# Consistent Neighborhood Search for Constrained Assignment Problems

Michel Vasquez, Nicolas Zufferey

$9^{th}$ *International Conference of Modeling, Optimization and Simulation - MOSIM'12*
*June 6-8, 2012 - Bordeaux - France*
*"Performance, interoperability and safety for sustainable development"*

# CONSISTENT NEIGHBORHOOD SEARCH FOR CONSTRAINED ASSIGNMENT PROBLEMS

**Michel Vasquez**

École des Mines d'Alès
LGI2P Research Center
30035 Nimes cedex 01 - France
michel.vasquez@mines-ales.fr

**Nicolas Zufferey**

HEC - University of Geneva
Blvd du Pont-d'Arve 40
1211 Geneva 4 - Switzerland
nicolas.zufferey-hec@unige.ch

**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 formally propose CNS, a new heuristic solution method. 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, satellite range scheduling and vehicle fleet management with maintenance constraints.*

**KEYWORDS:** *local search heuristics, combinatorial optimization*

## 1 INTRODUCTION

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. 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, i.e. the definition of the *neighborhood* structure, depends on the considered problem. Popular local search methods are simulated annealing (Kirkpatrick *et al.*, 1983), tabu search (Glover, 1989), threshold algorithms (Charon & Hudry, 1993), variable neighborhood search (Mladenovic & Hansen, 1997), and guided local search (Voudouris & Tsang, 1999).
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), 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 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). In this paper, 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 be considered as belonging to the CNS methodology, for various NP-hard constrained assignment problems.

The paper is organized as follows. In Section 2, the CNS methodology is proposed. Then, heuristics for various problems are presented within a CNS environment: graph coloring (Section 3), frequency assignment (Section 4), satellite range scheduling (Section 5), and car fleet management with maintenance (Section 6). We end up the paper with a conclusion.

## 2 CNS METHODOLOGY

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

An important feature of CNS is the definition of the neighborhood structure in $S^{(partial)}$. 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 $N(s)$. In contrast, any move $m$ is performed in 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. *Repairing phase.* In order to keep the partial solution feasible, remove the value of all the variables of $C(m)$.

We have now all the ingredients to formulate a pseudo-code of CNS in Algorithm 1.

**Algorithm 1** CNS

**Initialization:** generate an initial solution $s$, set $s^* = s$ and $\hat{f}^* = \hat{f}(s)$;

**While a time limit is not met and $\hat{f}^* > 0$, 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) let $s^{cand}$ be the so obtained candidate neighbor solution (which might be non feasible at this stage);

   (c) update the best candidate move: 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: if $\hat{f}(s) < \hat{f}^*$, set $s^* = s$ and $\hat{f}^* = \hat{f}(s)$;

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

Many local search methods (e.g., tabu search, simulated annealing, random walk, threshold algorithms, etc.) can be adapted within the environment of 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 $f$ 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 (Glover & Laguna, 1997).
Tabu search adapted within the framework of CNS has the following specificities: working in $S^{(feasible)}$; 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 $s_i$, it is then tabu to remove the value of $s_i$ for a certain number of iterations.

In the four next sections, we present existing tabu search methods, but within a CNS unified framework. Four fields are involved: graph coloring, frequency assignment, satellite range scheduling and car fleet management. For each problem, CNS was implemented in C/C++ and the allocated time limits were always fair when compared to the other methods. All the details on the experimental conditions of the compared methods (e.g., time limit, computer type, programming language) can be found in the corresponding papers.

## 3 GRAPH COLORING

The main reference associated with this section is (Bloechliger & Zufferey, 2008). The authors proposed a tabu search heuristic for the graph coloring problem, which we denote here CNS-GCP for 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 $\chi$). Both problems are NP-hard (Garey & Johnson, 1979) and many heuristics were proposed to solve them. For a recent survey, the reader is referred to (Malaguti & Toth, 2010). Currently, no known exact solution method is able to solve all instances with up to 100 vertices (Herrmann & Hertz, 2002). For larger instances, upper bounds on the chromatic number can be obtained by using heuristic algorithms. 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 a CNS Approach

The best $k$-coloring heuristics are based on two approaches. In $S^{(penalty)}$, 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 (Hertz & de Werra, 1987).

In contrast, in $S^{(partial)}$, 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 $S^{(partial)}$ 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 $\hat{f}(s) = |s| = |\{s_i \mid s_i > 0\}|$. At each iteration, the best non tabu move is performed (ties are randomly broken).

