TECHNISCHE UNIVERSITÄT WIEN
Institut für Computergraphik und Algorithmen

An Integer Linear Programming Approach and a Hybrid Variable Neighborhood Search for the Car Sequencing Problem

Matthias Prandtstetter and Günther R. Raidl

Forschungsbericht / Technical Report

TR–186–1–05–01

23. November 2005

Favoritenstraße 9-11 / E186, A-1040 Wien, Austria
Tel. +43 (1) 58801-18601, Fax +43 (1) 58801-18699
www.cg.tuwien.ac.at

An Integer Linear Programming Approach and a Hybrid Variable Neighborhood Search for the Car Sequencing Problem ⋆

Matthias Prandtstetter, Günther R. Raidl

Vienna University of Technology,
Institute of Computer Graphics and Algorithms,
Favoritenstrasse 9–11 / E186–1, A–1040 Vienna, Austria
http://www.ads.tuwien.ac.at

Abstract

In this paper we present two major approaches to solve the car sequencing problem, in which the goal is to find an optimal arrangement of commissioned vehicles along a production line with respect to constraints of the form *"no more than l_c cars are allowed to require a component c in any subsequence of m_c consecutive cars"*. The first method is an exact one based on integer linear programming (ILP). The second approach is hybrid: it uses ILP techniques within a general variable neighborhood search (VNS) framework for examining large neighborhoods. We tested the two methods on benchmark instances provided by CSPLib and the automobile manufacturer RENAULT for the ROADEF Challenge 2005. These tests reveal that our approaches are competitive to previous reported algorithms. For the CSPLib instances we were able to shorten the required computation time for reaching and proving optimality. Furthermore, we were able to obtain tight bounds on some of the ROADEF instances. For two of these instances the proposed ILP-method could provide new optimality proofs for already known solutions. For the VNS, the individual contributions of the used neighborhoods are also experimentally analyzed. Results highlight the significant impact of each structure. In particular the large ones examined using ILP techniques enhance the overall performance significantly, so that the hybrid approach clearly outperforms variants including only commonly defined neighborhoods.

Key words: Car Sequencing Problem, Integer Linear Programming, Variable Neighborhood Search, Hybrid Meta-heuristics

⋆ This work is supported by the RTN ADONET under grant 504438.
Email addresses: prandtstetter@ads.tuwien.ac.at (Matthias Prandtstetter), raidl@ads.tuwien.ac.at (Günther R. Raidl).

1 Introduction

In automobile industry a cost-effective arrangement of commissioned cars along the production line is desired. Although the individual cars are similar, each automobile requires particular components to be installed by different working bays along the assembly line. In addition to the different configurations of cars to be arranged along the production line, each vehicle has to be painted with exactly one color. The arising problem in which the goal is to minimize the number of color changes while considering the constraints defined by various working bays is called car sequencing problem (CarSP). A feasible solution to CarSP is a permutation of all cars to be produced at a certain day taking all constraints into consideration.

The production line itself consists of three stages: the body shop, the paint shop, and the assembly shop. In the body shop the chassis of the cars are manufactured, the paint shop workers paint the cars, and in the assembly shop different options like air condition, sun roofs, or sound systems get installed. The constraints defined by the body shop and the assembly shop are similar to each other, whereas the paint shop constraints differ significantly. For the former, we consider restrictions which can be expressed as "*No more than l_c cars are allowed to require component c in any sequence of m_c consecutive cars.*" For the latter, we consider constraints of the form: "*At most s cars with the same color are allowed to be arranged consecutively.*" Changing the color after at most s cars is motivated by a more psychological reason: In automobile industry, the paint of a car is applied using an injector. During this spraying the paint slowly agglutinates. To obtain good results the injector has to be cleaned in regular intervals. If the same color would be applied after cleaning the injector again, the staff concerned with the cleaning process would get imprecise, because "*It's the same color again.*" This leads to improper painting results, which consequently evoke reclamations.

In Section 2 we first give a formal definition of CarSP, and in Section 3 we present a brief literature review. Section 4 proposes a new integer linear programming (ILP) formulation whose solution with the general purpose ILP solver CPLEX yields proven optimal solutions for small and medium-sized problem instances. For dealing with larger instances, a second approach based on general variable neighborhood search (VNS) is introduced in Section 5. Several types of neighborhoods are presented: in addition to some adopted from previous work, also new large neighborhoods that are examined by ILP methods are introduced. For comparison with results found in the literature, we use two different sets of benchmark instances. The first set is taken from the publicly available benchmark collection called CSPLib [2]. The other set is taken from the instances proposed by the French Operations Research Society ROADEF and the car manufacturer RENAULT for the ROADEF

Challenge 2005 [9]. Experimental results, which are presented in Section 6, indicate that the new ILP approach performs well in comparison to other exact methods; in particular we were able to approximately halve the time needed to solve some of the benchmark instances. The variants of our VNS approach that include the large neighborhoods examined using ILP techniques are shown to outperform those considering only traditional structures. In fact, an experimental analysis indicates that each neighborhood we propose in this article usually contributes significantly to the overall success. In particular, the VNS-variant including all neighborhoods turns out to be competitive to the leading algorithms from the ROADEF Challenge 2005. Conclusions complete this article.

2 Formal Definition

In this section, we present a formal definition of CarSP. The notation introduced here will be used throughout the whole document.

Given are a set of possible components C, including a set of colors $F \subseteq C$, and a set K of requested configurations

$$K = \{k : k \subseteq C,\ |k \cap F| = 1\},$$

i.e. each configuration k is a subset of components to be installed, and exactly one color is selected in each configuration.

If configuration k contains component c, a corresponding 0–1 constant a_{ck} is set to 1, otherwise to 0. Component vector $a_k^{\mathbf{T}} = (a_{0k}, \ldots, a_{|C|k})$ denotes the incidence vector for configuration k. Let function $\mathrm{H}(a_i^{\mathbf{T}}, a_j^{\mathbf{T}})$ be the Hamming distance of two configurations i and j. All configurations requested in an instance of CarSP are pairwise different, i.e. $\mathrm{H}(a_i^{\mathbf{T}}, a_j^{\mathbf{T}}) \geq 1$, for all pairs $(i, j) \in K^2, i \neq j$, and for each $k \in K$ there is further given an integer demand $\delta_k \geq 1$ indicating how many vehicles of configuration k have to be produced.

A solution to CarSP is a mapping $X = (x_1, \ldots, x_n) : \{1, \ldots, n\} \to K$ specifying a sequence of length $n = \sum_{k \in K} \delta_k$, which assigns exactly one configuration $k \in K$ to each position $i = 1, \ldots, n$. Since only commissioned cars shall be produced, the restriction $|\{x_i : x_i = k\}| = \delta_k$, for all $k \in K$, has to be fulfilled. Furthermore, the number of consecutive cars painted with the same color $f \in F$ has to be less than or equal to the maximum color block size s.

For each component $c \in C$ we are given a sliding window length $m_c \in \mathbb{N}$ and a quota $l_c \in \mathbb{N}$. Only l_c cars are allowed to require component c in any subsequence of m_c consecutive vehicles. Due to the constraints defined by the

paint shop l_f is equal to s and m_f is equal to $s+1$ for all colors $f \in F$. We denote the total demand of any component $c \in C$ by $d_c = \sum_{k \in K} a_{ck} \cdot \delta_k$.

The production of the last day needs to be considered. For this purpose additional constants $e_{ci} \in \{0,1\}$, $c \in C$, $i = 1, \ldots, m_c - 1$, are used: e_{ci} is set to 1 iff the i-th last car of the previous day required component c.

Often CarSP is formulated as an optimization problem in which the total costs of color changes and weighted assembly shop constraint violations have to be minimized. For this purpose, we associate constant costs $\gamma_f > 0$, $f \in F$, with a change to color f and costs $\gamma_c > 0$, $c \in C \setminus F$, with a violation of an assembly shop constraint. The goal is to find a sequence X of commissioned cars minimizing the objective function

$$\text{obj}(X) = \sum_{i=1}^{n} \text{costs}(i) \tag{1}$$

with the costs for each position i being

$$\text{costs}(i) = \text{change}(i) + \sum_{c \in C \setminus F} \text{viol}(i,c), \tag{2}$$

$$\text{change}(i) = \begin{cases} \gamma_f \cdot \max_{f \in F} \left\{ a_{fx_1} - e_{f1_1} \right\} & \text{if } i = 1 \\ \gamma_f \cdot \max_{f \in F} \left\{ a_{fx_i} - a_{fx_{i-1}} \right\} & \text{otherwise} \end{cases} \tag{3}$$

$$\text{viol}(i,c) = \begin{cases} \gamma_c \cdot \max \left\{ 0, \sum\limits_{j=i-m_c+1}^{i} a_{cx_i} - l_c \right\} & \text{if } i \geq m_c \\ \gamma_c \cdot \max \left\{ 0, \sum\limits_{j=1}^{i} a_{cx_i} + \sum\limits_{j=1}^{m_c - i} e_{cj} - l_c \right\} & \text{otherwise} \end{cases} \tag{4}$$

under the remaining hard constraints

$$\sum_{j=1}^{i} a_{fx_j} + \sum_{j=1}^{s-i+1} e_{fj} \leq s \qquad \text{for } i = 1, \ldots, s \tag{5}$$

$$\sum_{j=i-s}^{i} a_{fx_j} \leq s \qquad \text{for } i = s+1, \ldots, n \tag{6}$$

ensuring that no more than s cars of the same color are scheduled in a row. Expression change(i) represents the costs for a potential color change at position i of sequence X and viol(i,c) denotes the costs for possible assembly shop constraint violations at position i with respect to component c. The latter are computed as the sum of cars requiring component c within the last m_c cars (including the car at position i) minus the quota l_c; if this difference is less than 0, the number of violations is set to 0.

