more sophisticated moves.

A CPU time limit of 60 minutes on a Pentium 4 (2 GHZ, 512 MB of RAM) was considered for CNS-GCP. The first two columns of Table 1, respectively, indicate the name and the number n of vertices of the graph. The third column contains two numbers, the first one being the chromatic number (a "?" is put when it is not known), and the second one the best upper bound $k^\star$ ever found by a heuristic. Then, for every algorithm, the smallest k such that a feasible k-coloring was found is reported. We can observe that CNS-GCP is rather competitive with the best coloring methods. However, it is much simpler!

4. Frequency Assignment with Polarization

The main reference associated with this section is [18], where the frequency assignment problem with polarization (FAPP) was considered. The authors proposed that a tabu search approach working in $S^{(\text{partial})}$ denoted CNS-FAPP below.

4.1. Description of the Problem. The FAPP concerns a Hertzian telecommunication network made up of antennae located at a set of geographical sites. A Hertzian liaison joins two sites by one or more paths. Hence, a *path* is a unidirectional radioelectric bond, established between antennae at distinct sites, which has a given frequency and polarization. Let F_i and P_i, respectively, be the allowed frequency set and polarization set for path i, where $P_i \in \{\{-1\},\{1\},\{-1,1\}\}$. The FAPP consists in finding, for each path, a frequency and a polarization satisfying the following set of constraints.

Let IC be the set of the imperative constraints, which are of four types: $p_i = p_j$, $p_i \neq p_j$, $|f_i - f_j| = \varepsilon_{ij}$, and $|f_i - f_j| \neq \varepsilon_{ij}$, where $\varepsilon_{ij} \geq 0$. In addition, some electromagnetic compatibility constraints ($ECCs$) require a minimal distance between frequencies of two paths: $|f_i - f_j| \geq \gamma_{ij}$ if $p_i = p_j$ and $|f_i - f_j| \geq \delta_{ij}$ if $p_i \neq p_j$. This constraint controls the interference phenomenon, which is why the required distance between frequencies depends on their polarizations; it is smaller if the polarizations are different (i.e., $\delta_{ij} \leq \gamma_{ij}$). Unfortunately, most problems do not have feasible solutions because the domains are too restrictive or the requirements too numerous. Consequently, some deterioration is allowed by permitting some interference, which have to be minimized. With this aim, for the ECC constraints, a progressive relaxation is authorized and expressed by relaxation levels; level 0 corresponds to no relaxation, and going from level k to level $k+1$ involves the relaxation of some or all the frequency gaps, the maximum relaxation level being 10. Formally, we have:

$$\left|f_i - f_j\right| \geq \begin{cases} \gamma_{ij}^0 \geq \cdots \geq \gamma_{ij}^k \geq \cdots \geq \gamma_{ij}^{10} & \text{if } p_i = p_j \\ \delta_{ij}^0 \geq \cdots \geq \delta_{ij}^k \geq \cdots \geq \delta_{ij}^{10} & \text{if } p_i \neq p_j \end{cases} \tag{1}$$

since in the 11th level, $\gamma_{ij}^{11} = \delta_{ij}^{11} = 0$, so there is no ECC.

Let ECC_k be the set of ECC constraints at level k (for $0 \leq k \leq 10$). This means that each constraint belonging to ECC_k is affected to its γ_{ij}^k and δ_{ij}^k gaps. More precisely, $|f_i - f_j| \geq (|p_i + p_j|/2)\gamma_{ij}^{(k)} + (|p_i - p_j|/2)\delta_{ij}^{(k)}$. Accordingly, a feasible solution at level k is an assignment of all the paths satisfying all the strong constraints IC and all the ECC_k constraints. If such a solution exists, the problem is said to be k-feasible. Every problem is assumed to be 11-feasible.

