RAIRO Operations Research
Will be set by the publisher

LARGE NEIGHBORHOOD IMPROVEMENTS FOR SOLVING CAR SEQUENCING PROBLEMS

B. Estellon[1], F. Gardi[1, 2] and K. Nouioua[1]

Abstract. The $\mathcal{NP}$-hard problem of car sequencing has received a lot of attention these last years. Whereas a direct approach based on integer programming or constraint programming is generally fruitless when the number of vehicles to sequence exceeds the hundred, several heuristics have shown their efficiency. In this paper, very large-scale neighborhood improvement techniques based on integer programming and linear assignment are presented for solving car sequencing problems. The effectiveness of this approach is demonstrated through an experimental study made on seminal CSPlib's benchmarks.

Keywords: combinatorial optimization, car sequencing/scheduling, very large-scale neighborhood search, integer programming, assignment

Mathematics Subject Classification. 90C27, 90B35, 90C10

Résumé. Le problème $\mathcal{NP}$-difficile de l'ordonnancement de véhicules a reçu une attention particulière ces dernières années. Alors qu'une approche directe par programmation en nombres entiers ou programmation par contraintes est généralement vaine dès lors que le nombre de véhicules dépasse la centaine, plusieurs heuristiques ont démontré leur efficacité. Dans cet article, des techniques d'amélioration par recherche locale à voisinage large basées sur la programmation en nombres entiers et l'affectation linéaire sont présentées pour résoudre le problème d'ordonnancement de véhicules. L'efficacité de cette approche est démontrée au travers d'une étude expérimentale sur les jeux d'essais de la CSPlib.

submitted on –, revised on –.

[1] Laboratoire d'Informatique Fondamentale – CNRS UMR 6166, Université de la Méditerranée – Aix-Marseille II, Parc Scientifique et Technologique de Luminy, case 901, 163 avenue de Luminy, 13288 Marseille cedex 9, France (e-mail: `bertrand.estellon@lif.univ-mrs.fr` & `karim.nouioua@lif.univ-mrs.fr`)

[2] Experian-Prologia, Parc Scientifique et Technologique de Luminy, case 919, bâtiment CCIMP, 13288 Marseille cedex 9, France (e-mail: `fgardi@experian-prologia.fr`)

1. Introduction

The *car sequencing problem* consists in scheduling cars along an assembly line composed of different posts where are installed the equipments and options relative to each vehicle (radio, sun-roof, air-conditioning, etc). In order to smooth the workload on all the posts, it is necessary to space out in the sequence the vehicles for which setting options needs some heavy operations. In other words, the goal is to minimize the density of vehicles which require much work to assemble, to avoid overloading the posts where these vehicles are assembled. This need of spacing out vehicles is formalized by defining a *ratio constraint* for each option. For example, for an option to which is associated the ratio 3/7, one shall not find more than 3 vehicles affected by the option in any window (that is, set of adjacent positions) consisting of 7 vehicles. In the problem generally addressed in the literature [10, 12, 13, 16, 28, 32, 33], the goal is to find a sequence of vehicles satisfying all ratio constraints.

Here a slightly different problem is considered, closer to industrial realities [3, 4]: as it is not possible to know in advance if all the ratio constraints are satisfiable, these ones are defined as soft constraints. In this way, the objective is to minimize the number of violations of ratio constraints. In the previous example, if 5 vehicles have the option in a window of 7, then 2 violations are counted. (In this model, the violations could be weighted by some constants to distinguish the options having priority). In addition, as suggested by Gravel et al. [13] to simulate the presence of vehicles to be sequenced the previous (resp. following) day, the evaluation of the sequence is done beyond the first (resp. last) window by adding fictive vehicles requiring no option before the first (resp. last) vehicle of the sequence; these additional windows are called "side windows". The interested reader is referred to Figure 10 in Appendix for an illustration of the impact of the different ways for evaluating solutions.

The car sequencing problem is strongly $\mathcal{NP}$-hard [10, 16]. Historically, this problem was rather studied in constraints community [10, 11, 29, 32]. Car sequencing problems, referenced as Prob001 in CSPlib [11], serve as typical benchmarks for constraint programming solvers. However, a brute approach by using contraint programming (or even integer programming) softwares reaches its limit when one hundred vehicles with few options are considered (see the studies of [13, 29]). Then, several heuristics have been proposed to solve effectively car sequencing problems; in the literature appears three kinds of approaches, sometimes mixed the ones with the others: ant colony optimization [12, 13, 19, 32, 33], greedy [12], local search [12, 28]. These approaches have been intensively studied and experimented in the context of the ROADEF'2005 Challenge [3, 4], organized by the French Operations Research Society and the car manufacturer RENAULT (see [7, 30] for example). At the same time, some works have been initiated on the integration of local search techniques into a constraint programming environment dedicated to the resolution of ratio constraints [20, 23, 24].

1.1. Motivations

During the qualification stage of this Challenge, the authors [5, 7, 8] have designed and experimented an original approach mixing local search and integer programming to solve real-life car sequencing problems (including constraints and objectives from paint shop in addition to those of assembly shop). We have recently been informed of a similar work by Prandtstetter and Raidl [25, 26] (note that the results given in [25] are in fact erroneous [27]). Then, the present paper reports our last advances on the subject.

The paper is organized as follows. First, a new formulation of the problem as an integer linear program is described. As an example, this formulation is used to establish that the car sequencing instance 21-90 of CSPlib [11], the last one whose status remained unknown, is unsatisfiable. Then, we show how exploiting *integer linear programming* (ILP) in a *very large-scale neighborhood search* (VLNS) algorithm to solve car sequencing problems. In particular, a family of exponential neighborhoods is exhibited for which the best improving neighbor can be found in polynomial time and space by reduction to a linear assignment problem. Finally, this approach is compared to a classical but very efficient local search approach for sequencing problems (see [2, pp. 372–375]), experimented by Gottlieb et al. [12, 28] and revisited by the authors [6–8] to win ROADEF Challenge, who called it *very fast local search* (VFLS). An extensive computational study made on CSPlib's benchmarks [11] shows that VLNS is practically effective, but not competitive with a state-of-the-art implementation of VFLS. The good news is that the hybridization of the two techniques, performed by adding large neighborhood improvements to the local transformations employed in VFLS, is shown to be still more powerful than pure VFLS, in particular on large and hard instances.

All the terminology related to local search and employed throughout this paper is derived from the book edited by Aarts and Lenstra [2]. A comprehensive survey on very large-scale neighborhood search techniques has been recently published by Ahuja et al. [1].

2. Integer linear programming approach

2.1. A new ILP formulation

In this section, a new ILP formulation of the car sequencing problem is described. In order to reduce the number of variables, similar vehicles are grouped into classes (two vehicles belong to the same class if they share the same options). The number of possible positions in the sequence (that is, of vehicles) and the number of classes are respectively noted NPOS and NCL. The number of vehicles of class k is noted N_k, and $\mathsf{P}_i/\mathsf{Q}_i$ denotes the ratio attached to option i (with $1 \leq \mathsf{P}_i < \mathsf{Q}_i \leq \mathsf{NPOS}$).

To each pair class k/position j is associated a binary variable $cl_{k,j}$ whose value is 1 if the vehicle at position j belongs to the class k and 0 otherwise. The constraints (1) and (2), written below, ensure that each vehicle of a class is assigned to a position in the sequence and that a position is occupied by one and only one vehicle of a class.

$$\sum_{j=1}^{\mathsf{NPOS}} cl_{k,j} = \mathsf{N}_k \quad \forall k \in \{1,\dots,\mathsf{NCL}\} \tag{1}$$

$$\sum_{k=1}^{\mathsf{NCL}} cl_{k,j} = 1 \quad \forall j \in \{1,\dots,\mathsf{NPOS}\} \tag{2}$$

For each pair class k/option i, the constant $\mathsf{OP}_{k,i}$ equals 1 if the vehicles of class k have option i and 0 otherwise. Then, to each pair option i/position j is associated a binary variable $o_{i,j}$ whose value is 1 if the vehicle at position j has option i and 0 otherwise. Then, the constraints (3) express variables $o_{i,j}$ in function of variables $cl_{k,j}$ and constants $\mathsf{OP}_{k,i}$.

$$o_{i,j} = \sum_{k=1}^{\mathsf{NCL}} \mathsf{OP}_{k,i} \times cl_{k,j} \quad \forall i \in \{1,\dots,\mathsf{NOP}\} \;\; \forall j \in \{1,\dots,\mathsf{NPOS}\} \tag{3}$$

Now, denote by $v_{i,j}$ the variable which counts the number of violations on option i for the window ending at position j. Formally, the number of violations is determined by the constraint $v_{i,j} = \max\{0, \sum_{f=j-\mathsf{Q}_i+1}^{j} o_{i,f} - \mathsf{P}_i\}$. Since the $v_{i,j}$ are minimized, such a constraint is equivalent, for each option $i \in \{1,\dots,\mathsf{NOP}\}$, to the two linear inequalities

$$\begin{aligned} v_{i,j} &\geq s_{i,j} - \mathsf{P}_i \quad \forall j \in \{\mathsf{P}_i+1,\dots,\mathsf{NPOS}+\mathsf{Q}_i-\mathsf{P}_i-1\} && (4)\\ v_{i,j} &\geq 0 \quad \forall j \in \{\mathsf{P}_i+1,\dots,\mathsf{NPOS}+\mathsf{Q}_i-\mathsf{P}_i-1\} && (5)\end{aligned}$$

to which are added the equalities

$$\begin{aligned} s_{i,\mathsf{Q}_i} &= \sum_{f=1}^{\mathsf{Q}_i} o_{i,f} && (6)\\ s_{i,j} &= s_{i,j+1} - o_{i,j+1} \quad \forall j \in \{\mathsf{P}_i+1,\dots,\mathsf{Q}_i-1\} && (7)\\ s_{i,j} &= s_{i,j-1} + o_{i,j} - o_{i,j-\mathsf{Q}_i} \quad \forall j \in \{\mathsf{Q}_i+1,\dots,\mathsf{NPOS}\} && (8)\\ s_{i,j} &= s_{i,j-1} - o_{i,j-\mathsf{Q}_i} \quad \forall j \in \{\mathsf{NPOS}+1,\dots,\mathsf{NPOS}+\mathsf{Q}_i-\mathsf{P}_i-1\} && (9)\end{aligned}$$

Equality (6) defines the first complete window of the sequence for option i, whereas constraints (8) define dynamically the other windows from the left to the right of the sequence. Constraints (7) (resp. constraints (9)) define partial windows on the left (resp. right) side; note that side windows containing less than P_i vehicles are omitted, because they can not cause any violation.

