A hybrid heuristic for a multi-objective real-life car sequencing problem with painting and assembly line constraints

Celso C. Ribeiro[1,2], Daniel Aloise[2], Thiago F. Noronha[2], Caroline Rocha[1], and Sebastián Urrutia[3]

[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

[3] Department of Computer Science, Universidade Federal de Minas Gerais,
Av. Antônio Carlos 6627, Belo Horizonte, MG 31270-010, Brazil.
surrutia@dcc.ufmg.br

Abstract. We address a multi-objective version of the car sequencing problem, which consists in sequencing a given set of cars to be produced in a single day, minimizing the number of violations of assembly constraints and the number of paint color changes in the production line. We propose a set of heuristics for approximately solving this problem, based on the paradigms of the VNS and ILS metaheuristics, to which further intensification and diversification strategies have been added. Computational results on real-life test instances are reported. The work presented in this paper obtained the second prize in the ROADEF challenge 2005 sponsored by Renault.

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

1 Introduction

The car sequencing problem [11] consists in sequencing a given set of cars to be produced in a given working day in the assembly line of a car factory. The cars to be produced are different and may have different options (such as air-conditioning and sun-roof, among others). The machines that install these options can handle at most a certain number of cars passing consecutively along the assembly line. These restrictions are modeled as *ratio constraints*. A ratio constraint N/P means that in any sequence of P cars there are at most N cars with the respective option. For instance, if $N/P = 3/5$, there must be no more than three constrained cars in any sequence of five cars. Each option has its own ratio constraint. A car may have more than one option. The objective is to sequence the cars so as that no ratio constraint is violated. The decision version of the car sequencing problem is NP-complete [5]. It was formulated and solved as a constraint satisfaction problem [1, 15] and by constraint logic programming [4]. Approaches based on local search were proposed in [3, 8, 12]. It was also tackled by metaheuristics approaches, such as genetic algorithms [16] and ant colony optimization [6, 14].

In this work, we address a multi-objective version of the car sequencing problem faced at Renault car factories. This subject was proposed in the context of the ROADEF challenge 2005 [10], which involved 55 teams from 15 different countries. In this multi-objective version, there are two sets of ratio constraints handled with different priorities: *high priority ratio constraints* (HPRC) and *low priority ratio constraints* (LPRC). These two sets of ratio constraints are handled as soft constraints and their violations are penalized in the objective function, since there is no guarantee that a feasible sequence respecting all of them exists in real-life instances. Furthermore, a paint color is also assigned to each car. The spray guns have to be washed after a given number of cars is painted with some color. To minimize the consumption of paint solvent, the number of paint color changes in the sequence of scheduled cars must be minimized. The limitation on the number of cars that may be painted consecutively with the same color is a hard constraint, in the sense that paint color batches (sequence of cars painted with the same color) with more cars than the maximum load supported by the spray guns makes the solution infeasible.

To summarize, this multi-objective car sequencing problem consists in scheduling the production of a set of cars minimizing three different objectives: (a) the number of paint color changes (PCC), (b) the number of violations of high priority ratio constraints, and (c) the number of violations of low priority ratio constraints. These objectives are often conflicting. However, they can be considered with different priority levels varying from one factory to another or at different times at the same factory. Renault proposed to tackle the problem using a weighted sum method that groups the three objectives into a single objective function, by assigning different weights to each objective, according with their priority levels. Three problem classes with different objective functions are considered [2, 10], with weights $W_1 >> W_2 >> W_3$:

Class I: Objective function 1 = W_1 × number of paint color changes + W_2 × number of violations of HPRC + W_3 × number of violations of LPRC;

Class II: Objective function 2 = W_1 × number of violations of HPRC + W_2 × number of paint color changes + W_3 × number of violations of LPRC;

Class III: Objective function 3 = W_1 × number of violations of HPRC + W_2 × number of violations of LPRC + W_3 × number of paint color changes.

The values of W_1, W_2, and W_3 are defined in a way that the best solution is the one with the smallest value for the first objective. In case of ties in the first objective, the best solution is the one with the smallest value for the second objective. Finally, in case of ties in the first and second objectives, the best solution is the one with the smallest value for the third objective. In practice, Renault proposed to consider $W_1 = 1000000$, $W_2 = 1000$, and $W_3 = 1$. Problem instances are evenly distributed over the three classes in practice.

The rules of the ROADEF challenge 2005 stated that the proposed algorithms should stop after 600 seconds of computation time. This time limit was enforced by Renault and ROADEF, representing the maximum response time accepted by the users. The remaining of the paper is organized as follows. Section 2 describes the general approach proposed to tackle the multi-objective version of the car sequencing problem. Two constructive heuristics are proposed in Section 3. Neighborhood definitions and improvement heuristics based on the paradigms of the Iterated Local Search (ILS) [9] and Variable Neighborhood Search (VNS) [7] metaheuristics are presented in Section 4. Some relevant implementation issues, regarding data structures and quick neighborhood investigation, are commented in detail in Section 5. Computational results are reported in the last section.