Consequently, the objective function of the problem is, in order of priority to (1) minimize the lowest relaxation level k for which a k-feasible solution exists (2) minimize $V^{(k-1)}$: the number of constraints of $ECC_{(k-1)}$ violated at level $k-1$, and (3) minimize $\sum_{0\leq i<k-1} V^{(i)}$: the sum of the constraints of ECC_i violated at all levels i less than $k-1$.

4.2. Description of the Method within a CNS Framework. The strategy adopted for the resolution consists in transforming the FAPP optimization problem into 11 decision problems according to the relaxation level on the *ECC*; each FAPP(k) contains both the IC and the ECC_k constraints. This enables us to introduce some filtering treatments to reduce the frequency and the polarization domains. Starting from level $k = 11$ where an initial solution is provided by a greedy constructive method, the general algorithm works in a downward fashion: each time a k-feasible solution is found, a lower level is considered.

A solution $s = (s_1, \ldots, s_n)$ indicates for each path i its associated resource (f_i, p_i), where $f_i \in F_i$ and $p_i \in P_i$. Thus, $D_i = F_i \times P_i$. In the assignment phase, a pair (f_i, p_i) is given to the chosen nonassigned candidate path i. Then, in the repairing phase, this affectation is propagated to its neighbors in the constraint network, and, if necessary, the conflicting neighboring values are deleted in order to satisfy the IC and ECC_k constraints. This was done efficiently using incremental computing on specific data structures, allowing variable domains to be dynamically reduced.

A tabu list is needed to prevent cycling, which occurs when there is an attempt to instantiate the last deleted variables in the current partial solution. Indeed, all the values (f_j, p_j) likely to delete the variable $s_i = (f_i, p_j)$ affected by the move are classified tabu during some iterations; the tabu tenure is proportional to the number of times this resource was affected.

The considered problem was the subject of the Challenge ROADEF 2001 (organized by the French Society of Operations Research and Decision Analysis), involving 27 research teams (see http://uma.ensta-paristech.fr/conf/roadef-2001-challenge/). In Table 2 are presented the results obtained by the five best teams. During the competition, only one run was allowed, and the computing time was limited to one hour on a Pentium 3 (500 MHZ, 128 MB of RAM). Table 2 details the hierarchical objective function, by giving first the relaxation level k, then the sum of all the unsatisfied ECC_{k-1}, and finally the sum of all the unsatisfied ECC_i, where i varies from 0 to $k-2$. The first column indicates the instance names with the instance number and the number n of considered paths. For example, 02-0250 is instance 2 with 250 paths.

The first approach, developed by Bisaillon's team and referred to as TS-VN, is a local search based on tabu search with a variable neighborhood. The algorithm *H+CP*, developed by Caseau, combines constraint propagation with heuristics such as *Large Neighborhood Search* and *Limited Discrepancy Search*. The third method, *LS-CC*, developed by Gavranovic, is a typical local search guided by the constraint cost; at each level, it builds frequency trees, ignoring the polarization constraints, and then it tries to optimize the polarization allocation. In a similar way, classical tabu search procedures (simply denoted Tabu) working with complete solutions were implemented by Schindl's team. Finally, the last column gives the results obtained by CNS-FAPP.

We can observe the efficiency of CNS-FAPP when compared to the other methods. Care is needed because the indicated values are the best among 10. CNS-FAPP finds the optimal k level for 37 instances out of 40. And last but not least, CNS-FAPP was the winner of the Challenge!

5. Fleet Management with Maintenance Constraints

The main reference associated with this section is [19], where a rather complex solution method was proposed for a problem which can be formulated as a car fleet management problem with maintenance constraints (but denoted CAR for the sake of simplicity). The particularity of the problem is that feasible solutions are very easy to find, but can cost a lot. Thus, $S^{(\text{partial})}$ was designed to avoid to assign the most expensive value to each variable.

