An efficient implementation of a VNS/ILS heuristic for a real-life car sequencing problem

Celso C. Ribeiro[1], Daniel Aloise[2], Thiago F. Noronha[2], Caroline Rocha[1], and Sebastián Urrutia[2]

[1] Department of Computer Science, Universidade Federal Fluminense
Rua Passo da Pátria 156
Niterói, RJ 24210-240, Brazil.
celso@inf.puc-rio.br, crocha@ic.uff.br

[2] Department of Computer Science, Catholic University of Rio de Janeiro
Rua Marquês de São Vicente 225
Rio de Janeiro, RJ 22453-900, Brazil.
aloise@inf.puc-rio.br, tfn@inf.puc-rio.br, useba@inf.puc-rio.br

Abstract. The car sequencing problem consists in sequencing a given set of cars to be produced in a single day. We address one of the variants of this problem, in which the objective of minimizing the number of violations of assembly constraints has a stronger weight than the minimization of the number of paint color changes. In this paper, we present and describe in details a VNS/ILS approach for approximately solving this problem. Computational results on real-life test instances are reported. The work presented in this paper obtained the second prize in the challenge ROADEF'2005 sponsored by Renault.

Keywords: Car sequencing problem, metaheuristics, VNS, ILS, heuristics.

1 Problem statement

In car factories, different configurations of the same car are set up using the same production line. The production line can be considered as a linear manufacturing process composed by three plant shops: body shop, paint shop, and assembly line. Customer orders are transmitted daily in real-time to the factories. The scheduling of the production line is divided in two main steps: (1) assign a production day to each ordered vehicle, according to production line capacities and delivery dates, and (2) schedule the cars in the production line for each day, satisfying as many requirements as possible of the plant shops: body shop constraints, paint shop constraints, and assembly line constraints. The first step above can be well-solved by linear programming [5]. This work addresses the second step, which is called the Car Sequencing Problem (CSP). Since the body shop does not set requirements for the daily schedule, we focus only on the requirements of the paint shop and of the assembly line. This variant of the car sequencing problem was proposed by Renault in the challenge ROADEF'2005 [5].

In the context of the CSP, one must schedule a set of cars to be produced in a specific day D. For each of these cars, we are given a unique identifier, a paint color, and two subsets of high priority level and low priority level ratio constraints associated with it. Given a set of n cars to be produced, a sequence of previously scheduled cars remaining from production day $D-1$, and the ratio constraints, CSP consists in scheduling the n cars to be produced minimizing the paint shop and assembly line objectives:

Paint shop objective: The paint shop attempts to minimize the consumption of paint solvent, which is used to wash spray guns each time the paint color is changed between two consecutive vehicles. Therefore, the paint shop requires the minimization of the number of paint color changes in the sequence of scheduled vehicles. Paint color batches (sequences of cars that have to be painted with the same color) have an upper bound on their size, since spray guns have to be washed regularly, even if there is no paint color change in the sequence of vehicles. This limitation is a hard constraint, in the sense that the presence of a paint color batch with more cars than the upper bound on the batch size makes the solution infeasible.

Assembly line objective: To smooth the workload on the assembly line, vehicles that require special assembly operations have to be evenly distributed along the sequence of processed cars. These vehicles are considered hard to assemble and may not exceed a given quota over any sequence of vehicles. This requirement is modeled by a ratio constraint N/P. Ratio constraints are associated with car characteristics which require extra operations in the assembly line (such as sunroof and air conditioning). A ratio constraint N/P means that at most N cars in any consecutive sequence of P cars can be associated with this constraint. For instance, if $N/P = 3/5$, there must be no more than three constrained cars in any sequence of five vehicles. If $N/P = 1/P$, any two constrained cars must be separated by at least $P-1$ consecutive non-constrained vehicles. For instance, if a ratio constraint is described by $N/P = 1/5$, then `X____X` is an acceptable sequence, where the symbol `X` represents a vehicle associated with this ratio constraint and the symbol `_` denotes a car not concerned by this constraint. The problem may be over-constrained, i.e., it may be impossible to satisfy all ratio constraints when a production day is scheduled. Hence, the objective is to minimize the number of violations of ratio constraints. We refer to [1, 5] for more details about the calculation of violations in ratio constraints.

In summary, there are three objectives: (a) minimization of the number of paint color changes (PCC), (b) minimization of the number of violations of high priority ratio constraints (HPRC), and (c) minimization of the number of violations of low priority ratio constraints (LPRC). These objectives are usually conflicting. However, they can be considered with different priority levels, depending on the factory where the vehicles are being produced. Renault proposes to tackle the problem using a weighted sum method that groups the three objectives into one single objective function, by assigning different weights to each objective, according with their priority levels [5]. In this work, we address the problem variant in which the objective of minimizing the number of violations of assembly constraints has the highest weight, where the objective function is given by $1,000,000$ multiplied by the number of violations of HPRC plus $1,000$ multiplied by the number of violations of LPRC plus the number of PCC.

We propose a four-phase approach for solving the above variant of CSP. In the first phase, a constructive heuristic is used to build an initial solution. In the second phase, the first objective is minimized. Next, we seek a solution minimizing the second objective that does not deteriorate the first. Finally, in the last phase the algorithm searches for a solution that minimizes the third objective without deteriorating the two first objectives. This strategy corresponds to the following scheme:

1. Use a constructive heuristic to find an initial solution.
2. Use an ILS heuristic to optimize the first objective.
3. Use a VNS heuristic to optimize the second objective.
4. Use a repair heuristic to make the current solution feasible.
5. Use a VNS heuristic to optimize the third objective.

In this paper, we propose and describe in details a VNS/ILS heuristic for approximately solving the above variant of CSP. The remaining of this paper is organized as follows. A constructive heuristic is given in Section 2. Neighborhood definitions and local search algorithms are described in Section 3. Improvement heuristics based on the paradigms of the Variable Neighborhood Search (VNS) [3] and Iterated Local Search (ILS) [4] metaheuristics to optimize the first, second, and third objectives are presented, respectively, in Sections 4, 5, and 6. Computational results are reported in Section 7 and concluding remarks are draw in the last section.

2 Greedy heuristic

The constructive heuristic attempts to build an initial solution minimizing the number of violations on high priority ratio constraints.

We use the same formalization proposed in [2] for the classic version of the car sequencing problem, which consists in minimizing the number of violations of a single set of ratio capacity constraints.