### 3.3 Comparisons with Other Methods

In Table 1 are considered difficult benchmark instances from the DIMACS Challenge (see `ftp://dimacs.rutgers.edu/pub/challenge/graph/`).
Below, CNS-GCP is compared with other state-of-the-art coloring heuristics, which are: Tabucol (Hertz & de Werra, 1987), GH (Galinier & Hao, 1999), MOR (Morgenstern, 1996), and MMT (Malaguti *et al.*, 2008). Tabucol is a standard tabu search working in $S^{(penalty)}$. 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 is reported the smallest $k$ such that a feasible $k$-coloring was found. We can observe that CNS-GCP is rather competitive with the best coloring methods. However, it is much simpler.

Table 1: CNS-GCP versus other methods

| Graph | $n$ | $\chi$, $k^\star$ | 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 |

## 4 FREQUENCY ASSIGNMENT

The main reference associated with this section is (Dupont *et al.*, 2004), where the Frequency Assignment Problem with Polarization (FAPP) was considered. The authors proposed a tabu search approach working in $S^{(partial)}$, 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 radio-electric bond, established between antennae at distinct sites, which has a given frequency and polarization. A frequency resource for a path is a pair (frequency, polarization), whose components are respectively associated with the carrying frequency of the transmitted signal and the wave polarization. The polarization is a binary variable (i.e. only vertical or horizontal). With each path $s_i$ is associated a set of available resources $\{(f_i,p_i),\ f_i \in F_i$ and $p_i \in P_i\}$, where $F_i$ is the ordered frequency domain representing the authorized wave band, and $P_i$ is the polarization information which may include a required polarization. Thus, 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 ($ECC$) 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:

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

since in the $11^{th}$ 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 $\gamma_{ij}^k$ and $\delta_{ij}^k$ gaps. More precisely, $|f_i - f_j| \geq \frac{|p_i+p_j|}{2}\gamma_{ij}^{(k)} + \frac{|p_i-p_j|}{2}\delta_{ij}^{(k)}$. 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: (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 $k-1$; (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 a CNS Approach

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 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 non assigned 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.

### 4.3 Comparisons with Other Methods

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.fr/conf/roadef-2001-challenge/`). In Table 2 are presented the results obtained by the four 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 $TS\text{-}VN$, 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*. 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.

Table 2: CNS-FAPP versus others methods

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

## 5 SATELLITE SCHEDULING

The main reference associated with this section is (Zufferey *et al.*, 2008), where the Multi-Resource Satellite Range Scheduling Problem (denote here by SAT for sake of simplicity) is tackled.

### 5.1 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. The SAT is a NP-hard problem (Barbulescu *et al.*, 2004) 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 *U.S. 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 (Parish, 1994). The human schedulers themselves state that minimizing the number of conflicts reduces (1) their workload, (2) communication with outside agencies, and (3) the time required to produce a conflict-free schedule (Barbulescu *et al.*, 2004).

### 5.2 Description of a CNS Approach

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 main idea is to 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.

### 5.3 Comparisons with other Methods

It is relevant to compare the three following heuristics: CNS-SAT (which also uses diversification procedures), AMA (which is an adaptive memory algorithm using CNS-SAT as intensification procedure) and GENITOR (Barbulescu *et al.*, 2004). The latter method is a genetic algorithm which usually provides the best solutions for the SAT up to 2007. GENITOR is based on the permutation search space, where, as proposed by several researchers (e.g. (Globus *et al.*, 2004), (Barbulescu *et al.*, 2004)), a solution is encoded as a permutation $\pi$ of the $n$ jobs to schedule. Let $S^{(permutation)}$ be the search space containing all the possible permutations. From a permutation $\pi$ in $S^{(permutation)}$, it is possible to generate a schedule in $S^{(feasible)}$ 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 (Zufferey *et al.*, 2008)). Each instance was built by a well-known generator described and used in (Barbulescu *et al.*, 2004). It produces instances of the SAT by modeling realistic features. A maximum time limit of 15 minutes Pentium 4 (2 GHZ, 512 MB of RAM) is considered in Table 3, which is consistent with the ones used in (Barbulescu *et al.*, 2004), and is appropriate in a practical standpoint. In Table 3, we compare the average number of bumped jobs of GENITOR, CNS-SAT 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-SAT, and 51.5 bumped jobs by AMA. Such results obviously show that CNS-SAT is more efficient than the previous best existing algorithm for the SAT.

Table 3: CNS-SAT versus other methods

| 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 |

## 6 FLEET MANAGEMENT

The main reference associated with this section is (Hertz *et al.*, 2009), 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 here by CAR for sake of simplicity). The particularity of the problem is that feasible solutions are very easy to find, but can cost a lot. Thus, $S^{(partial)}$ was designed to avoid to assign the most expensive value to each variable.