5.1. Description of the Problem. The problem retained for the Challenge ROADEF 1999 was an inventory management problem (see http://www.roadef.org/content/roadef/challenge.htm for the details), where a cost function has to be minimized. A car rental company manages a stock of cars of different types. It receives requests from customers asking for cars of specific types for a given time horizon. Basically a *request* is characterized by its start and end times, by a required car type, and by the number of required cars. All requests are supposed to be known for the considered time horizon. The satisfaction of all customer requests is mandatory. If there are not enough cars available in stock, the company can react in three different ways: (1) *upgrading*: it can offer a better car type to the customer (but the company encounters the additional associated cost); (2) *subcontracting*: the company can decide to subcontract some requests to other providers, which is generally the most expensive alternative; (3) *purchasing* new cars, which then belong to the stock of the company for the rest of the time horizon.

Two types of maintenance constraints make the problem difficult: (1) a maximum time of use without maintenance is given for each car type (each maintenance has a duration, a cost and a number of workers needed to perform it); (2) the company has a fixed number of maintenance workers, which means that the maintenances should be scheduled so that the capacity of the workshop is never exceeded. In addition, the following costs are also known: the costs (fixed and time dependent) associated with the assignment of a car to a request, and the inventory cost per day of a car in stock (rented or not). The goal is to satisfy all the requests while minimizing the total cost.

5.2. Description of the Method within a CNS Framework. The general pseudocode of the method, denoted CNS-CAR,

Table 2: Comparison of CNS-FAPP with others methods.

FAPP	TS-VN			H+CP			LS-CC			Tabu			CNS-FAPP		
	k	$V1$	$SV2$	k	$V1$	$SV2$	k	$V1$	$SV2$	k	$V1$	$SV2$	k	$V1$	$SV2$
01-0200	4	4	56	4	6	279	4	14	165	5	1	281	4	14	233
02-0250	2	7	86	2	18	248	2	21	160	11	1	1274	2	20	195
03-0300	7	10	341	7	27	1076	7	16	420	7	13	589	7	32	892
04-0300	1	31	0	1	164	0	3	9	224	7	1	3678	1	184	0
05-0350	11	1	372	11	892	12364	11	1	1467	11	7	2284	11	364	5694
06-0500	5	12	246	5	53	1029	7	15	879	7	15	1210	5	31	811
07-0600	9	22	714	9	132	4419	10	28	3070	9	33	1585	9	106	3375
08-0700	5	16	266	5	53	1359	5	37	691	5	26	625	5	73	1225
09-0800	3	28	195	3	63	937	4	24	573	10	1	3678	3	104	846
10-0900	6	18	475	6	82	2365	6	39	1146	8	5	2871	6	103	2003
11-1000	8	8	1015	8	119	5206	9	30	3736	10	1	5108	8	119	4191
12-1500	3	83	1698	7	180	6538	11	17	2634	9	70	7682	2	62	1310
13-2000	3	49	2003	7	229	7503	11	59	6164	10	13	9651	5	132	3645
14-2500	4	35	3485	8	18	10661	11	3	5574	10	101	15718	5	217	5045
15-3000	5	15	1569	7	333	9988	11	46	9523	10	61	14010	5	192	4727
16-0260	11	5	56	11	572	5779	11	67	913	11	5	57	11	514	5189
17-0300	4	4	34	4	4	36	4	4	35	4	4	34	4	4	36
18-0350	8	4	55	8	4	55	8	4	57	8	4	55	8	4	59
19-0350	6	2	51	6	3	79	6	2	53	6	2	51	6	3	70
20-0420	10	5	97	10	6	145	10	5	99	10	5	97	10	7	142
21-0500	4	2	10	4	2	12	4	2	11	4	2	10	4	2	12
22-1750	7	15	187	7	16	356	7	16	194	7	15	187	7	25	503
23-1800	9	16	187	9	17	197	9	16	189	9	16	187	9	17	197
24-2000	7	6	71	7	7	90	7	7	79	7	6	71	7	9	91
25-2230	3	7	32	3	7	33	3	7	33	3	7	32	3	7	33
26-2300	7	9	74	7	10	81	7	9	74	7	9	74	7	10	86
27-2550	11	4	64	5	7	46	5	8	37	5	4	20	5	11	54
28-2800	3	13	32	3	32	129	3	25	58	3	13	32	3	42	142
29-2900	6	25	239	6	28	351	6	25	212	6	25	212	6	25	310
30-3000	11	1166	12029	7	17	602	7	16	190	7	13	148	7	48	1045
31-0400	5	4	1180	5	161	2131	5	34	1151	5	16	1400	5	117	1896
32-0550	10	52	1739	6	16	388	6	5	71	11	25	2166	6	10	235
33-0650	5	7	66	5	16	332	5	7	77	11	5	1310	5	10	235
34-0750	4	2	46	4	35	767	4	6	213	10	1	1701	4	22	565
35-1500	7	3	1280	6	74	1919	6	16	431	11	24	5870	6	62	1375
36-2000	7	99	2153	9	3	2478	8	25	970	11	16	4652	7	63	1643
37-2250	11	3	12229	5	56	1745	8	13	975	11	14	10353	5	51	1288
38-2500	11	79	14058	3	39	572	3	14	174	11	53	13355	9	125	6717
39-2750	3	356	2844	3	2567	10470	3	747	4603	11	36	13267	11	3947	40473
40-3000	11	39	16755	4	77	1562	8	20	1261	11	867	13684	4	64	1252