2.1 Formalization

The car sequencing problem can be defined as tuple (V, O, p, q, r), where $V = \{v_1, \ldots, v_n\}$ is the set of cars to be scheduled and $O = \{o_1, \ldots, o_m\}$ is the set of possible options required by any car. The ratio capacity constraints associated to each option $o_j, j = 1, \ldots, m$, are defined by the functions $p : O \rightarrow \mathbb{N}$ and $q : O \rightarrow \mathbb{N}$, which determine that each sequence with $q(o_j)$ consecutive cars can have at most $p(o_j)$ cars requiring option $o_j, j = 1, \ldots, m$. For any $i = 1, \ldots, n$ and $j = 1, \ldots, m$, we define $r(v_i, o_j) = 1$ if option o_j is associated with car v_i, $r(v_i, o_j) = 0$ otherwise.

A sequence of k cars is denoted by $\pi =< v_{i_1}, v_{i_2}, \ldots, v_{i_k} >$. The number of cars in a sequence π is denoted by $|\pi|$. The concatenation of two sequences π_1 and π_2, represented by $\pi_1|\pi_2$, is the sequence formed by the cars in π_1 immediately followed by those in π_2. For any $j = 1, \ldots, m$, the number of cars that require an option o_j in a sequence $\pi =< v_{i_1}, v_{i_2}, \ldots, v_{i_k} >$ is given by

$$r(\pi, o_j) = \sum_{l=1}^{k} r(v_{i_l}, o_j).$$

For any $j = 1, \ldots, m$, the number of cars that require an option o_j in the whole set V of vehicles to be scheduled is given by

$$r(V, o_j) = \sum_{i=1}^{n} r(v_i, o_j).$$

The cost of a sequence $\pi =< v_{i_1}, v_{i_2}, \ldots, v_{i_k} >$ is the number of violations of ratio capacity constraints, given by

$$cost(\pi) = \sum_{j=1}^{m} \sum_{\substack{\pi' \\ |\pi'|=q(o_j) \\ \pi' \subseteq \pi}} \max\{0, r(\pi', o_j) - p(o_j)\}.$$

For any $j = 1, \ldots, m$, the utilization rate [7] of an option o_j is formally expressed by

$$utilizationRate(o_j) = \left(\frac{r(V, o_j)}{n}\right) \Big/ \left(\frac{p(o_j)}{q(o_j)}\right).$$

Thus, the overall utilization rate of an option o_j is given by the ratio between the number of cars requiring this option and the total number of cars to be scheduled, divided by the corresponding value of the ratio constraint. Therefore, if the utilization rate of option o_j is greater than 1, the demand for cars with this option is greater than the capacity provided for its production and at least one violation will inevitably occur in the solution. Otherwise, if the utilization rate is very small and close to 0, there is a small demand for option o_j compared with the capacity of the assembly line.

2.2 Greedy criterion

The heuristic iteratively builds the initial solution, by choosing the next car to be scheduled at the end of the partial sequence accordingly with a greedy criterion [2]. This straightforward greedy criterion consists in choosing, at each iteration, the car that induces the smallest number of new violations. Given a partial sequence π, the next car $v \in V\backslash < \pi >$ to be selected is the one which minimizes

$$\sum_{j=1}^{m} \left(r(v, o_j) \times \sum_{l=p(o_j)+1}^{q(o_j)} \min\{1, \max\{0, r(lastCars(\pi| < v >, l), o_j) - p(o_j)\}\} \right),$$

where

$$lastCars(\pi', k) = \begin{cases} \text{sequence with the last } k \text{ cars of } \pi', & \text{if } |\pi'| \geq k, \\ \pi', \text{ otherwise.} \end{cases}$$

However, there are usually many candidate cars which minimize the number of new induced violations. Therefore, the greedy criterion may be enriched by a tie breaking criterion. This criterion is more elaborated, since it encourages a more homogeneous distribution of the required options among the cars already in the partial sequence and those yet unscheduled. The distribution is based on the following idea: if the average number of cars demanding an option is lower in the partial sequence π than in the whole set V of vehicles to be scheduled, then these cars should be encouraged to enter in the sequence, and vice versa. Therefore, the next car $v \in V\backslash < \pi >$ to be scheduled is the one maximizing

$$\eta_1(v, \pi) = \sum_{j=1}^{m} \left\{ (r(v_i, o_j) = 0) \text{ XOR } \left(\frac{r(V, o_j)}{n} > \frac{r(\pi, o_j)}{|\pi|} \right) \right\}.$$

In case of ties for the first position of π, the first scheduled car is randomly chosen among those with the largest number of required options, since the expression above cannot be used when $|\pi| = 0$.

It is not rare that ties still remain after the application of this criterion. In this case, a second criterion is added, favoring cars that require options with higher utilization rates. The utilization rates are now dynamic in the sense that they are updated whenever a new car is added in the sequence. The dynamic utilization rate of an option o_j, for $j = 1, \ldots, m$, regarding the partial sequence π of cars already scheduled is given by

$$dynamicUtilizationRate(o_j, \pi) = \left(\frac{r(V\backslash < \pi >, o_j)}{|V\backslash < \pi > |} \right) \Big/ \left(\frac{p(o_j)}{q(o_j)} \right).$$

The second tie breaking criterion based on the sum of the dynamic utilization rates leads to the selection of the car $v \in V \backslash < \pi >$ which maximizes

$$\eta_2(v, \pi) = \sum_{j=1}^{m} r(v_i, o_j) \cdot dynamicUtilizationRate(o_j, \pi).$$

If ties still exist after the application of the last above criterion, one car is randomly chosen among those minimizing the number of violations of LPRC.

3 Neighborhood search

3.1 Moves and local search

The moves applied by the local search procedures are based on two neighborhood structures:

- Car exchange moves: the positions of two cars are exchanged.
- Car insertion moves: a car is moved from its current position to a new specific position and the positions of all cars in between the new and the former positions are shifted.

The local search procedure proposed in this work is neither first improvement, nor best improvement. The neighborhood is divided into a number of sub-neighborhoods. Each sub-neighborhood contains all neighbors that are generated by exchanging or shifting a different car. Therefore, there are n sub-neighborhoods. The neighborhood search consists in iteratively evaluating each sub-neighborhood, from first to last. At each iteration, the search moves to the best non-deteriorating solution in the corresponding sub-neighborhood, even if the latter has the same cost of the current solution

Figure 1 shows the pseudo-code of the local search procedures, where $cost(\pi)$ denotes the cost of a solution π and $move(\pi, i, j)$ means either a car exchange move between cars in positions i and j or a car insertion move of the car in position i into position j, depending on the neighborhood being used. The loop in lines 1–20 is performed until no further improvement can be found in the current solution. The cost ϕ of the current solution is initialized in line 2. For each car $i = 1, \ldots, n$, the loop in lines 3–19 handles the moves involving car i. Since only non-deteriorating moves can be accepted, variable Δ^* containing the value of the best acceptable move is initialized with zero in line 4. The set L of best moves involving car i is initialized in line 5. The loop 6–14 investigates all moves associated with car i. The reduction Δ in the cost of the current solution if $move(\pi, i, j)$ is performed is computed in line 7. If the non-deteriorating