Finally, the objective function of the program is written as follows, with CV_i the cost of one violation on option i (here $\mathsf{CV}_i = 1$ for all options).

$$\text{Minimize} \quad \sum_{i=1}^{\mathsf{NOP}} \sum_{j=\mathsf{P}_i+1}^{\mathsf{NPOS}+\mathsf{Q}_i-\mathsf{P}_i-1} \mathsf{CV}_i \times v_{i,j} \tag{10}$$

One shall remark that if the variables $cl_{k,j}$ are integral, then any solution of the program is integral, even if all the other variables are unbounded. Inversely, if the sole variables $o_{i,j}$ are integral, then any solution of the program is integral too. Consequently, the domains of the variables can be defined as follows: if $\mathsf{NCL} \leq \mathsf{NOP}$ then

$$cl_{k,j} \in \{0,1\} \qquad \forall k \in \{1,\dots,\mathsf{NCL}\} \quad \forall j \in \{1,\dots,\mathsf{NPOS}\} \tag{11}$$

and variables $o_{i,j}$, $s_{i,j}$, $v_{i,j}$ are free, else

$$o_{i,j} \in \{0,1\} \qquad \forall i \in \{1,\dots,\mathsf{NOP}\} \quad \forall j \in \{1,\dots,\mathsf{NPOS}\} \tag{12}$$

and variables $cl_{k,j}$, $s_{i,j}$, $v_{i,j}$ are free. In conclusion, the total number of variables is lower than $(\mathsf{NCL} + 5 \cdot \mathsf{NOP}) \cdot \mathsf{NPOS}$, of which $\min\{\mathsf{NCL}, \mathsf{NOP}\} \cdot \mathsf{NPOS}$ are binary, and the total number of constraints is lower than $\mathsf{NCL} + (7 \cdot \mathsf{NOP} + 1) \cdot \mathsf{NPOS}$. Unlike the formulations proposed by Gravel et al. [13] or Prandtstetter and Raidl [25,26], the number of integer variables in this ILP formulation is not determined by the sole number of classes, but by the number of options too. Moreover, the introduction of the variables $s_{i,j}$ enables to reduce the number of nonzeros of the matrix by factorizing the sums of $o_{i,j}$ which normally appear in the constraint (4) when counting up the number of violations.

2.2. Unsatisfiability of CSPlib's instance 21-90

Using the above formulation, we show how proving the unsatisfiability of the CSPlib's instance 21-90 [11], the last one whose status remained unknown. This instance is composed of 100 vehicles distributed in 23 classes according to 5 options, to which are associated respectively the ratio constraints 1/2, 2/3, 1/3, 2/5 and 1/5. Figure 1 describes precisely the assignment of the option to the different classes of vehicles. For example, the class K contains 3 vehicles which must satisfy the ratio constraints 1/2, 2/3 and 2/5.

class	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
option 1/2		×			×	×		×	×		×	×	×			×			×	×			
option 2/3	×		×	×					×		×	×	×	×	×	×	×		×	×	×		
option 1/3			×	×		×			×	×			×	×				×	×	×			
option 2/5			×		×		×	×	×		×	×			×		×			×			×
option 1/5				×	×				×			×	×				×	×			×	×	×
cardinal	14	11	2	1	1	3	5	4	1	5	3	2	2	7	9	14	3	2	6	2	1	1	1

Figure 1. CSPlib's instance 21-90.

Considering solely one option, the following observation is useful to determine if one solution exists or not, which satisfies the ratio constraint associated to this option.

Proposition 2.1 (Necessary condition for satisfiability)**.** *Let* N_i *be the number of vehicles concerned by an option* i *having ratio constraint* $\mathsf{P}_i/\mathsf{Q}_i$ *and* NPOS *the total number of positions. Then, the ratio constraint* $\mathsf{P}_i/\mathsf{Q}_i$ *is satisfiable only if*

$$\mathsf{N}_i \leq \mathsf{P}_i \cdot \lfloor \mathsf{NPOS}/\mathsf{Q}_i \rfloor + \min\{\mathsf{P}_i, \mathsf{NPOS} \bmod \mathsf{Q}_i\}$$

Proof. Consider a partition of the sequence into windows of size Q_i. Such a partition is composed of $\lfloor \mathsf{NPOS}/\mathsf{Q}_i \rfloor$ windows of size Q_i, plus one of size $\mathsf{NPOS} \bmod \mathsf{Q}_i$ if NPOS is not a multiple of Q_i. Since each window can not contain more than P_i vehicles of ratio $\mathsf{P}_i/\mathsf{Q}_i$ without causing violations, we obtain by summing on all the windows that the number N_i of vehicles concerning by this ratio can not exceed $\mathsf{P}_i \cdot \lfloor \mathsf{NPOS}/\mathsf{Q}_i \rfloor + \min\{\mathsf{P}_i, \mathsf{NPOS} \bmod \mathsf{Q}_i\}$ when the ratio constraint $\mathsf{P}_i/\mathsf{Q}_i$ is satisfiable. □

Remark 2.2. The necessary condition for satisfiability becomes sufficient when only one option i is considered in input of the problem. To observe that, place P_i cars concerned by the option at the beginning of each window of size exactly Q_i, and $\min\{\mathsf{P}_i, \mathsf{NPOS} \bmod \mathsf{Q}_i\}$ cars in the window which remains.

According to Proposition 2.1, the value $\delta_i = \mathsf{P}_i \cdot \lfloor \mathsf{NPOS}/\mathsf{Q}_i \rfloor + \min\{\mathsf{P}_i, \mathsf{NPOS} \bmod \mathsf{Q}_i\} - \mathsf{N}_i$ seems to be an appropriate index to determine how hard is the ratio constraint associated to an option. Following this observation, an idea is to solve the ILP presented in the previous section by branching first on variables $o_{i,j}$ which corresponds to the hardest options (i.e., having the smallest δ_i). When some ratio constraints are stiff, such a technique allows early cuts in the branch-and-bound tree, particularly if the objective is to find a sequence without violation. The unsatisfiability of instance 21-90 is precisely established by using this technique.

	cardinal	limit	δ_i
option 1/2	49	50	1
option 2/3	67	67	0
option 1/3	31	34	3
option 2/5	33	40	7
option 1/5	15	20	5

FIGURE 2. The hardness of options in instance 21-90.

Figure 2 shows for each option of instance 21-90, the number of vehicles concerned by the option and the limit given by Proposition 2.1: here the hardest options are 1/2 and 2/3. In effect, there are only 12239 valid assignments of the $o_{i,j}$ corresponding to options 1/2 and 2/3. At this point, computing the continuous relaxation for each subproblem obtained shows that no solution exists without violation, which proves the unsatisfiability of instance 21-90.

3. Very large-scale neighborhood search

In this section is described the very large-scale neighborhood search approach (VLNS). Roughly speaking, this approach is based on a simple *descent method* where each iteration consists in solving exactly a subproblem by ILP. Mixing local search and integer linear programming, this approach can also be viewed as hybrid. This kind of approach was previously formalized by Mautor and Michelon [18] and applied with success to the resolution of a ressource-constrained project scheduling problem [22] (see also [21] for more details).

3.1. k-permutation neighborhoods

Given a sequence, a very simple neighborhood is defined by the permutation of two vehicles of the sequence; this technique forms the basis of the very fast local search approach, described in the next section. A natural extension of this paradigm is the local search by K-permutation: choose K positions in the current sequence and find a permutation of the vehicles at these K positions which does not increased the cost of the sequence. In this way, the neighborhood of the current solution is defined as the set of sequences which are obtainable by permuting these K movable vehicles. The size of the neighborhood is exactly $\mathsf{K}!$, which becomes exponential with respect to the size of the input data when $\mathsf{K} = \Omega(\mathsf{NPOS})$. Thus, the goal is to find among all these neighbors one having a better cost than the current solution or the best possible one. Even for small values of K, computing such a neighbor seems to be difficult in a reasonable lapse of time. Next we show how taking advantage of ILP to explore efficiently K-permutation neighborhoods.

3.2. ILP-based local search

Having computed randomly or greedily an initial sequence (see [8, 12] for good $O(\mathsf{NPOS}^2)$-time greedy algorithms), the local search is done as follows. At each iteration, a K-permutation is performed by solving the integer linear program described in section 2.1, where all variables corresponding to non-movable vehicles are fixed. The resolution of the program is done using a basic branch-and-bound procedure, in a lapse of time limited to $\mathsf{T\text{-}MAX}$ seconds. Once the resolution finished or interrupted for lack of time, the current solution is updated and the values of K and $\mathsf{T\text{-}MAX}$ are adjusted to plan a new iteration. To summary, the broad lines of the VLNS heuristic appear on Figure 3.