2 Solution approach

The three objectives must be optimized and their weighted sums clearly differentiate the importance of each of them. Therefore, we optimize each objective individually according to the decreasing order of their weights in the objective function. First, we optimize the most important objective. Next, we attempt to optimize the second objective, without deteriorating the first. Finally, we attempt to optimize the third objective, without deteriorating the first and second objectives. For example, in the case of of Class II problems, we first minimize the number of violations of HPRC. Next, we attempt to minimize the number of PCC, without increasing the the number of violations of HPRC. Finally, we attempt to minimize the number of violations of LPRC, without deteriorating the number of violations of HPRC and the number of PCC. Triggers based on the computation time and on the stabilization of the best known solution are used to interrupt the minimization of one objective and to activate the minimization of the next. The same strategy is also used for Class I and Class III problems.

Each of the three above steps can be seen as a different subproblem, with its specific characteristics. We developed specific codes for each problem class and for each subproblem, to take advantage of their specific characteristics. The computation of the cost function (which usually accounts for the largest fraction of the time spent by local search algorithms) can be performed quickly, since not all objectives are evaluated in every subproblem. The initial solution to the algorithm that handles the first objective is provided by a greedy heuristic. The algorithms for the other subproblems start from the solution provided by the algorithm for the previous subproblem.

Table 1 summarizes the different algorithms implemented and described in the next section. The minimization of the number of paint color changes (first subproblem of class I problems), in the absence of other constraints, is solvable in polynomial time by algorithm `Gre1` described in Section 3. The first objective is the same for class II and class III problems; therefore, the same algorithm can be used for optimizing the first subproblem in both classes. We also notice that the third subproblem is the same for problems in classes I and II; therefore, the same algorithm can be used in these two cases.

Table 1. Greedy algorithms and improvement heuristics.

<table>
<tr><th></th><th>Class I</th><th>Class II</th><th>Class III</th></tr>
<tr><td>Construction</td><td rowspan="2">Gre1: builds a solution minimizing the number of paint color changes (greedy).</td><td colspan="2">Gre2: builds a solution minimizing the number of violations of high priority ratio constraints (greedy).</td></tr>
<tr><td>First objective</td><td colspan="2">Imp3: attempts to reduce the number of violations of high priority ratio constraints (ILS).</td></tr>
<tr><td>Second objective</td><td>Imp4: attempts to reduce the number of violations of high priority ratio constraints, without increasing the number of paint color changes (ILS).</td><td>Imp5: attempts to reduce the number of paint color changes, without increasing the number of violations of high priority ratio constraints (VNS).</td><td>Imp6: attempts to reduce the number of violations of low priority ratio constraints, without increasing the number of violations of high priority ratio constraints (VNS).</td></tr>
<tr><td>Third objective</td><td colspan="2">Imp7: attempts to reduce the number of violations of low priority ratio constraints, without increasing the number of violations of high priority ratio constraints and the number of paint color changes (ILS).</td><td>Imp8: attempts to reduce the number of paint color changes, without increasing the number of violations of high and low priority ratio constraints (VNS).</td></tr>
</table>

The two constructive algorithms `Gre1` and `Gre2` are greedy heuristics described in Section 3. The other algorithms are improvement heuristics based on the paradigms of the Iterated Local Search and the Variable Neighborhood Search metaheuristics, as reported in Section 4.

3 Constructive heuristics

We denote by $V = \{v_1, \ldots, v_n\}$ the set of cars to be scheduled and by $O = \{o_1, \ldots, o_m\}$ the set of options that a car may have. The set of colors used to paint the cars is denoted by $C = \{c_1, \ldots, c_t\}$. Ratio constraints associated with each option $o \in O$ are defined by two functions $p : O \rightarrow \mathbb{N}^*$ and $q : O \rightarrow \mathbb{N}^*$: there must be at most $p(o)$ cars requiring option o in any sequence of $q(o)$ consecutive cars. High priority ratio constraints (HPRC) are those that should preferably be enforced, while the others are low priority ratio constraints (LPRC). 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. We denote by $\pi =< v_{i_1}, v_{i_2}, \ldots, v_{i_k} >$ a sequence of $|\pi| = k$ consecutive (and different) cars, with $v_{i_\ell} \in V$ for every $\ell = 1, \ldots, k$ and $v_{i_{\ell_1}} \neq v_{i_{\ell_2}}$ for any $\ell_1, \ell_2 = 1, \ldots, k$ with $\ell_1 \neq \ell_2$.