$move(\pi, i, j)$ is better than the best move identified so far, the latter is updated in lines 9–10. If the non-deteriorating $move(\pi, i, j)$ is as good as the best move identified so far, the new move is appended to the list of best moves in line 12. In case at least one non-deteriorating move involving car i was found, then the move to be applied to the current solution is randomly selected from the set of best moves in line 16 and applied to it in line 17. The solution obtained by local search is returned in line 19.

```
procedure LocalSearch(π);
1  repeat
2      φ ← cost(π)
3      for i = 1, ..., |π| do
4          Δ* ← 0
5          L ← ∅
6          for j = 1, ..., |π| do
7              Δ ← cost(move(π, i, j)) − cost(π)
8              if Δ < Δ* then
9                  L ← {j}
10                 Δ* ← Δ
11             else if Δ = Δ* then
12                 L ← L ∪ {j}
13             end-if
14         end-for
15         if L ≠ ∅ then
16             Choose k ∈ L at random
17             π ← move(π, i, j)
18         end-if
19     end-for
20 until φ = cost(π)
21 return π
end LocalSearch;
```

Fig. 1. Pseudo-code of the general local search procedure.

Data structures play a very important role in move evaluations. The use of efficient data structures allows performing moves in both neighborhoods in time $O(m)$ and, consequently, the local search procedures can both be very fast.

3.2 Data structures

Move evaluations are performed much more often than solution updates. Therefore, we decided to implement data structures optimizing the complexity of move evaluations.

The variation on the number of paint color changes can be easily evaluated in time $O(1)$, since it amounts to checking the colors of adjacent cars. The evaluation of the feasibility of the solution obtained by a given move can also be done in time $O(1)$.

Each solution is represented as a vector that stores the car sequence π. To be able to quickly evaluate the number of violations of the ratio constraints, we also keep three additional auxiliary matrices associated to the sequence π. Each of their rows is associated with an option $o_j \in O, j = 1, \ldots, m$. The first matrix, M_1, stores in position $[j, i]$ the number of cars which require option o_j in the subsequence of $q(o_j)$ cars starting at position i in the sequence π. The second matrix, M_2, stores in position i of row j the number of subsequences of $q(o_j)$ cars starting at positions 1 up to i in which the number of cars that require this option is greater than $p(o_j)$. The third matrix, M_3, is similar to M_2, except by the fact that it stores in each position of the row j the number of subsequences in which the number of cars which require an option o_j is greater than or equal to $p(o_j)$. This matrix is used to count the number of subsequences in which the addition of a car requiring option o_j increases the number of violations of the ratio constraint associated with that option. Figure 2 presents a single example for $n = 10$ and $m = 1$, with $p(o_1) = 1$ and $q(o_1) = 4$.

1	2	3	4	5	6	7	8	9	10
X	_	X	X	_	_	_	X	_	X

M1 = [3 2 2 1 1 1 2 2 1 1]
M2 = [1 2 3 3 3 3 4 5 5 5]
M3 = [1 2 3 4 5 6 7 8 9 10]

Fig. 2. Auxiliary matrices M_1, M_2, and M_3 (an "X" in the sequence denotes a car requiring option 1, otherwise the symbol "_" is used).

Let us now examine how these matrices contribute to the efficient evaluation of exchange moves. The number of violations of a ratio constraint associated with an option $o_j \in O$, for a given $j = 1, \ldots, m$, can only change after an exchange move if one of the cars has this option, but not the other. Let us suppose that these cars are v_{i_a} and v_{i_b}, respectively. Notice that a and b are the corresponding positions of these cars before the move.

The decrease in the number of violations of the ratio constraint associated with option o_j due to the replacement of car v_{i_a} by v_{i_b} is given by

$$\Delta_j^1 = \begin{cases} M_2[j][a] - M_2[j][a - q(o_j)], \text{ if } a - q(o_j) > 0; \\ M_2[j][a], \text{ otherwise.} \end{cases}$$

The value Δ_j^1 gives the number of subsequences of π with size $q(o_j)$ that contain v_{i_a} and that violate the ratio constraint associated with option o_j. Analogously, the increase in the number of violations of the ratio constraint associated with option o_j due to the replacement of car v_{i_b} by v_{i_a} is given by

$$\Delta_j^2 = \begin{cases} M_3[j][b] - M_3[j][b - q(o_j)], \text{ if } b - q(o_j) > 0; \\ M_3[j][b], \text{ otherwise.} \end{cases}$$

In this case, Δ_j^2 gives the number of subsequences of π with size $q(o_j)$ that contain v_{i_b} and that will have a new violation after the exchange of the positions of cars v_{i_a} and v_{i_b}.

Therefore, the variation on the number of violations of the ratio constraint associated with option o_j is given by $\Delta_j = \Delta_j^2 - \Delta_j^1$. Special care with these computations is required when the distance between positions a and b is smaller than $q(o_j)$. The total variation for all ratio constraints is given by $\sum_{j=1}^{m} \Delta_j$ and can be computed in time $O(m)$.

Only matrix M_1 is necessary to perform the evaluation of insertion moves. In the proposed local search algorithm, one searches for the best insertion move considering each car in the sequence $\pi =< v_{i_1}, v_{i_2}, \ldots, v_{i_n} >$ at a time. Suppose that we are considering the insertion of car v_{i_a} at position b. First, we remove this car from π and update each row of M_1 as follows:

$$M_1[j][k] = \begin{cases} M_1[j][k], \text{ for } k = 1, \ldots, a - q(o_j); \\ M_1[j][k] - r(v_{i_a}, o_j) + r(v_{i_{k+q(o_j)-1}}, o_j), \text{ for } k = a - q(o_j) + 1, \ldots, a - 1; \\ M_1[j][k+1], \text{ for } k = a, \ldots, n-1, \end{cases}$$

All positions of matrix M_1 associated to subsequences that contained the car v_{i_a} to be moved (subsequences starting at positions $k = a - (q(o_j) - 1), \ldots, a - 1$) should be updated. The new values $M_1[j][k]$ are obtained by subtracting one unity from the old value when car v_{i_a} requires option o_j and by adding one unity when the shifted car $v_{i_{k+q(o_j)-1}}$ requires that option.

The evaluation of insertion moves is based on the following idea. Suppose that the variation on the number of violations has already been computed for the insertion of car v_{i_a} in the first possible position. We now show that if the variation on the number of violations associated to the insertion of this car in a given position was already computed, then the evaluation of the variation for the insertion at the next position is straightforward. We notice that for any option $o_j \in O, j = 1, \dots, m$, the insertion of car v_{i_a} at position b affects $q(o_j)$ subsequences of cars in the current solution: $q(o_j) - 1$ subsequences are changed and a new subsequence starting at position b is created.