The efficiency of such a heuristic depends essentially on two points. First, the values given to the parameters K and $\mathsf{T\text{-}MAX}$ are crucial. If K is too large or $\mathsf{T\text{-}MAX}$ too low, the resolution of the restricted ILP may often fail (no integer solution found), which considerably slows down the descent. To prevent that, the idea is to initialize K to a small value and increase it as the iterations go. When one resolution fails, the value of K is stabilized in order to choose the largest number of movable vehicles for a given value of $\mathsf{T\text{-}MAX}$. The second point concerns the

```
ILP-based local search(TIME-LIMIT)
Begin;
    initialize K and T-MAX;
    compute initial solution greedily;
    while TIME-LIMIT is not reached do
        choose K movable vehicles;
        construct ILP restricted to these K movable vehicles;
        run branch-and-bound on ILP during T-MAX seconds;
        update current solution and adjust K and T-MAX;
    end do;
    return solution;
End;
```

Figure 3. The VLNS heuristic for car sequencing problems.

choice of movable vehicles. From our experimentations arises the following property, which is the key to drastically lower the time necessary to obtain an optimal integer solution by branch-and-bound: further are the movable vehicles the ones from the others in the sequence, better is the quality of the continuous relaxation of the restricted ILP. Some elements of polyhedral analysis are detailed in the following section which explains this phenomenon. In fact, a simple and efficient way to proceed is to choose randomly the positions of movable vehicles, and complete with positions where violations appear. Long subsequences of contiguous vehicles are to avoid because in this case, the continuous relaxation of the program is generally poor (very fractional, with cost equal to zero) and a basic branch-and-bound procedure has trouble finding one integer solution; on the other hand, allowing small subsequences of contiguous vehicles (a dozen or so) is not prohibitory and supports the diversification of the search.

3.3. A sufficient condition for polynomial VLNS

In certain cases, the exploration of K-permutation neighborhoods can be done in polynomial time and space. Assume that the K positions selected to perform the K-permutation satisfy the following property, called *one-per-window*: two selected positions j and j' are such that $|j - j'| \geq \mathsf{Q}_{\max}$, where $\mathsf{Q}_{\max}$ is the larger denominator among all the ratio constraints. According to this property, the number of violations caused by the assignment of one of the K vehicles to one of the K positions does not depend on the assignment of the other vehicles, but is fixed. Then, the problem of finding the best K-permutation is reduced to a linear assignment problem (LAP) [31, pp. 285–300] as follows. Build the complete bipartite graph $B = (V, P, E)$ where the set V (resp. P) corresponds to movable vehicles (resp. positions). For any edge $e \in E$ connecting one vehicle $v \in V$ to one position $p \in P$, the weight of the edge e is defined as the number of violations caused by the assignment of v to p. Clearly, a minimum weight perfect matching in this complete bipartite graph (that is, a minimum assignment) corresponds exactly to a best K-permutation.

Since the inequality $\mathsf{K} \leq \mathsf{NPOS}/\mathsf{Q}_{\max}$ is a necessary condition to satisfy the one-per-window property, the value of $\mathsf{Q}_{\max}$ is determining for the application of the previous technique. Small values of $\mathsf{Q}_{\max}$ offer the following remarkable case.

Proposition 3.1. *If $\mathsf{Q}_{\max} = O(1)$, then neighborhoods of size $\Omega(\mathsf{NPOS}!)$ can be explored in $O(\mathsf{NPOS}^3 + \mathsf{NPOS}^2 \cdot \mathsf{NOP})$ time and $O(\mathsf{NPOS}^2 \cdot \log \mathsf{NOP})$ space, which is polynomial in the size of the input data.*

Proof. When $\mathsf{Q}_{\max} = O(1)$, the technique exposed above is applicable to find the best K-permutation on $\Omega(\mathsf{NPOS})$ movable vehicles. Indeed, choosing $\Omega(\mathsf{NPOS})$ positions satisfying the one-per-window property is simply done in $O(\mathsf{NPOS})$ time. Now, $O(\mathsf{NOP} \cdot \mathsf{Q}_{\max})$ operations are necessary to compute the weight of each edge $vp \in E$ of the bipartite graph B. Note that the weight is a positive integer bounded by the value $\mathsf{NOP} \cdot \mathsf{Q}_{\max}$. Hence, the construction of the graph B requires $O(\mathsf{K}^2 \cdot \mathsf{NOP} \cdot \mathsf{Q}_{\max})$ time and $O(\mathsf{K}^2 \cdot \log(\mathsf{NOP} \cdot \mathsf{Q}_{\max}))$ space. Finally, computing a minimum assignment in B can be done $O(\mathsf{K}^3)$ time [31, p. 288]. The desired complexity is obtained by using the relations $\mathsf{Q}_{\max} = O(1)$ and $\mathsf{K} \leq \mathsf{NPOS}$. $\square$

Remark 3.2. By computing first the number of violations for each pair option/position, the construction of the graph B can be done in $O(\mathsf{K} \cdot \mathsf{NOP} \cdot (\mathsf{Q}_{\max} + \mathsf{NCL}))$ time and $O(\mathsf{K} \cdot \mathsf{NCL} \cdot \log(\mathsf{NOP} \cdot \mathsf{Q}_{\max}))$ space. By reasoning on classes versus positions rather than on vehicles versus positions (as done in ILP formulation), the problem can also be viewed as a transportation problem [31, p. 343].

3.3.1. *Polyhedral consequences*

The one-per-window property has some interesting consequences from the polyhedral point of view. Denote by $P_{j_1,\ldots,j_p}$ the continuous version of the ILP restricted to positions $j_1, \ldots, j_p$.

Proposition 3.3. *If the positions $j_1, \ldots, j_p$ of movable vehicles satisfy the one-per-window property, then any basic optimal solution of $P_{j_1,\ldots,j_p}$ is integral.*

This proposition, which confirms our empirical observations, says in other words: if the movable vehicles are further enough the ones from the others in the sequence, then any basic optimal solution of the continuous relaxation of the restricted ILP is integer. Its proof is sketched below.

Proof. First, we show that when positions $j_1, \ldots, j_p$ satisfy the one-per-window property, $P_{j_1,\ldots,j_p}$ can be reduced to another program $P'_{j_1,\ldots,j_p}$ having a totally unimodular matrix and a linear objective function.

Having reintroduced variables $o_{i,j}$ in contraints (4), artificial variables $s_{i,j}$ can be eliminated with the contraints (6), (7), (8) and (9). Now, let us analyse the constraints (4) and (5) after rewriting, that is, for all option i and position j:

$$\begin{cases} v_{i,j} \geq \sum_{f=j-\mathsf{Q}_i+1}^{j} o_{i,f} - \mathsf{P}_i \\ v_{i,j} \geq 0 \end{cases} \tag{13}$$

When the positions $j_1, \ldots, j_p$ satisfy the one-per-window property, at most one variable o_{i,j^*} with $j-\mathsf{Q}_i+1 \leq j^* \leq j$ remains not fixed into the sum $\sum_{f=j-\mathsf{Q}_i+1}^{j} o_{i,f}$. Hence, constraints (13) can be rewritten as

$$\begin{cases} v_{i,j} \geq o_{i,j^*} + \Delta_{i,j} \\ v_{i,j} \geq 0 \end{cases} \tag{14}$$

where the integer constant $\Delta_{i,j}$ equals the number of non-movable vehicles having option i in the window ending at j, minus the numerator P_i. Then, the following observation is crucial. If $\Delta_{i,j} \geq 0$, then the two inequalities of (14) are equivalent to the sole equality $v_{i,j} = o_{i,j^*} + \Delta_{i,j}$ (because $o_{i,j^*} + \Delta_{i,j} \geq 0$ and the $v_{i,j}$ are minimized). On the other hand, if $\Delta_{i,j} \leq -1$, then these ones are equivalent to $v_{i,j} = 0$ (because $o_{i,j^*} + \Delta_{i,j} \leq 0$). Since in both cases constraints (14) are reduced to only one equality, these ones and the variables $v_{i,j}$ can be eliminated by introducing the right member of the corresponding equality into the objective function. Because the $o_{i,j}$ are just some linear combinations of the $cl_{k,j}$, they can be eliminated too with the constraints (3) by introducing the right member into the objective function. After this elimination process, only variables $cl_{k,j}$ with constraints (1) and (2) remain in the program, which is renamed in $P'_{j_1,\ldots,j_p}$.

Now, the matrix induced by constraints (1) and (2), which express all possible sequences of vehicles, is shown to be totally unimodular. This is immediate by using the characterization of Ghouila-Houri [31, p. 76]: a matrix is totally unimodular if and only if each collection of rows can be partitioned into two classes X and Y such that the sum of the rows in X, minus the sum of the rows in Y, is a vector with entries $-1, 0, 1$ only. Here each column of the matrix in question contains exactly two 1s: one in the bloc of constraints (1) and one in the bloc of constraints (2). Thus, for any collection of rows of the matrix, place the rows of (1) in X and the rows of (2) in Y.

Since the objective function of $P'_{j_1,\ldots,j_p}$ remains linear with integer coefficients and all the second members are integers, the total unimodularity of the matrix implies that any basic optimal solution of $P'_{j_1,\ldots,j_p}$ is integral [31, pp. 75–76] and also that any non-integral optimal vector of variables $cl_{k,j}$ can be expressed as a convex combination of integral ones. According to the previous discussion, variables $o_{i,j}$, $s_{i,j}$, $v_{i,j}$ are just integer linear combinations of variables $cl_{k,j}$, for any solution of $P_{j_1,\ldots,j_p}$. Hence, any non-integral optimal solution of $P_{j_1,\ldots,j_p}$ is a convex combination of integral ones. □

3.3.2. *Generalizations*

In fact, Propositions 3.1 and 3.3 are just some corollaries of a more general proposition. Consider the window (i, j) associated to option i and position j and denote by $J^* \subseteq \{j_1, \ldots, j_p\}$ the subset of non-fixed positions between $j - \mathsf{Q}_i + 1$

and j. The constraint for counting violations in window (i, j) can be written as

$$\begin{cases} v_{i,j} \geq \sum_{f \in J^*} o_{i,f} + \Delta_{i,j} \\ v_{i,j} \geq 0 \end{cases} \tag{15}$$

where $\Delta_{i,j}$ equals the number of non-movable vehicles having option i in the window (i, j), minus P_i. If $\Delta_{i,j} \geq 0$, we have $\sum_{f \in J^*} o_{i,f} - \Delta_{i,j} \geq 0$ and then $v_{i,j} = \sum_{f \in J^*} o_{i,f} - \Delta_{i,j}$. On the other hand, if $\Delta_{i,j} \leq -|J^*|$, we have $\sum_{f \in J^*} o_{i,f} - \Delta_{i,j} \leq 0$ and then $v_{i,j} = 0$. Thus, by using the same arguments than in the proofs of Propositions 3.1 and 3.3, we obtain the following result.