The first constructive heuristic, named `Gre1`, starts with a subsequence that cannot be modified, formed exclusively by cars already scheduled in the previous day that have not been produced. At each iteration, a new car selected among those not yet scheduled is considered for insertion into the current partial solution. The insertion criterion consists in searching for the best position to schedule this car into the sequence formed by the cars already scheduled. The best position corresponds to that leading to the smallest increase (possibly negative) in the cost function. Insertions into positions corresponding to infeasible partial solutions are discarded. This procedure is repeated until all cars have been scheduled. Figure 1 illustrates an example.

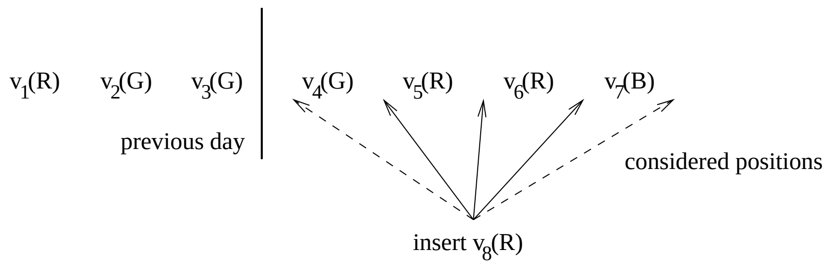

Fig. 1. Evaluation of an insertion by `Gre1` in terms of the number of paint color changes.

In this figure, cars are represented by identifiers v_i and by their colors (R, G, or B). For ease of understanding, this example considers only the number of paint color changes. The number of paint color changes is not increased either if car v_8 is scheduled in between cars v_5 and v_6, or in between cars v_5 and v_6 or in between cars v_6 and v_7. In case of ties in the variation of the number of paint color changes, the number of violations of high priority ratio constraints is also evaluated by the heuristic. Figure 2 illustrates the insertion of a car chosen by

Gre1 in the current partial solution. There is only one position to be considered for insertion in the first iteration (the first one), while in the last iteration there are as many possible positions as the number of cars.

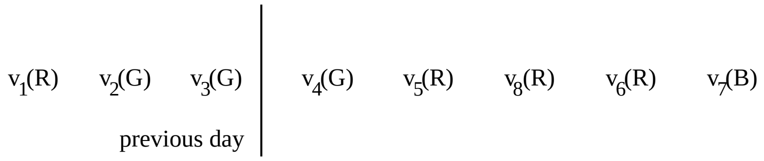

Fig. 2. Example of an insertion in the current partial solution performed by Gre1.

```
procedure Gre1(π_0, V);
1  π ← π_0;
2  S ← V;
3  while S ≠ ∅ do
4      Δ*, Δ*_color ← ∞;
5      v ← randomly chosen element of S;
6      for i = |π_0| + 1, ..., |π| + 1 do
7          Δ_color ← cost_color(π, v, i);
8          Δ ← cost(π, v, i);
9          if (Δ_color < Δ*_color) or (Δ_color = Δ*_color and Δ < Δ*)
10         then do;
11             Δ*_color ← Δ_color;
12             Δ* ← Δ;
13             best_pos ← i;
14         end-if;
15     end-for;
16     Insert car v at position best_pos of the current partial solution π;
17     S ← S − {v};
18 end-while;
19 return π;
end Gre1;
```

Fig. 3. Pseudo-code of the heuristic Gre1.

Figure 3 presents the pseudo-code of algorithm Gre1. The latter receives as inputs the initial solution formed by the sequence π_0 of cars already scheduled in the previous day and the set V of cars to be scheduled in the current day. The current solution π is initialized with π_0 in line 1. In line 2, the set S of yet unscheduled cars is initialized with V. The loop in lines 3–18 iterates until all cars have been scheduled. At each iteration, the best position to insert a yet unscheduled car is determined. Variables Δ^* and Δ^*_{color} are initialized in line 4 and store, respectively, the smallest solution cost (low priority ratio constraints are not taken into account) and the smallest number of paint color changes over all possible insertion positions. A car v still to be scheduled is randomly selected in line 5. The loop in lines 6–15 investigates all possible insertion positions for this car. The number of paint color changes and the cost of the solution obtained by inserting car v at position i of the current partial solution π is computed by functions `cost_color`(π, v, i) and `cost`(π, v, i) in lines 7 and 8, respectively (`cost_color`$(\pi, v, i)= \infty$ if car v cannot be inserted at position i). In line 9, we compare the solution obtained by inserting car v at position i of solution π with that obtained by performing the best move found so far. The smallest number of paint color changes, the smallest solution cost, and the best insertion position, are possibly updated in lines 10–14. In line 16, car v is inserted at position *best_pos* of the current partial solution π and the latter is updated. The set of yet unscheduled cars is updated in line 17 and a new iteration starts. The algorithm ends when $S = \emptyset$, returning a complete sequence π with all cars in V scheduled. The heuristic runs in time $O(mn^2)$, if implemented with data structures directly derived from those presented in Section 5.2.