Let us denote by b_0 the first possible position of π (corresponding to the position of the first car of the current production day). The variation on the number of violations of ratio constraints due to the insertion of car v_{i_a} at position b_0 is given by the number of violations in the new subsequence starting at position b_0 plus the variation on the number of violations in the modified subsequences. The number of violations of the ratio constraint associated to option $o_j \in O$, for a given $j = 1, \dots, m$, in the new subsequence is given by

$$\Delta^1_{j,b_0} = \max\{0, M_1[j][b_0] + r(v_{i_a}, o_j) - r(v_{i_{b_0+q(o_j)-1}}, o_j)) - p(o_j)\}$$

while the variation due to the modified subsequences is given by

$$\Delta^2_{j,b_0} = \sum_{k=b_0-q(o_j)+1}^{b_0-1} B_k,$$

with

$$B_k = \begin{cases} -1, \text{ if } r(v_{i_a}, o_j) < r(v_{i_{k+q(o_j)-1}}, o_j) \text{ and } M_1[j][k] > p(o_j); \\ +1, \text{ if } r(v_{i_a}, o_j) > r(v_{i_{k+q(o_j)-1}}, o_j) \text{ and } M_1[j][k] \geq p(o_j); \\ 0, \text{ otherwise.} \end{cases}$$

Then, the variation on the number of violations of ratio constraints associated to the insertion of v_{i_a} at position b_0 is given by $\sum_{j=1}^{m} \Delta^1_{j,b_0} + \Delta^2_{j,b_0}$. The variations associated to insertions in the other positions are sequentially computed. For any subsequent position b, the value of $\Delta^1_{j,b}$ is computed as before, while the value of $\Delta^2_{j,b}$ is simply obtained by taking advantage of the value previously computed for position $b-1$. The value $\Delta^2_{j,b}$ associated to the insertion of car v_{i_a} at position $b = b_0 + 1, \dots, n$ is given by

$$\Delta^2_{j,b} = \Delta^2_{j,b-1} - \delta_{b-q(o_j)} + \delta_{b-1},$$

with

$$\delta_{b-q(o_j)} = \begin{cases} -1, \text{ if } r(v_{i_a}, o_j) < r(v_{i_{b-1}}, o_j) \text{ and } M_1[j][b-q(o_j)] > p(o_j); \\ +1, \text{ if } r(v_{i_a}, o_j) > r(v_{i_{b-1}}, o_j) \text{ and } M_1[j][b-q(o_j)] \geq p(o_j); \\ 0, \text{ otherwise,} \end{cases}$$

and

$$\delta_{b-1} = \begin{cases} -1, \text{if } r(v_{i_a}, o_j) < r(v_{i_{b+q(o_j)-2}}, o_j) \text{ and } M_1[j][b-1] > p(o_j); \\ +1, \text{if } r(v_{i_a}, o_j) > r(v_{i_{b+q(o_j)-2}}, o_j) \text{ and } M_1[j][b-1] \geq p(o_j); \\ 0, \text{ otherwise.} \end{cases}$$

3.3 Complexity

Both local search algorithms, using either exchange or insertion moves, follow the same principle. First, all moves involving a given car are evaluated and the best one is selected. If the latter is an improving move or leads to a neighbor solution with the same cost of the current one, then it is performed and the current solution is updated. Otherwise, the next car is considered and, as before, all associated moves are evaluated. The procedure stops after performing a full search in which all cars are considered and no improving move is found. We notice that at most n feasible moves are evaluated for each car, while only one solution update is performed. The data structures described in the previous section are used in these procedures.

We first consider the local search algorithm using exchange moves. The evaluation of each exchange move can be performed in time $O(m)$. Once a move involving cars v_{i_a} and v_{i_b} is chosen, matrices M_1, M_2, and M_3 have to be updated. At most $2q(o_j)$ positions of each row j of matrix M_1 have to be updated, corresponding to all subsequences to which the exchanged cars belong. Then, matrix M_1 can be updated in time $O(mq_{max})$, where $q_{max} = max_{o \in O}\{q(o)\}$. Updating matrices M_2 and M_3 requires time $O(mn)$, because for each row $j = 1, \ldots, m$ all entries from column $\min\{a, b\} - (q(o_j) - 1)$ to the last should be updated.

Each local search iteration involves the evaluation of $O(n^2)$ moves. Since the evaluation of each of them can be performed in time $O(m)$, all evaluation operations take time $O(mn^2)$. Because only one update has to be performed for each car (corresponding to the best move for this car), all update operations take time $O(mn^2)$. Therefore, each local search iteration in the `CarExchange` neighborhood is performed in time $O(mn^2)$.

Similar arguments show that each local iteration in the `CarInsertion` neighborhood can also be performed in time $O(mn^2)$.

3.4 Quick neighborhood investigation: non-improving moves

The following observations concern relevant implementation issues that are quite important to speedup neighborhood searches and move evaluations:

Identical cars: Two cars are said to be identical if they have to be painted with the same color and have the same options (in which case they are subject to the same ratio constraints). An exchange move involving two identical cars does not change the cost function. Therefore, the local search procedures do not consider exchange moves of identical cars. Using the appropriate data structures, one can check in time $O(1)$ if two cars are identical.

Cars with the same options: Given an option associated to a given ratio constraint, an exchange move involving two cars having both or none of them this option do not change the number of violations of the associated constraint. Therefore, the term of the objective function corresponding to the variation on the number of violations of this ratio constraint does not have to be evaluated.

Cars not involved in violations: An exchange move of two cars that are not involved in violations of a given constraint does not decrease the number of violations of this constraint. Therefore, exchange moves between pairs of cars not involved in violations of any constraint are discarded (and not even evaluated) when the neighborhood is sought for moves strictly decreasing the number of violations of ratio constraints.

4 First objective optimization

This section concerns the optimization of the first objective of CSP, regarding the minimization of the number of violations of the high priority ratio constraints.

Since the constructive heuristic obtains an initial solution investigating exclusively the ratio constraints, the latter may be infeasible by violating paint batch limit sizes). The algorithm presented in this section can also explore infeasible regions, leaving feasibility Restoration for a later moment.

The proposed algorithm is based on the Iterated Local Search (ILS) metaheuristic [4]. The basic strategy of this metaheuristics consists in successively applying perturbations and local search to the current solution. Figure 3 presents the pseudo-code of the basic ILS algorithm. We denote by `Perturbation(s)` and `Local_Search(s)` the generic perturbation and local search procedures applied to a solution s, respectively.