is summarized in Algorithm 2. First, an initial solution is greedily generated. Step 1 of the main loop tries to improve the current solution without changing the set of purchased cars (with the use of two tabu search procedures working in $S^{(\text{partial})}$, denoted TS1-CAR and TS2-CAR below), while the second step generates a new solution with a different set of purchased cars. The stopping criterion is a time limit of one hour, as imposed by the organizers of the Challenge.

In TS1-CAR, a solution s can be modeled as follows. Let $s_r = t$ if request r is performed by a car of type t of the

Initialization: generate an initial solution s;
While the time limit is not reached, do
1. try to improve s without changing the set of purchased cars, with the successive use of TS1-CAR and TS2-CAR;
2. update s by purchasing a car or by removing a previously purchased car (the requests associated with a removed car are initially subcontracted).

Algorithm 2: Algorithm CNS-CAR.

fleet (purchased or not), and s_r has no value (or an artificial value, say 0) if request r is subcontracted to another provider. Thus, $S^{(\text{partial})}$ is defined in order to minimize the number of subcontracted requests. A neighbor s' of a solution s is obtained by assigning a car c of type t to a subcontracted request r (i.e., the corresponding s_r equals t instead of 0). To make such a change feasible, in the repairing phase, requests covered by c that overlap with r are subcontracted (i.e., the associated s_j values are set to 0), and the maintenances of car c are possibly rescheduled in a greedy fashion while satisfying the maintenance constraints. If it is not possible, other s_j's such that $s_j = t$ might be set to 0 in order to create more room to schedule the maintenances. If it is still not possible to generate a feasible schedule for the maintenances (because of the maintenances schedule of the other car types), such a move is not considered further.

TS2-CAR is an extension of TS1-CAR in the following sense: (1) it works on several car types during the same move; (2) it tries to reduce the total cost not only by assigning cars to subcontracted requests, but also by avoiding upgrades; (3) a reassignment phase is performed; (4) the repairing phase has more options to validate the move proposed in the assignment phase. A neighbor s' of a solution s is obtained by assigning a car of type t to a request r, where r is subcontracted or covered by a car of type $t' \neq t$ in s, where type t' is an upgrade of type t. In other words, s_r equals t instead of 0 or t'. The reassignment and repairing phases are performed simultaneously as follows: all the requests C_t covered by the cars of type t might be reassigned within car type t (while considering an exact method for a specific case of the graph coloring problem), and it is allowed to subcontract some requests of C_t. In such a phase, the maintenance schedule of all the cars might change (in a greedy fashion or by the use of an exact method). In the two tabu procedures, when a request r is assigned to a car type t (i.e., s_r is set equal to t), it is then tabu to remove the value t from s_r for a certain number of iterations.