The second constructive heuristic, named `Gre2`, also builds the initial solution using a greedy strategy. It takes the number of additional violations of high priority ratio constraints as the greedy criterion to define the next car to be introduced into the sequence. Ties are broken using an extension of the strategy proposed in [6], which favors a more equitable car distribution. In our case, we also considered second and third tie breaking criteria based on the hardness of each constraint and on the cost of the second objective, respectively. Harder constraints are those applied to more cars and that have smaller ratios. Cars associated with harder constraints are scheduled first. Heuristic `Gre2` also runs in time $O(mn^2)$.

4 Improvement heuristics

Six neighborhoods are used by the improvement heuristics `Imp3` to `Imp8` described in the next subsections:

1. Car exchange (`CarExchange`): the positions of two cars are exchanged.
2. Car insertion (`CarInsertion`): 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.
3. Group exchange (`GroupExchange`): two maximal subsequences of consecutive cars painted with the same color are exchanged.
4. Group inversion (`GroupInversion`): a maximal subsequence of consecutive cars painted with the same color is inverted.

5. Reinsertion (`Reinsertion`): a subset of cars is discarded and reinserted using the same principles of heuristic `Gre1`.
6. Multiple car exchanges (k-`CarExchange`): k pairs of cars are randomly chosen and the positions of the two cars in each pair are exchanged.

`GroupExchange` and `GroupInversion` neighborhoods preserve the number of paint color changes of the current solution. The `GroupInversion` neighborhood preserves the number of paint color changes even if the subsequence of consecutive cars painted with the same color considered by the movement is not maximal. The number of violations of ratio constraints between cars belonging to the inverted subsequence is not affected by movements in this neighborhood. If only maximal subsequences of consecutive cars painted with the same color are considered, a big portion of the subsequence of cars affected by the movement remains unchanged in terms of both the number of paint color changes and the number of violations of ratio constraints.

Moves in neighborhoods `CarExchange` and `CarInsertion` can be quickly evaluated (see Section 5), making them good candidates to be used in local search procedures. Since moves in the other four neighborhoods are harder to evaluate, they are used as perturbations that are only sporadically applied.

Neighborhood `CarExchange` is defined by moves that swap the positions a and b of two cars of a sequence

$$\pi =< v_{i_1}, \ldots, v_{i_a}, \ldots, v_{i_b}, \ldots, v_{i_n} > .$$

The application of move $exchange(\pi, a, b)$ to sequence π results in the sequence

$$\pi' =< v_{i_1}, \ldots, v_{i_b}, \ldots, v_{i_a}, \ldots, v_{i_n} > .$$

Neighborhood `CarInsertion` is defined by moves that insert into position b the car currently at position a of a sequence

$$\pi =< v_{i_1}, \ldots, v_{i_{a-1}}, v_{i_a}, v_{i_{a+1}}, \ldots, v_{i_{b-1}}, v_{i_b}, v_{i_{b+1}}, \ldots, v_{i_n} > .$$

Move $insertion(\pi, a, b)$ results in the sequence

$$\pi' =< v_{i_1}, \ldots, v_{i_{a-1}}, v_{i_{a+1}}, \ldots, v_{i_{b-1}}, v_{i_b}, v_{i_a}, v_{i_{b+1}}, \ldots, v_{i_n} >,$$

for $b > a$ (a similar definition applies for $a > b$).

The local search procedures used in this work are based exclusively on exchange and insertion moves. They terminate when the objective function cannot be improved after the evaluation of all moves in the neighborhood.

4.1 Framework

Metaheuristics are general high-level procedures that coordinate simple heuristics and rules to find good (often optimal) approximate solutions to computationally difficult combinatorial optimization problems. Among them, we find simulated annealing, tabu search, GRASP, genetic algorithms, scatter search, VNS, ant colonies, and others. They are based on distinct paradigms and offer different mechanisms to escape from locally optimal solutions, contrarily to greedy algorithms or local search methods. Metaheuristics are among the most effective solution strategies for solving combinatorial optimization problems in practice and they have been applied to a very large variety of areas and situations. The customization (or instantiation) of some metaheuristic to a given problem yields a heuristic to the latter.