Proposition 3.4. *If the positions $j_1, \ldots, j_p$ of movable vehicles are such that for any window (i, j), the value of $\Delta_{i,j}$ satisfies either $\Delta_{i,j} \geq 0$ or $\Delta_{i,j} \leq -|J^*|$, then any basic optimal solution of $P_{j_1,\ldots,j_p}$ is integral. Moreover, such a solution can be combinatorially computed in polynomial time and space by reduction to a linear assignment problem (or a transportation problem).*

Propositions 3.1 and 3.3 are obtained with $|J^*| = 1$. Observe that when the number $|J^*|$ of non-movable vehicles into the window (i, j) grows, the number of values of $\Delta_{i,j}$ for which the conditions are satisfied decreases, and so the chances to satisfy the proposition decreases too. When $|J^*| = \mathsf{Q}_i$, the conditions can not be satisfied any more.

Remark 3.5. Formally, the academic car sequencing problem can be viewed as an assignment problem with a special objective function, according to the following formulation (side windows are omitted for the sake of simplicity):

$$\text{Minimize} \quad \sum_{i=1}^{\mathsf{NOP}} \sum_{k=1}^{\mathsf{NCL}} \left(\mathsf{OP}_{k,i} \times \sum_{j=\mathsf{Q}_i}^{\mathsf{NPOS}} \max\{0, \sum_{f=j-\mathsf{Q}_i+1}^{j} cl_{k,f} - \mathsf{P}_i\} \right)$$

$$\sum_{k=1}^{\mathsf{NPOS}} cl_{k,j} = \mathsf{N}_k \qquad \forall k \in \{1, \ldots, \mathsf{NCL}\}$$

$$\sum_{j=1}^{\mathsf{NCL}} cl_{k,j} = 1 \qquad \forall j \in \{1, \ldots, \mathsf{NPOS}\}$$

$$cl_{k,j} \in \{0, 1\} \qquad \forall k \in \{1, \ldots, \mathsf{NCL}\} \;\; \forall j \in \{1, \ldots, \mathsf{NPOS}\}$$

The objective function is non-linear but convex (because non-decreasing piecewise linear) and the matrix defined by the constraints is totally unimodular. In this context, Proposition 3.4 gives a way to linearize the objective function by fixing variables of the program. Indeed, under conditions of Proposition 3.4, it becomes possible to decide a priori if $\sum_{f=j-\mathsf{Q}_i+1}^{j} cl_{k,f} - \mathsf{P}_i \geq 0$ or not, which allows to remove all max operators from the objective function and implies immediately its linearity.

3.4. Experimental results

Figure 4 reports the results obtained by the VLNS heuristic on CSPlib's benchmarks [11], using a computer with 3 GHz Pentium 4 processor and 1024 Mo of RAM. The VLNS heuristic have been implemented in C ANSI programming language (about 1500 lines of code), with GLPK [17] as integer linear programming library and the Jonker-Volgenant's code [14] for solving linear assignment problems.

Instances	Average cost	Average time
4-72	0.0	21.04
6-76	6.0	0.00
10-93	3.0	123.13
16-81	0.0	32.19
19-71	2.0	8.59
21-90	2.0	9.65
26-82	0.0	3.52
36-92	2.0	14.13
41-66	0.0	0.11
200-01	0.1	240.31
200-02	2.4	261.49
200-03	5.8	253.45
200-04	7.2	281.69
200-05	6.0	38.79
200-06	6.0	15.84
200-07	0.0	18.98
200-08	8.0	40.84
200-09	10.0	59.43
200-10	19.0	232.75

Instances	Average cost	Average time
300-01	1.2	155.65
300-02	12.0	34.16
300-03	13.0	75.21
300-04	8.8	321.81
300-05	33.1	451.31
300-06	3.1	341.05
300-07	0.0	192.82
300-08	8.0	34.81
300-09	8.0	109.28
300-10	21.4	441.05
400-01	1.6	315.99
400-02	16.3	403.03
400-03	12.0	38.21
400-04	20.1	85.91
400-05	0.0	100.62
400-06	0.0	187.98
400-07	4.6	287.43
400-08	4.1	281.48
400-09	8.3	391.85
400-10	0.0	45.47

Figure 4. Results of the VLNS heuristic on CSPlib's instances.

Initial solutions are computed by a greedy algorithm inspired from the DSU heuristic of Gottlieb et al. [12] (see also [8]). Then, two procedures are employed to select mobile positions and solve the corresponding subproblem. The first one selects random positions or random small blocs of consecutive positions. In this case, the value of the parameter K varies from 10 to 50 vehicles all along the search, for a value of T-MAX limited to 1 second. (For a greater value of T-MAX like 10 seconds as used in [8], the value of K can be increased to select near from the two thirds of the vehicles, but this tends to slow the descent). The restricted ILP is solved optimally by using the branch-and-bound procedure of [17] (without particular tuning); the computation of the continuous relaxation of the ILP is facilitated by exhibiting a realizable basis from the current solution. The second procedure selects mobile positions in order to satisfy the one-per-window property. In this case, around $\mathsf{NPOS}/\mathsf{Q}_{\max}$ positions are selected and the best K-permutation is determined by using the Jonker-Volgenant's algorithm [14] for

LAP, very efficient in practice. The use of this second procedure, baptized LAP-based VLNS, considerably speeds up the descent in comparison with the sole use of ILP.

The results of Figure 4 are obtained with 10 trials per instance, with a running-time limited to 10 minutes for each trial. The column “average cost” shows the average number of violations found on the 10 trials and the column “average time” gives the average running-time in seconds to find this average cost. For classical instances (first table top on the left), the number of violations found is always the best known according to the results reported in CSPLib [11] (see also [12, 13, 29]). The three other tables report the results obtained on the three sets of instances recently proposed by Gagne et al. [13], which involve more vehicles (respectively 200, 300 and 400 cars). For 24 instances on 39, VLNS always finds the best known number of violations on the 10 trials. For 23 (resp. 36) instances, the average cost obtained by VLNS is strictly better (resp. better or equal) than the one obtained by the heuristic of Gagne et al. [13], which mixes ant colony optimization and basic local search. (Within sight of the results presented in [19], pure ant colony optimization is not competitive with VLNS at this point). On the other hand, the VLNS heuristic consumes generally much more time than the other efficient approaches [12, 13, 19] to obtain these results. Note that the results of VLNS on CSPLib's benchmarks 60/65/70/75/80/85/90 are not mentioned because all the instances of these benchmarks are solved to optimality (0 violation) in less than 0.01 sec. (The interested reader is referred to Figure 11 in Appendix for additional experimental results concerning the greedy algorithm employed).

The above results attest the practical effectiveness and robustness of the VLNS heuristic. Note that contrary to ant colony optimization [12, 13, 19] which is a constructive approach, VLNS seems to converge towards a global optimum, even if this convergence is very slow.

4. Hybridization with very fast local search

In this section, very large-scale neighborhood search is hybridized with the best known approach for solving car sequencing problems: very fast local search (VFLS). This approach, quite classical in essence [2, pp. 372–375], was first experimented by Gottlieb et al. [12, 28] and then revisited by the authors [6–8] to win the ROADEF'2005 Challenge, where a real-life car sequencing problem was posed as subject by the car manufacturer RENAULT. Having reminded the core ingredients of the VFLS approach and its results on CSPLib's benchmarks, two hybridizations with LAP-based VLNS are described which show very good computational results, in particular on the hardest instances.

4.1. Very fast local search

Very fast local search is a *first-improvement descent* heuristic based on very fast explorations of small neighborhoods. More precisely, the algorithm applies at each iteration one transformation to the current sequence which modifies it only very locally. If the transformation does not increase the cost of the current solution, this one is really performed. Four basic transformations are used here: swap, forward insertion, backward insertion and reflection.

A swap simply consists in exchanging the positions of two vehicles in the sequence. A forward insertion localized on a portion $v_j, x, \dots, y, v_{j'}$ of the sequence consists in extracting $v_{j'}$, shifting the vehicles $v_j, x, \dots, y$ to the right, and reinserting $v_{j'}$ at the position which remains unfilled (the former position of v_j); after the transformation, the initial portion contains in order the vehicles $v_{j'}, v_j, x, \dots, y$. A backward insertion is defined in a symmetric way, by extracting v_j instead of $v_{j'}$. A reflection localized on a portion $v_j, x, \dots, y, v_{j'}$ of the sequence consists in reversing this portion, so as to contain in order the vehicles $v_{j'}, y, \dots, x, v_j$. The neighborhood defined by these four transformations is only of size $O(\mathsf{NPOS}^2)$. Moreover, the selection of the neighbor is guided by no sophisticated rule: the first neighbor lowering or even equaling the cost of the current solution is retained for a new search. The acceptance of transformations which do not improve the cost is crucial: coupled to a fast evaluation procedure, this is a way to diversify widely the search and then to avoid local optima during the descent. Note that the transformation "random shuffle", defined in [12, 28] to diversify the search, is not employed here because useless according to our computational experimentations.

Transformations	Variants	%
swap	generic	69.6
	consecutive	3.2
	similar	2.5
forward insertion	generic	3.2
	denominator	3.8
backward insertion	generic	3.2
	denominator	3.8
reflection	generic	6.9
	denominator	3.8

FIGURE 5. The composition of VFLS.

The generic way for choosing the positions where transformations are applied is to pick the positions j and j' randomly. But to increase the probability of success of one transformation, cleverer choices are necessary. Thus, the following variants are defined. The choice "consecutive" consists in picking a position j randomly and set $j' = j + 1$. The choice "similar" consists in picking the two positions j and j' such that the corresponding vehicles share some options. These two special choices, first introduced by Gottlieb et al. [12, 28], are used for swaps. The choice "denominator", introduced by the authors [7, 8] and used for insertions and reflections, consists in picking a position j and an option i randomly and set $j' = j + \mathsf{Q}_i$. The percentage of attempted transformations for each variant is detailed on Figure 5. For example, the number of attempted generic swaps represents 69.6 % of the total number of attempted transformations during the search.

