Two local search approaches for solving real-life car sequencing problems

Bertrand Estellon [a], Frédéric Gardi [a,b], Karim Nouioua [a]

[a] *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*

[b] *Bouygues SA/DGITN/e-lab, 32 avenue Hoche, 75008 Paris, France*

Abstract

The NP-hard problem of car sequencing appears as the heart of the logistic process of many car manufacturers. The subject of the ROADEF'2005 Challenge addressed a car sequencing problem proposed by the car manufacturer RENAULT, more complex than the academic problem generally addressed in the literature. This paper describes two local search approaches for this problem. In the first part, a new approach by very large-scale neighborhood search is presented. This approach, designed during the qualification stage preceding the final, is based on an original integer linear programming formulation. The second part is dedicated to the approach which enabled us to win the ROADEF'2005 Challenge. Inspired by the latest works on the subject, this one is based on very fast explorations of small neighborhoods. Our contribution here is mainly algorithmic, in particular by showing how much exploiting invariants speeds up the neighborhood evaluation and contributes to the diversification of the search. Finally, the two approaches are compared and discussed through an extensive computational study on RENAULT's benchmarks. The main conclusion drawn at this point is that sophisticated metaheuristics are useless to solve car sequencing problems. More generally, our victory on ROADEF'2005 Challenge demonstrates that algorithmic aspects, sometimes neglected, remain the key ingredients for designing and engineering high-performance local search heuristics.

Key words: combinatorial optimization, real-life car sequencing/scheduling, integer linear programming, local/neighborhood search, invariants

1991 MSC: 90C27, 90B35, 90C10

⋆ The second author is on leave from the firm Experian-Prologia, Marseille, France.
Email addresses: bertrand.estellon@lif.univ-mrs.fr (Bertrand Estellon), fgardi@bouygues.com (Frédéric Gardi), karim.nouioua@lif.univ-mrs.fr (Karim Nouioua).

1 Introduction

A modern car factory is composed of three major workshops: the sheet metal workshop where the body of the car is assembled, the paint workshop where the car is painted, and the assembly line where the equipments and options of each vehicle are set. The car sequencing problem consists in determining the order in which a set of vehicles should go through these three workshops so as to make easier the whole process of fabrication. An ordered set of vehicles is called sequence.

Each workshop is subject to specific constraints, which tend to be in conflict with each other. In order to limit the complexity of the problem, the previous works on the subject [8–11,18,22] dealt only with the constraints and objectives related to the assembly line. In the context of the ROADEF'2005 Challenge organized by the French Operations Research Society, a more complex problem was posed by the car manufacturer RENAULT, including both the needs of the paint workshop and the ones of the assembly line.

1.1 The RENAULT's car sequencing problem

For the paint workshop, the objective is to minimize the consumption of solvent used to purge the spray guns at each change of color in the sequence. In a more explicit way, the goal is to minimize the number of purges (or color changes), which amounts to group vehicles which have the same color. Nevertheless, the number of consecutive vehicles having the same color must not exceed a certain value, noted PAINT-LIMIT (the color must be changed regularly to allow a better visual checking of the quality).

In order to smooth the workload on the different stations composing the assembly line, it is necessary to space out 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 stations 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 three vehicles affected by the option in any contiguous subsequence consisting of 7 vehicles (such subsequences are called windows). 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 for all the ratio constraints. In the previous example, if 5 vehicles have the option in a window of 7, then 2 violations are counted. To distinguish certain options having priority, the violations are weighted by some constants.

Here the options are classified into two kinds: priority or non-priority.

In this real-life version, the objective function is composed of three terms which are minimized in lexicographic order: the number of color changes (RAF), the number of violations on priority options (EP) and the number of violations on non-priority options (ENP). For example, the ordering EP/RAF/ENP means that the first objective is EP, the second is RAF and the third ENP. In this way, three kinds of objectives are possible: EP/ENP/RAF, EP/RAF/ENP and RAF/EP/ENP. In fact, these different orderings of objectives lead to car sequencing problems which are, by nature, quite different with each other.

1.2 Previous works

The car sequencing problem is strongly NP-hard, even if only ratio constraints are considered [11], and the brute force use of constraint programming or integer programming softwares reaches its limit when one hundred or so vehicles with few options are considered (see the studies of [10,20]). Then, several heuristics have been proposed to solve effectively car sequencing problems like ant colony optimization [9,10,22], greedy algorithms [9] or local search [9,18].

These approaches have been intensively studied and experimented in the context of ROADEF'2005 Challenge [3,4]. In effect, the heuristics designed by the different competitors to the Challenge are principally based on local search techniques, integrated into improvement heuristics based on classical schemes like simulated annealing or tabu search, or more exotic ones like iterated local search, variable neighborhood search or greedy randomized adaptive search procedures (see for example the hybrid heuristic proposed by Ribeiro et al. [19], second team of the competition). The main characteristic of all these heuristics is the small size of the neighborhood explored at each iteration.

Recently, some approaches based on large neighborhood search have been proposed to solve car sequencing problems. Although appearing as an improvement heuristic in many classical algorithms like augmenting path algorithms for solving matching problems, the concept of large neighborhood search has been really formalized and studied only for ten years. Unfortunately, the formal definition of what is large neighborhood search in combinatorial optimization seems to differ slightly from one scientific community to another. In constraint programming community, large neighborhood search is viewed as the hybridization of local search and exact resolution techniques like constraint programming or mixed integer programming, without precise notion of complexity (see [21] and more recent works like [5,13,15]). On the other hand, operations researchers define very large-scale neighborhoods as neighborhoods whose size grows exponentially with respect to the size of input data,

and work on designing polynomial algorithms to explore such neighborhoods. A comprehensive survey on very large-scale neighborhood search techniques was recently written by Ahuja et al. [2].

Perron et al. [14,15] have worked on the integration of local search into a constraint programming environment dedicated to the resolution of ratio constraints. During the redaction of this paper, we have been informed of the work of Prandtstetter and Raidl [17] hybridizing integer linear programming and local search. In both cases, the experimental results reported by the authors are not competitive with pure local search (note that the results given in [17] on RENAULT's instances are in fact erroneous [16]).

All the terminology related to local search which is employed throughout the paper is derived from the book of Aarts and Lenstra [1].

1.3 Organization of the paper

The paper describes two local search approaches for the RENAULT's car sequencing problem, both integrated into a simple descent heuristic.

The first part is dedicated to a new approach by very large-scale neighborhood search. This approach, designed during the qualification stage preceding the final, is based on an original integer linear programming formulation. Our attention is focused on the treatment of RENAULT's constraints and objectives. An oncoming paper [7] deals with this approach and its theoretical foundations in the context of academic car sequencing problems (i.e., including only ratio constraints); because resulting of a more recent research, these achievements are not detailed here.