### 6.1 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.

### 6.2 Description of a CNS Approach

The general pseudo-code of the method, denoted CNS-CAR 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 $S^{(partial)}$, 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.

**Algorithm 2** Algorithm CNS-CAR

**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).

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 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, $S^{(partial)}$ 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.

### 6.3 Comparisons with Other Methods

In Table 4 and Table 5 are reported the results for the 16 benchmark instances of the Challenge. 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 triplet $(x, y, z, w)$, 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 4 (for instances where purchases are forbidden) and Table 5 (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 CNS-CAR gives in average better results than those obtained by the four best competitors of the Challenge.

Table 4: Results on 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 5: Results on 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% |

## 7 CONCLUSION

In this paper, we propose and discuss a generic method, CNS, for constrained assignment problems. Its consideration within various fields shows that CNS is efficient, robust, quick and relatively easy to implement. Other heuristics, which were not discussed here, can obviously be considered as belonging to the CNS environment (e.g., (Morgenstern, 1996), (Vasquez & Hao, 2001b), (Vasquez & Hao, 2001a), (Dupont *et al.*, 2009)).

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. Graph coloring can be tackled by the search of $k$-colorings with decreasing values of $k$. 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. 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 also a flexible method for at least two reasons. Firstly, it can consider various types of constraints. It is particularly well adapted for constraints linking two or three variables together, because the repairing phase is usually straightforward in such situations. Secondly, 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 non assigned variable corresponds to a subcontracted request for the car rental company).

**References**


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

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

Charon, I., & Hudry, O. 1993. The noising method: a new method for combinatorial optimization. *Operations Research Letters*, **14**, 133–137.

Dupont, A., Alverhne, E., & Vasquez, M. 2004. Efficient Filtering and Tabu Search on a Consistent Neighbouhood for the Frequency Assignment Problem with Polarisation. *Annals of Operations Research*, **130**, 179 – 198.

Dupont, A., Carneiro-Linhares, A., Artigues, Ch., Feillet, D., Michelon, Ph., & Vasquez, M. 2009. The dynamic frequency assignment problem. *European Journal of Operational Research*, **195 (1)**, 75 – 88.

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

Garey, M., & Johnson, D.S. 1979. *Computer and Intractability: a Guide to the Theory of NP-Completeness*. San Francisco: Freeman.

Globus, A., Crawford, J., Lohn, J., & Pryor, A. 2004. A comparison of techniques for scheduling earth observing satellites. *In: Proceedings of the Sixteenth Innovative Applications of Artificial Intelligence Conference (iaai-04)*.

Glover, F. 1989. Tabu search - part I. *ORSA Journal on Computing*, **1**, 190–205.

Glover, F., & Laguna, M. 1997. *Tabu Search*. Boston: Kluwer Academic Publishers.

Herrmann, F., & Hertz, A. 2002. Finding the chromatic number by means of critical graphs. *ACM Journal of Experimental Algorithmics*, **7 (10)**, 1 – 9.

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

Hertz, A., Schindl, D., & Zufferey, N. 2009. A solution method for a car fleet management problem with maintenance constraints. *Journal of Heuristics*, **15 (5)**, 425 – 450.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. 1983. Optimization by simulated annealing. *Science*, **220 (5498)**, 671–680.

Malaguti, E., & Toth, P. 2010. A survey on vertex coloring problems. *International Transactions in Operational Research*, **17 (1)**, 1 – 34.

Malaguti, E., Monaci, M., & Toth, P. 2008. A metaheuristic approach for the vertex coloring problem. *INFORMS Journal on Computing*, **20 (2)**, 302 – 316.

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

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

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

Vasquez, M., & Hao, J.-K. 2001a. A heuristic approach for antenna positioning in cellular networks. *Journal of Heuristics*, **7**(5), 443 – 472.

Vasquez, M., & Hao, J.-K. 2001b. A Logic-Constrained Knapsack Formulation and a Tabu Algorithm for the Daily Photograph Scheduling of an Earth Observation Satellite. *Computational Optimization and Applications*, **20**(2), 137 – 157.

Voudouris, C., & Tsang, E. 1999. Guided local search. *European Journal of Operational Research*, **113 (2)**, 469Ű–499.

Zufferey, N., Amstutz, P., & Giaccari, P. 2008. Graph colouring approaches for a satellite range scheduling problem. *Journal of Scheduling*, **11 (4)**, 263 – 277.