The efficiency of this approach relies on the huge number of attempted transformations, ensuring a large diversification of the search. Fortunately, the singular structure of the four transformations reveals some invariants which are exploitable using special data structures to quickly evaluate their impact on the cost of the current solution. Thus, whereas a poor implementation attempts few hundred thousands of transformations per minute on a basic computer and converges slowly, an advanced one (as described in [7,8]) allows several millions of attempted transformations per minute and converges quickly (note that, in comparison, the VLNS heuristic performs only a few thousands of iterations per minute). The reader is referred to [8] for more details on how to speed up the evaluation of transformations.

4.1.1. *Experimental results*

The VFLS heuristic have been implemented in C ANSI programming language (about 1500 lines of code). Figure 6 shows the results obtained by VFLS under the same conditions than VLNS: 10 trials for each instance with a running-time limited to 10 minutes per trial, using a computer with 3 GHz Pentium 4 processor and 1024 Mo of RAM. Initial solutions are computed by the same greedy algorithm than for VLNS, inspired from the DSU heuristic of Gottlieb et al. [8, 12]. (The interested reader is referred to Figure 11 in Appendix for additional experimental results concerning the greedy algorithm employed).

Instances	Average cost	Average time
4-72	0.0	1.52
6-76	6.0	0.00
10-93	3.0	3.36
16-81	0.0	0.56
19-71	2.0	0.51
21-90	2.0	0.11
26-82	0.0	0.36
36-92	2.0	0.05
41-66	0.0	0.03
200-01	0.0	3.40
200-02	2.0	1.69
200-03	$\boxed{3.0}$	93.32
200-04	7.0	2.74
200-05	6.0	1.89
200-06	6.0	0.52
200-07	0.0	0.57
200-08	8.0	0.18
200-09	10.0	0.32
200-10	19.0	0.18

Instances	Average cost	Average time
300-01	0.0	20.15
300-02	12.0	0.57
300-03	13.0	0.68
300-04	7.0	1.46
300-05	$\boxed{28.0}$	141.61
300-06	2.0	13.93
300-07	0.0	4.26
300-08	8.0	0.53
300-09	7.0	1.16
300-10	21.0	3.13
400-01	1.0	55.55
400-02	$\boxed{15.0}$	16.59
400-03	12.0	0.53
400-04	19.0	0.47
400-05	0.0	1.01
400-06	0.0	1.57
400-07	4.0	8.26
400-08	4.0	8.31
400-09	5.0	3.40
400-10	0.0	6.16

FIGURE 6. Results of the VFLS heuristic on CSPLIB's instances.

The results obtained by the VFLS heuristic are impressive: it always finds the best known number of violations on the 10 trials, in less than 1 sec. on average for

17 instances on 39 and in less than 10 sec. on average for 33 instances. (Results on CSPLIB's benchmarks 60/65/70/75/80/85/90 are not mentioned because all instances of these benchmarks are solved to zero violation in less than 0.01 sec.) Note that the cost found for instance 400-03 is 12 and not 9 as in [13], because here we count the number of violations and not the number of violated windows (see Figure 10 in Appendix for more details). In addition, three results of Gagne et al. [13] are improved: 3 for instance 200-03 (instead of 4), 28 for instance 300-05 (instead of 29), 15 for instance 400-02 (instead of 16). In conclusion, VFLS dramatically outperforms VLNS, just as any other algorithm from the literature (see the results presented in [12,13,19,20,23,24,29]).

4.2. Hybridization

We have shown that Proposition 3.4 is useful to speed up the convergence of the VLNS heuristic. But may it be useful to speed up the convergence of VFLS heuristic? The answer is yes, particularly for hardest instances, that is, those which are solved in more than a few seconds by VFLS (like instances 200-03, 300-05, 400-01).

The hybridization is done by adding a VLNS transformation to the four basic transformations employed in VFLS. This transformation is drawn from the LAP-based VLNS procedure used in the VLNS heuristic: select around $\mathsf{K} = \mathsf{NPOS}/\mathsf{Q}_{\max}$ positions which respects the one-per-window property (Proposition 3.1) and find a best K-permutation by using the Jonker-Volgenant's algorithm [14] for LAP. Such a transformation must be used parsimoniously because it is very time-consuming in comparison with the four basic transformations. Our experimentations on CSPLIB's benchmarks lead us to fix the percentage of attempted VLNS transformations to 0.2 %; more generally, this percentage must be chosen in such a way that the time consumed by VLNS transformations remains balanced with the time consumed by swap transformations.

Two variants of hybridization have been implemented. In the first one, the LAP-based VLNS procedure is designed to find solutions with better or equal cost (which helps to diversification). In the second one, the LAP-based VLNS procedure is specialized to find solutions with strictly better cost (which does not help to diversification, but is slightly faster). The implementation of each variant is detailed below:

- HYBRID A. In this variant, solutions of equal cost are accepted. Our observations reveal that in this case, the restricted subproblem admits generally many solutions of equal cost, which implies a high success rate for the VLNS transformation. To ensure diversification, the weights of the edges of the bipartite graph $B = (V, P)$ (defined in section 3.3) are perturbed: the edges appearing in the current solution are penalized by increasing their weight of $\epsilon < 1/\mathsf{NPOS}$. In this way, among several optimal assignments, the one having the least number of edges in common with the current solution is chosen, which contributes to diversification.

- HYBRID B. Here only solutions of better cost are accepted, which implies a low success rate for the VLNS transformation. To reduce the running-time of the transformation, a cheaper test is performed before running the Jonker-Volgenant's algorithm [14] for LAP. This test consists in detecting a negative cycle in the residual graph computed from current solution, because according to network flow theory [34, p. 101], a strictly better assignment exists if and only if the residual graph contains a negative cycle (see [34, pp. 97–123] for more details). The negative cycle detection is done via the Bellman-Ford algorithm for shortest path problems with negative distances [34, pp. 91–94]. Since in our case the residual graph is unlikely to contain negative cycles, pass counting [34, p. 93] strongly reduces the practical running-time of the Bellman-Ford algorithm (note in addition that edges having a weight better than or equal to the weight of the current assignment are removed from the graph). Although not contributing to diversification, this advanced implementation makes faster the VLNS transformation in practice.

4.2.1. *Experimental results*

HYBRID A and HYBRID B heuristics have been implemented in C ANSI programming language (about 2000 lines of code each one). To make valuable comparisons with pure VFLS, experimentations have been done on the 10 hardest CSPLIB's instances (according to the results obtained by VFLS on Figure 6). Moreover, these tests are done under sharper conditions than previously: 100 trials for each instance with a running-time limited to 10 minutes per trial, always using a computer with 3 GHz Pentium 4 processor and 1024 Mo of RAM. Figure 7 shows the results obtained by VFLS in these conditions. Here the column "Avg. iterations" is added, which gives the average number of iterations done by the algorithm to obtain its best solution on 10 minutes. Surprisingly, a solution with 27 violations is found for instance 300-05, which still improves the best known upper bound for this instance. (Improved solutions for CSPLIB's instance 200-03, 300-05 and 400-02 are presented on Figure 12 in Appendix.)

Instances	VFLS		
	Avg. cost	Avg. time	Avg. iterations
200-03	3.02	161.74	151 413 326
300-01	0.00	24.13	23 600 151
300-05	27.81	235.79	206 131 189
300-06	2.00	11.93	11 140 807
300-07	0.00	3.47	3 231 827
400-01	1.00	41.93	40 553 796
400-02	15.00	14.61	12 637 527
400-07	4.00	9.08	8 550 350
400-08	4.00	5.36	4 953 331
400-10	0.00	6.02	5 809 615

FIGURE 7. Sharper results for VFLS on the hardest CSPLIB's instances.

Figure 8 shows the results obtained by the two hybrid algorithms under these sharper conditions. In order to facilitate the comparison with pure VFLS, the columns "% impr." give the percentage of improvement compared to VFLS for the concerned data. In both cases, the convergence of the descent is speeded up for almost all instances. On average, HYBRID A (resp. HYBRID B) heuristic converges 20.6 % (resp. 15.5 %) faster than VFLS heuristic, while the number of iterations is reduced of 66.7 % (resp. 58.3 %). Such results demonstrate clearly that the addition of VLNS transformations is beneficial: despite a small number of attempts (only 0.2 %), they substantially reduce the number of iterations as well as the time of the descent (nevertheless, the reduction of the number of iterations is compensated by the time spent into VLNS transformations). Note that the results obtained by HYBRID A are globally better than the ones of HYBRID B, which suggests that supporting diversification through VLNS transformations helps.

Instances	HYBRID A				
	Avg. cost	Avg. time	% impr.	Avg. iterations	% impr.
200-03	3.02	153.24	5.3	27 381 333	81.9
300-01	0.00	20.38	15.5	7 362 280	68.8
300-05	27.79	211.88	10.1	71 499 964	65.3
300-06	2.00	9.28	22.2	4 337 777	61.0
300-07	0.00	3.10	10.7	1 389 923	57.0
400-01	1.00	33.64	19.8	13 145 998	67.6
400-02	15.00	11.67	20.1	5 334 379	57.8
400-07	4.00	4.99	45.0	2 585 487	69.8
400-08	4.00	4.50	16.0	1 591 986	67.9
400-10	0.00	3.51	41.7	1 733 348	70.2

Instances	HYBRID B				
	Avg. cost	Avg. time	% impr.	Avg. iterations	% impr.
200-03	3.01	153.07	5.4	38 076 073	74.9
300-01	0.00	18.91	21.6	8 617 242	63.5
300-05	27.85	231.95	1.6	74 580 841	63.8
300-06	2.00	10.75	9.9	5 291 699	52.5
300-07	0.00	3.89	-12.1	2 073 856	35.8
400-01	1.00	36.04	14.0	15 737 308	61.2
400-02	15.00	11.90	20.1	6 181 598	51.1
400-07	4.00	5.37	40.9	3 206 521	62.5
400-08	4.00	4.50	16.0	2 174 497	56.1
400-10	0.00	3.77	37.4	2 235 781	61.5

FIGURE 8. Results of hybrid algorithms on the hardest CSPLIB's instances.