Diversification procedures were also used, based on the following idea: the requests which were not subcontracted from a large number of iterations are subcontracted, in order to make room for other requests in the schedule.

5.3. Numerical Comparison with Other Methods. In Tables 3 and 4 the results for the 16 benchmark instances of the Challenge are reported. CNS-CAR is compared with the four best methods (among the thirteen proposed heuristics) of the Challenge. The winners of the contest were Briant and Bouzgarrou. Their algorithm mainly combines linear programming ignoring the maintenance constraints and then adjust the solution according to the maintenance constraints. The name of an instance is coded with a vector (x, y, z, w), where x is the number of requests, y is the number of car types, z is the capacity of the workshop, and w is equal to b if purchases are allowed, and to nb otherwise. The time horizon of all instances is $[0, 730]$ corresponding to a period of 2 years.

The algorithm was run with the time limit equivalent to one hour on a Pentium Pro (200 MHZ, 64 MB of RAM), as imposed by the organizers of the challenge. The results are shown in Table 3 (for instances where purchases are forbidden) and Table 4 (associated with the same instances, but where purchases are allowed). The column labeled *Best* contains the best known solution for each instance. An asterisk is put when CNS-CAR was able to equal or improve the previous best known solution. Some of these best results have been obtained when using different parameters from those mentioned above (for tuning purposes) or by running CNS-CAR for more than one hour. The next four columns contain the percentage gap with respect to *Best* obtained by the four best methods, labeled with the initials of the members of each team, namely, BB (for Briant and Bouzgarrou), AGHKU (for Asdemir, Karslioğlu, Gürbüz, and Ünal), B (for Bayrak), and DD (for Dhaenens-Flipo and Durand). The next column contains the percentage gap with respect to *Best* obtained with CNS-CAR. For each instance, ten runs of CNS-CAR were executed, and average results are reported. The last line of each column indicates average results. We can observe that CNS-CAR gives in average better results than those obtained by the four best competitors of the Challenge.

6. Satellite Range Scheduling

The main reference associated with this section is [20], in which the problem is referred to as the daily photograph scheduling of an earth observation satellite (but only denoted SAT below). The authors proposed a tabu search approach working in $S^{(\text{partial})}$ denoted CNS-SAT below. Note that CNS kind of approaches were also very successfully adapted to other satellite range scheduling problems: the multi-resource satellite range scheduling problem [21] where more than one resource are available, and a satellite range scheduling with partial acquisition and transition times [22].

6.1. Description of the Problem. The considered satellite range scheduling problem can be described as follows [23].

Table 3: Results for the instances without purchase.

Instance	Best	BB	AGHKU	B	DD	CNS-CAR
(80, 8, 2, *nb*)	1162285*	0.00%	0.05%	1.31%	8.47%	0.00%
(150, 7, 2, *nb*)	3280230*	0.87%	5.85%	5.03%	9.73%	0.00%
(160, 12, 2, *nb*)	3333599*	14.63%	19.68%	29.56%	20.51%	0.81%
(200, 12, 2, *nb*)	5450785*	7.77%	22.56%	25.52%	26.02%	2.59%
(200, 7, 2, *nb*)	5156915*	6.36%	12.61%	21.02%	31.93%	3.62%
(200, 7, 4, *nb*)	4558728*	0.00%	0.66%	3.26%	14.45%	0.00%
(210, 9, 2, *nb*)	5810288*	5.67%	10.76%	18.48%	27.11%	2.42%
(210, 9, 4, *nb*)	5135237*	1.82%	1.44%	3.46%	13.09%	0.12%
Average	4236008	4.64%	9.20%	13.46%	18.91%	1.20%

Table 4: Results for the instances with purchase.