Note that for computing the costs of color changes considering not only the new color to but also the color of the previous car would be more precise. Since

typically color changes are penalized using a constant factor independently of the involved colors, we use this simplified problem description.

3 Previous Work

Gent [1] showed that the decision problem associated with the car sequencing problem whether there exists an optimal solution without any violations of the constraints defined by the assembly line is NP-hard. Kis [8] proved that this decision problem is NP-hard in a strong sense, and Hu [6] stated that the optimization problem including the color constraints as defined in Section 2 is NP-hard, too.

Several different approaches have been made to solve CarSP or variants of it. The methods used vary from greedy heuristics to meta-heuristics like ant colony optimization, whereas only a few exact algorithms have been described.

3.1 Exact methods

Gravel et al. [4] proposed an ILP approach for a variant of CarSP without the constraints specifically related to the paint shop. It is able to solve commonly used benchmark instances with about 200 cars and 5 components to proven optimality within practically reasonable time. The main idea applied in this formulation is to group cars with the same configuration into classes to avoid symmetries.

Hu [6] describes another ILP approach that also takes constraints defined by the paint shop into account. Unfortunately, the size of practically solvable instances is limited to about 30 cars with 8 components.

Another exact ILP approach is presented in [12], which, in contrast to the approach by Hu and analogously to the formulation of Gravel et al., classifies cars with the same configuration into groups. Thereby, the size of practically solvable instances is enlarged to at most 300 cars with about 8 components.

3.2 Greedy heuristics

Gottlieb et al. [3] proposed greedy heuristics using different evaluation strategies. They construct sequences of cars by always adding the next best car in respect to some evaluation function to a current partially filled sequence. Once a car is placed at a position, it is never reconsidered again. Some of

the proposed evaluation functions take the currently available cars and the already existing partial sequence into account, whereas others only compute a global value indicating whether a car is hard to arrange without constraint violations or not.

Many other approaches like the following local search based techniques utilize similar greedy heuristics for computing initial solutions.

3.3 Local search based methods

Many attempts for solving CarSP are based on the concept of local search. Puchta et al. [3,13] proposed an approach that makes use of six different types of neighborhood structures which are defined by the following moves: exchanging two cars (swap moves), removing one car and inserting it at another position (insert move), swapping to consecutive cars (transposition moves), swapping two cars similar with respect to their configuration (similar swap moves), inverting a subsequence of cars (Lin2Opt moves) and randomly rearranging cars in a subsequence (random move). Within a local search framework, the type of move and the affected positions are chosen at random. In contrast, Jaskiewicz et al. [7] decide the initial position, where the move is applied to, by means of a greedy heuristic. Then they look for the best move to be applied at this position.

Perron et al. [10] define similar moves, but they apply them to subsequences, i.e. swap moves exchange two subsequences of cars and insert moves shift a subsequence of cars to another position.

In [12], we presented a general variable neighborhood search approach for CarSP, which combines well known neighborhood structures with newly defined ones examined using ILP techniques. The current article extends this work and discusses several aspects in more detail. For instance, the conclusions drawn by investigating preliminary results presented in [12] were used to adjust and refine the applied search parameters and data structures for evaluating single (improvement) moves. New variants of utilized neighborhood structures where included within the VNS approach as proposed in this article. Furthermore, a new ILP formulation is presented.

3.4 Ant colony optimization

Gravel et al. [4] and Gottlieb et al. [3] presented different variants of ant colony optimization (ACO) approaches for CarSP. The proposed variants differ in the local heuristics they apply for selecting the next car and in the

neighborhoods used during local search. The iterative solution construction process is guided by local heuristics, pheromone information, and random decisions. A pheromone value is maintained for each ordered pair of cars, and it is strengthened when in good candidate solutions the second car is scheduled directly after the first. In some variants of this ACO approach, each candidate solution as well as the final result is further improved by local search.

4 A New Integer Linear Programming Formulation

This section introduces a new ILP formulation for CarSP. We will later see that for many practical instances it can be significantly faster solved to optimality than previous formulations. This new ILP formulation differs from the previous ones therein that the assignment of individual components to positions of the production line is emphasized. Suitable constraints guarantee that the commissioned cars are produced.

The following variables are used: For each position $i \in \{1, \ldots, n\}$ and each component $c \in C$, 0–1 variables b_{ci} indicate whether component c is to be installed at the i-th car ($b_{ci} = 1$) or not ($b_{ci} = 0$). Furthermore, for each $k \in K$ and each position $i \in \{1, \ldots, n\}$ a binary variable p_{ki} is set to 1 iff the components assigned to position i correspond to configuration k. For each component $c \in C \setminus F$ and position $i \in \{1, \ldots, n\}$ variable g_{ci} represents the number of constraint violations occurring at the corresponding position with respect to the constraints defined for component c. A binary variable w_{fi} is set to 1 iff a change to color f occurs at position i.

The formulation as an integer linear program is as follows:

$$\min \quad \sum_{c \in C \setminus F} \gamma_c \cdot \sum_{i=1}^{n} g_{ci} + \sum_{f \in F} \gamma_f \cdot \sum_{i=1}^{n} w_{fi} \tag{7}$$

subject to

$$\sum_{f \in F} b_{fi} = 1 \qquad i \in \{1, \ldots, n\} \tag{8}$$

$$\sum_{i=1}^{n} b_{ci} = d_c \qquad c \in C \tag{9}$$

$$p_{ki} \leq a_{ck} \cdot b_{ci} + (1 - a_{ck}) \cdot (1 - b_{ci}) \qquad k \in K,\ c \in C,\ i \in \{1, \ldots, n\} \tag{10}$$

$$b_{ci} = \sum_{k \in K} a_{ck} \cdot p_{ki} \qquad i \in \{1, \ldots, n\},\ c \in C \tag{11}$$

$$\sum_{i=1}^{n} p_{ki} = \delta_k \qquad k \in K \tag{12}$$

$$g_{ci} \geq \sum_{j=1}^{i} b_{cj} + \sum_{j=1}^{m_c - i} e_{cj} - l_c \qquad i \in \{1, \ldots, m_c - 1\},\ c \in C \backslash F \qquad (13)$$

$$g_{ci} \geq \sum_{j=i-m_c+1}^{i} b_{cj} - l_c \qquad i \in \{m_c, \ldots, n\},\ c \in C \backslash F \qquad (14)$$

$$w_{f1} \geq b_{f1} - e_{f1} \qquad f \in F \qquad (15)$$

$$w_{fi} \geq b_{fi} - b_{f(i-1)} \qquad i \in \{2, \ldots, n\},\ f \in F \qquad (16)$$

$$\sum_{j=1}^{i} b_{fj} + \sum_{j=1}^{s+1-i} e_{fj} \leq s \qquad i \in \{1, \ldots, s\},\ f \in F \qquad (17)$$

$$\sum_{j=i-s}^{i} b_{fj} \leq s \qquad i \in \{s+1, \ldots, n\},\ f \in F \qquad (18)$$

$$g_{ci} \geq 0 \qquad i \in \{1, \ldots, n\},\ c \in C \backslash F \qquad (19)$$

$$w_{fi} \geq 0 \qquad i \in \{1, \ldots, n\},\ f \in F \qquad (20)$$

$$b_{ci} \in \{0, 1\} \qquad c \in C,\ i \in \{1, \ldots, n\} \qquad (21)$$

$$p_{ki} \in \{0, 1\} \qquad k \in K,\ i \in \{1, \ldots, n\} \qquad (22)$$

Objective function (7) corresponds to function $obj(X)$ in Eq. (1) and aims at minimizing the costs for color changes and constraint violations. Since each car has to be painted with exactly one color, constraints (8) are introduced. Equations (9) ensure that in total d_c cars requiring component $c \in C$ are produced. Constraints (10) allow variables p_{ki} to be set to 1 only if the components assigned to position i correspond to the requirements of configuration k. In addition, we introduce the equations (11) which guarantee that each variable b_{ci} is set in accordance to the values of the corresponding variables p_{ki}. Equalities (12) assure the production of the correct amount δ_k of each configuration $k \in K$.

To ensure that the number of occurring constraint violations is correctly counted we introduce constraints (13), (14) and (19). For counting the correct number of color changes, we add inequalities (15), (16) and (20).

Finally, we have to ensure the hard constraints defined by the paint shop, which state that in any subsequence of $s + 1$ consecutive cars at least one color change has to occur. For this purpose, we introduce constraints (17) and (18).

Like for other so far published ILP models the application of this formulation is limited to moderately sized instances with up to 300 cars and about 8 components. For approaching larger instances, we also developed an algorithm based on general variable neighborhood search in which several types of large neighborhoods are explored via integer linear programming.

5 A Heuristic Approach Based on Variable Neighborhood Search

5.1 General Variable Neighborhood Search Framework

General variable neighborhood search (VNS) is a meta-heuristic concept which follows the idea of exploiting different neighborhood structures within a local search framework in order to escape from local optima. This method was first proposed by Hansen and Mladenović, and a detailed introduction can be found in [5].