All improvement heuristics `Imp3` to `Imp8` proposed in this work follow the same framework. They are based on the paradigms of the Variable Neighborhood Search (VNS) [7] and the Iterated Local Search (ILS) [9] metaheuristics. We added to their basic templates two additional phases: intensification and diversification. These phases are not applied at every iteration of the algorithm. The goal of the intensification phase is to perform a more effective (and, consequently, also more time consuming) search in the neighborhood of the current solution. The goal of the diversification phase is to drive the search to another region of the solution space. The intensification and diversification phases are triggered by different events in each heuristic.

```
procedure ImprovementHeuristic(S0);
  S, S* ← LocalSearch(S0);
  while .NOT.StoppingCriterion do;
      S' ← Perturbation(S);
      S' ← LocalSearch(S');
      if S' is not worse than S then do;
          S ← S';
          S* ← UpdateBestSolution(S, S*);
      end-if;
      if IntensificationCriterion then do;
          S ← IntensificationProcedure(S);
          S* ← UpdateBestSolution(S, S*);
      end-if;
      if DiversificationCriterion then do;
          S ← DiversificationProcedure(S);
          S* ← UpdateBestSolution(S, S*);
      end-if;
  end-while;
  return S*;
end ImprovementHeuristic;
```

Fig. 4. General framework of the improvement heuristics.

The pseudo-code in Figure 4 summarizes the framework of the main steps of our improvement heuristics. They all start from a feasible solution S_0. A local search procedure is applied to the initial solution and returns a locally optimal solution S, which is also used to initialize the best solution found S^*. The while loop in lines 2–17 runs until the stopping criterion is satisfied. A perturbation is applied to the current solution S to produce a new solution S' in line 3. A local search procedure is applied to the solution obtained by the perturbation, returning a locally optimal solution S' in line 4. If solution S' is not worse than the current solution S, then the latter is updated in line 6. The best solution found S^* is possibly updated in line 7. An intensification phase is performed whenever an intensification criterion is met in line 9. The current solution is updated in line 10 after the intensification procedure. The best solution S^* is possibly updated in line 11. Similarly, the diversification procedure is performed whenever a diversification criterion is met in line 13. The current solution is always updated in line 14 after diversification, to drive the search to another region of the solution space. The best solution S^* is possibly updated in line 15.

The improvement heuristics `Imp3` to `Imp8` differ from each other by the subproblem they tackle, the neighborhoods they use for local search and perturbations, the stopping criteria, and also by the intensification and diversification phases. All the improvement heuristics consider that an improving neighbor solution cannot have a higher objective value, if the objectives are considered from that with the highest to that with the lowest priority.

4.2 Class I problems

The optimization of the first objective of Class I problems is performed in polynomial time by the constructive heuristic `Gre1` described in Section 3.

Heuristic `Imp4` takes the two first minimization objectives into account: the number of paint color changes and the number of violations of high priority ratio constraints. It follows the paradigm of the ILS metaheuristic. Local search is based on the neighborhood `CarExchange`, while moves in the `GroupExchange` and `GroupInversion` neighborhoods are used as perturbations. The intensification and diversification phases start after a given number of iterations since the last time the best solution found was updated. Intensification is performed by means of a VND local search strategy [7], using neighborhoods `CarExchange` and `CarInsertion`. Diversification is performed by means of restarts using algorithm `Gre1`. The procedure stops after a given number of restarts since the last time the best solution was updated or after a given fraction of the maximum time limit (this criterion is used for triggering the optimization of the last objective).

Heuristic `Imp7` is very similar to `Imp4`, but tackles all three objectives. It uses the same strategies of the latter for local search, perturbations, and intensification. Diversification consists in restarts from a new solution constructed by a variant of `Gre2` that does not change the value of the first two objectives. In this variant, one can only select a car for a given position if its paint color and its options associated to high priority ratio constraints are the same as those of the car at this position in the best known solution. The intensification and diversification phases start after a given number of iterations since the last time the best solution was updated. The algorithm stops when the maximum time limit is reached.

4.3 Class II problems

The initial solution is built with the greedy algorithm `Gre2`. Next, the ILS heuristic `Imp3` handles exclusively the first objective, i.e. the minimization of the number of violations of high priority ratio constraints. Local search is based on the `CarExchange` neighborhood. Moves in the `Reinsertion` neighborhood are used as perturbations, after the removal of five cars involved in violations. The intensification and diversification phases are activated after a given number of iterations since the last time the best solution found was updated. Intensification is performed by means of a VND local search strategy using neighborhoods `CarExchange` and `CarInsertion`. Diversification is performed via restarts from new solutions constructed by algorithm `Gre2`. The heuristic stops after a given number of restarts since the last time the best solution was updated or after a given fraction of the maximum time limit.