The second part is devoted to the local search approach which enabled us to win the ROADEF'2005 Challenge. Inspired by the recent works of Gottlieb et al. [9,18] on the subject, this one is based on very fast explorations of small neighborhoods. Our contribution stands at two levels. First, the neighborhood functions proposed in [9,18] are enriching and adapting to treat the RENAULT's car sequencing problem at best, in particular the contraints and objectives related to the paint workshop. Besides, we explain how to make the exploration very efficient by maintaining special data structures incrementally. This second contribution, mainly algorithmic, is probably the most important part of our work, by showing how much exploiting invariants speeds up the neighborhood evaluation and contributes to the diversification of the search (the reader is referred to the work of Michel and Van Hentenryck [12] for an introduction to the concept of invariants in local search).

Finally, the two approaches are compared and discussed through an extensive

computational study on RENAULT's benchmarks. The main conclusion drawn at this point, quite surprising, is that sophisticated metaheuristics are useless to solve car sequencing problems. On the other hand, the paper ends with promising research ideas to improve the efficiency of these approaches.

A preliminary version of this paper has been published in French in [6].

2 Very large neighborhood search

In this section, we present a heuristic based on very large-scale neighborhood search which qualified our team to the final of the ROADEF'2005 Challenge. This one can be viewed as hybrid in the sense that it mixes local search and integer linear programming (ILP).

2.1 ILP formulation

An original ILP formulation of the RENAULT's car sequencing problem is described here. The number of vehicles, of options and of colors are respectively denoted by NPOS, NOP and NCOL. To each option i is attached the ratio constraint $\mathsf{P}_i/\mathsf{Q}_i$ where $\mathsf{P}_i \leq \mathsf{Q}_i$ are positive integers; Q-MAX corresponds to the maximum value among all the Q_i's. In order to reduce the number of variables, similar vehicles are grouped into classes (two vehicles are similar if they share the same options and the same color). The number of classes is denoted by NCL and each class k contains N_k vehicles.

First, 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 class k and 0 otherwise. The basic constraints (1) and (2) ensure that all the vehicles of a class are assigned to a position and that a position is occupied by one and only one vehicle of a class:

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

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

Now, a binary variable $o_{i,j}$ is associated to each pair option i/position j whose value is 1 if the vehicle at position j has option i and 0 otherwise. Having defined for each pair class k/option i a constant $\mathsf{OP}_{k,i}$ which equals 1 if the vehicles of class k have option i and 0 otherwise, each variable $o_{i,j}$ can be

expressed as a linear function of the $cl_{k,j}$'s:

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

Then, counting up the number $v_{i,j}$ of violations on option i for the window beginning at position j is done via the inequality

$$v_{i,j} \geq \sum_{f=j}^{j+\mathsf{Q}_i-1} o_{i,f} - \mathsf{P}_i \quad \forall i \in \{1, \ldots, \mathsf{NOP}\} \ \forall j \in \{1, \ldots, \mathsf{NPOS} - \mathsf{Q}_i + 1\} \tag{4}$$

coupled with the positivity constraint $v_{i,j} \geq 0$, since the $v_{i,j}$'s are minimized. In effect, $\mathsf{Q}_i - 1$ similar constraints should be added to count violations caused by the last (resp. first) vehicles sequenced the previous (resp. next) day, but for the sake of simplicity these ones are omitted here.

The variables and constraints concerning colors are defined in a similar way. To each pair color i/position j is associated a binary variable $c_{i,j}$ whose value is 1 if the vehicle at position j has color i and 0 otherwise. Having defined for each pair class k/color i a constant $\mathsf{CO}_{k,i}$ which equals 1 if vehicles of class k must be painted with color i and 0 otherwise, each variable $c_{i,j}$ can be expressed linearly in function of the $cl_{k,j}$'s:

$$c_{i,j} = \sum_{k=1}^{\mathsf{NCL}} \mathsf{CO}_{k,i} \times cl_{k,j} \quad \forall i \in \{1, \ldots, \mathsf{NCOL}\} \ \forall j \in \{1, \ldots, \mathsf{NPOS}\} \tag{5}$$

Then, to each position j is associated a variable p_j whose value is 1 when a purge is performed between positions $j-1$ and j. The activation of the p_j's is done via the pair of constraints

$$\begin{aligned} p_j &\geq c_{i,j} - c_{i,j-1} \\ p_j &\geq c_{i,j-1} - c_{i,j} \end{aligned} \quad \forall i \in \{1, \ldots, \mathsf{NCOL}\} \ \forall j \in \{2, \ldots, \mathsf{NPOS}\} \tag{6}$$

which is equivalent to the XOR constraint $p_j = c_{i,j-1} \oplus c_{i,j}$, since the p_j's are minimized. Then, the PAINT-LIMIT constraints are expressible using the $c_{i,j}$'s too:

$$\sum_{f=j}^{j+\mathsf{PAINT\text{-}LIMIT}} c_{i,f} \leq \mathsf{PAINT\text{-}LIMIT} \quad \begin{aligned} &\forall i \in \{1, \ldots, \mathsf{NCOL}\} \\ &\forall j \in \{1, \ldots, \mathsf{NPOS} - \mathsf{PAINT\text{-}LIMIT}\} \end{aligned} \tag{7}$$

To conclude, the objective function of the ILP is written, with CV_i the cost of

one violation on option i and CP the cost of one purge:

$$\text{Minimize} \quad \sum_{i=1}^{\mathsf{NOP}} \sum_{j=1}^{\mathsf{NPOS}-\mathsf{Q}_i+1} \mathsf{CV}_i \times v_{i,j} + \sum_{j=2}^{\mathsf{NPOS}} \mathsf{CP} \times p_j \tag{8}$$

The values of the constants CV_i and CP are chosen in such a way that the three objectives EP, ENP and RAF are minimized in the desired lexicographic order.

One can observe that the integrality of the $cl_{k,j}$'s ensures the integrality of any solution of the ILP. Conversely, if the sole $o_{i,j}$'s and $c_{i,j}$'s are integral, any solution of the ILP is integral too. Consequently, the domains of variables can be defined as follows: if $\mathsf{NCL} \leq \mathsf{NOP} + \mathsf{NCOL}$ then

$$cl_{k,j} \in \{0,1\} \quad \forall k \in \{1,\ldots,\mathsf{NCL}\} \;\; \forall j \in \{1,\ldots,\mathsf{NPOS}\} \tag{9}$$

else

$$\begin{aligned} o_{i,j} &\in \{0,1\} \quad \forall i \in \{1,\ldots,\mathsf{NOP}\} \;\; \forall j \in \{1,\ldots,\mathsf{NPOS}\} \\ c_{i,j} &\in \{0,1\} \quad \forall i \in \{1,\ldots,\mathsf{NCOL}\} \;\; \forall j \in \{1,\ldots,\mathsf{NPOS}\} \end{aligned} \tag{10}$$

Unlike the more classical formulation proposed by Gravel et al. [10] (which omits the intermediate variables $o_{i,j}$ and $c_{i,j}$) or even the one given recently by Prandtstetter and Raidl [17], the number of binary variables is not only determined by the number of classes, but by the number of options and colors too.