Algorithm 5.1: VND(X)

Input: an initial solution X
Output: a local optimum in respect to all available neighborhoods $\mathcal{N}_t$, for $t = 1, \ldots, t_{\max}$

$t \leftarrow 1$
repeat
  find $X^* \in \mathcal{N}_t(X)$ with $f(X^*) \leq f(X'),\ \forall X' \in \mathcal{N}_t(X)$
  if $f(X^*) < f(X)$ **then**
    $X \leftarrow X^*$
    $t \leftarrow 1$
  else
    $t \leftarrow t + 1$
until $t > t_{\max}$
return X

Variable neighborhood descent (VND) is used as a subroutine for finding solutions that are local optima with respect to a set of $t_{\max}$ different neighborhood structures $\{\mathcal{N}_1, \ldots, \mathcal{N}_{t_{\max}}\}$. Algorithm 5.1 shows this procedure which performs a local search and systematically switches between the neighborhoods. Typically, the sizes of the neighborhoods or the time complexities for evaluating them induce a natural order of them such that the smallest or fastest neighborhood is examined first followed by the more complex ones. Different step functions can be applied, but it is most common to choose a best-improvement or a next-improvement strategy.

VND is embedded in the general VNS framework, which uses a second set of neighborhood structures $\{N_1, \ldots, N_{u_{\max}}\}$ and focuses more on diversification, see Alg. 5.2. For escaping local optima, a shaking operation is performed that randomly chooses a feasible solution from the u-th neighborhood $N_u(X)$.

Neighborhood structures $\{N_1, \ldots, N_{u_{\max}}\}$ should be designed in such a way that the similarity between the incumbent solution X and the candidate solutions in $N_u(X)$ decreases with an increasing parameter u. If no improvement

Algorithm 5.2: VNS()

Output: the best heuristic solution found

generate initial solution X
repeat
  $u \leftarrow 1$
  repeat
    $X \leftarrow$ `Shaking(`u, X`)`
    $X' \leftarrow$ `VND(`X`)`
    if $f(X') < f(X)$ **then**
      $X \leftarrow X'$
      $u \leftarrow 1$
    else
      $u \leftarrow u + 1$
  until $u > u_{\max}$
until *stopping condition is met*
return X

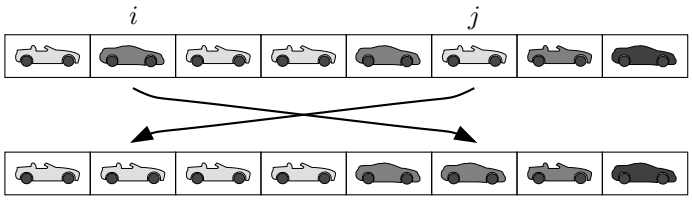

Fig. 1. Swap move: Configurations at positions i and j are swapped. All other configurations stay at their positions.

could be achieved during the last VND iteration attempts are made to escape the current local optimum by steadily increasing u. Otherwise, if the algorithm was able to find a better solution, it restarts with N_1. The whole procedure is repeated until some stopping criterion is met.

5.2 Neighborhoods for VND

In the following, we present the neighborhood structures $\mathcal{N}_1$ to $\mathcal{N}_{t_{\max}}$, which are used within VND. They utilize in general three different types of basic moves: swap, shift, and κ-exchange. In our VNS for CarSP we represent a candidate solution naturally by its configuration vector $X = (x_1, \ldots, x_n)$ and in the following we denote by π_i a subsequence of X of arbitrary length, by (x_i) the subsequence consisting of a single configuration x_i, and by "$\cdot$" the concatenation operator.

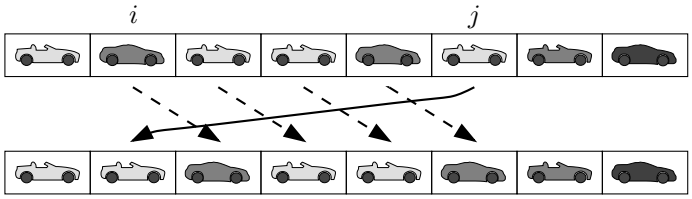

Fig. 2. Backward shift move: One configuration is moved from position j to position i. All configurations between these two positions are shifted one position backward.

5.2.1 Swap moves

A swap move $SWP(X, i, j)$ swaps the positions of two configurations in the current arrangement X, see Fig. 1; i.e. configuration x_i at position i is exchanged with configuration x_j at position j. The configurations at all other positions k, $k \neq i$ and $k \neq j$, are unaffected by this move:

$$SWP\Big(\pi_1 \cdot (x_i) \cdot \pi_2 \cdot (x_j) \cdot \pi_3, i, j\Big) = \pi_1 \cdot (x_j) \cdot \pi_2 \cdot (x_i) \cdot \pi_3 \tag{23}$$

For saving computation time in the evaluation of neighborhoods induced by this type of move, we use an incremental method for computing objective values. For this purpose, we define an array of size $n \cdot |C|$, whose entries represent the number of occurrences of each component $c \in C$ within each sliding window of size m_c. Thereby, a potential change in the number of violations for component $c \in C$ can be computed in constant time. Using a second array of size n whose entries point to the first configuration within the current color block it is further possible to detect and evaluate potential color changes and violations of the maximum color block size in constant time. For this purpose it is only necessary to save the length of each color block. In total, evaluating a single move only takes time $O(\sum_{c \in C \setminus F} m_c)$.

5.2.2 Shift moves

There are two types of shift moves depending on the displacement of the affected car: backward shift $BSH(X, i, j)$ and forward shift moves $FSH(X, i, j)$. If a backward shift move is applied to the current arrangement the configuration at position j is moved to position i with $i < j$, whereas all cars at locations k, with $k = i, \ldots, j-1$, are shifted one position backward along the production line, see also Fig. 2:

$$BSH\Big(\pi_1 \cdot (x_i) \cdot \pi_2 \cdot (x_j) \cdot \pi_3, i, j\Big) = \pi_1 \cdot (x_j) \cdot (x_i) \cdot \pi_2 \cdot \pi_3 \quad \text{(backward)} \tag{24}$$

The forward shift move is defined correspondingly:

$$FSH\Big(\pi_1 \cdot (x_i) \cdot \pi_2 \cdot (x_j) \cdot \pi_3, i, j\Big) = \pi_1 \cdot \pi_2 \cdot (x_j) \cdot (x_i) \cdot \pi_3 \quad \text{(forward)} \tag{25}$$

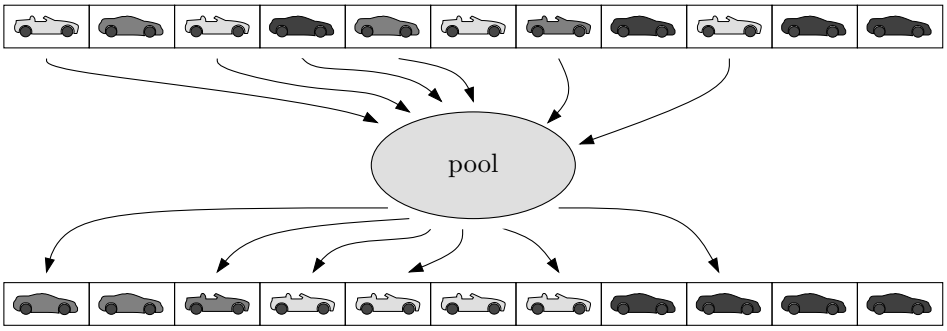

Fig. 3. κ-exchange move: A set of configurations is selected and set free, i.e. put into a pool. Then these configurations are reassigned to the free positions.

Exploiting the same data structures as for swap moves it is possible to achieve an incremental and efficient evaluation of shift moves in time $O(\sum_{c \in C \setminus F} m_c)$. Since it is required to update the utilized data structure for each affected position when a shift move is applied, the worst case consists of repositioning the last configuration of the current arrangement to the first position along the production line (or vice versa), which can only be done in time $O(n \cdot |C|)$.

5.2.3 κ-Exchange moves

For the above mentioned moves the improvement possibilities are limited, since the contribution costs for at most $2 \cdot \max_{c \in C}\{m_c\}$ configurations are changed. To enhance the potential improvement we introduce κ-exchange moves $EXG(X, S)$. For this type of moves a set S of κ different positions in the current sequence X is selected in some way, e.g. randomly or by some heuristic, and the corresponding configurations are set free and reassigned along the production line, see also Fig. 3:

$$EXG\Big(\pi_1 \cdot (x_{i_1}) \cdot \pi_2 \cdot \ldots \cdot (x_{i_\kappa}) \cdot \pi_{\kappa+1}, S\Big) = \pi_1 \cdot (x_{j_1}) \cdot \pi_2 \cdot \ldots \cdot (x_{j_\kappa}) \cdot \pi_{\kappa+1}$$
$$\text{with } (j_1, \ldots, j_\kappa) \text{ being a permutation of } S = \{i_1, \ldots, i_\kappa\} \subseteq \{1, \ldots, n\} \quad (26)$$

Using the same data structure as for swap and shift moves, an evaluation and application of this move can in general be done in time $O(n \cdot |C|)$.

Based on these three introduced types of moves, we define the following neighborhood structures.