The contribution of VLNS transformations is still illustrated on Figure 9, which details the average number of transformations improving (strictly) the cost of the solution during the descent, for instances 200-03, 300-05 and 400-10. When pure VFLS is employed, improving transformations are always classified in the following order (for the 10 hardest CSPLIB's instances): generic swaps (about 50 %), generic reflections (about 20 %), consecutive swaps (about 12 %), denominator reflections (about 10 %), similar swaps (less than 5 %), and at last insertions. (Observe that the percentage of improving transformations is not correlated with the percentage of attempted transformations given on Figure 5; in particular, consecutive swaps and reflections have a high success rate.)

200-03		VFLS		HYBRID A		HYBRID B	
Transformations	Variants	Nb.	%	Nb.	%	Nb.	%
swap	generic	13.75	56.1	10.56	43.6	9.83	41.0
	consecutive	2.48	10.1	2.06	8.5	2.06	8.6
	similar	1.02	4.2	0.85	3.6	0.69	2.9
forward insertion	generic	0.11	0.4	0.08	0.3	0.10	0.4
	denominator	0.19	0.8	0.10	0.4	0.12	0.5
backward insertion	generic	0.09	0.4	0.08	0.3	0.09	0.4
	denominator	0.25	1.0	0.12	0.5	0.15	0.6
reflection	generic	5.07	20.7	4.13	17.0	4.17	17.4
	denominator	1.55	6.3	1.48	6.1	1.30	5.4
VLNS	–	–	–	4.78	$\boxed{19.7}$	5.48	$\boxed{22.8}$
Total		24.51		24.24		23.99	

300-05		VFLS		HYBRID A		HYBRID B	
Transformations	Variants	Nb.	%	Nb.	%	Nb.	%
swap	generic	20.37	47.8	13.48	31.3	13.65	31.3
	consecutive	5.59	13.1	4.68	10.4	4.10	9.4
	similar	1.83	4.3	0.96	2.2	0.94	2.2
forward insertion	generic	0.13	0.3	0.24	0.6	0.25	0.6
	denominator	0.55	1.3	0.47	1.1	0.38	0.9
backward insertion	generic	0.30	0.7	0.22	0.5	0.32	0.7
	denominator	0.37	0.9	0.46	1.1	0.37	0.8
reflection	generic	10.04	23.6	7.81	18.2	8.54	19.6
	denominator	3.45	8.1	3.09	7.2	2.81	6.4
VLNS	–	–	–	11.79	$\boxed{27.4}$	12.27	$\boxed{28.1}$
Total		42.63		43.00		43.63	

400-10		VFLS		HYBRID A		HYBRID B	
Transformations	Variants	Nb.	%	Nb.	%	Nb.	%
swap	generic	6.46	52.6	2.62	20.9	2.34	19.0
	consecutive	1.38	11.2	0.87	6.9	0.62	5.0
	similar	0.36	3.0	0.22	1.9	0.12	1.0
forward insertion	generic	0.05	0.4	0.06	0.5	0.13	1.1
	denominator	0.24	2.0	0.14	1.1	0.08	0.7
backward insertion	generic	0.15	1.2	0.09	0.7	0.08	0.7
	denominator	0.21	1.7	0.08	0.6	0.20	1.6
reflection	generic	2.12	17.2	1.02	8.1	1.13	9.2
	denominator	1.32	10.7	0.90	7.2	0.50	4.1
VLNS	–	–	–	6.53	$\boxed{52.1}$	7.09	$\boxed{57.7}$
Total		12.29		12.53		12.29	

FIGURE 9. Details of the descent on CSPLIB's instances 200-03, 300-05 and 400-10.

When HYBRID A or HYBRID B is employed, VLNS becomes the first improving transformation or the second behind generic swaps (for instances 200-03 and 300-05 only). The percentage of improving VLNS transformations is the lowest for instances 200-03 and 300-05 (top of Figure 9) and the highest for instance 400-10 (bottom of Figure 9); the average percentage on the 10 hardest CSPLIB instances is about 35 % with HYBRID A and about 38 % with HYBRID B. Within sight of the very small number of attempts (only 0.2 %), such statistics demonstrate that in both cases, the VLNS transformation has a very high success rate compared to the four basic transformations and contributes significantly to the progression of the search.

5. Conclusion

The experimental results obtained by pure very large-scale neighborhood search (VLNS) show that this approach is effective. The VLNS algorithm is robust (not requiring special adjustments for each instance) and converge towards good solutions (or even optimal ones). But its convergence is slow, and in particular slower than very fast local search (VFLS) which, surprisingly, never meets local optimum in practice. As already mentioned in [8], the behavior of VFLS calls a deep theoretical study, in particular to answer to the following question: does VFLS always converge towards a global optimal solution? And if counterexamples exist, are there some classes of instances for which this is the case?

On the other hand, the addition of VLNS transformations based on linear assignment (LAP-based VLNS) into the VFLS heuristic, which can be viewed as a hybridization VFLS/VLNS, is shown to improve significantly the convergence of pure VFLS, by reducing the number of iterations and the time of the descent. Unfortunately, the reduction of the number of iterations is partially compensated by the time spent into VLNS transformations. Thus, speeding up LAP-based VLNS transformations could drastically reduce the time of the descent. Note that adding LAP-based VLNS transformations to VFLS should be still more profitable for real-life car sequencing problems where vehicles can be assigned to a subset of positions in the sequence, and not to all positions (as encountered in many plants of the car manufacturer RENAULT [15]).

An other research axis, in the context of exact approaches, is to dedicate a general branch-and-bound procedure for car sequencing problems, which integrates the ideas and results developed through this paper into an efficient local branching scheme (see [9] for more details on local branching). Indeed, the proof of unsatisfiability for instance 21-90 shows that the branch-and-bound procedure can be specialized to be more efficient, in particular by branching first on variables $o_{i,j}$ corresponding to the most constraint ratios. Then, the choose and the resolution of subproblems for bounding could be guided to satisfy the conditions of Proposition 3.4.

We are grateful to Prof. Gérard Cornuéjols (LIF, Marseille) for discussion of the theoretical results of section 3.3 and to Prof. Marc Gravel (Université du Québec, Chicoutimi) and Dr. Christine Solnon (LIRIS, Lyon) for communicating to us their solutions to CSPLib's instance 400-03, which appears on Figure 10. We also thank Prof. Van-Dat Cung (GILCO, Grenoble), Mr Youssef Khacheni (RENAULT/DL, Guyancourt), Mr Alain Nguyen (RENAULT/DTSI, Le Plessis Robinson) for their encouragements and the many interesting discussions that we had about the car sequencing problem. Finally, we express our gratitude to the three anonymous referees for their suggestions, which have resulted in an improved paper.

References

[1] R.K. Ahuja, Ö. Ergun, J.B. Orlin and A.P. Punnen (2002). A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics* 123, pp. 75–102.

[2] E. Aarts and J.K. Lenstra (1997). *Local Search in Combinatorial Optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons, Chichester, UK.

[3] V-.D. Cung and A. Nguyen (2003). Challenge ROADEF'2005. `http://www.prism.uvsq.fr/~vdc/ROADEF/CHALLENGES/2005/`

[4] V-.D. Cung and A. Nguyen (2005). Le problème du Car Sequencing RENAULT et le Challenge ROADEF'2005. In *Actes des JFPC 2005, les 1ères Journées Francophones de Programmation par Contraintes* (C. Solnon, ed), pp. 3–10. Lens, France.

[5] B. Estellon, F. Gardi and K. Nouioua (2005). Ordonnancement de véhicules dans les usines RENAULT: une approche par recherche locale à voisinage large. In *Actes de ROADEF 2005, le 6ème Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision* (J.-C. Billaut, ed), pp. 33–34. Tours, France.

[6] B. Estellon, F. Gardi and K. Nouioua (2005). Ordonnancement de véhicules dans les usines RENAULT: une approche par recherche locale à voisinage réduit. In *Actes de ROADEF 2005, le 6ème Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision* (J.-C. Billaut, ed), pp. 35–36. Tours, France.

[7] B. Estellon, F. Gardi and K. Nouioua (2005). Ordonnancement de véhicules : une approche par recherche locale à voisinage large. In *Actes des JFPC 2005, les 1ères Journées Francophones de Programmation par Contraintes* (C. Solnon, ed), pp. 21–28. Lens, France.

[8] B. Estellon, F. Gardi and K. Nouioua (2006). Two local search approaches for solving real-life car sequencing problems. (submitted to *European Journal of Operational Research*)

[9] M. Fischetti and A. Lodi (2003). Local branching. *Mathematical Programming Series B* 98, pp. 23–47.

[10] I.P. Gent (1998). Two results on car sequencing problem. APES Research Report 02-1998. Department of Computer Science, University of Strathclyde, Glasgow, Scotland, UK.

[11] I.P. Gent and T. Walsh (1999). CSPLIB: a benchmark library for constraints. APES Research Report 09-1999. Department of Computer Science, University of Strathclyde, Glasgow, Scotland, UK. `http://www.csplib.org/`

[12] J. Gottlieb, M. Puchta and C. Solnon (2003). A study of greedy, local search and ant colony optimization approaches for car sequencing problems. In *Proceedings of EvoWorkshops 2003 on Applications of Evolutionary Computing* (G.R. Raidl et al., eds), *LNCS* 2611, pp. 246–257. Springer-Verlag, Berlin, Germany.

[13] M. Gravel, C. Gagné and W.L. Price (2005). Review and comparison of three methods for the solution of the car sequencing problem. *Journal of the Operational Research Society* 56, pp. 1287–1295.

[14] R. Jonker and A. Volgenant (1987). A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing* 38, pp. 325–340. `http://www.magiclogic.com/assignment.html`

[15] Y. Khacheni and A. Nguyen (2006). Personnal communication.