In conclusion, the total number of variables is $(\mathsf{NCL} + \mathsf{NOP} + \mathsf{NCOL}) \cdot \mathsf{NPOS}$, of which $\min\{\mathsf{NCL}, \mathsf{NOP} + \mathsf{NCOL}\} \cdot \mathsf{NPOS}$ are binary. The total number of constraints is $\mathsf{NCL} + (3 \cdot \mathsf{NOP} + 4 \cdot \mathsf{NCOL} + 1) \cdot \mathsf{NPOS}$. The number of nonzero coefficients of the matrix is bounded by $\mathsf{NPOS} \cdot (2 \cdot \mathsf{NCL} + (\mathsf{NCL} + \mathsf{Q\text{-}MAX} + 3) \cdot \mathsf{NOP} + (\mathsf{NCL} + \mathsf{PAINT\text{-}LIMIT} + 7) \cdot \mathsf{NCOL})$. Thus, assuming that $\mathsf{Q\text{-}MAX}$ and $\mathsf{PAINT\text{-}LIMIT}$ are small constants in practice, the matrix of the ILP is quite sparse.

2.2 ILP-based neighborhood search

The ILP formulation given above allows us to solve exactly some instances consisting of nearly one hundred vehicles with few options and colors in a reasonable lapse of time by using state-of-the-art ILP solvers. Unfortunately, for large-scale instances (of the order of one thousand or so vehicles), such tentatives are doomed to fail: the linear relaxation of the program is generally very fractional with cost equal to zero and a classical branch-and-bound procedure has trouble finding one integer solution. In this section, we show how to take

advantage of ILP to explore effectively exponential neighborhoods. Here are the broad lines of the improvement heuristic.

VLNS heuristic(TIME-LIMIT)
Begin;
    initialize K, OBJ and T-MAX;
    compute initial sequence;
    while TIME-LIMIT is not reached **do**
        choose K movable vehicles;
        construct ILP restricted to these K movable vehicles and objectives OBJ;
        run branch-and-bound on ILP during T-MAX seconds;
        update current sequence and adjust K, OBJ and T-MAX;
    end do;
    return current sequence;
End;

At each iteration of the descent, K vehicles of the sequence are chosen in order to exchange their positions; these K vehicles are called movable. The neighborhood of the current solution, called K-permutation neighborhood, is defined as the set of sequences which are obtainable by permuting these movable vehicles. Thus, the number of neighbors is potentially K!, with equality when PAINT-LIMIT constraints are not considered. Having defined OBJ as a subset of the three objectives EP, ENP and RAF, the goal is to find a neighbor having a better (or equal) cost than the current solution, or even the best possible one, in accordance with objectives in OBJ. To do that, the ILP restricted to objectives OBJ is solved where all the variables corresponding to non-movable vehicles are fixed. The resolution is done using a basic branch-and-bound procedure in a lapse of time limited to T-MAX seconds; in order to speed up the computation of the linear relaxation, a realizable basis built from the current solution is given to the simplex algorithm. Once the resolution finished or interrupted for lack of time, the current solution is updated and the parameters K, OBJ and T-MAX are adjusted to plan a new iteration.

The efficiency of the heuristic is based on the choice of the movable vehicles and the three parameters K, OBJ and T-MAX. These ones must be chosen in such a way to perform the best improvements at each iteration, in particular to decrease quickly the cost of the initial solution. Note that solutions of equal cost are necessary to diversify the search when finding better solutions becomes difficult.

2.2.1 The choice of movable vehicles

Two kinds of optimization must be considered: the case where the prime objective is EP and the case where the prime objective is RAF.

When EP is the prime objective, the following remark is a key to lower the time necessary to obtain an optimal integer solution of the restricted ILP: the further apart are the movable vehicles in the sequence, the better is the quality of the linear relaxation of the ILP. This phenomenon, initially verified through experimentations, has been recently characterized by using some elements of polyhedral analysis. In particular, we have proved that if any two movable vehicles are distant from at least Q-MAX positions, then any basic optimal solution of the linear relaxation of the restricted ILP is integer. More details shall appear in [7]. Thus, a simple way to proceed is to choose movable vehicles in windows where violations appear and complete by movable vehicles picked randomly into the sequence.

When RAF is the prime objective, an optimal assignment of colors is initially computed (see Section 4 for details). Then, the goal is to determine an optimal assignment of colors which minimizes the number of violations on objectives EP and ENP. In this case, movable vehicles are chosen into random blocks of consecutive positions to preserve the optimality of the assignment.

2.2.2 The choice of objectives

Since the objectives are lexicographically ordered, more computing time is allocated to the realization of the first objective than to the second and the third ones. Thus, the whole heuristic is composed of three phases: approximately 70% of the total computing time is dedicated to the realization of the first objective, 25% to the realization of the second objective, and 5% to the realization of the third objective.

When more than one objective is considered, fixing variables corresponding to higher-priority objectives in the ILP is done to intensify the search on a lower-priority objective; not fixing these variables can be viewed as a diversification of the search.

Because considering several objectives during the neighborhood search augments the size of the ILP, the values of the parameters K and T-MAX must be carefully chosen. If K is too large or T-MAX too low, the resolution of the restricted ILP often fails (no integer solution found), which considerably slows down the descent. To prevent that, an idea is to initialize K to a small value and to 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 T-MAX.

3 Very fast local search

This section is devoted to the local search heuristic which enabled us to win the ROADEF'2005 Challenge. Based on very fast explorations of small neighborhoods, this first-improvement descent heuristic differs radically from the one exposed in the previous section. More precisely, the algorithm applies at each iteration one transformation to the current sequence which modifies it only very locally. If the transformation is good, that is, respects the PAINT-LIMIT constraints and does not increase the cost of the current solution, this one is really performed. This approach is inspired by the recent works of Gottlieb et al. [18,9] on the subject. Here are the broad lines of the heuristic.

VFLS heuristic(TIME-LIMIT)
Begin;
    compute initial sequence;
    while TIME-LIMIT is not reached **do**
        choose transformation and positions where applying it;
        if transformation is good **then**
            update current sequence by performing it;
        end if;
    end do;
    return current sequence;
End;

Having presented the different transformations which are employed, we shall give the keys which made the success of this approach.

3.1 The transformations

Five basic transformations are used: swap, forward insertion, backward insertion, reflection and random shuffle (see Figure 1). A swap simply consists in exchanging the positions of two vehicles of the sequence. A forward insertion localized on a portion v_k, x, y, z, v_l of vehicles consists in extracting v_l, shifting the vehicles v_k, x, y, z to the right, and reinserting v_l at the position which remains unfilled (the former position of v_k); after the transformation, the initial portion contains in order the vehicles v_l, v_k, x, y, z. A backward insertion is defined in a symmetric way, by extracting v_k instead of v_l. A reflection between two vehicles v_k and v_l consists in reversing the portion of vehicles between v_k and v_l. Finally, a random shuffle between v_k and v_l consists in shuffling randomly the vehicles in the portion defined by these two positions.