5.2.4 Neighborhood Swapping

The swapping neighborhood $\mathcal{S}(X)$ consists of all feasible candidate solutions which can be derived from a current solution X by applying one single swap

move $SWP(X, i, j)$, i.e.

$$\mathcal{S}(X) = \Big\{X' : X' = SWP(X, i, j), \text{ for all } i = 1, \ldots, n-1,\ j = i+1, \ldots, n\Big\} \tag{27}$$

To penalize infeasible arrangements of cars, i.e. sequences violating the constraints defined by the paint shop, we define a penalization factor for violations of the paint shop constraints equal to $n \cdot |C| \cdot \max_{c \in C}\{\gamma_c\}$. In this way, it is guaranteed that the objective value of each feasible candidate solution is lower than the objective value of infeasible arrangements.

The maximum size of this neighborhood is bounded above by $\frac{n^2-n}{2}$. Therefore, completely examining this neighborhood can be achieved in time $O(n^2 \cdot \sum_{c \in C \setminus F} m_c)$. For our implementation time complexity is even in $O(n \cdot |K| \cdot \sum_{c \in C \setminus F} m_c)$ since we are caching the change of the objective function for moving configuration $k \in K$ from position j to position i. If another swap move $SWP(X, i, j')$ with $j' > j$ and $x_j = x_{j'}$ is evaluated, the cached value for position i can be reused. Since $|K| \leq n$ the additional space needed for caching the partial results is justified.

5.2.5 Neighborhood Shifting

All candidate solutions which can be derived from X by using either an backward shift move $BSH(X, i, j)$ or a forward shift move $FSH(X, i, j)$ compose the shifting neighborhood $\mathcal{SH}(X)$:

$$\begin{aligned}\mathcal{SH}(X) = \Big\{X' : X' = BSH(X, i, j) \vee X' = FSH(X, j, i),\\ \text{for all } i = 1, \ldots, n-1,\ j = i+1, \ldots, n\Big\}\end{aligned} \tag{28}$$

In addition to all feasible candidate solutions the infeasible arrangements derived from X by applying a shift move are also regarded using the same penalization strategy as for neighborhood $\mathcal{S}(X)$. Since this neighborhood consists of up to n^2-n pairwise disjoint candidate solutions and the worst case time complexity for evaluating an shift move is in $O(\sum_{c \in C \setminus F} m_c)$, completely iterating through this neighborhood can be done in time complexity $(n \cdot |K| \cdot \sum_{c \in C \setminus F} m_c)$. To achieve this result it is crucial to compute the objective values of multiple occurrences of partial arrangements only once. Therefore, the same caching strategies as described for the neighborhood $\mathcal{S}(X)$ are applied.

5.2.6 Neighborhood Greedy Swapping

The greedy swapping neighborhood $\mathcal{GS}_\kappa(X)$ of a current solution X is defined on a restricted set of all possible single swap moves $SWP(X, i, j)$ and therefore, $\mathcal{GS}_\kappa(X)$ is a subset of $\mathcal{S}(X)$. Let $\mathcal{W}_\kappa$ be the set of $\kappa \geq 1$ positions with

maximum contributions $costs(i)$ in the objective function. Ties are broken randomly. Allowed swap moves are all those where the first position i is in $\mathcal{W}_\kappa$, while the second position can be freely chosen from $\{1,\ldots,n\}\setminus\{i\}$. Therefore, this neighborhood consists of up to $n\cdot\kappa$ different candidate solutions and a complete examination is possible in time $O(n\cdot\kappa\cdot\sum_{c\in C\setminus F} m_c)$:

$$\mathcal{GS}_\kappa(X) = \Big\{X' : X' = SWP(X,i,j),\ i\in\mathcal{W}_\kappa,\ j\in\{1,\ldots,n\}\setminus\{i\}\Big\} \tag{29}$$

5.2.7 Neighborhood Greedy Shifting

The greedy shifting neighborhood $\mathcal{GSH}_\kappa(X)$ consists of all candidate solutions which can be obtained by applying a restricted single shift move to a current solution X. If $\mathcal{W}_\kappa$ is again the set of $\kappa\geq 1$ positions with maximum contributions $costs(i)$ in the objective function, only shift moves $BSH(X,i,j)$ and $FSH(X,i,j)$ whose position j is in $\mathcal{W}_\kappa$ are allowed. Again, the other position i can be freely chosen from $\{1,\ldots,n\}\setminus\{j\}$. The size of this neighborhood is limited to at most $n\cdot\kappa$ different neighbors, and examining this neighborhood can be done in $O(n\cdot\kappa\cdot\sum_{c\in C\setminus F} m_c)$.

$$\begin{aligned}\mathcal{GSH}_\kappa(X) = \Big\{X' :\ & X' = BSH(X,i,j),\ j\in\mathcal{W}_\kappa,\ i<j\\ & \vee\ X' = FSH(X,i,j),\ j\in\mathcal{W}_\kappa,\ i>j\Big\}\end{aligned} \tag{30}$$

5.2.8 Neighborhood Similar Swapping

Another special case of the neighborhood $\mathcal{S}(X)$ constitutes the similar swapping neighborhood $\mathcal{SS}_\kappa(X)$ which consists of all candidate solutions which can be derived by applying a swap move $SWP(X,i,j)$ on configurations differing in at least one but no more than κ components, i.e.

$$1\leq \mathrm{H}(x_i,x_j)\leq\kappa. \tag{31}$$

This neighborhood is formally defined as

$$\begin{aligned}\mathcal{SS}_\kappa(X) = \Big\{X' : X' = SWP(X,i,j),\ & 1\leq \mathrm{H}(x_i,x_j)\leq\kappa,\\ & \text{for all } i=1,\ldots,n-1,\ j=i+1,\ldots,n\Big\}\end{aligned} \tag{32}$$

Although this neighborhood might include up to $\frac{n^2-n}{2}$ neighbors, the size is in general much smaller. Since $\mathcal{SS}_\kappa(X)\subseteq\mathcal{S}(X)$, the time needed for completely examining this neighborhood is bounded by $O(n\cdot|K|\cdot\sum_{c\in C\setminus F} m_c)$.

5.2.9 Neighborhood Similar Shifting

Analogously to $\mathcal{SS}_\kappa(X)$ we define the similar shifting neighborhood $\mathcal{SSH}_\kappa(X)$, which is a subset of neighborhood $\mathcal{SH}(X)$ and consists of all solutions that can be produced by a shift move under restriction (31):

$$\mathcal{SSH}_\kappa(X) = \Big\{X' :\ X' = BSH(X,i,j) \vee X' = FSH(X,j,i), \\ 1 \leq \mathrm{H}(x_i, x_j) \leq \kappa, \text{ for all } i = 1,\ldots,n-1,\ j = i+1,\ldots,n\Big\} \tag{33}$$

Again, the size of this neighborhood is in practice typically much smaller than the theoretically possible size of $n^2 - n$, and the evaluation time is bounded by $O(n \cdot |K| \cdot \sum_{c \in C \setminus F} m_c)$.

5.2.10 Neighborhood κ-Exchange with Random Selection

Since all so far mentioned neighborhoods are defined by applying either a single swap or shift move, all contained candidate solutions are relatively similar to the original solution X. To increase diversification among the investigated candidate solutions the neighborhood $\mathcal{R}_\kappa(X)$ is defined exploiting a κ-exchange move $EXG(X,S)$. The set S of mutable positions along the production line is randomly chosen with $|S| = \kappa$:

$$\mathcal{R}_\kappa(X,S) = \Big\{X' : X' = EXG(X,S)\Big\} \tag{34}$$

Already for small values of κ the size of this neighborhood is huge, because there are in the worst case $\kappa!$ pairwise disjoint arrangements of the corresponding configurations. Therefore, we decided to solve the subproblem of examining this neighborhood by using an exact method based on integer linear programming.

The idea behind our approach is to define a subproblem which can be solved more efficiently than the initially stated problem and to substitute the obtained results into the original model. For this purpose, we fix all variables in this model with exception of those which correspond to the elected positions in set S. Doing this using the formulation presented in Section 4, the additional constraints

$$p_{ki} = \begin{cases} 1 & k = x_i \\ 0 & \text{otherwise} \end{cases} \qquad \text{for } i \notin S, \tag{35}$$

Algorithm 5.3: Shaking(u, X)

Input: the index u of the neighborhood to be used, the current best solution X

Output: the modified solution

for $i \leftarrow 1$ ***to*** u **do**
  swap two randomly chosen configurations x_i and x_j
return X

which ensure that only configurations corresponding to positions $i \in S$ are affected, are added. In addition, the constraints

$$b_{ci} = \begin{cases} 1 & c \in x_i \\ 0 & \text{otherwise} \end{cases} \qquad \text{for } i \notin S \tag{36}$$

for guaranteeing that the fixed configurations are properly assembled have to be added. Note that variables g_{ci} and w_{fi} cannot be fixed to certain constants since the number of constraint violations and color changes still have to be computed. Using a general purpose ILP solver, we solve the thereby defined subproblem whose feasible solutions are all valid arrangements for the CarSP. It can be guaranteed that at least one feasible solution exists, because the original solution X is part of the solution set.

5.2.11 Neighborhood κ-Exchange with Greedy Selection

We define the κ-exchange with greedy selection neighborhood $\mathcal{G}_\kappa(X)$ analogously to the neighborhood $\mathcal{R}_\kappa(X)$, but this time the set of mutable positions along the production line corresponds to set $\mathcal{W}_\kappa$ as defined in Section 5.2.6, i.e. the κ configurations with highest costs $costs(i)$ are selected:

$$\mathcal{G}_\kappa(X) = \left\{X' : X' = EXG(X, \mathcal{W}_\kappa)\right\} \tag{37}$$

Since the size of this neighborhood is again limited to at most $\kappa!$ disjoint arrangements, we once more use the ILP based methods described in the previous section for examining the neighborhood.