Heuristic `Imp5` takes the solution obtained by `Imp3` and considers the two first minimization objectives: the number of violations of high priority ratio constraints and the number of paint color changes. Again, local search is based on the `CarExchange` neighborhood. Perturbations follow the VNS strategy, generating moves in the neighborhoods k-`CarExchange`, for $k = 3, \ldots, 20$. The intensification phase is activated every time the highest order of the k-`CarExchange` neighborhood is attained. Intensification is performed by a technique similar to that used in `Imp3`, based on a VND local search strategy using neighborhoods `CarExchange` and `CarInsertion`. The diversification phase is activated after the highest neighborhood order has been attained a given number of times, since the last time the best solution found was updated. Diversification is performed via restarts from new solutions constructed by a variant of algorithm `Gre2` that does not change the value of the first objective. In this variant, one can only select a car for a given position if its options associated to high priority ratio constraints are the same as those of the car at this position in the best known solution. The heuristic stops after diversification has been applied a given number of times since the last time the best solution was updated or after a given time limit.

The optimization of the third objective of Class II problems is performed by the same heuristic `Imp7` described in Section 4.2, since the last objective of Class I and Class II problems is the same.

4.4 Class III problems

The construction of the initial solution and the optimization of the first objective of Class III problems is performed by the same algorithms used for Class II problems, respectively `Gre2` and `Imp3`, as described in Section 4.3, since the first objective of the problems in these classes is the same.

Heuristic `Imp6` takes the solution obtained by `Imp3` and considers the two first minimization objectives: the number of violations of high priority ratio constraints and the number of violations of low priority ratio constraints. It follows the VNS paradigm, once again with local search based on the `CarExchange` neighborhood. `Imp6` oscillates between the `Reinsertion` and k-`CarExchange` neighborhoods to perform perturbation moves, changing the type of perturbation whenever the highest neighborhood order of the current perturbation is attained. The number of cars removed in each move within neighborhood `Reinsertion` ranges from three to eight, while for moves in the k-`CarExchange` neighborhood k varies from five to ten. The intensification phase is activated every time the highest neighborhood order of the current perturbation is attained. The intensification strategy is similar to that used in `Imp3`, consisting on a VND local search strategy using neighborhoods `CarExchange` and `CarInsertion`. There is no diversification phase. The heuristic stops after a given number of applications of the intensification phase since the last time the best solution was updated or after a given fraction of the maximum time limit.

Finally, the VNS heuristic `Imp8` tackles all three objectives, taking the solution produced by `Imp6` as the initial solution. However, the latter may be infeasible, since `Imp6` does not consider the paint color changes constraints. Therefore, a repair procedure is applied to make the solution feasible. This algorithm is still another variant of `Gre2`. In this case, one can only select a car for a given position if its options associated to ratio constraints are the same as those of the car at this position in the solution produced by `Imp6`. If the solution obtained by this algorithm is still infeasible, we apply insertion moves to enforce feasibility by breaking infeasible paint color batches. The selected moves are the best ones, in the sense that they do not deteriorate too much the solution cost and make the solution feasible. As before, local search is based on the `CarExchange` neighborhood. `Imp8` also oscillates between the `Reinsertion` and k-`CarExchange` neighborhoods to perform perturbation moves, changing the type of perturbation whenever the highest neighborhood order of the current perturbation is attained. The number of cars removed in each move within neighborhood `Reinsertion` ranges from two to six, while $k = 2, \ldots, 8$ for moves in the k-`CarExchange` neighborhood. The intensification phase is similar to those used by `Imp3` and `Imp6`, consisting on a VND local search strategy using neighborhoods `CarExchange` and `CarInsertion`. Also in this case, there is no diversification phase. The algorithm stops when the maximum time limit is reached.

5 Implementation issues

In this section, we give further details concerning implementation issues that are quite important to speedup the neighborhood search and move evaluation.

5.1 Quick neighborhood investigation: non-improving moves

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 improving heuristics `Imp3` to `Imp8` 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.

Isolated cars: A car is said to be isolated in a given solution if its color is different from those of the cars scheduled immediately before and immediately after it. An insertion move can only reduce the number of paint color changes if the reinserted car was isolated before the move. To improve the performance of the local search based on insertion moves in algorithm `Imp5`, we consider not only isolated cars, but also cars that are in small batches of size at most equal to three. This case is also considered because such moves can make the batches smaller and, subsequently, some cars may become isolated.

Extreme cars: A car is said to be extreme in a given solution if its color is different from that of the car scheduled immediately before or immediately after it, i.e. if it is the first or the last of a sequence of consecutive cars painted with the same color. If none of the cars involved in an exchange move is an extreme car, then the number of paint color changes necessarily increases. Exchange moves that do not meet this condition are discarded whenever the number of paint color changes is part of the objective function being considered.

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 are discarded (and not even evaluated) when the neighborhood is sought for moves strictly decreasing the number of violations of ratio constraints.

Perturbations and local search: Moves in the neighborhoods `GroupExchange` and `GroupInversion` do not modify the number of paint color changes. Let us suppose that a move in one of these neighborhoods was applied as a perturbation to a locally optimal solution, followed by a local search procedure using exchange moves. Then, an exchange move can only reduce the solution cost if it involves at least one car affected by the previous perturbation move. A car is affected by a perturbation move if at least one of the cars scheduled at a distance lower than or equal to the maximum value of $q(o_j)$ over all options $j = 1, \ldots, m$ was moved by the perturbation.