Instance	Best	BB	AGHKU	B	DD	CNS-CAR
(80, 8, 2, *b*)	1145181*	0.00%	1.55%	2.82%	7.23%	0.04%
(150, 7, 2, *b*)	2811138	0.00%	9.40%	3.98%	13.23%	0.01%
(160, 12, 2, *b*)	3064397*	11.99%	29.40%	26.08%	21.98%	1.47%
(200, 12, 2, *b*)	4517706*	12.42%	34.97%	35.93%	38.13%	4.88%
(200, 7, 2, *b*)	4990499*	6.88%	15.36%	27.76%	31.38%	4.98%
(200, 7, 4, *b*)	4092002*	0.00%	3.00%	3.46%	12.76%	0.01%
(210, 9, 2, *b*)	5380588*	7.38%	18.91%	29.31%	34.15%	3.52%
(210, 9, 4, *b*)	4147087*	8.71%	10.92%	9.19%	14.91%	0.01%
Average	3787302	5.92%	15.44%	17.32%	21.72%	1.86%

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 monophotograph 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 monophotograph $p_i \in P$ are associated three pairs of elements $(p_i, \text{camera\_1}), (p_i, \text{camera\_2})$, and $(p_i, \text{camera\_3})$. 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 stereophotographs 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 monophotographs and p_3 is a stereophotograph, 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.

The SAT is to find a subset P' of P which satisfies all the imperative constraints and maximizes the sum of the profits of the photographs in P'. 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, that is, (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, that is, $f(s) = \sum_{i=1}^{m} g_i \cdot s_i$.

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, that is, (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 $\sum_{i=1}^{m} c_i \cdot s_i \leq \text{Max\_capacity}$.

Binary constraints involving the nonoverlapping 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 and 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 and 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 as follow $s_i + s_j + s_l \leq 2$. 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, that is, C3 = C3_1 ∪ C3_2.

6.2. Description of the Method. In contrast with the previous problems where all the constraints are considered to define the search space, a partially constrained search space C is considered here, which is composed of all binary vectors of m elements satisfying constraints C2 and C3 above. The relaxation of the capacity constraint helps to obtain better results and to accelerate the search. Let $s = (s_1, s_2, \ldots, s_m) \in$ C and $s' = (s_{1'}, s_{2'}, \ldots, s_{m'})$, and then s' is a neighbor of s, that is, $s' \in N(s)$, 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, for all $(s_i, s_j) \in$ C2, $s_{j'} = 0$ for $1 \leq j \leq m$, and

(3) for the above i, for all $(s_i, s_j, s_k) \in$ 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 and camera) (i.e., flipping a variable s_i from 0 to 1) in the current schedule and then dropping some pairs (photo and camera) (i.e., flipping some s_j's from 1 to 0) to repair binary and ternary constraint violations.

During the search, the capacity constraint may be violated by the current solution $s = (s_1, s_2, \ldots, s_m)$; that is, 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 elements s_i which have the worst ratio g_i/c_i until the capacity constraint is satisfied.

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. Note that intensification and diversification procedures were also used to enhance the efficiency of the general method.

6.3. Numerical Comparison with Other Methods. Experiments are carried out on a set of 20 realistic instances provided by the CNES (French National Space Agency) and described in details in [23]. 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-SAT algorithm. The other six instances have from 488 to 1057 candidate photographs, giving up to 2355 binary variables and 35933 constraints. Existing exact algorithms are unable to solve optimally these instances.

The best known nonexact algorithm was a tabu search TS-SAT proposed by the CNES. The main differences with CNS-SAT algorithm are the following. (1) TS-SAT 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-SAT 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-SAT 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 5 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^*_{TS} and the associated computing time time_{TS} (in seconds) obtained with TS-SAT. Columns 6-7 give the average profit value $\overline{f}_{\text{CNS}}$ and the average time time_{CNS} needed by CNS-SAT to find such a solution. It is easy to see that CNS-SAT is much more efficient and quicker than TS-SAT (see also the line labeled "Average").