[16] T. Kis (2004). On the complexity of the car sequencing problem. *Operations Research Letters* 32, pp. 331–335.

[17] A. Makhorin (2006). GLPK library (GNU Linear Programming Kit, version 4.9). `http://www.gnu.org/software/glpk/`

[18] T. Mautor and P. Michelon (1997). MIMAUSA: a new hybrid method combining exact solution and local search. In *MIC 97, the 2nd Metaheuristics International Conference*. Sophia-Antipolis, France.

[19] S. Morin, C. Gagné, M. Gravel and W.L. Price (2006). Trace de phéromone spécialisée dans un algorithme de fourmi pour le problème de "car sequencing". In *Actes de MOSIM 2006, la*

6ème Conférence Francophone de Modélisation et Simulation (M. Gournand and F. Riane, eds), pp. 22–29. Rabat, Maroc.

[20] L. Michel and P. Van Hentenryck (2002). A constraint-based architecture for local search. In *Proceedings of OOPSLA 2002, the 2002 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications*, *SIGPLAN Notices* 37(11), pp. 83–100. ACM Press, New York, NY.

[21] M. Palpant (2005). Recherche exacte et approchée en optimisation combinatoire: schémas d'intégration et applications. PhD Thesis. Laboratoire d'Informatique d'Avignon, Université d'Avignon et des Pays de Vaucluse, Avignon, France.

[22] M. Palpant, C. Artigues and P. Michelon (2004). LSSPER: solving the ressource-constrained project scheduling problem with large neighborhood search. *Annals of Operations Research* 31(1), pp. 237–257.

[23] L. Perron and P. Shaw (2004). Combining forces to solve the car sequencing problem. In *Proceedings of CPAIOR 2004, the 1st International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems* (J.-C. Régin and M. Rueher, eds), *LNCS* 3011, pp. 225–239. Springer-Verlag, Berlin, Germany.

[24] L. Perron, P. Shaw and V. Furnon (2004). Propagation guided large neighborhood search. In *Proceedings of CP 2004, the 10th International Conference on Principles and Practice of Constraint Programming* (M. Wallace, ed), *LNCS* 3258, pp. 468–481. Springer-Verlag, Berlin, Germany.

[25] M. Prandtstetter and G.R. Raidl (2005). A variable neighborhood search approach for solving the car sequencing problem. In *Proceedings of the 18th Mini Euro Conference on Variable Neighborhood Search.* (P. Hansen, N. Mladenovic, J.A. Moreno Pérez, J.M. Moreno Vega and B. Melián Batista, eds). Tenerife, Spain.

[26] M. Prandtstetter and G.R. Raidl (2005). An integer linear programming approach and a hybrid variable neighborhood search for the car sequencing problem. Technical Report TR-186-1-05-01. Institut for Computer Graphics and Algorithms, Vienna University of Technology, Vienna, Austria.

[27] M. Prandtstetter (2006). Personnal communication.

[28] M. Puchta and J. Gottlieb (2002). Solving car sequencing problems by local optimization. In *Proceedings of EvoWorkshops 2002 on Applications of Evolutionary Computing* (S. Cagnoni et al., eds), *LNCS* 2279, pp. 132–142. Springer-Verlag, Berlin, Germany.

[29] J.-C. Régin and J.-F. Puget (1997). A filtering algorithm for global sequencing constraints. In *Proceedings of CP 97, the 3rd International Conference on Principles and Practice of Constraint Programming* (G. Smolka, ed), *LNCS* 1330, pp. 32–46. Springer-Verlag, Berlin, Germany.

[30] C. Ribeiro, D. Aloise, T. Noronha, C. Rocha and S. Urrutia (2005). A hybrid heuristic for a real-life car sequencing problem with multiple requirements. In *Proceedings of the 18th Mini Euro Conference on Variable Neighborhood Search.* (P. Hansen, N. Mladenovic, J.A. Moreno Pérez, J.M. Moreno Vega and B. Melián Batista, eds). Tenerife, Spain.

[31] A. Schrijver (2003). *Combinatorial Optimization: Polyhedra and Efficiency.* Algorithms and Combinatorics 24, Vol. A, Springer-Verlag, Berlin, Germany.

[32] C. Solnon (2000). Solving permutation contraint satisfaction problems with artificial ants. In *Proceedings of ECAI 2000, the 14th European Conference on Artificial Intelligence* (H. Werner, ed), pp. 118–122. IOS Press, Amsterdam, The Netherlands.

[33] C. Solnon (2006). Des fourmis pour le problème de l'ordonnancement de voitures. In *Actes des JFPC 2006, les 2èmes Journées Francophones de Programmation par Contraintes* (L. Henocque, ed), pp. 305–316. Nîmes, France.

[34] R.E. Tarjan (1983). *Data Structures and Network Algorithms.* CBMS-NSF Regional Conference Series in Applied Mathematics 44, SIAM Publications, Philadelphie, PA.

Appendix

15	08	18	04	18	07	14	06	17	03
17	07	16	06	11	03	19	06	11	10
17	00	21	06	11	09	17	02	20	01
17	03	18	06	14	06	17	04	06	17
03	17	07	14	06	17	03	12	06	14
06	11	10	11	08	14	06	11	09	11
07	20	01	17	09	13	06	09	01	17
03	19	06	14	06	11	08	20	01	17
09	13	06	16	06	17	00	21	06	11
09	11	08	14	06	17	03	18	06	11
10	17	00	20	06	11	09	18	01	17
09	11	07	10	11	06	14	02	11	09
17	01	22	06	11	09	06	12	10	17
01	20	07	11	10	17	01	20	06	12
10	17	01	20	06	12	10	17	01	20
06	12	10	11	06	21	00	17	05	17
06	14	07	06	16	06	17	03	19	06
14	06	17	04	17	06	03	17	08	14
06	17	05	18	06	16	06	11	09	13
06	17	03	01	18	05	11	06	14	08
11	05	17	06	00	21	06	11	09	18
01	21	06	11	09	17	00	21	06	11
10	17	02	20	06	11	10	17	00	20
06	01	20	06	12	09	11	06	14	07
17	05	17	06	14	07	11	09	17	01
21	07	11	09	17	01	20	07	01	20
06	14	08	17	03	11	06	21	00	17
10	11	06	10	12	06	20	01	17	10
12	06	14	06	11	10	18	01	20	06
11	09	07	11	10	11	06	21	00	17
09	11	06	19	05	17	06	14	08	17
03	17	06	15	06	17	05	17	00	21
06	11	09	18	01	21	06	11	09	18
01	21	06	11	04	17	06	14	06	07
14	06	17	03	18	01	21	06	11	07
21	01	17	09	12	06	21	01	17	03
18	01	21	06	11	09	12	06	21	01
17	03	18	06	16	06	17	05	18	06
16	06	07	16	06	11	10	12	01	17
09	11	07	20	01	17	09	12	08	22

22	06	11	10	17	02	20	06	11	10
18	01	20	06	11	04	17	06	16	06
12	10	11	06	21	00	17	10	11	06
12	10	17	01	20	07	11	10	11	06
21	00	17	10	11	06	21	00	17	05
17	01	18	09	11	06	21	00	17	09
11	06	22	01	06	14	06	19	03	17
01	20	07	11	10	11	06	21	00	17
09	11	06	21	00	17	09	11	06	21
00	17	10	11	06	21	00	11	06	21
01	18	09	11	06	21	00	17	10	11
06	21	00	17	05	17	06	14	08	17
03	11	06	21	02	17	09	11	06	21
01	18	09	11	06	21	02	17	09	11
01	18	03	17	06	14	08	17	01	20
06	13	09	17	01	20	08	11	09	17
01	18	09	11	06	21	01	18	09	11
06	21	01	18	09	11	06	21	01	18
09	11	06	20	01	18	05	17	06	16
07	11	10	17	01	12	09	17	01	20
08	11	09	17	01	20	08	11	09	17
01	20	01	18	09	11	06	14	08	17
05	06	17	04	17	06	14	06	18	05
17	06	16	06	18	03	17	06	14	07
17	05	17	06	14	07	17	05	17	06
14	08	17	05	17	06	12	09	11	06
14	08	17	03	17	06	14	08	17	03
17	06	15	06	06	14	06	18	03	11
06	14	07	17	03	17	06	14	06	13
09	17	01	20	01	18	09	11	06	14
06	12	09	11	06	14	06	12	09	17
01	20	07	11	10	17	01	20	07	11
10	17	01	20	07	11	10	17	01	20
07	11	10	17	01	20	07	11	09	17
03	19	06	14	06	06	14	06	12	10
11	06	14	06	19	03	17	06	14	06
12	10	06	11	03	19	06	16	06	06
16	07	06	16	06	06	12	03	17	06
16	07	17	03	06	06	13	03	06	06
03	12	07	15	07	12	04	12	07	04

22	06	11	10	17	02	20	06	01	20
08	11	06	20	01	19	09	11	06	14
08	11	09	17	01	13	03	11	06	14
07	06	14	06	17	04	17	06	16	06
12	10	11	06	20	00	21	06	11	09
18	01	20	06	14	08	09	11	06	21
00	17	06	16	06	19	03	17	06	14
07	17	01	21	06	12	09	11	06	03
18	01	20	06	11	07	21	01	17	09
12	06	21	01	17	04	17	06	16	06
18	03	17	06	14	07	17	05	17	06
12	10	11	06	16	07	06	14	06	17
04	17	06	14	06	12	10	11	06	16
07	17	03	17	06	13	10	17	03	17
08	14	06	11	09	13	06	20	01	17
00	21	06	11	09	18	01	17	05	17
07	14	06	17	05	18	06	16	06	19
03	17	06	14	07	17	05	17	09	11
07	20	01	17	10	12	06	17	03	17
08	14	06	17	05	18	06	11	10	17
00	20	06	11	10	12	03	17	06	16
00	17	09	11	06	15	06	11	10	06
12	10	06	11	10	18	01	20	06	11
10	07	11	09	17	01	18	05	11	06
20	02	17	09	11	08	05	17	06	03
18	09	11	10	17	00	21	06	11	10
18	01	17	09	11	07	21	01	11	09
18	05	17	06	14	06	18	03	17	06
14	08	17	03	17	01	22	06	11	03
11	07	20	01	17	03	18	06	14	06
06	12	09	11	06	21	02	17	09	11
06	20	00	17	01	21	06	12	09	17
01	21	07	14	06	06	14	08	17	03
17	01	19	09	11	06	09	12	06	21
01	17	04	17	06	14	01	18	09	11
01	20	00	17	01	21	06	12	09	17
01	21	07	11	06	21	01	07	14	06
11	10	08	11	06	20	01	18	09	11
06	21	00	17	10	11	06	15	06	11
09	17	00	21	06	11	10	11	07	20