5.2 Quick neighborhood investigation: 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 check the colors of adjacent cars. Solution feasibility (checking if the size of the subsequences of cars painted with the same color do not exceed the maximum limit) number of can be performed in time $O(1)$ as well.

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 with 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 5 presents a simple example for $n = 10$ and $m = 1$, with one unique ratio constraint with $p(o_1) = 1$ and $q(o_1) = 4$, where the number of violations is equal to six.

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

	1	2	3	4	5	6	7	8	9	10
	X	_	X	X	_	_	_	X	_	X
M_1 =	[3	2	2	1	1	1	2	2	1	1]
M_2 =	[1	2	3	3	3	3	4	5	5	5]
M_3 =	[1	2	3	4	5	6	7	8	9	10]

Fig. 5. 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).

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

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 $\Delta_j = \Delta_j^2 - \Delta_j^1$. 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 $o_j \in O, j = 1, \ldots, m$, 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 $j = 1, \ldots, m$, in the new subsequence is given by

$$\Delta^1_{j,b_0} = \begin{cases} (M_1[j][b_0] + r(v_{i_a}, o_j) - r(v_{i_{b_0+q(o_j)-1}}, o_j)) - p(o_j), \\ \quad \text{if } M_1[j][b_0] + r(v_{i_a}, o_j) - r(v_{i_{b_0+q(o_j)-1}}, o_j) > p(o_j); \\ 0, \text{ otherwise,} \end{cases}$$

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 $b = b_0+1, \ldots, n$ 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

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

5.3 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 $n-1$ feasible moves are evaluated for each car, while only one solution update is performed. The data structures described in the previous section are used.

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 starting at column $\min\{a, b\} - (q(o_j) - 1)$ 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 $O(mn^2)$.

6 Computational experiments

The algorithms described in the previous sections were 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 [13].

The problem instances in the test set X were provided by Renault and are available from the challenge web site [10]. 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 cars of the current production day to be scheduled, the remaining cars scheduled but not produced in the previous day, and the ranking of the optimization objectives (which defines the objective function).

The computational experiments were performed by the challenge organizers on a Pentium IV 1.7 GHz machine with 1 Gbyte of RAM memory running under Linux. The same machine was used to run the algorithms proposed by all teams taking part in the challenge. All algorithms should stop after 600 seconds of computation time and the best solution found should be returned. The results for all test instances are presented in Tables 2 to 4. The first column gives the name of each instance. The second, third, and fourth columns give average results obtained by our algorithms over five independent runs, as reported by the organizers, 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. We refer [10] for more detailed information about the individual performance of each team.

Figure 6 provides a plot depicting the number of instances for which our algorithms obtained a certain ranking position, in comparison with the results obtained by the other teams. The set of algorithms described in this paper obtained the second place in the ROADEF challenge 2005.

Table 2. Average results for class I test problems.

	Proposed algorithm			Best result		
Instances	PCC	HPRC	LPRC	PCC	HPRC	LPRC
022_S49_J2	**12.0**	**2.0**	**3.0**	12.0	2.0	3.0
035_CH1_S50_J4	**5.0**	**10.0**	**0.0**	5.0	10.0	0.0
035_CH2_S50_J4	**6.0**	**56.0**	**0.0**	6.0	56.0	0.0

Table 3. Average results for class II test problems.

	Proposed algorithm			Best result		
Instances	HPRC	PCC	LPRC	HPRC	PCC	LPRC
023_S49_J2	0.0	193.0	77.2	0.0	192.4	66.0
024_S49_J2	0.0	352.8	12.0	0.0	337.0	6.0
029_S49_J5	0.0	111.8	66.0	0.0	110.2	98.4
034_VP_S51_J1_J2_J3	0.0	58.0	643.6	0.0	55.2	794.8
034_VU_S51_J1_J2_J3	**8.0**	**87.0**	**35.8**	8.0	87.0	36.4
039_CH1_S49_J1	0.0	69.0	479.6	0.0	69.0	239.0
039_CH3_S49_J1	0.0	231.0	2162.6	0.0	231.0	30.0
048_CH1_S50_J4	0.0	196.0	1016.0	0.0	196.0	1005.6
048_CH2_S49_J5	31.0	79.0	1128.4	31.0	76.8	1116.2
064_CH1_S49_J1	61.0	190.8	81.4	61.0	187.2	29.8
064_CH2_S49_J4	**0.0**	**37.0**	**0.0**	0.0	37.0	0.0
655_CH1_S51_J2_J3_J4	**0.0**	**30.0**	**0.0**	0.0	30.0	0.0
655_CH2_S52_J1_J2_S01_J1	**153.0**	**34.0**	**0.0**	153.0	34.0	0.0

Table 4. Average results for class III test problems.