5.3 The VNS Framework for the CarSP

5.3.1 Initialization and Shaking

To provide an initial solution to VNS, we generate a permutation of all commissioned cars. Using this strategy it is possible that the generated sequence is invalid in respect to the constraints defined by the paint shop. Therefore,

Table 1
Order of the neighborhood structures for VND.

$\mathcal{N}_1 = \mathcal{GS}_1$	$\mathcal{N}_4 = \mathcal{GSH}_5$	$\mathcal{N}_7 = \mathcal{SS}_2$	$\mathcal{N}_{10} = \mathcal{SH}$	$\mathcal{N}_{13} = \mathcal{R}_{130}$
$\mathcal{N}_2 = \mathcal{GSH}_1$	$\mathcal{N}_5 = \mathcal{GS}_{20}$	$\mathcal{N}_8 = \mathcal{S}$	$\mathcal{N}_{11} = \mathcal{R}_{65}$	$\mathcal{N}_{14} = \mathcal{G}_{130}$
$\mathcal{N}_3 = \mathcal{GS}_5$	$\mathcal{N}_6 = \mathcal{GSH}_{20}$	$\mathcal{N}_9 = \mathcal{SSH}_2$	$\mathcal{N}_{12} = \mathcal{G}_{65}$	

we use a factor equal to $n \cdot |C| \cdot \max_{c \in C}\{\gamma_c\}$ for penalizing each paint shop constraint violation. In the further context we refer to this method for generating initial solutions as random arrangement (RA).

The shaking algorithm used for trying to escape local optima is straightforward, see Alg. 5.3. Neighborhoods $N_u(X)$, $u = 1, \ldots, u_{\max}$, with $u_{\max} = \frac{3}{4} \cdot n$, are implicitly defined by a series of u individual swaps of two randomly chosen configurations. Using this strategy has two advantages: firstly, evaluating and applying swap moves can be done efficiently and secondly, diversification and therefore the chance of escaping local optima grows with increasing values of u.

5.3.2 The Order of the Neighborhoods for VND

Our implementation of VND applies $t_{\max} = 14$ different neighborhood structures $\mathcal{N}_1, \ldots, \mathcal{N}_{14}$. The order of these neighborhoods is crucial for runtime performance of the neighborhood search. Therefore, we decided to examine them by increasing order of time complexity. Thus our VND implementation explores the more expensive neighborhoods in the end. The order we used for our test runs is printed in Table 1.

In neighborhoods $\mathcal{N}_1$ to $\mathcal{N}_6$, we turn the attention on minimizing the maximum contributions *costs*(i) to the objective function. The selection of the values for κ, i.e. $\kappa = 1, 5, 20$, is based on preliminary tests. If there was no improvement within these neighborhoods, we change our strategy for finding new and better solutions by trying to rearrange cars which are similar to each other ($\mathcal{N}_7$ and $\mathcal{N}_9$). To profit by exchanges of cars which are not very similar to each other we introduce neighborhoods $\mathcal{N}_8$ and $\mathcal{N}_{10}$. The neighborhoods based on swap moves are examined first, because exploring them can be done faster than evaluating neighborhoods utilizing shift moves. In the end the most expensive neighborhoods based on κ-exchange moves are examined—once with 65 configurations set free and the second time with 130 cars to be rearranged. These values are chosen, because preliminary tests showed that problems of this size can be solved in acceptable time using exact methods based on integer linear programming.

5.3.3 Examining the Neighborhoods

For examining neighborhoods $\mathcal{N}_1$ to $\mathcal{N}_{10}$, we use two alternative strategies: *best improvement* and *next improvement*. For the remaining neighborhoods $\mathcal{N}_{11}$ – $\mathcal{N}_{14}$, we use CPLEX 10.0 as general purpose ILP solver to solve the ILP formulation presented in [12] to explore them. Analogously to best and next improvement, we define a *limited best improvement* and a *limited next improvement* strategy. Using limited next improvement the evaluation process terminates as soon as a better integer feasible solution was found or a time limit has been reached. If no improvement could be achieved the best so far found solution is returned.

6 Tests and Results

Two different sets of benchmark instances are considered here: the first one taken from CSPLib [2] to compare our ILP approach with the one presented by Gravel et al. in [4], and the second one taken from the instances provided by ROADEF and the automobile manufacturer RENAULT for the ROADEF Challenge 2005 [9] for testing our methods on larger instances. Only the latter instances include constraints defined by the paint shop.

All experiments were performed on a dual AMD Opteron 2.4GHz PC with 4GB RAM. Our algorithm has been implemented in C++ and for solving the ILP formulations the general purpose ILP solver CPLEX 10.0 by ILOG has been used.

6.1 Results on CSPLib *Instances*

The CSPLib library itself is split into two subsets of instances. The first one contains 70 instances which are all satisfiable, i.e. there exists an optimal solution with zero costs. All of these instances consist of exactly 200 cars with 5 different components and 17 to 30 configurations. They are grouped into seven classes, each of them indicating the utilization rate of the components in the instances, i.e. the percentage of cars requiring a component in regard to all cars of the instance. Each class has ten members and we refer to each of these groups as *util-rate*-*, e.g. 60-* for the group of instances with an utilization rate of 60%.

The second subset of CSPLib benchmarks consists of nine additional instances which are partly not completely satisfiable and typically harder to solve. These instances consider 100 cars with 5 different components and 19 to 26 configu-

Table 2
Average time for solving the CSPLib instances to optimality in seconds. Values in parentheses indicate standard deviation.

			60-*	65-*	70-*	75-*	80-*	85-*	90-*
C-ILP			4.85 (5.8)	**6.65 (5.4)**	**6.65 (1.7)**	**13.16 (8.4)**	**18.65 (10.6)**	**18.53 (8.7)**	**33.63 (15.1)**
K-ILP			9.38 (6.0)	9.20 (5.1)	15.6 (3.6)	22.0 (12.5)	30.43 (12.0)	37.30 (13.7)	61.03 (26.1)
GRAVEL			**3.27 (2.4)**	6.87 (3.5)	11.94 (5.5)	15.85 (7.6)	27.86 (16.1)	40.39 (22.1)	77.26 (36.4)
best	NA	hybr.	1.7 (1.7) 98	3.1 (12.1) 99	8.6 (39.1) 94	3.1 (3.8) 93	4.5 (6.7) 93	14.6 (45.0) 97	40.2 (81.7) 93
		heur.	1.2 (0.6) 85	1.3 (0.7) 86	1.5 (0.6) 88	1.8 (0.7) 78	2.5 (0.9) 66	2.9 (1.2) 43	3.6 (1.2) 10
	RA	hybr.	0.6 (0.4) 100	0.6 (0.3) 100	0.9 (0.8) 99	2.8 (16.0) 98	6.5 (26.2) 96	4.9 (10.5) 97	10.3 (23.7) 93
		heur.	0.5 (0.2) 98	0.6 (0.3) 97	0.6 (0.3) 90	1.0 (0.3) 85	1.4 (0.4) 60	1.6 (0.6) 45	2.1 (0.6) 16
next	NA	hybr.	0.9 (1.0) 100	1.4 (2.5) 100	2.0 (4.1) 100	2.2 (3.1) 98	8.5 (22.4) 100	7.6 (25.7) 98	27.7 (72.0) 96
		heur.	0.8 (0.5) 91	0.8 (0.4) 90	1.0 (0.4) 89	1.3 (0.6) 80	2.1 (0.7) 60	2.2 (0.9) 42	3.0 (1.0) 16
	RA	hybr.	0.7 (2.5) 100	0.6 (0.7) 100	0.9 (1.2) 100	1.6 (2.4) 100	5.9 (14.9) 98	6.0 (29.3) 99	11.9 (27.2) 99
		heur.	0.4 (0.2) 94	0.5 (0.2) 91	0.6 (0.2) 94	0.8 (0.3) 80	1.2 (0.4) 57	1.4 (0.5) 42	1.7 (0.5) 18

rations.

Table 2 shows the results obtained for the test runs on the first subset of CSPLib instances. The first three rows represent the mean time for each class needed by the three ILP based exact approaches for finding optimal solutions (including proofs of optimality). The values in parentheses stand for the corresponding standard deviations. The second part of the table depicts the results obtained using the heuristic approaches. Each line contains the mean time for reaching the optimum, the corresponding standard deviation in parentheses and below these two values the number of runs reaching the optimal solution with zero costs. For the heuristic approaches ten runs per instance were performed. Therefore each mean value corresponds to 100 individual runs.

We refer with *C-ILP* to the ILP formulation presented in Section 4. *K-ILP* stands for the formulation presented in [12] and *GRAVEL* denotes the results obtained by using the formulation by Gravel et al. [4]. Since this formulation uses a method of counting constraint violations different to the strategy proposed for the ROADEF Challenge 2005, we adapted their objective function such that beside the number of positions with constraint violations the exact number of constraint violations is counted. In addition, we added a method analogously to the method presented in Section 4 for counting the number of color changes. Finally, the weighted sum of all constraint violations and the number of color changes is computed, see [11] for details. We indicate the results obtained by using our VNS approach with (limited) best improvement with *best*, whereas *next* denotes the setting with (limited) next improvement

used as step function for examining the neighborhoods.