400-03	W	W'	V	V'
Gravel's solution	$\boxed{9}$	15	19	30
Solnon's solution	$\boxed{9}$	12	27	33
VFLS solution	12	12	12	$\boxed{12}$

Figure 10. Three solutions to CSPlib's instance 400-03. In each table, the solution must be read from the left to the right and from the top to the down. The solutions on the left side and in the center have been respectively communicated to us by Gravel [13, 19] and Solnon [33], whereas the solution on the right side has been obtained by the VFLS heuristic. W (resp. W') represents the number of violated windows without (resp. with) side windows and V (resp. V') the number of violations without (resp. with) side windows. Note that for any instance, we have $W \leq W' \leq V'$ and $W \leq V \leq V'$. Gravel et al. [13, 19] and Solnon [33] use W as criterion for evaluating solutions, whereas throughout this paper, V' is employed.

Instances	10 iterations	10 minutes	
	Average cost	Average cost	Average time
4-72	4.0	0.0	5.94
6-76	6.0	6.0	0.00
10-93	10.3	3.8	158.53
16-81	5.9	0.0	97.18
19-71	7.0	2.0	18.52
21-90	5.3	2.0	8.29
26-82	3.4	0.0	43.88
36-92	6.4	2.0	15.17
41-66	2.1	0.0	0.04
200-01	14.0	5.0	184.24
200-02	12.0	4.8	175.62
200-03	26.9	14.6	197.32
200-04	16.4	10.0	143.19
200-05	13.1	7.6	211.71
200-06	12.9	7.0	182.40
200-07	9.1	2.2	159.38
200-08	19.1	9.4	94.89
200-09	17.8	11.8	137.82
200-10	36.9	27.2	196.82

Instances	10 iterations	10 minutes	
	Average cost	Average cost	Average time
300-01	11.2	5.6	48.24
300-02	22.2	14.4	90.27
300-03	22.8	15.8	197.48
300-04	26.4	13.2	233.56
300-05	71.4	53.8	178.97
300-06	21.3	10.0	262.01
300-07	15.9	5.4	138.14
300-08	18.2	10.2	205.73
300-09	21.1	11.0	184.22
300-10	47.0	35.0	169.85
400-01	14.7	4.6	263.37
400-02	39.7	27.2	220.68
400-03	28.2	16.8	261.46
400-04	41.8	31.0	209.11
400-05	12.2	3.4	281.67
400-06	15.2	5.2	264.66
400-07	33.8	13.0	380.51
400-08	28.4	13.0	128.94
400-09	36.2	23.4	328.51
400-10	12.8	5.4	174.12

Instances	Worst cost	Instances	Worst cost
60-01	0	65-01	0
60-02	0	65-02	0
60-03	3	65-03	1
60-04	0	65-04	0
60-05	0	65-05	0
60-06	1	65-06	0
60-07	0	65-07	0
60-08	1	65-08	1
60-09	0	65-09	1
60-10	0	65-10	0
70-01	0	75-01	0
70-02	0	75-02	0
70-03	0	75-03	1
70-04	1	75-04	0
70-05	0	75-05	0
70-06	1	75-06	4
70-07	0	75-07	0
70-08	2	75-08	1
70-09	1	75-09	0
70-10	0	75-10	0

Instances	Worst cost	Instances	Worst cost
80-01	0	85-01	0
80-02	0	85-02	0
80-03	0	85-03	0
80-04	0	85-04	1
80-05	1	85-05	0
80-06	0	85-06	0
80-07	0	85-07	0
80-08	0	85-08	2
80-09	2	85-09	0
80-10	2	85-10	0
90-01	2		
90-02	0	4-72	6
90-03	1	6-76	6
90-04	0	10-93	12
90-05	1	16-81	8
90-06	0	19-71	10
90-07	0	21-90	7
90-08	0	26-82	5
90-09	0	36-92	8
90-10	1	41-66	4

Figure 11. Results of our greedy algorithm on CSPlib's instances. In the top table, the second column shows the average cost obtained after 10 iterations of the greedy algorithm (with 10 trials per instance), as used in VLNS, VFLS and HYBRID algorithms. The third and fourth columns show the average results of the greedy algorithm iterated on 10 minutes (with 10 trials per instance). Note that the greedy algorithm always finds the best known number of violations for 8 instances (of the first set) in a reasonable amount of time, which seems to indicate that these instances are not really hard to solve. In the bottom table, the second and fourth columns show the worst cost found by the greedy algorithm after 10 iterations, with 10 trials per instance. Note that the greedy algorithm always finds zero (resp. zero or one) violation for 48 (resp. 63) instances on 79, which confirms that instances of CSPlib's benchmarks 60/65/70/75/80/85/90 (all satisfiable) have a limited interest for experimentations.

05	08	06	17	06	21	05	13	06	17
08	22	03	13	06	19	07	01	23	06
01	24	08	13	10	21	00	21	10	13
08	24	01	06	23	01	20	08	17	06
21	03	22	06	03	21	06	16	10	06
15	10	09	13	10	08	13	09	10	13
08	10	16	06	10	13	08	12	13	06
23	01	22	10	13	06	19	07	21	03
06	21	04	21	06	03	08	20	01	23
06	15	10	20	01	23	08	13	11	08
13	08	23	00	21	10	15	06	24	01
06	21	03	20	08	03	08	13	07	19
06	15	10	20	01	23	08	13	11	08
13	08	10	14	08	10	15	06	24	01
06	21	03	07	08	03	21	06	04	21
06	01	23	07	15	10	08	17	07	08
17	08	06	18	08	06	15	10	20	01
23	08	13	11	08	13	08	10	14	08
23	01	23	07	15	10	15	06	23	02
23	08	13	10	16	06	10	15	06	12

19	05	11	07	14	01	16	07	11	14
18	01	15	07	11	05	18	00	14	07
01	14	10	11	14	09	11	16	07	11
05	12	06	11	07	14	01	17	05	11
07	14	13	05	11	07	14	13	05	11
07	14	01	17	05	11	07	14	01	18
04	11	07	05	01	15	07	11	05	18
00	14	07	01	14	10	11	14	09	11
06	07	11	05	12	06	11	07	14	01
16	07	11	14	18	01	15	07	11	05
18	00	14	03	14	05	02	14	05	12
05	04	12	05	14	03	04	14	03	14
05	02	14	05	12	05	04	12	05	14
01	08	14	01	14	18	00	14	09	11
05	17	11	05	12	01	15	07	01	14
18	00	14	09	11	05	17	11	05	11
18	04	11	09	14	01	17	14	01	14
18	00	14	09	11	05	17	11	05	12
05	04	12	05	14	03	04	14	03	14
01	08	14	01	14	18	00	14	09	11
05	08	11	05	07	01	15	07	01	14
18	00	14	03	14	01	17	11	05	12
01	15	07	01	14	09	00	14	09	11
05	08	11	05	14	03	04	14	03	14
05	02	14	05	12	05	11	08	05	11
09	14	00	18	14	01	07	15	01	12
05	14	02	05	14	03	14	04	03	14
05	12	04	05	12	05	11	08	05	11
09	14	00	18	05	11	07	15	01	18
14	11	07	16	01	14	07	11	05	19

21	06	14	09	05	14	10	07	12	09
18	00	21	05	12	04	05	01	20	05
14	20	08	12	09	14	05	11	12	05
21	12	08	09	12	07	20	01	20	05
14	09	07	00	20	07	12	07	10	12
07	09	12	08	16	07	05	12	11	12
07	20	05	12	11	05	14	09	05	13
04	07	05	16	06	07	04	05	05	02
19	07	16	05	12	11	07	12	09	05
01	20	07	12	05	21	00	18	09	12
07	20	01	20	05	14	09	08	12	05
21	05	01	20	05	14	02	06	07	16
05	18	03	05	18	02	18	06	16	07
12	07	20	00	07	09	14	05	20	01
20	07	12	09	08	12	07	09	12	08
02	18	05	16	05	18	04	06	05	02
18	07	16	06	05	04	18	05	17	05
18	04	05	05	03	07	18	02	05	05
04	08	05	16	05	18	03	05	18	02
18	06	02	05	18	02	18	05	03	18
05	02	18	06	16	05	18	04	13	05
20	07	14	10	12	05	02	07	08	16
05	05	04	07	06	16	05	18	02	05
01	20	07	12	05	21	00	18	02	12
07	20	01	20	05	14	09	15	05	02
18	05	15	09	12	07	20	01	20	05
14	09	15	05	09	14	05	21	00	20
07	12	07	10	12	07	09	14	06	09
14	05	20	01	20	07	12	09	07	13
07	20	05	14	10	07	12	09	14	06
07	16	05	18	02	19	05	16	07	12
10	07	12	21	05	13	09	14	07	20
00	20	07	14	09	05	00	21	05	14
09	05	01	20	07	12	09	07	00	21
05	12	09	18	01	20	05	12	04	18
06	16	05	18	04	05	06	02	18	07
16	06	05	04	05	18	03	05	07	02
18	06	16	07	05	02	18	06	04	05
05	04	06	05	04	05	07	16	06	18
02	18	05	17	07	05	02	14	06	21

FIGURE 12. Best solutions to CSPLIB's instances 200-03 (3 violations), 300-05 (27 violations) and 400-02 (15 violations). Solutions must be read from the left to the right and from the top to the down.