ILS starts from a local optimum s^* built in lines 1 to 2. Then, new local optima s' are iteratively obtained by the combination of a perturbation move applied to the current local optimum s^* with a local search procedure applied to the neighbor solution obtained by perturbation. The current local optimal solution s^* is replaced by s' accordingly to the defined acceptance criterion. The loop is interrupted in line 3 when a given stopping condition is met by the algorithm.

```
procedure ILS();
  Build an initial solution s0 using a constructive heuristic.
  s* ← Local_Search(s0)
  while stopping condition is not satisfied do
      s' ← Perturbation(s*)
      s' ← LocalSearch(s')
      if s* ← AcceptanceCriterion(s*, s') then s* ← s'
  end-while
  return s*
end ILS;
```

Fig. 3. Pseudo-code of the ILS metaheuristic.

4.1 Perturbations

The perturbations applied within the ILS algorithm designed to optimize the first objective consist in removing a given number of cars from the current solution and, subsequently, reinserting them into the latter accordingly with a greedy criterion.

The cars to be removed are selected from those involved in violations of high priority ratio constraints. Each car is reinserted into the partial solution in the position corresponding to the smallest increase in the first objective. Figure 4 illustrates an example of the use of this perturbation.

4.2 Local search

The local search procedure is based on the scheme presented in Section 3 for the `CarExchange` neighborhood, with one additional improvement.

There may exist a car exchange improvement move with respect to the first objective only if at least one the cars to be exchanged is involved in violations of the ratio constraints considered. Therefore, a *fast local search* procedure was developed considering as exchange moves only those for which at least one of the cars to be exchanged is involved in violations. This search performs efficiently whenever there are not too many cars involved in violations in the current solution. This fast local search was used whenever the fraction of cars involved in violations was smaller than 40%.

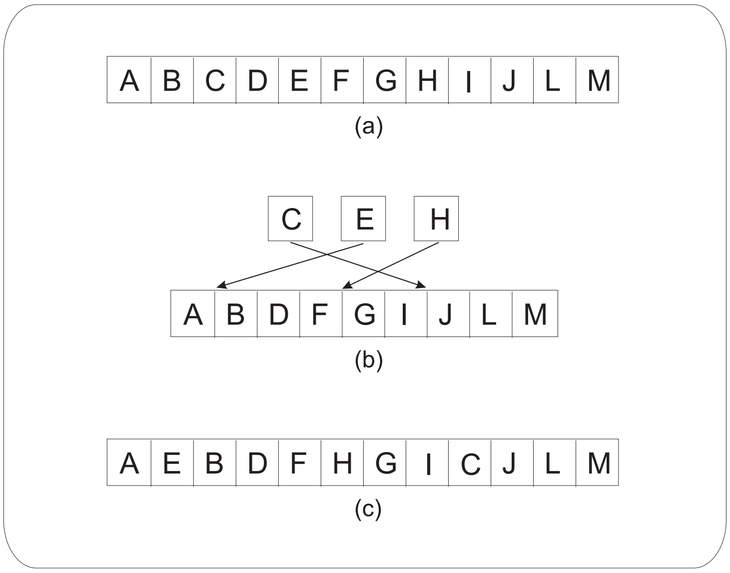

Fig. 4. Illustration of the use of perturbations in the ILS algorithm for optimizing the first objective: (a) original sequence with all cars scheduled (each letter represents a different car), (b) three cars are removed and subsequently reinserted into the partial sequence in the positions pointed by the arrows, and (c) the resulting solution.

4.3 Intensification

This phase was added to the basic ILS framework. It consists in the consecutive application of the two local search procedures whenever no improvement was obtained in the current solution after a given number of iterations. It begins by a local search in the car insertion neighborhood, followed by a local search in the car exchange neighborhood.

This phase may also de viewed as a diversification procedure because, besides optimizing in the neighborhood of the current solution, non-improving moves with the same cost are also allowed and randomly chosen. Therefore, it is possible to visit many distinct solutions with the same cost along the same application of local search, extending the set of visited solutions. Therefore, after that, the current solution may be very different from that used at the beginning of the procedure.

4.4 Restarts

A restart phase was also added to basic ILS algorithm. It is applied whenever the current solution is not improved after a given number of iterations, aiming to obtain new initial solutions for the algorithm. The new starting point for the search is obtained either by a stronger perturbation in the current solution that removes a larger number of cars, or by the application of the constructive heuristic. The first option is preferred, since no information will be kept from the previous iterations when we restart from scratch with the solution obtained by the constructive heuristic.

After a given number of stronger perturbations without improvement in the best known solution, the constructive heuristic is applied and a new starting point is obtained. If no improvement in the current solution is obtained after a given number of iterations and its cost is bigger than the cost of the best known solution, then the latter is assumed as the new current solution and the search resumes from this point.

The number of restarts is also used to define the stopping criterion for the algorithm.

4.5 Stopping criteria

The choice of appropriate stopping criteria is critical in the context of the strategy conceived for the problem and also due to the time limit on the available computation time. The algorithm should be able to quit the optimization of the first objective and switch to the next, avoiding loosing too much time in the former.

The restart phase guided the definition of the first stopping criterion. Whenever no best overall solution is found after three consecutive restarts performed by the constructive heuristic, the algorithm that optimizes the first objective halts and the algorithm for the optimization of the second objective is triggered.

The second stopping criterion is based on the computation time, since an overall time limit was imposed to the whole algorithm. Even if the first criterion was not met, the optimization of the second objective is activated after a given fraction of the time limit was spent on the first objective.

Finally, the third and most obvious stopping criterion is met if there are no violations of high priority ratio constraints in the current solution.

4.6 Algorithm

We now present an overview of the algorithm for the optimization of the first objective.

The algorithm starts from an initial solution built by the constructive heuristic described in Section 2. At each iteration, the current solution is perturbed and a local search (fast or complete, depending on the number of cars involved on violations of high priority ratio constraints) in the car exchange neighborhood is applied. The resulting solution replaces the current if it is at least as good as the latter (acceptance criterion) and the best known solution is possibly updated. If the current solution is not improved after a given number of iterations α, the intensification phase is performed. The restart phase is applied whenever no improvement is obtained after a given number of iterations β and the current solution has the same cost of the best known solution. The algorithm is halted when one of the stopping criteria is satisfied.

Figure 5 presents the pseudo-code of the above algorithm for the optimization of the first objective. In line 1, the best known solution π^* and the current solution π are initialized with the solution obtained by the constructive heuristic. The loop in lines 2–25 is performed until one of the stopping criteria is met. Each iteration of the ILS algorithm starts in line 3 with a perturbation applied to the current solution π, leading to an intermediary solution π'. Local search in the car exchange neighborhood is applied to π' in line 4. The new solution π' is accepted in line 6 to replace the current solution π if it is at least as good as the latter. The best known solution π^* is updated in line 9 if it is better than the incumbent. If no improvement was obtained in the current solution π after α iterations, the intensification phase is performed in line 12 and the current solution is possibly updated in line 14. The best known solution π^* is updated in line 17 if it is better than the incumbent. If no improvement in the current solution π is obtained after β iterations (with $\alpha < \beta$), a restart is performed if the cost of π is equal to the cost of the best known solution. Otherwise, the latter is restored as the current solution (line 23).