To elaborate the impact of hybrid neighborhoods on the final results, we tested our approach using two different settings: The first one is based on the VNS-framework including all neighborhoods as described in Section 5 (we denote this setting by *hybr.*) and the other one uses the same neighborhoods except those taking advantage of the ILP based methods for examining them (denoted with *heur.*).

Since our VNS approach did not yield promising results within preliminary tests for some instances using random arrangement (RA) for generating initial solutions, we implemented a second method to which we refer as naive arrangement (NA). NA builds a sequence in a greedy way by placing all cars with configuration $k_1 \in K$ at the beginning, followed by cars with configuration $k_2 \in K$ and so on.

To summarize, the ILP formulation presented in Section 4 is in general the fastest one on these kind of instances. Since each of the instances can be solved without any constraint violations and the objective function of the ILP formulations includes no negative terms, the corresponding LP relaxation trivially is 0. Therefore, CPLEX consumes no extra time for proving optimality. Furthermore, as soon as our VNS approach reaches an objective value of 0, an optimal solution is found. Thus, the comparison of run times given in Table 2 is fair.

For the heuristic approaches, the hybrid VNS-approach yields better results than the purely heuristic one with a forfeit of computation time across the board. Further, the (limited) next improvement strategy provides faster at least as good results than using (limited) best improvement as step function and the initial solution generated by RA seems to have a positive impact on the solution process for these instances. In general, the VNS-approaches are usually significantly faster than the exact methods with the drawback that sometimes the optimal solution might not be reached.

For the second subset of CSPLib not all instances are satisfiable. Therefore, the first row of Table 3 covers the so far best known (and partly optimal) solution. The next three rows indicate the objective values obtained by solving the ILP models. Underneath these values the consumed computation times are printed, whereas the solution process was stopped after 600 seconds and the so far best solution was used. The remaining eight rows reproduce the results obtained using the VNS approaches, where the average number of violations over 10 runs, the standard deviation in parentheses and below these values the total number of runs finding the best known solutions are printed for each instance and test setting.

The time needed for solving the ILP model as proposed in Section 4 is in

Table 3
Computational results for the set of hard CSPLib instances. The first three rows present the results obtained by the exact methods whereas the remaining rows give an overview over results gained by the VNS approach.

			10-93	16-81	19-71	21-90	26-82	36-92	41-66	4-72	6-76
best sol.			3	0	2	2	0	2	0	0	6
C-ILP			8 600.0 s	0 51.9 s	2 600.0 s	2 600.0 s	0 28.8 s	2 600.0 s	0 11.9 s	0 24.1 s	6 600.0 s
K-ILP			11 600.0 s	0 280.6 s	4 600.0 s	2 600.0 s	0 23.0 s	7 600.0 s	0 13.0 s	0 35.6 s	6 600.0 s
GRAVEL			6 600.0 s	0 49.4 s	3 600.0 s	4 600.0 s	0 32.1 s	2 600.0 s	0 14.5 s	0 41.4 s	6 600.0 s
best	NA	hybr.	16.7 (6.0) 0	19.6 (7.0) 0	9.5 (4.7) 1	4.1 (4.6) 8	3.4 (5.7) 6	9.4 (2.9) 0	0.0 (0.0) 10	7.9 (6.5) 1	7.3 (1.8) 4
		heur.	25.9 (2.6) 0	24.8 (2.8) 0	17.1 (3.0) 0	15.2 (2.6) 0	12.4 (4.2) 0	17.3 (2.1) 0	6.9 (1.7) 0	15.4 (2.7) 0	12.0 (2.1) 0
	RA	hybr.	18.3 (5.0) 0	18.7 (5.5) 0	8.4 (5.0) 1	3.4 (2.4) 7	4.9 (5.2) 4	8.7 (1.5) 0	0.9 (2.0) 8	9.3 (5.5) 1	7.3 (1.4) 4
		heur.	26.5 (3.5) 0	22.6 (4.4) 0	15.9 (3.2) 0	12.7 (2.5) 0	13.2 (3.3) 0	15.2 (2.4) 0	7.7 (1.4) 0	15.1 (2.1) 0	11.8 (1.0) 0
next	NA	hybr.	7.1 (1.4) 0	0.0 (0.0) 10	3.5 (1.5) 2	3.1 (1.4) 3	0.0 (0.0) 10	3.9 (0.7) 0	0.0 (0.0) 10	0.0 (0.0) 10	6.0 (0.0) 10
		heur.	25.9 (2.8) 0	22.8 (3.8) 0	18.3 (3.8) 0	14.3 (2.3) 0	13.4 (4.5) 0	15.8 (2.6) 0	6.5 (1.5) 0	14.8 (2.4) 0	11.0 (1.3) 0
	RA	hybr.	8.0 (1.1) 0	0.0 (0.0) 10	3.2 (1.2) 3	3.0 (0.9) 3	0.0 (0.0) 10	3.7 (1.4) 2	0.0 (0.0) 10	0.2 (0.6) 9	6.0 (0.0) 10
		heur.	25.2 (2.7) 0	20.7 (3.5) 0	18.0 (2.5) 0	12.8 (2.1) 0	11.6 (2.5) 0	16.4 (3.1) 0	8.0 (2.0) 0	15.9 (1.4) 0	11.4 (1.6) 0

general shorter than the computation time needed for solving the model introduced by Gravel et al. [4]. Is is evident that the third formulation is the worst one on these instances. Results obtained for the VNS approaches are similar to the previously presented results. On this second subset of CSPLib instances the hybrid approach is much better than the pure heuristic one, which was not able to find the best known solution for any of the instances. Again, the (limited) next improvement strategy yields better results in shorter time than (limited) best improvement and the strategy for generating an initial solution seems to have no noticeable impact on the solution process.

In conclusion, the hybrid VNS approach with (limited) next improvement as step function and RA used for generating initial solutions yields the best results on CSPLib instances. The results obtained using the hybrid variants of the VNS approach justify the extra computation time needed during the solution process. For the second subset of instances the traditional approach was even not able to reach the so far best known solution for any instance. The high performance of (limited) next improvement in contrast to (limited) best improvement can be explained by the extra time available for additional VNS iterations, because especially in the large neighborhoods examined using ILP based methods much time is consumed for trying to prove optimality for

a found solution.

6.2 ROADEF *instances*

ROADEF and the automobile manufacturer RENAULT published three sets of test instances for the ROADEF Challenge 2005: set A, B, and X [9]. We used set X for the experiments documented here, since this set was also used for the final evaluation procedure and ranking of the candidates. There are 19 instances with 65 to 1319 cars, 5 to 20 colors, 5 to 26 components, and 10 to 328 configurations in set X. For the final evaluation process at the ROADEF Challenge 2005 the available computation time was limited to 600 seconds on a Pentium IV 1.6 GHz with 1 GB RAM. Since the objective function defined by RENAULT for the ROADEF Challenge 2005 is slightly different from function (1) stated previously, the number of violations which definitely occur at the first positions of the production of the next day are added to the objective function for these tests.

Although the ILP formulation presented in Section 4 yields good results on CSPlib instances, preliminary tests showed that the formulation presented in [12] yields faster at least as good results on instances including paint shop constraints. Therefore, we used this formulation for examining large neighborhoods.

Each column in Tables 4 and 5 represents one instance from set X. In the heading, the instance number, the number of cars, the number of components, and the number of colors are presented. Rows 1 to 8 show average results for each instance over ten runs using 600 seconds as time limit. The values beneath the averages indicate the standard deviations. A triple of values $v_1/v_2/v_3$ indicates v_1 violations of the highest order objective, v_2 violations of the second objective and v_3 violations of the least important objective. Again *best*/*next* represents the step function used, NA/RA denotes the strategy for generating the initial solution and *hybr.*/*heur.* indicates which sets of neighborhoods have been used for the test runs. The two rows labeled with *R-best* and *R-worst* indicate the objective value of the best and worst solution among the candidates obtained during the final evaluation procedure of the ROADEF Challenge 2005, respectively. The last three rows present results obtained by solving the instances using the different ILP formulations. Cells without any entry indicate that even solving the LP-relaxations for these instances was not possible within the given time limits. If one entry is given, this value represents a lower bound on the corresponding instance. If two values are printed in one cell, there are two possibilities: the first one indicates an upper and a lower bound for the optimum. The other value indicates the objective value of the optimum and the solution time in seconds (including the proof of optimality)

Table 4. Results obtained for set X of the ROADEF Challenge 2005. The first eight rows represent the objective values (and standard deviations below) obtained using different settings of the VNS approach. The next 2 lines represent the best and worst results obtained during the ROADEF Challenge 2005, respectively. The last three rows indicate the results obtained using CPLEX solving the discussed ILP models.