7. Conclusion

In this paper, we propose and discuss a generic method for combinatorial optimization problems. Its consideration within various fields shows that CNS is very efficient, robust, quick, and relatively easy to implement. Note that other heuristic solution methods which were not discussed here could be considered within the CNS framework (e.g., [24, 25]). On the contrary to tree search, CNS mainly focuses on the bottom part of the search tree (i.e., it evolves close to the leaves). In contrast with standard local search methods, on the one hand, it can perform jumps in the search tree, which means that the search space is likely to be connected. On the

Table 5: Comparison between TS-SAT and CNS-SAT.

Instance	n	m	f^*_{TS}	Time_{TS}	$\overline{f}_{CNS}$	Time_{CNS}
1401	488	914	174058	846	176055	120
1403	665	1317	174137	1324	176134	332
1405	855	1815	174174	1574	176175	1314
1021	1057	2355	174238	2197	176241	2422
1504	605	1253	124238	1011	124241	405
1506	940	2060	165244	1945	168224	1423
Average			164348.17	1272.86	166178.33	859.57

other hand, there is no need to extend the search space by considering unfeasible solutions and penalizing them with a function which can be difficult to design and to tune.

CNS is especially well adapted when the optimization problem can be divided into a series of decision problems. It was the case for three of the presented applications.

(i) Graph coloring can be tackled by the search of k-colorings with decreasing values of k.

(ii) The frequency assignment problem can be approached at level k by only considering interference constraints at level k and imperative constraints. Then, if a feasible solution is found at level k, level $k-1$ is considered.

(iii) The car fleet management problem can be considered with a fixed number k of cars in stock (k being first equal to the existing cars in stock), and the provided solution will be the less costly solution among the different considered k values (k can augment if cars are purchased).

CNS is a very flexible method for at least four reasons.

(i) It can manage various ways of representing a solution; the component i of solution s, denoted s_i, can involve only one information (e.g., the color of a vertex, the car type of a client request, a binary decision value associated with a selection of a photograph or not), or several data of different nature (e.g., a frequency and a polarization, a binary decision value for the selection of a photograph, and the associated camera). This allows to better manage the repairing phase.

(ii) It 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. If a specific constraint involves several variables, such a constraint can be relaxed (at least to save CPU time), as it was the case for the satellite range scheduling problem when the capacity constraint was only considered at specific iterations.

(iii) It is also well adapted for some problems where the unassigned variables actually correspond to an expensive assignment for the considered problem (e.g., a nonassigned variable corresponds to a subcontracted request for the car rental company).

(iv) Other significant ingredients can be easily added within the framework of CNS to enhance its efficiency, such as intensification or diversification procedures.

CNS can be easily combined with evolutionary heuristics, like genetic or adaptive memory algorithms. We can consider that it was already successfully performed for graph coloring [17] and a satellite range scheduling problem [21]. In both cases, the resulting methods are the best for the considered problems. Therefore, a relevant avenue of research would consist in hybridizing CNS with other techniques for other optimization problems.

References

[1] M. Garey and D. S. Johnson, *Computer and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, Calif, USA, 1979.

[2] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[3] F. Glover, "Tabu search—part I," *Journal on Computing*, vol. 1, no. 3, pp. 190–250, 1989.

[4] I. Charon and O. Hudry, "The noising method: a new method for combinatorial optimization," *Operations Research Letters*, vol. 14, no. 3, pp. 133–137, 1993.

[5] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers and Operations Research*, vol. 24, no. 11, pp. 1097–1100, 1997.

[6] C. Voudouris and E. Tsang, "Guided local search and its application to the traveling salesman problem," *European Journal of Operational Research*, vol. 113, no. 2, pp. 469–499, 1999.