The neighborhood which can be explored according to swap, insertion and reflection transformations is only of size $O(\mathsf{NPOS}^2)$. Moreover, the selection

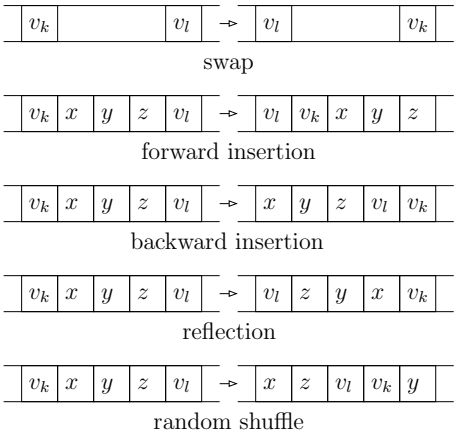

Fig. 1. The five transformations.

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. Note that the acceptance of transformations which do not improve the cost is crucial; coupled with a very fast evaluation procedure, this is a way, in addition to the random shuffle transformation, to diversify widely the search and then to avoid local optima during the descent.

Since the effectiveness of this approach relies directly on the number of performed transformations, two questions must be considered carefully: how to increase the probability of success of one transformation? and how to increase the number of attempted transformations? The answer to the second question – make the evaluation of transformations fast – is detailed finally. Now we answer to the first question by giving some hints to choose cleverly the positions where transformations must be applied.

3.2 Choosing the positions

The simplest way to choose the positions where to apply swap, insertion and reflection transformations consists in picking out randomly the positions k and l. This generic strategy is essentially employed early during the descent to decrease the cost quickly when EP is the prime objective. However, sharper strategies are useful when finding better solutions becomes difficult. Of course, such strategies depend on what is the prime objective and which kind of transformation is applied. All these strategies are exposed on Figure 2.

For swaps, choosing vehicles sharing some options or having the same color augments the chance of success of the transformation. In the same way, choosing adjacent positions in the sequence limits the risk of deterioration while making the evaluation faster. For insertions and reflections, a good strategy is to choose k and l such that the distance $|l - k|$ is equal to the denominator

Q_i of one ratio constraint. When RAF is the prime objective, choosing the positions k and l as the beginning or the ending of a sequence of vehicles having the same color limits the chances to break such sequences.

Variants	Description
Generic	Pick two positions k and l randomly.
Similar	Pick two positions k and l such that the corresponding vehicles share some options.
Consecutive	Pick a position k randomly and set $l = k + 1$.
Violation	Pick a position k where a violation appears and choose l randomly.
Denominator	Pick a position k and an option i randomly and set $l = k + \mathsf{Q}_i$.
Same color	Pick two positions k and l such that the corresponding vehicles have the same color.
Border block one	Pick a position k at the beginning or at the ending of a sequence of vehicles having the same color. Choose l randomly.
Border block two	Pick the positions k and l at the beginning or at the ending of a sequence of vehicles having the same color.
Violation same color	Pick a position k where a violation appears and choose l such that the corresponding vehicle has the same color than the one at position k.

Fig. 2. The different strategies for choosing positions.

The global heuristic is composed of four optimization subroutines: Greedy which computes an initial solution, OptA which decreases the number of violations on ratio constraints (when optimizing first on EP or ENP objectives), OptB which builds some blocks of colors (when optimizing RAF objective after EP or ENP objectives), and OptC which decreases the number of violations while maintaining blocks of colors (when optimizing EP or ENP objectives after RAF objective). The percentage of total time consumed by each optimization subroutine is given on Figure 3, for each kind of objective function. For example, the first line of the table means that 60% of the total time is spent in subroutine OptA for EP objective, 25% in subroutine OptA for EP and ENP objectives, and 15% in subroutine OptB for the three objectives. The exact composition of each optimization subroutine is detailed on Figure 4; for example, the number of generic swaps which are attempted represents 66% of the total number of attempted transformations for subroutine OptA. More details on Greedy subroutines are given in Section 4.

All these values have been determined by conducting extensive experimentations on RENAULT's instances. To summary, these ones teach us the following general rules. When optimizing firstly on ratio contraints (optA), the number of swaps dominate. More than 66% of attempted transformations are basic

swaps; the number of insertions and reflections should be around 15%. To create or maintain blocks of colors (optB, optC), the number of swap, insertion and reflection transformations is more homogeneous, respectively around 40%, 25% and 35%. Remark that random shuffles are used sparsely (1% of attempted transformations) because time-consuming. These ones, which are done on short portions of sequence (no more than a dozen of vehicles), seem to ensure in practice the convergence towards a global optimal solution. They serve as second level of diversification, after the acceptation of solutions of equal cost.

Objectives	**Phase 1**	**Phase 2**	**Phase 3**	**Phase 4**
EP/ENP/RAF	Greedy EP	60% OptA	25% OptA	15% OptB
EP/RAF/ENP	Greedy EP	60% OptA	25% OptB	15% OptC
EP/RAF	Greedy EP	50% OptA	50% OptB	-
RAF/EP/ENP	Greedy RAF	80% OptC	20% OptC	-
RAF/EP	Greedy RAF	100% OptC	-	-

Fig. 3. The optimization sequences for each objective function.

Transformations	**Variants**	**OptA**	**OptB**	**OptC**
Swap	Generic	66%	18%	-
	Similar	2%	-	-
	Consecutive	2%	4%	5%
	Same color	-	8%	25%
	Border block two	-	10%	5%
	Violation	2%	2%	-
	Violation same color	-	1%	5%
Insertion	Generic	8%	-	-
	Denominator	8%	-	-
	Same color	-	30%	12%
	Border block one	-	8%	12%
Reflection	Generic	7%	-	-
	Denominator	4%	-	-
	Same color	-	8%	10%
	Border block one	-	6%	10%
	Border block two	-	4%	15%
Shuffle	Generic	1%	1%	1%

Fig. 4. The composition of each optimization subroutine.

Choosing the positions where is applied one transformation is not really time-consuming. The bottleneck in term of complexity of one iteration of the VFLS heuristic is clearly the evaluation. Fortunately, the singular structure of the first four transformations (swap, forward insertion, backward insertion and reflection) reveals some invariants which are exploitable thanks to special data structures to evaluate quickly the impact of these transformations on the cost of the current solution.

For violations, the crucial remark is that the number of windows which are perturbed by one of the first four transformations (that is, which must be reevaluated) depends only on the denominator of each ratio constraint, generally small in practice. For swaps, only windows containing the two exchanged vehicles v_k and v_l are perturbed. In the case of forward and backward insertions, the windows which are entirely contained between the vehicles v_k and v_l are shifted of one position. Thus, only windows containing v_k and v_l must be considered for the evaluation of insertion transformations. The same idea holds for reflections since windows entirely contained between extremal positions k and l are reversed, which lets the number of violations into these ones unchanged. This remark is formalized through the following proposition.