inst. n $\lvert C\rvert$ $\lvert F\rvert$			(1) 704 12 13	(2) 1260 12 13	(3) 1319 18 15	(4) 996 20 20	(5) 325 26 15	(6) 65 6 5	(7) 780 7 14	(8) 921 8 10	(9) 231 7 8	(10) 90 1 6
best	NA	hybr.	13.0/2.1/4.6 0.0/0.3/1.6	0.0/260.4/57.4 0.0/ 5.7/ 7.1	80.0/537.4/61.1 8.3/ 12.4/13.8	5.5/329.7/637.3 4.2/ 25.9/ 8.0	46.5/429.3/ 99.8 2.6/ 11.6/ 4.6	0.0/0.0/3.0 0.0/0.0/0.0	0.0/181.2/ 12.3 0.0/ 8.4/ 11.0	0.0/116.1/599.3 0.0/ 6.8/ 23.2	8.8/ 95.1/49.9 1.7/ 1.4/ 7.0	5.7/10.3/0.0 0.5/ 0.5/0.0
		heur.	13.0/4.8/9.2 0.0/0.4/1.8	0.0/257.0/64.7 0.0/ 3.5/ 9.4	79.3/542.3/61.7 11.7/ 9.9/22.2	4.5/310.3/636.5 5.4/ 18.0/ 10.7	49.7/447.6/ 98.1 4.5/ 13.0/ 2.9	0.0/0.0/4.0 0.0/0.0/0.0	0.0/186.6/ 14.1 0.0/ 6.1/ 9.9	0.0/117.0/605.7 0.0/ 11.5/ 21.6	11.3/102.5/54.2 2.8/ 3.1/ 9.2	6.0/11.0/0.0 0.0/ 0.0/0.0
	RA	hybr.	111.5/0.2/0.6 7.4/0.6/1.1	0.0/394.1/40.1 0.0/ 7.1/ 4.4	67.8/573.5/37.3 6.2/ 8.9/14.0	10.7/312.9/648.4 3.6/ 20.6/ 10.8	46.0/425.3/ 96.2 3.6/ 14.4/ 2.9	0.0/0.0/3.1 0.0/0.0/0.3	0.0/213.8/ 7.3 0.0/ 12.1/ 9.8	0.0/219.1/491.0 0.0/ 8.2/ 8.4	8.6/ 95.5/45.6 1.1/ 2.1/ 6.8	6.5/10.4/0.0 1.6/ 0.5/0.0
		heur.	170.3/0.3/0.5 11.2/0.7/0.8	0.0/398.0/41.6 0.0/ 10.1/ 8.3	68.6/570.1/34.1 8.2/ 13.4/11.5	10.3/302.3/640.5 4.0/ 15.4/ 11.1	49.2/448.2/ 96.6 3.1/ 12.9/ 4.0	0.0/0.0/4.3 0.0/0.0/0.5	0.0/218.5/ 5.5 0.0/ 12.2/ 2.4	0.0/230.1/491.9 0.0/ 5.9/ 8.4	11.5/107.2/37.1 2.5/ 2.9/ 4.1	16.8/10.9/0.0 3.6/ 0.9/0.0
next	NA	hybr.	13.0/2.0/8.5 0.0/0.0/2.5	0.0/330.3/44.5 0.0/ 11.7/ 7.9	78.0/562.2/39.1 8.2/ 16.0/13.3	6.4/353.9/637.6 3.3/ 36.8/ 8.7	44.2/428.9/ 96.6 1.8/ 11.4/ 2.6	0.0/0.0/3.0 0.0/0.0/0.0	0.0/181.2/ 7.0 0.0/ 8.4/ 6.4	0.0/175.4/506.0 0.0/ 10.5/ 9.3	8.9/ 95.7/43.6 2.2/ 2.3/ 5.2	5.2/10.8/0.0 0.4/ 0.4/0.0
		heur.	13.0/3.3/9.6 0.0/1.1/3.6	0.0/335.8/43.8 0.0/ 7.9/ 5.9	83.5/566.3/37.9 12.0/ 21.5/13.9	6.5/359.5/643.0 2.0/ 45.4/ 10.1	45.9/450.2/ 97.9 3.2/ 13.2/ 3.7	0.0/0.0/4.1 0.0/0.0/0.3	0.0/184.6/ 4.9 0.0/ 7.2/ 3.4	0.0/176.4/514.9 0.0/ 6.7/ 8.8	12.7/106.9/44.8 3.2/ 3.2/10.2	6.0/11.0/0.0 0.0/ 0.0/0.0
	RA	hybr.	85.0/0.5/1.8 11.8/0.8/1.2	0.0/376.3/42.2 0.0/ 10.8/ 4.4	81.6/578.0/31.1 6.5/ 13.5/ 5.3	8.8/349.2/634.2 3.1/ 15.5/ 15.7	45.5/434.0/ 98.9 4.1/ 18.9/ 3.6	0.0/0.0/3.0 0.0/0.0/0.0	0.0/195.4/ 3.5 0.0/ 5.9/ 1.1	0.0/206.7/484.7 0.0/ 4.4/ 8.0	8.6/ 94.7/44.8 1.9/ 1.2/ 5.5	5.5/10.7/0.0 1.1/ 0.5/0.0
		heur.	139.7/0.8/0.9 14.2/1.0/1.1	0.0/387.4/44.2 0.0/ 6.2/ 6.5	69.2/582.9/34.2 8.2/ 22.2/12.9	7.9/345.1/638.4 4.7/ 31.6/ 11.5	45.6/448.1/ 97.7 3.2/ 11.4/ 4.2	0.0/0.0/4.4 0.0/0.0/0.5	0.0/198.2/ 3.0 0.0/ 9.3/ 0.9	0.0/215.2/499.1 0.0/ 7.2/ 9.3	10.8/106.9/42.2 2.2/ 2.1/ 8.0	15.9/10.5/0.0 1.9/ 0.7/0.0
R-best			12.0/2.0/3.0	0.0/192.4/66.0	0.0/337.0/ 6.0	0.0/160.0/407.6	36.0/341.4/ 95.4	0.0/0.0/3.0	0.0/110.2/ 98.4	0.0/ 55.2/794.8	8.0/ 35.8/87.0	5.0/10.0/0.0
R-worst			12.2/2.0/4.4	0.0/314.0/44.0	49.4/444.8/33.8	16.0/950.0/509.0	338.2/408.0/118.2	0.0/9.4/3.0	0.0/180.8/223.8	0.0/145.0/530.0	12.0/103.0/52.0	6.0/10.0/0.0
C-ILP			– –	– –	– –	– –	– –	0.0/0.0/3.0 0.33 s	– –	– –	– 1 001 501.0	5.0/10.0/0.0 18.98 s
K-ILP			– –	– –	– –	– –	– –	3.0 0.25 s	– –	– –	– 1 001 501.0	5.0/10.0/0.0 48.38 s
GRAVEL			– –	– –	– –	– –	– –	3.0 17.98 s	– –	– –	– 1 001 023.0	7 012 000.0 1 552 111.0

Table 5. Results obtained for set X of the ROADEF Challenge 2005. The first eight rows represent the objective values (and standard deviations below) obtained using different settings of the VNS approach. The next 2 lines represent the best and worst results obtained during the ROADEF Challenge 2005, respectively. The last three rows indicate the results obtained using CPLEX solving the discussed ILP models.