[7] M. Vasquez and N. Zufferey, "Consistent neighborhood search for constrained assignment problems," in *Proceedings of the 9th International Conference on Modeling, Optimization & Simulation (MOSIM '12)*, Bordeaux, France, June 2012.

[8] A. Dupont, M. Vasquez, and D. Habet, "Consistent neighbourhood in a Tabu search," in *Metaheuristics: Progress as Real Problem Solverschapter*, no. 17, pp. 367–386, Springer, 2005.

[9] F. Glover and M. Laguna, *Tabu Search*, Kluwer Academic, Boston, Mass, USA, 1997.

[10] V. Chvátal, "Resolution search," *Discrete Applied Mathematics*, vol. 73, no. 1, pp. 81–99, 1997.

[11] I. Blöchliger and N. Zufferey, "A graph coloring heuristic using partial solutions and a reactive tabu scheme," *Computers and Operations Research*, vol. 35, no. 3, pp. 960–975, 2008.

[12] P. Galinier and A. Hertz, "A survey of local search methods for graph coloring," *Computers and Operations Research*, vol. 33, no. 9, pp. 2547–2562, 2006.

[13] A. Hertz and D. de Werra, "Using tabu search techniques for graph coloring," *Computing*, vol. 39, no. 4, pp. 345–351, 1987.

[14] P. Galinier, A. Hertz, and N. Zufferey, "An adaptive memory algorithm for the k-coloring problem," *Discrete Applied Mathematics*, vol. 156, no. 2, pp. 267–279, 2008.

[15] P. Galinier and J.-K. Hao, "Hybrid evolutionary algorithms for graph coloring," *Journal of Combinatorial Optimization*, vol. 3, no. 4, pp. 379–397, 1999.

[16] C. Morgenstern, "Distributed coloration neighborhood search," *Discrete Mathematics and Theoretical Computer Science*, vol. 26, pp. 335–3357, 1996.

[17] E. Malaguti, M. Monacci, and P. Toth, "A metaheuristic approach for the vertex coloring problem," *INFORMS Journal on Computing*, vol. 20, no. 2, pp. 302–316, 2008.

[18] A. Dupont, E. Alvernhe, and M. Vasquez, "Efficient filtering and Tabu search on a consistent neighbourhood for the Frequency Assignment Problem with Polarisation," *Annals of Operations Research*, vol. 130, no. 1–4, pp. 179–198, 2004.

[19] A. Hertz, D. Schindl, and N. Zufferey, "A solution method for a car fleet management problem with maintenance constraints," *Journal of Heuristics*, vol. 15, no. 5, pp. 425–450, 2009.

[20] 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," *Computational Optimization and Applications*, vol. 20, no. 2, pp. 137–157, 2001.

[21] N. Zufferey, P. Amstutz, and P. Giaccari, "Graph colouring approaches for a satellite range scheduling problem," *Journal of Scheduling*, vol. 11, no. 4, pp. 263–277, 2008.

[22] D. Habet, M. Vasquez, and Y. Vimont, "Bounding the optimum for the problem of scheduling the photographs of an Agile Earth Observing Satellite," *Computational Optimization and Applications*, vol. 47, no. 2, pp. 307–333, 2010.

[23] E. Bensana, M. Lemaître, and G. Verfaillie, "Earth observation satellite management," *Constraints*, vol. 4, no. 3, pp. 293–299, 1999.

[24] M. Vasquez and J.-K. Hao, "A heuristic approach for antenna positioning in cellular networks," *Journal of Heuristics*, vol. 7, no. 5, pp. 443–472, 2001.

[25] A. Dupont, A. C. Linhares, C. Artigues, D. Feillet, P. Michelon, and M. Vasquez, "The dynamic frequency assignment problem," *European Journal of Operational Research*, vol. 195, no. 1, pp. 75–88, 2009.