Proposition 1 *For any ratio constraint $\mathsf{P}_i/\mathsf{Q}_i$, the number of windows which require to be reevaluated following one swap, one insertion or one reflection is at most $2\mathsf{Q}_i$, which is the best possible bound. Then, evaluating the new number of violations following one of these transformations is done in $O(\mathsf{Q}_i)$ time.*

The second part of the proposition rises from the definition of the data structure used to maintain the number of violations. This one is quite simple, storing for each option and each window associated to this option, the number of vehicles which require it. Then, the evaluation of swaps is done by simulating the exchange of v_k by v_l in every window containing v_k and the exchange of v_l by v_k in every window containing v_l. By using this data structure, simulating the insertion or the deletion of one vehicle in a window associated to an option is done in constant time. Since at most $2\mathsf{Q}_i$ windows are concerned for an option with ratio constraint $\mathsf{P}_i/\mathsf{Q}_i$, the evaluation of one swap transformation takes $O(\mathsf{Q}_i)$ time for this option. By applying the same idea, the evaluation of forward and backward insertions is done in $O(\mathsf{Q}_i)$ time too for each option i. The evaluation of reflections is more delicate because the number of modified positions in each modified window (that is, containing v_k and v_l) is not constant. Here is the way to process. At iteration j, evaluate the window ending at the position v_{k+j} and the one starting at the position v_{l-j}. Now, start the evaluation with $j = 0$ and stop it when every window which is likely to act upon the number of violations has been considered. At iteration j, the

vehicles moving into the two considered windows are composed of those at position v_{k+j} and v_{l-j}, plus the ones contained in the two windows treated at iteration $j-1$. By using this property, each iteration can be computed in constant time. Since $O(\mathsf{Q}_i)$ iterations are performed, the reflection is evaluated in $O(\mathsf{Q}_i)$ time for any option i.

Proposition 2 *Detecting violations of PAINT-LIMIT constraints and evaluating the new number of purges following one swap, one insertion or one reflection is done in $O(1)$ time.*

Another special data structure is used for maintaining informations concerning colors. This one contains for each position j of the sequence, the indices of the beginning and the ending of the block of vehicles having the same color than the vehicle at j and containing it. For the first four transformations (swap, forward insertion, backward insertion and reflection), such informations suffice to check in $O(1)$ time if the transformation respects the PAINT-LIMIT constraints and to evaluate in $O(1)$ time too the new number of purges, since only a constant number of positions around v_k and v_l needs to be scanned to do this.

The update of the two previous structures can be done efficiently, but this is not crucial here because such an operation is performed in case of success only. Since the number of attempted transformations dominates largely the number of performed transformations, the complexity of updating becomes negligible compared to the one of evaluating. The remark is also valid concerning the initialization of the data structures. In consequence, these aspects are left to the reader.

3.4 Speeding up the evaluation

Since there is no compensation between objectives EP, ENP and RAF, the order of their evaluation is significant. In many cases, the evaluation process can be stopped before to have evaluated the three objectives. For example, if a transformation deteriorates (resp. improves) the current solution for the first objective, the decision to reject (resp. accept) this transformation can be taken without evaluating the last two objectives (because the first one dominates the two others). Indeed, the second objective needs to be evaluated only if the value of the first objective is not modified by the transformation. Obviously, this remark holds for the evaluation of the second and the third objectives. In the same way, starting the evaluation by checking if the transformation respects the PAINT-LIMIT constraints seems to be judicious, since the verification takes only $O(1)$ time. Figure 5 gives the general scheme of an evaluation.

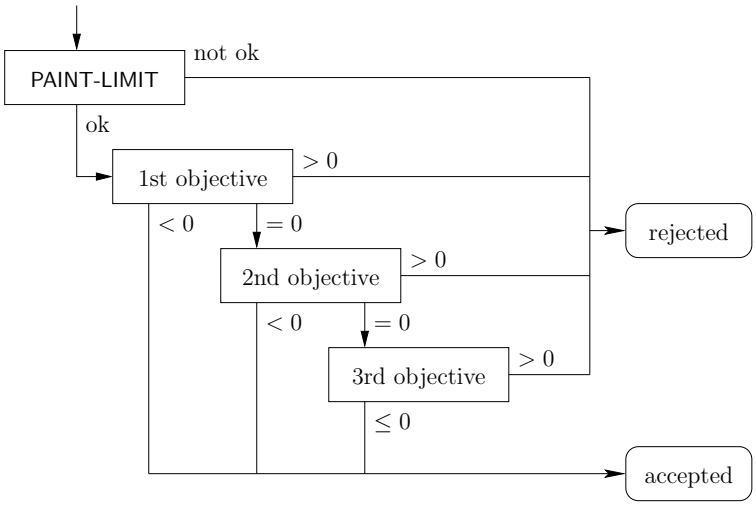

Fig. 5. General evaluation scheme of one transformation.

The evaluation can be further improved heuristically for ratio constraints. In effect, the order in which the ratio constraints are treated is significant too. Suppose that after the evaluation of one transformation for some ratio constraints, the number of violations newly created is greater than the total number of violations on the remaining ratio constraints (that is, which are not evaluated yet). Then, the evaluation can be stopped immediately and the transformation rejected, because in all cases the transformation shall deteriorate the sequence.

option	before	max profit	after	profit
A	3	14	5	–2
B	0	14	3	–5
C	1	13	0	–4
D	0	13	2	–6
E	7	6	10	–9
F	4	2	STOP	
G	2	0		

option	before	max profit	after	profit
E	7	10	10	–3
F	4	6	8	–7
A	3	3	STOP	
G	2	1		
C	1	0		
B	0	0		
D	0	0		

Fig. 6. Speeding up the evaluation: without ordering (left), with ordering (right).

Accordingly, evaluating ratio constraints in the decreasing order of the number of violations seems to be a good heuristic to decide earlier of the rejection. Figure 6 shows an example of evaluation without special ordering on ratio constraints (left) and the same case having ordered options according to the number of violations (right). The column named “before” (resp. “after”) corresponds to the number of violations for each option before (resp. after) the transformation. The column named “max profit” gives for an option X the maximum profit obtainable on options evaluated after X; the column named

"profit" gives for an option X the profit obtained on options evaluated until X (negative profits mean an increase in the cost of the current sequence).

4 Computing initial solutions

4.1 Greedy algorithms

Greedy algorithms are useful to obtain good initial solutions, and even become crucial when prime objective is RAF. The greedy algorithms that we employed are inspired from the DSU heuristic from Gottlieb et al. [9], which is based on the dynamic utilization rate

$$\mathrm{dur}(i, \pi_j) = \frac{\mathsf{Q}_i}{\mathsf{P}_i} \cdot \frac{\mathsf{N}_i(\Pi) - \mathsf{N}_i(\pi_j)}{|\Pi| - |\pi_j|}$$

defined for each option i, where π_j represents the partial sequence built until position j, Π the set of cars to be produced, $\mathsf{N}_i(\pi_j)$ the number of vehicles requiring option i in the sequence π_j, and $\mathsf{N}_i(\Pi)$ the number of vehicles requiring option i in the set Π.

If the prime objective is to minimize the number of violations on priority options (EP), our algorithm, named Greedy EP, chooses the next vehicle so as to maximise the sum of dynamic utilization rates computed only on priority options, while not violating PAINT-LIMIT constraints.

If the prime objective is to minimize the number of purges (RAF), our algorithm, named Greedy RAF, starts by choosing the color of the next vehicle. The number NC_i of vehicles requiring color i among those which remain to insert is stored for each color i. If a vehicle having the same color than the last one of the sequence under construction can be inserted without violating PAINT-LIMIT, the algorithm makes it. Otherwise, another color is chosen such that the value of NC_i is as large as possible; such a choice is useful to prevent the case where a lot of vehicles sharing the same color remains without vehicle having a different color to separate them. Having fixed the color of the next vehicle, the algorithm chooses the one which maximises the sum of dynamic utilization rates on priority options.