	Proposed algorithm			Best result		
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

References

1. M. Butaru and Z. Habbas, "The car-sequencing problem as n-ary CSP: Sequential and parallel solving", *Lecture Notes in Computer Science* 3809 (2005), 875–878.
2. V.-D. Cung, "The car sequencing problem", *European Journal of Operational Research*, this issue.
3. A. J. Davenport and E. P. K. Tsang, " Solving Constraint Satisfaction Sequencing Problems By Iterative Repair", *In Proceedings of the First International Conference on the practical Applications of Constraint Technologies and Logic Programming*, pages 345–357, 1999.
4. M. Dincbas, H. Simonis and P. Hentenryck, "Solving car sequencing problem in constraint logic programming", *Proceedings of European Conference on AI*, pages 290–295, 1988.

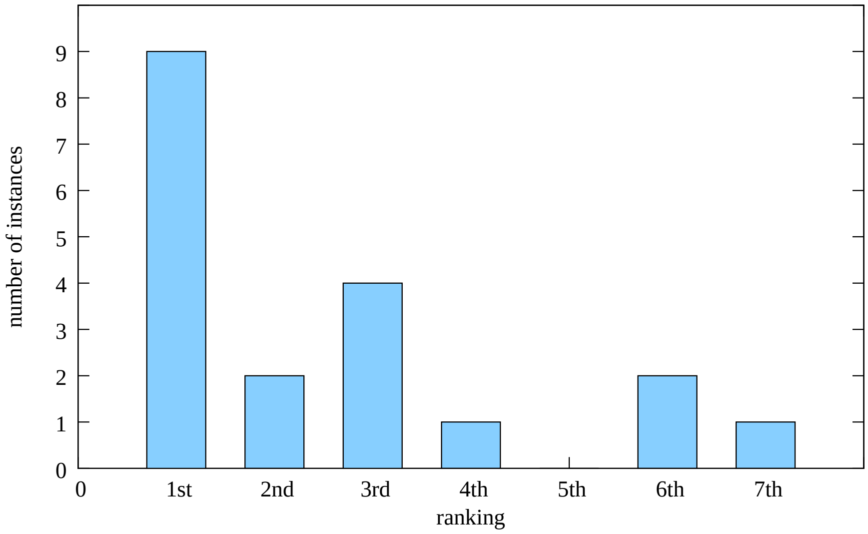

Fig. 6. Number of instances for which the proposed algorithms obtained a given ranking.

5. I. Gent, "Two results on car sequencing problems", Technical Report APES, University of St. Andrews, School of Computer Science, 1998.
6. J. Gottlieb, M. Puchta, and C. Solnon, "A study of greedy, local search and ant colony optimization approaches for car sequencing problems", *Lecture Notes in Computer Science* 2611 (2003), 246–257.
7. P. Hansen and N. Mladenovic, "Variable Neighborhood Search", in *Handbook of Metaheuristics* (F. Glover and G. Kochenberger, eds.), pages 145–184, Kluwer, 2002.
8. J. H. M. Lee, H. F. Leung and H.W. Won. "Performance of a Comprehensive and Ecient Constraint Library using Local Search", *Lecture Notes in Artificial Intelligence* 1502 (1998), 191–202.
9. H.R. Lourenço, O.C. Martin, and T. Stützle, "Iterated local search", in *Handbook of Metaheuristics* (F. Glover and G. Kochenberger, eds.), pages 321–353, Kluwer, 2002.
10. A. Nguyen, "Challenge ROADEF'2005: Car Sequencing Problem", online reference at `http://www.prism.uvsq.fr/~vdc/ROADEF/CHALLENGES/2005/`, last visited on December 11, 2005.
11. B.D. Parello, W.C. Kabat, and L. Wos, "Job-shop scheduling using automated reasoning: A case study of the car sequencing problem", *Journal of Automated Reasoning* 2 (1986), 1–42.
12. M. Puchta and J. Gottlieb, "Solving car sequencing problems by local optimization" , *Lecture Notes in Computer Science* 2279 (2002), 132–142.
13. L. Schrage, "A more portable FORTRAN random number generator", *ACM Transactions on Mathematical Software* 5 (1979): 132–138.

14. C. Solnon, “Solving permutation constraint satisfaction problems with artificial ants”, *Proceedings of the European Conference on Artificial Intelligence*, pages 118–122, 2000.
15. E. Tsang, “Foundations of Constraint Satisfaction”, *Academic Press*, 1993.
16. T. Warnick and E. Tsang, “Tackling car sequencing problems using a genetic algorithm”, *Evolutionary Computation* 3 (1995), 267–298.