5 Second objective optimization

This section presents the algorithm for the optimization of the second objective, based on the VNS (Variable Neighborhood Search) metaheuristic proposed by Hansen and Mladenović [3], which is briefly described below.

The VNS metaheuristic is based on a systematic change of neighborhoods $N_1, \ldots, N_{k_{max}}$ where perturbations are applied. The pseudo-code in Figure 3 shows that each iteration basically consists in a perturbation phase (line 5), followed by a local search procedure (line 6). The search starts by the exploration of smaller neighborhoods (line 3). The search region is enlarged whenever an improving solution was not found within the current neighborhood (line 10).

The proposed algorithm for the optimization of the second objective starts with a solution obtained by the algorithm described in Section 4 for the optimization of the first objective. The first (number of violations of high priority

```
procedure ILS_HPRC(π0);
  π ← π0, π* ← π0
  while stopping criteria not met do
      π' ← Perturbation(π)
      π' ← LocalSearch(π')
      if cost(π) ≥ cost(π') then
          π ← π'
      end-if
      if cost(π) < cost(π*) then
          π* ← π
      end-if
      if the number of iterations since the last improvement
      in π is equal to α then
          π' ← Intensification(π)
          if cost(π) ≥ cost(π') then
              π ← π'
          end-if
          if cost(π) < cost(π*) then
              π* ← π
          end-if
      end-if
      if the number of iterations since the last improvement
      in π is equal to β then
          if cost(π) = cost(π*) then
              π ← Restart(π)
          else π ← π*
      end-if
  end-while
  return π*
end ILS_HPRC;
```

Fig. 5. Pseudo-code of the algorithm for the minimization of the first objective.

ratio constraints) and the second (number of violations of low priority ratio constraints) objectives are taken into account. The second objective is optimized without deteriorating the first one, which can also be further improved along this process.

5.1 Perturbations

The perturbation phase makes use of two neighborhood structures in an oscillating way, changing the type of perturbation whenever the highest neighborhood order of the current perturbation is attained. With the first neighborhood structure, car exchange moves are randomly applied to pairs of cars of the same type,

```
procedure VNS();
  Build an initial solution s*
  while stopping criterion not met do
      k ← 1
      while k ≤ k_max do
          s' ← Perturbation(s*, N_k)
          s' ← LocalSearch(s')
          cost(s') < cost(s*) then
              s* ← s'
              k ← 1
          else k ← k + 1
      end-while
  end-while
  return s*
end VNS;
```

Fig. 6. VNS pseudo-code.

preserving the value of the first objective cost. Here, cars of same type are those associated to the same set of high priority ratio constraints. Car insertion moves in the second neighborhood structure consist in removing some cars involved in violations from the current solution and reinserting them at new chosen positions.

We denote by N_k^1, $k = k_1, \ldots, k_{max}$, the sequence of car insertion neighborhoods, where k is the number of cars that are removed and subsequently reinserted. Similarly, we denote by N_q^2, $q = q_1, \ldots, q_{max}$, the sequence of car exchange neighborhoods, where q is the number of car exchanges.

5.2 Local search

The fast local search strategy presented in Section 4.2 is used. Solutions perturbed by car insertion moves can have their first objective deteriorated. Therefore, the local search procedure attempts to optimize the weighted sum of the first and second objectives. The solution obtained by local search is discarded if it deteriorates the first objective.

In the case of solutions perturbed by car exchange moves, the local search considers only pairs of cars of the same type, consequently preserving the cost associated to the first objective.

5.3 Intensification

The intensification procedure is similar to that described in Section 4.3 for the optimization of the first objective. It is applied whenever the type of perturbation is changed. The first and the second objectives may be improved, since both of them are considered in the optimization.

5.4 Stopping criteria

As described in Section 4.5 for the optimization of the first objective, three stopping criteria are used. The first is the number of intensification phases i_{max} performed since the last improvement of the current solution. The second criterion is met when a given fraction of the time limit was spent on the second objective, to ensure that the third objective will be also considered. Finally, the third and most obvious stopping criterion is satisfied if there are no violations of low priority ratio constraints in the current solution. The optimization of the second objective is stopped whenever any of these three criteria is met.

5.5 Algorithm

The pseudo-code in Figure 7 provides an overview of the algorithm for the optimization of the second objective. Some variables and parameters are described below. Let π^* denote the current solution (which is also the best known solution) and π' a solution obtained by some perturbation. We denote by ϕ^* the cost of the current solution and by ϕ the cost of the solution obtained at the last intensification phase. The counter i accounts for the number of intensification phases performed since the last improvement in the current solution. The upper limit for this counter is the parameter i_{max}. Counters k and q represent the order of the current neighborhood used for generating car insertion and car exchange perturbations, respectively. Parameters k_1 and q_1 (resp. k_{max} and q_{max}) define the lowest (resp. highest) neighborhood orders for the car insertion and car exchange perturbations, respectively. Finally, P is a binary variable whose value is 1 if the current perturbation consists of car insertion moves, and 0 otherwise.

This algorithm starts with the solution obtained by the heuristic `ILS_HPRC` used for the optimization of the first objective, presented in Figure 5. Initializations are performed in lines 1–2. The loop in lines 3–24 is performed until one of the stopping criteria is met. Initial neighborhood orders and the cost of the current solution are initialized in lines 4 and 5, respectively. An iteration of the while loop in lines 6–18 is composed by the application of a perturbation to the current solution π^* resulting in the solution π' in line 7, followed by a local search applied to the latter in line 8. The new solution π' is accepted in line 9 to replace the current solution π^* if it is at least as good as the latter. The if statement in lines 12–17 updates the order of the neighborhood that will be used in the next iteration. If any improvement is performed on the current solution, the search will resume from the lower order neighborhood (line 13) and the cost ϕ^* of the best known solution is updated (line 14). Otherwise, the search will resume from a higher neighborhood order (line 16). When the current perturbation attains its highest order (k_{max} or q_{max}, depending on its type), the while loop in lines 6–18 finishes and the type of perturbation is changed in line 19. The intensification procedure is applied to the current solution in line 20 and the cost of the best known solution is updated in line 21. The variable i used for the stopping criterion is updated either in line 22 or in line 23, depending on the result of intensification procedure. The algorithm stops whenever one of the stopping criteria is met: the maximum number of intensification phases performed since the last improvement is attained (i.e. $i = i_{max}$), the computation time exceeds the established limit, or there are no violations of low priority ratio constraints.