Theoretically speaking, the heuristic Greedy RAF is not exact because some sequences may be built which does not respect PAINT-LIMIT. But in practice, PAINT-LIMIT constraints are not so hard to satisfy (RENAULT ensures us an admissible solution) and such a heuristic allows to obtain good solutions for the prime objective RAF but also for the second objective EP. In the following section, an optimal algorithm is presented to solve the problem for colors.

4.2 An optimal algorithm for colors

Here is described an optimal algorithm to solve the problem for colors (that is, for objective RAF only). In input, we have the PAINT-LIMIT and for each color i, the number NC_i of vehicles which require this color (the computation of the NC_i's can be done in $O(\mathsf{NPOS})$ from the characteristics of the vehicles). A maximal subsequence of vehicles having the same color is called a block. Then, the problem is to determine a minimum ordered partition of the NPOS vehicles into blocks such that the size of each block does not exceed the PAINT-LIMIT, if such a partition exists. Having such a solution, the number of purges equals the number of blocks minus one.

The quantity $\lceil \mathsf{NC}_i/\mathsf{PAINT\text{-}LIMIT} \rceil$, which corresponds to the minimum number of blocks necessary to partition the vehicles of color i, is denoted by NB_i. Let c be the color for which NC_i, or equivalently NB_i, is maximum.

Proposition 3 *The problem has a solution if and only if* $\mathsf{NB}_c \leq \sum_{i \neq c} \mathsf{NC}_i + 1$. *When this condition is satisfied, the optimal number of blocks is given by the expression* $\max\{2\mathsf{NB}_c - 1, \sum_i \mathsf{NB}_i\}$.

The "only if" part of the first assertion is verified by observing that when $\mathsf{NB}_c > \sum_{i \neq c} \mathsf{NC}_i + 1$, there are not enough vehicles to separate each pair of blocks with color c by one vehicle with color $i \neq c$. Assuming that $\mathsf{NB}_c \leq \sum_{i \neq c} \mathsf{NC}_i + 1$, the rest of the proposition is demonstrated constructively.

According to the definition of the NB_i's, an optimal sequence is composed of at least $\sum_i \mathsf{NB}_i$ blocks. On the other hand, each pair of blocks with color c must be separated by at least one block with color $i \neq c$, which implies the second lower bound $2\mathsf{NB}_c - 1$. Consider first the case where $2\mathsf{NB}_c - 1 > \sum_i \mathsf{NB}_i$, which can be rewritten as $\mathsf{NB}_c > \sum_{i \neq c} \mathsf{NB}_i + 1$. In this case, a solution is obtained by subdividing blocks of color $i \neq c$ while NB_c remains strictly greater than the sum of blocks of color $i \neq c$ plus one (this is always possible since $\mathsf{NB}_c \leq \sum_{i \neq c} \mathsf{NC}_i + 1$) and then ordering alternatively blocks of color c and blocks of color $i \neq c$. Now, consider the case where $\mathsf{NB}_c \leq \sum_{i \neq c} \mathsf{NB}_i + 1$. Here a sequence is built by applying the following rule: the color of the next block to be placed in the sequence is, among all the colors differing from the one of the previous block, the one in which remains the largest number of blocks. Applying this rule iteratively maintains the validity of the condition $\mathsf{NB}_c \leq \sum_{i \neq c} \mathsf{NB}_i + 1$ after each iteration, ensuring that no two blocks having the same color remain at the end of the sequence. According to the previous discussion, we obtain the following algorithmic result.

Proposition 4 *Determining if the problem for colors has one solution takes* $O(\mathsf{NPOS} + \mathsf{NCOL})$ *time. If there is one, computing it can be done in* $O(\mathsf{NPOS} \cdot \mathsf{NCOL})$ *time.*

4.3 Sampling techniques

To speed up the VLNS heuristic, sampling techniques have been also employed. The idea consists in considering the problem with $\lfloor \mathsf{N}_i/\alpha \rfloor$ vehicles of each class i, with $\alpha > 1$. Having obtained a solution to this reduced problem (called sample solution), a solution to the initial problem is built by repeating α times the sample solution and completed by N_i mod α vehicles of each class i.

In practice, sampling with $\alpha = 2$ allows to obtain good solutions for starting the whole local search, in particular, better than greedy algorithms. Indeed, the reduced problem is solved by the VLNS heuristic too and in many cases, optimal solutions for the first objective are obtained (the heuristic finishes by selecting all the vehicles of sequence as movable, i.e., $\mathsf{K} = \mathsf{NPOS}$).

5 Experimental results

The VLNS and VFLS algorithms have been implemented in C ANSI programming language. The experimentations have been realized on a computer with 1.6 GHz Pentium 4 and 1024 Mo of RAM. The VFLS algorithm obtained the best results of the Challenge, whereas the VLNS algorithm was ranked around the sixteen position among the 24 algorithms which were submitted (see [3,4] for more details). In both cases, our algorithms greatly improves the results obtained by RENAULT according to an approach based on simulated annealing. Figures 9 and 10 presented at the end of the paper report results obtained on the RENAULT's benchmarks in the context of ROADEF'2005 Challenge. For each instance, the best result obtained by the two algorithms among 10 trials is given, each trial having a time limit of 10 minutes. The column "EP" denotes the number of violations on priority ratio constraints, "ENP" the number of violations on non-priority ratio constraints and "RAF" the number of purges. For example, if the name of an instance ends in _EP_RAF_ENP, then the objectives to minimize are EP, RAF and ENP in lexicographic order.

The VLSN algorithm uses ILOG-CPLEX 9.0 as integer linear programming library. The parameter T-MAX is fixed around 10 seconds and the value of the parameter K, initialized around 70 vehicles, is quickly increased to select from the half to the two thirds of the total number of vehicles. On average, the time spent to solve optimally the restricted ILP by branch-and-bound is about 3 seconds (note that the computation of the linear relaxation is the most time-consuming), which leads approximately to 20 iterations per minute. In comparison, the VFLS algorithm performs several millions of iterations per minute (on some instances, we have observed until 20 millions of iterations per minute).