procedure `VNS_LPRC`(π_0);
1 $\pi^* \leftarrow \pi_0$, $\phi^* \leftarrow cost(\pi_0)$
2 $i \leftarrow 0$, $P \leftarrow 1$
3 **while** stopping criteria not met **do**
4     $k \leftarrow k_1$, $q \leftarrow q_1$
5     $\phi \leftarrow \phi^*$
6   **while** ($P = 1$ **and** $k \leq k_{max}$) **or** ($P = 0$ **and** $q \leq q_{max}$) **do**
7         $\pi' \leftarrow$ `Perturbation`(π^*, P, N_k^1, N_q^2)
8         $\pi' \leftarrow$ `LocalSearch`(π')
9   **if** $cost(\pi') \leq cost(\pi^*)$ **then**
10             $\pi^* \leftarrow \pi'$
11   **end-if**
12   **if** $cost(\pi^*) < \phi^*$ **then**
13             $k \leftarrow k_1$, $q \leftarrow q_1$
14             $\phi^* \leftarrow cost(\pi^*)$
15   **else**
16             $k \leftarrow k + 1$, $q \leftarrow q + 1$
17   **end-if**
18   **end-while**
19     $P \leftarrow 1 - P$
20     $\pi^* \leftarrow$ `Intensification`(π^*)
21     $\phi^* \leftarrow cost(\pi^*)$
22   **if** $\phi^* < \phi$ **then** $i \leftarrow 0$
23   **else** $i \leftarrow i + 1$
24 **end-while**
25 **return** π^*
end `VNS_LPRC`;

Fig. 7. Pseudo-code of the algorithm for the minimization of the second objective.

6 Third objective optimization

This section presents the algorithm for the optimization of the third objective, also based on the VNS (Variable Neighborhood Search) metaheuristic.

All objectives are simultaneously considered by this algorithm. The first objective is preserved, the second can be still further improved, and the third is optimized. The initial solution obtained by algorithm `VNS_LPRC` used to optimize the second objective may be infeasible, i.e. it can have paint color batches whose sizes exceed the upper limit. Therefore, the first phase of the algorithm for the third objective consists in making feasible the initial solution, whenever necessary. The next section presents the repair procedures used to reach feasibility, while the subsequent sections describe the remaining phases.

6.1 Feasibility

In the process of making feasible the initial solution, some care should be taken in order to do not deteriorate the objectives previously optimized. Two repair procedures were developed. If the first procedure is not able to reach feasibility, the second is applied. Only the latter ensures feasibility, although at the cost of possibly deteriorating the value of the original (infeasible) solution:

- The first procedure applies car exchange moves to break violations of paint color batches. The moves are applied only to pairs of cars sharing the same ratio constraints, preserving the cost of the first and second objectives.
- The second procedure applies car insertion moves to remove cars from color batches violating the upper limit on their sizes, reinserting them into new feasible positions. The new positions are those that lead to the smaller variations on solution cost. Such car insertion moves are performed until the solution turns up to be feasible.

6.2 Perturbations

Car insertion and car exchange moves are used as perturbations in the same way previously described for the algorithm to optimize the second objective. As before, the type of perturbation is changed whenever the highest order of the current perturbation is attained.

Solutions perturbed by car insertion moves whose first objective deteriorates are discarded and local search is applied to the current solution. This issue does not apply in the case of solutions perturbed by car exchange moves, since the cost associated to the first objective is preserved if only cars of the same type are involved.

The neighborhood sets N_s^1, $s = s_1, \ldots, s_{max}$ and N_t^2, $t = t_1, \ldots, t_{max}$ correspond, respectively, to car insertion and car exchange moves. The values of s_{max} and t_{max} should be smaller than those used in the algorithm for the optimization of the second objective, otherwise a large amount of perturbed solutions will be discarded.

6.3 Local search

Solutions obtained by perturbations have the value of the first objective preserved, but may have the value of the second objective deteriorated. Hence, the local search optimizes the weighted sum of the second and third objectives. The complete local search procedure is adopted, but only car exchange moves involving pairs of cars of the same type (with respect to their high level priority ratio constraints) are considered. The solution obtained by local search is discarded if it deteriorates the second objective.

6.4 Intensification

The intensification phase is quite similar to those previously presented. The only difference is that all objectives are simultaneously considered. An intensification phase is performed whenever the type of perturbation is changed.

6.5 Algorithm

Figure 8 presents the pseudo-code of the algorithm proposed for the optimization of the third objective. Its structure is very similar to that of algorithm `VNS_LPRC` for the minimization of the second objective, described in Section 5.5. The only differences consist in the addition of the repair procedure to make the solution feasible (line 1) and of the filtering step performed on the solutions obtained by car insertion perturbations (lines 8 to 10). The algorithm halts when the time limit is exceeded (line 4).

7 Computational experiments

The heuristic described in the previous sections was coded in `C++` and compiled with version 3.2.2 of the `gcc` compiler with the optimization flag `-O3`. We have used our own implementation of the random number generator described in [6]. The algorithm is supposed to halt after 10 minutes of CPU time.

We used two sets of real-life instances to evaluate and tune the heuristic. They are referred to as set A and set B. The first set has four instances, which have

procedure `VNS_PCC`(π_0);
  if π_0 is infeasible **then** $\pi_0 \leftarrow$ `MakeFeasible`(π_0)
  $\pi^* \leftarrow \pi_0$, $\phi^* \leftarrow cost(\pi^*)$
  $P \leftarrow 1$
  while time limit is not exceeded **do**
    $s \leftarrow s_1$, $t \leftarrow t_1$
    while ($P = 1$ **and** $s \leq s_{max}$) **or** ($P = 0$ **and** $t \leq t_{max}$) **do**
      $\pi' \leftarrow$ `Perturbation`(π^*, P, N_s^1, N_t^2)
      if the first objective of π' is greater than
        the first objective of π^* **then**
        $\pi' \leftarrow \pi^*$
      end-if
      $\pi' \leftarrow$ `LocalSearch`(π')
      if $cost(\pi') \leq cost(\pi^*)$ **then**
        $\pi^* \leftarrow \pi'$
      end-if
      if $cost(\pi^*) < \phi^*$ **then**
        $s \leftarrow s_1$, $t \leftarrow t_1$
        $\phi^* \leftarrow cost(\pi^*)$
      else
        $s \leftarrow s + 1$, $t \leftarrow t + 1$
      end-if
    end-while
    $P \leftarrow 1 - P$
    $\pi^* \leftarrow$ `Intensification`(π^*)
    $\phi^* \leftarrow cost(\pi^*)$
  end-while
  return π^*
end `VNS_PCC`;

Fig. 8. Pseudo-code of the algorithm for the minimization of the third objective.

from 618 to 1329 cars, while the second has fifteen instances, which have from 92 to 1319 cars. The problem instances in these sets were provided by Renault and are available from the challenge web site [5]. Each instance corresponds to production data from a different factory. They are defined by the high priority level ratio constraints, the low priority level ratio constraints, the upper bound on the size of the paint color batches, the vehicles of the current production day to be scheduled, and the remaining cars scheduled but not produced in the previous day. The computational experiments of these sets were performed on a Pentium IV 1.8 GHz machine with 512 Mbytes of RAM memory running under Linux.

The computational results for set A and set B are presented in Table 1. The first four instances belong to set A and the other to set B. The first column gives the name of each instance. The second, third, and fourth columns give average results obtained by our algorithms over twenty independent runs respectively for the first, second, and third objectives. The three last columns present the best results obtained over all other teams taking part in the challenge ROADEF'2005. The total cost of the objective function is not reported, however it can be easily calculated applying the weights reported for the objective function described in Section 1.

Table 1. Average results for set A and set B.

	Our algorithm			Best results		
Instances	HPRC	LPRC	PCC	HPRC	LPRC	PCC
024_38_3	4.0	1.1	351.1	4	0	302
024_38_5	4.0	36.4	339.6	4	34	309
025_38_1	0.0	121.4	774.3	0	99	720
048_39_1	0.0	63.2	304.0	0	61	290
022_S22_J1	0.0	3.0	111.6	0	3	109
023_S23_J3	48.0	0.0	316.2	48	0	316
024_V2_S22_J1	1074.2	871.1	545.1	1074	850	430
025_S22_J3	0.0	3912.0	501.8	0	3912	479
028_ch1_S22_J2	54.0	3.0	74.5	54	3	76
028_ch2_S23_J3	0.0	70.0	6.0	0	70	6
029_S21_J6	35.0	2150.0	165.8	35	2150	167
035_ch1_S22_J3	67.0	52.0	49.0	67	52	49
035_ch2_S22_J3	385.0	341.0	205.0	385	341	205
039_ch1_S22_J4	0.0	29.0	160.4	0	29	117
039_ch3_S22_J4	0.0	0.0	197.7	0	0	197
048_ch1_S22_J3	0.0	0.0	200.0	0	0	200
048_ch2_S22_J3	3.0	0.0	340.0	3	0	337
064_ch1_S22_J3	0.0	0.0	194.2	0	0	182
064_ch2_S22_J4	0.0	69.0	130.0	0	69	130

We observe from Table 1 that the proposed heuristic obtains average results that are very close to the best known solutions for most instances. Except for instance 024_V2_S22_J1, it was able to find out solutions as good for the first objective as the best solution over all runs. Except for instances 025_38_1 and 024_V2_S22_J1, the average results for the second objective are the same or just a little worse than the best solution: the maximum and the average deviations are, respectively, 2.4 and 0.4 units in the number of violations of the second objective. We highlight that our heuristic obtained average results for instances 028_ch1_S22_J2 and 029_S21_J6 that were better than the best solution obtained by the other teams.

The proposed heuristic was submitted to the final phase of the challenge ROADEF'2005. The algorithms of all participating teams were executed on the same Pentium IV machine, with 1 Gbyte of RAM memory and a 1.7 GHz clock. The set of test instances used in this phase is called set X and was not known beforehand by the authors. It has instances with 78, 460, and 1071 cars that are similar to those in set A and set B. Instances from set X are also available from the challenge web site [5].

Table 2 reports the performance of the proposed heuristic in the last phase of ROADEF challenge. The first column gives the name of each instance. The second, third, and fourth columns give average results obtained by our algorithm over five independent runs, as reported by the organization committee, respectively for the first, second, and third objectives. The three last columns present the same results for the best among the other teams taking part in the challenge. Results in bold indicate that our algorithm found the best solution over all teams.

Our heuristic obtained very good results when compared to the other algorithms. It was the only that found the best solution for instance 028_CH1_S50_J4. It also found a solution as good as the best solution found by the other teams for instance 028_CH2_S51_J1. The worst result was obtained for instance 025_S49_J1, for which the value for the first and second objective were the same as in the best solution, but the value of the third objective was worse. However, the third objective is the one with smallest priority.

Table 2. Average results for set X.

	Our algorithm			Best results		
Instances	HPRC	LPRC	PCC	HPRC	LPRC	PCC
025_S49_J1	0.0	160.0	602.2	0.0	160.0	407.6
028_CH1_S50_J4	**36.0**	**341.4**	**95.4**	36.0	360.0	92.4
028_CH2_S51_J1	**0.0**	**0.0**	**3.0**	0.0	0.0	3.0

8 Concluding remarks

We described a VNS/ILS approach for approximately solving a real-life version of the car sequencing problem proposed by Renault in the context of the Challenge ROADEF'2005.

The three different objectives are tackled independently, in order to explore their particular characteristics. We proposed an ILS based heuristic for optimizing the first objective and a VNS based heuristic for optimizing the second and third objectives. We also proposed optimized data structures for the efficient implementation of the heuristics, which are essential for their efficiency. We recall that all algorithms submitted to the challenge ROADEF'2005 were supposed to run in at most 600 seconds of CPU time.

Computational experiments have shown that our heuristic obtained very good results when compared to the other algorithms. For three instances, it obtained better average results than the other participating teams. For the other instances, it obtained results as good as or very close to best solutions found by the other algorithms. The heuristic proposed in this work obtained the second prize in the challenge ROADEF'2005 over 52 teams participating in the first phase.

References

1. V.-D. Cung. The car sequencing problem. *European Journal of Operational Research*, to appear.
2. J. Gottlieb, M. Puchta, and C. Solnon. A study of greedy, local search, and ant colony optimization approaches for car sequencing problems. In *Applications of Evolutionary Computing*, volume 2611 of *Lecture Notes in Computer Science*, pages 246–257. Springer, 2003.
3. P. Hansen and N. Mladenović. Variable Neighborhood Search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 145–184. Kluwer Academic Publishers, 2002.
4. H. P. Lourenço, O. Martin, and T. Stützle. Iterated Local Search. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 321–353. Kluwer Academic Publishers, 2002.
5. A. Nguyen. Challenge ROADEF'2005: Car sequencing problem. Online reference at `http://www.prism.uvsq.fr/~vdc/ROADEF/CHALLENGES/2005/`, last visited on March 23, 2005.
6. L. Schrage. A more portable FORTRAN random number generator. *ACM Transactions on Mathematical Software*, 5:132–138, 1979.
7. B.M. Smith. Succeed-first or fail-first: A case study in variable and value ordering heuristics. In *Proceedings of the Third International Conference on Principles and Practice of Constraint Programming*, volume 1330 of *Lecture Notes in Computer Science*, pages 321–330. Springer, 1997.