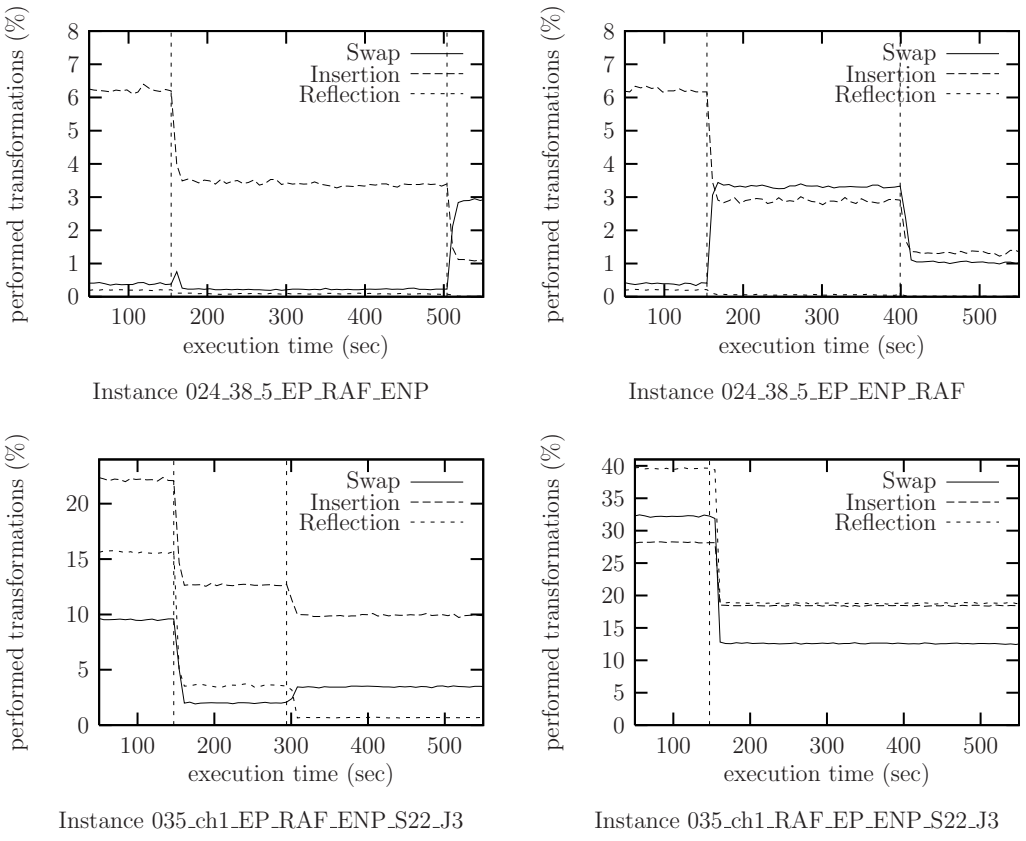

Fig. 7. Percentage of transformations really performed by the VFLS algorithm.

Figure 7 shows the percentage of transformations which are really performed among all the attempted transformations by the VFLS algorithm for four instances with different objectives. The percentage of random shuffles is too small to be mentioned, but this one is not nul. The vertical dotted lines mark the transitions between the different optimization phases during the execution of the VFLS algorithm (phases 2, 3, 4 which are described on Figure 3). For instance 024_38_5_EP_ENP_RAF, the percentage of performed transformations is low (of the order of 1 %), whereas for instance 035_ch1_EP_RAF_ENP_S22_J3, the same percentage is quite high (of the order of 10 %). Figure 8 details the curves during the second optimization phase for objective ENP. The number of performed swaps and insertions are given on the left ordinate, and the number of violations on non priority options on the right ordinate.

Here the major remark is that *in almost all cases, the percentage of performed transformations remains constant while the objective function is lowered (during one optimization phase).* This surprising property explains why no local optimum is met during the descent in practice: since the number of attempted transformations is huge, having a constant percentage of accepted transformations, even small, ensures a great diversification of the search. This fact inspires to us the following question, which calls a deep theoretical study: given one solution, does a sequence of swap, insertion or reflection transformations exists

leading to a solution having a better cost?

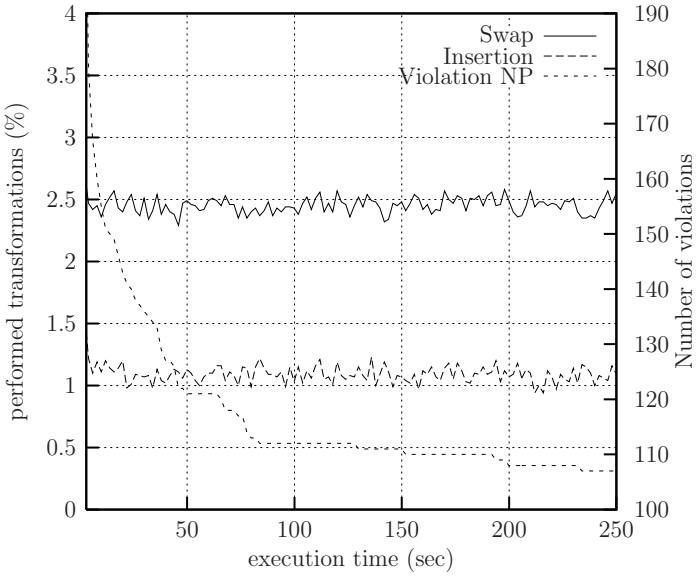

Fig. 8. A zoom on ENP optimization on instance 025_38_1_EP_ENP_RAF.

6 Conclusion

Correctly implemented, a VFLS-like algorithm seems to be the most judicious choice to tackle car sequencing problems. Concise implementations allow to get good results quickly, without using special external libraries (like integer linear programming library). Moreover, this approach remains as robust as very large neighborhood search, because local optima are rarely met in practice.

Nevertheless, the convergence of the VLNS algorithm could be largely speeded up by writing a branch-and-bound procedure and a simplex algorithm dedicated to the problem. In the same way, the hybridation of VFLS and VLNS techniques, useless at this point, seems to be promising according to the new results that we obtained on very large-scale neighborhood search restricted to ratio constraints [7].

Acknowledgements

We are grateful to Dr. Sofiane Oussedik from ILOG company who put at our's disposal the state-of-the-art commercial software ILOG-CPLEX 9.0 for our experimentations, and to Dr. Andrew Makhorin from Moscow Aviation Institute for developing and maintaining the free software GLPK that

we used during the qualification phase of ROADEF'2005 Challenge. We also thank Prof. Van-Dat Cung (GILCO, Grenoble), Mr Youssef Khacheni (RENAULT/DL, Guyancourt), Mr Alain Nguyen (RENAULT/DTSI, Le Plessis Robinson) and Dr. Christine Solnon (LIRIS, Lyon) for their constant encouragements and the many interesting discussions that we had about the car sequencing problem.

References

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

[2] 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.

[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 1ères Journées Francophones de Programmation par Contraintes, JFPC 2005* (C. Solnon, ed), pp. 3–10. Lens, France.

[5] E. Danna, E. Rothberg and C. Le Pape (2005). Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming* 102(1), pp. 71–90.

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

[7] B. Estellon, F. Gardi and K. Nouioua (2006). Solving car sequencing problems by very large-scale neighborhood search. (submitted to *RAIRO Operations Research*)

[8] 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/`

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

[10] 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.

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

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

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

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

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

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

[17] 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.

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

[19] 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.

[20] J.-C. Régin and J.-F. Puget (2002). A filtering algorithm for global sequencing constraints. In *Principles and Practice of Constraint Programming* (G. Smolka, ed), *LNCS* 1330, pp. 32–46. Springer-Verlag, Berlin, Germany.

[21] P. Shaw (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming, CP 1998*

(M.J. Maher, J.-F. Puget, eds), *LNCS* 1520, pp. 417–431. Springer-Verlag, Berlin, Germany.

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

Instances	VLNS			VFLS		
	EP	ENP	RAF	EP	ENP	RAF
022_3_4_EP_RAF_ENP	0	0	66	0	1	31
022_3_4_RAF_EP_ENP	39	1	11	39	1	11
024_38_3_EP_ENP_RAF	4	43	875	4	0	306
024_38_3_EP_RAF_ENP	4	101	336	4	83	249
024_38_5_EP_ENP_RAF	4	62	899	4	34	309
024_38_5_EP_RAF_ENP	4	381	576	4	79	280
025_38_1_EP_ENP_RAF	0	398	809	0	99	720
025_38_1_EP_RAF_ENP	0	2263	285	0	2452	229
039_38_4_EP_RAF_ch1	24	0	291	13	0	131
039_38_4_RAF_EP_ch1	251	0	68	161	0	68
048_39_1_EP_ENP_RAF	4	85	454	0	61	291
048_39_1_EP_RAF_ENP	0	676	210	0	615	175
064_38_2_EP_RAF_ENP_ch1	0	769	173	0	759	112
064_38_2_RAF_EP_ENP_ch1	433	783	63	423	782	63
064_38_2_EP_RAF_ENP_ch2	0	36	45	0	51	34
064_38_2_RAF_EP_ENP_ch2	367	56	27	367	52	27
022_EP_ENP_RAF_S22_J1	0	4	174	0	3	109
022_EP_RAF_ENP_S22_J1	0	135	45	0	144	19
022_RAF_EP_ENP_S22_J1	22	154	13	22	148	13
023_EP_ENP_RAF_S23_J3	55	0	702	48	0	317
023_EP_RAF_ENP_S23_J3	58	76	518	48	8	310
023_RAF_EP_ENP_S23_J3	1347	35	50	1327	31	50
024_V2_EP_ENP_RAF_S22_J1	1129	1162	889	1074	850	430
024_V2_EP_RAF_ENP_S22_J1	1099	2462	609	1074	1068	298
024_V2_RAF_EP_ENP_S22_J1	2186	1679	132	2022	1158	132
025_EP_ENP_RAF_S22_J3	0	3966	890	0	3912	479
025_EP_RAF_ENP_S22_J3	0	5353	254	0	5180	167
025_RAF_EP_ENP_S22_J3	135	6336	126	122	5589	126
028_ch1_EP_ENP_RAF_S22_J2	54	10	95	54	3	79
028_ch1_EP_RAF_ENP_S22_J2	54	271	85	54	124	49
028_ch1_RAF_EP_ENP_S22_J2	102	253	38	98	201	38
028_ch2_EP_ENP_RAF_S22_J3	0	70	3	0	70	6
028_ch2_EP_RAF_ENP_S22_J3	0	72	6	0	71	4
028_ch2_RAF_EP_ENP_S22_J3	0	72	6	0	71	4
029_EP_ENP_RAF_S21_J6	35	2150	265	35	2150	167
029_EP_RAF_ENP_S21_J6	35	2209	277	35	2170	165
029_RAF_EP_ENP_S21_J6	723	2170	52	709	2171	52
035_ch1_EP_ENP_RAF_S22_J3	67	52	55	67	52	49
035_ch1_EP_RAF_ENP_S22_J3	67	62	40	67	64	36
035_ch1_RAF_EP_ENP_S22_J3	156	91	6	156	90	6

Fig. 9. Results on Base A, B and X (first part).

Instances	VLNS			VFLS		
	EP	ENP	RAF	EP	ENP	RAF
035_ch2_EP_ENP_RAF_S22_J3	385	341	206	385	341	205
035_ch2_EP_RAF_ENP_S22_J3	385	351	187	385	351	187
035_ch2_RAF_EP_ENP_S22_J3	652	671	7	651	671	7
039_ch1_EP_ENP_RAF_S22_J4	0	29	876	0	29	117
039_ch1_EP_RAF_ENP_S22_J4	0	558	105	0	89	78
039_ch1_RAF_EP_ENP_S22_J4	45	337	55	45	96	55
039_ch3_EP_ENP_RAF_S22_J4	0	0	244	0	0	197
039_ch3_EP_RAF_ENP_S22_J4	0	262	206	0	146	189
039_ch3_RAF_EP_ENP_S22_J4	214	704	59	214	671	59
048_ch1_EP_ENP_RAF_S22_J3	0	0	294	0	0	200
048_ch1_EP_RAF_ENP_S22_J3	0	497	215	0	378	161
048_ch1_RAF_EP_ENP_S22_J3	124	761	64	115	670	64
048_ch2_EP_ENP_RAF_S22_J3	3	0	381	3	0	337
048_ch2_EP_RAF_ENP_S22_J3	3	1131	123	3	1029	93
048_ch2_RAF_EP_ENP_S22_J3	306	1212	75	282	1180	58
064_ch1_EP_ENP_RAF_S22_J3	0	4	519	0	0	182
064_ch1_EP_RAF_ENP_S22_J3	0	270	172	0	187	130
064_ch1_RAF_EP_ENP_S22_J3	111	359	62	95	288	62
064_ch2_EP_ENP_RAF_S22_J4	0	69	134	0	69	130
064_ch2_EP_RAF_ENP_S22_J4	0	78	136	0	69	130
064_ch2_RAF_EP_ENP_S22_J4	52	190	31	52	178	31
022_RAF_EP_ENP_S49_J2	2	3	12	2	3	12
023_EP_RAF_ENP_S49_J2	0	76	246	0	66	192
024_EP_RAF_ENP_S49_J2	31	42	536	0	6	337
025_EP_ENP_RAF_S49_J1	11	224	694	0	160	407
028_CH1_EP_ENP_RAF_S50_J4	42	421	108	36	370	94
028_CH2_EP_ENP_RAF_S51_J1	0	0	4	0	0	3
029_EP_RAF_ENP_S49_J5	0	28	156	0	42	110
034_VP_EP_RAF_ENP_S51_J1_J2_J3	0	643	95	0	586	55
034_VU_EP_RAF_ENP_S51_J1_J2_J3	8	44	93	8	8	87
035_CH1_RAF_EP_S50_J4	10	0	5	10	0	5
035_CH2_RAF_EP_S50_J4	56	0	6	56	0	6
039_CH1_EP_RAF_ENP_S49_J1	0	471	88	0	239	69
039_CH3_EP_RAF_ENP_S49_J1	0	271	249	0	30	231
048_CH1_EP_RAF_ENP_S50_J4	1	1064	224	0	1044	196
048_CH2_EP_RAF_ENP_S49_J5	31	1065	116	31	1116	76
064_CH1_EP_RAF_ENP_S49_J1	61	135	278	61	29	187
064_CH2_EP_RAF_ENP_S49_J4	0	0	60	0	0	37
655_CH1_EP_RAF_ENP_S51_J2_J3_J4	0	0	45	0	0	30
655_CH2_EP_RAF_ENP_S52_J1_J2_S01_J1	153	0	50	153	0	34

Fig. 10. Results on Base A, B and X (second part).