inst.		n	(11) 376	(12) 1247	(13) 1073	(14) 519	(15) 459	(16) 875	(17) 273	(18) 264	(19) 219
$\|C\|$		$\|F\|$	2 7	12 15	12 17	22 13	20 13	11 13	6 12	5 9	4 7
best	NA	hybr.	7.0/56.0/0.0 0.0/ 0.0/0.0	0.0/119.9/1381.6 0.0/ 5.0/ 218.9	0.0/264.2/560.7 0.0/ 4.8/ 96.0	0.0/227.9/1008.2 0.0/ 3.4/ 26.9	33.3/121.7/ 960.3 1.3/ 5.7/ 30.0	85.7/263.5/ 66.0 4.1/ 6.2/ 10.1	0.0/58.5/ 0.0 0.0/ 3.4/ 0.0	0.0/42.7/0.0 0.0/ 2.4/0.0	153.0/47.5/0.0 0.0/ 2.3/0.0
		heur.	7.0/57.0/0.0 0.0/ 0.0/0.0	0.0/119.4/1447.3 0.0/ 3.1/ 202.5	0.0/263.6/627.7 0.0/ 4.5/144.7	0.0/233.2/1010.2 0.0/ 4.8/ 20.8	37.3/129.2/ 956.6 2.9/ 3.9/ 24.6	88.1/263.5/ 65.8 4.3/ 7.3/ 9.9	0.0/73.3/ 0.0 0.0/ 2.8/ 0.0	0.0/48.8/0.0 0.0/ 1.8/0.0	153.0/56.5/0.0 0.0/ 2.8/0.0
	RA	hybr.	30.7/49.4/0.0 4.1/ 2.2/0.0	0.0/296.1/ 379.1 0.0/ 14.8/ 40.3	0.0/355.8/192.5 0.0/ 12.3/ 69.0	0.0/240.5/1005.8 0.0/ 4.9/ 20.9	33.1/130.4/ 932.6 1.9/ 6.0/ 26.8	66.8/323.2/ 69.4 3.9/ 8.8/ 7.4	0.0/72.3/ 0.0 0.0/ 4.0/ 0.0	0.0/40.2/0.0 0.0/ 2.5/0.0	153.0/48.7/0.0 0.0/ 3.5/0.0
		heur.	88.1/37.4/0.0 4.7/ 2.5/0.0	0.0/282.8/ 350.2 0.0/ 7.4/ 43.9	0.0/346.8/163.9 0.0/ 12.8/ 43.0	0.0/246.4/1007.8 0.0/ 4.0/ 26.0	36.0/146.5/ 875.2 2.2/ 3.7/ 19.8	65.3/323.7/ 66.5 3.0/ 9.2/ 7.1	0.0/87.1/ 0.0 0.0/ 6.0/ 0.0	0.0/66.3/0.0 0.0/ 3.6/0.0	153.0/58.2/0.0 0.0/ 4.0/0.0
next	NA	hybr.	7.0/56.0/0.0 0.0/ 0.0/0.0	0.0/118.8/1108.9 0.0/ 5.7/ 151.3	0.0/296.4/207.5 0.0/ 3.4/ 45.2	0.0/232.5/1007.9 0.0/ 4.0/ 17.6	32.5/125.0/ 933.6 1.6/ 10.9/ 45.0	66.5/310.1/ 74.1 4.2/ 6.7/ 6.6	0.0/59.3/ 0.0 0.0/ 5.7/ 0.0	0.0/36.9/0.0 0.0/ 1.9/0.0	153.0/48.3/0.0 0.0/ 2.2/0.0
		heur.	7.0/56.9/0.0 0.0/ 0.7/0.0	0.0/119.9/1136.5 0.0/ 6.3/ 163.8	0.0/293.2/210.9 0.0/ 10.0/ 39.0	0.0/239.3/1001.8 0.0/ 4.9/ 27.9	35.3/140.4/ 894.4 2.3/ 5.3/ 25.2	67.1/309.5/ 73.8 3.3/ 10.3/ 8.7	0.0/74.3/ 0.0 0.0/ 5.8/ 0.0	0.0/44.4/0.0 0.0/ 3.0/0.0	153.0/59.1/0.0 0.0/ 2.4/0.0
	RA	hybr.	27.5/49.9/0.0 3.8/ 1.7/0.0	0.0/204.9/ 634.0 0.0/ 15.3/ 77.0	0.0/330.3/173.6 0.0/ 8.4/ 56.0	0.0/238.1/1003.8 0.0/ 4.3/ 15.2	33.5/128.7/ 939.1 2.0/ 4.5/ 34.6	65.6/320.0/ 69.2 3.7/ 8.0/ 9.0	0.0/61.9/ 0.0 0.0/ 4.8/ 0.0	0.0/38.4/0.0 0.0/ 2.8/0.0	153.0/48.3/0.0 0.0/ 2.9/0.0
		heur.	74.8/40.0/0.0 8.1/ 2.6/0.0	0.0/198.6/ 682.3 0.0/ 18.1/ 71.0	0.0/326.8/180.9 0.0/ 10.5/ 66.1	0.0/249.2/ 998.6 0.0/ 4.6/ 13.1	34.5/149.1/ 875.9 2.4/ 5.3/ 27.8	64.7/321.9/ 67.9 1.9/ 5.1/ 8.6	0.0/85.5/ 0.0 0.0/ 3.1/ 0.0	0.0/64.2/0.0 0.0/ 4.0/0.0	153.0/59.2/0.0 0.0/ 2.8/0.0
R-best			6.0/56.0/0.0	0.0/ 69.0/ 239.0	0.0/231.0/ 30.0	0.0/196.0/1005.6	31.0/ 76.8/1116.2	61.0/187.2/ 29.8	0.0/37.0/ 0.0	0.0/30.0/0.0	153.0/34.0/0.0
R-worst			7.0/56.0/0.0	0.0/ 87.8/ 437.0	0.0/285.0/101.0	2.2/209.6/1107.2	34.8/150.6/ 938.0	67.0/263.0/119.6	0.0/72.4/ 0.0	0.0/53.0/0.0	306.0/32.0/0.0
C-ILP			32 116 000.0 1 176 181.0	– –	– –	– –	– –	– –	139 120 000.0 1 679.0	48 003.0 5 100.0	221 060 000.0 146 010 000.0
K-ILP			31 103 000.0 6 000 000.0	– –	– –	– –	– –	– –	– 810.0	49 000.0 7 900.0	– 146 000 000.0
GRAVEL			54 098 000.0 1 000 000.0	– –	– –	– –	– –	– –	106 090 000.0 1 000.0	55 000.0 1 012.0	– 146 000 000.0

Table 6
Mapping of short names to names used by ROADEF and RENAULT for the ROADEF Challenge 2005.

short name	original name	# cars	# components	# colors
(1)	022_RAF_EP_ENP_S49_J2	704	12	13
(2)	023_EP_RAF_ENP_S49_J2	1260	12	13
(3)	024_EP_RAF_ENP_S49_J2	1319	18	15
(4)	025_EP_ENP_RAF_S49_J1	996	20	20
(5)	028_CH1_EP_ENP_RAF_S50_J4	325	26	15
(6)	028_CH2_EP_ENP_RAF_S51_J1	65	6	5
(7)	029_EP_RAF_ENP_S49_J5	780	7	14
(8)	034_VP_EP_RAF_ENP_S51_J1_J2_J3	921	8	10
(9)	034_VU_EP_RAF_ENP_S51_J1_J2_J3	231	7	8
(10)	035_CH1_RAF_EP_S50_J4	90	1	6
(11)	035_CH2_RAF_EP_S50_J4	376	2	7
(12)	039_CH1_EP_RAF_ENP_S49_J1	1247	12	15
(13)	039_CH3_EP_RAF_ENP_S49_J1	1073	12	17
(14)	048_CH1_EP_RAF_ENP_S50_J4	519	22	13
(15)	048_CH2_EP_RAF_ENP_S49_J5	459	20	13
(16)	064_CH1_EP_RAF_ENP_S49_J1	875	11	13
(17)	064_CH2_EP_RAF_ENP_S49_J4	273	6	12
(18)	655_CH1_EP_RAF_ENP_S51_J2_J3_J4	264	5	9
(19)	655_CH2_EP_RAF_ENP_S52_J1_J2_S01_J1	219	4	7

labeled with s. A mapping of the names used in Tables 4–5 to the original names as used by ROADEF and RENAULT can be found in Table 6.

Referring to Table 4 and 5, it can be seen that for instances 6 and 10 we were able to find best solutions and prove their optimality using the exact methods based on integer linear programming. Further, we were able to compute lower bounds for instances 11 and 19 which indicate that the currently known best solution is close to the optimum.

In general, the instances proposed for the ROADEF Challenge 2005 are too large to be solved exactly. Even computing the LP-relaxations of currently available ILP formulations for ROADEF instances is in general too expensive in practice. The VNS approach proposed in this paper provides competitive results in comparison with the best methods of the challenge. Especially the combination of the VNS scheme with the exact ILP approach for searching large neighborhoods leads to excellent results in comparison with the pure heuristic VNS approach.

We further studied the individual contributions of each neighborhood during a run of VNS. Figure 4 exemplarily shows for instance 15 measured improvement ratios, i.e. the ratios of neighborhood evaluations yielding improved solutions to the total number of evaluations. The first plot are the results for

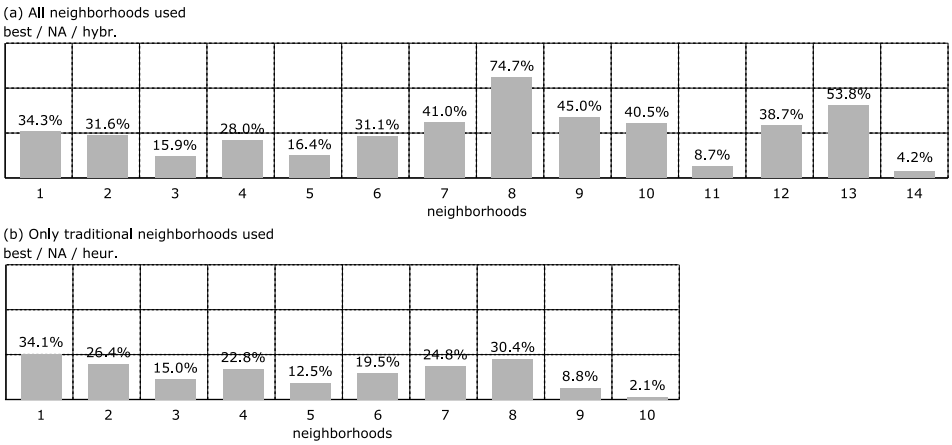

Fig. 4. This plot presents a characteristic graphical output of statistics indicating the ratio of improvements and total searches in the individual neighborhoods.

the hybrid approach with (limited) best strategy and NA for generating initial solutions. The second plot show ratios obtained by using the same setting but without those neighborhoods examined using ILP based methods. These results—together with those presented in Table 4—verify that examining the neighborhoods examined using ILP based methods is beneficial and can improve the final results after 600 seconds CPU time.

It is interesting that the approaches using NA yield better results than the approaches using RA for generating initial solutions for 14 out of 19 ROADEF instances. This happens, because initial solutions generated by NA are mostly infeasible, since the constraints defined by the paint shop are violated in general. Therefore, the first iterations of VND are typically used for repairing these solutions with respect to these constraints. However, once such a solution is feasible with respect to the paint shop constraints, it will usually contain substantially fewer color changes than a random solution derived by RA. In the further optimization, it seems to be hard for the VNS to make large improvements with respect to the number of color changes, while focusing at the same time on the other objective of reducing l_c/m_c constraint violations.

We were not able to notice a significant performance difference between (limited) best and next improvement, although a trend towards (limited) next improvement being more promising can be detected.

7 Conclusions

In this paper we presented an ILP formulation which can be used to solve instances with up to about 300 cars to proven optimality. On CSPLib instances, this formulation leads to significantly shorter computation time than compa-

rable formulations from literature. This in particular holds for instances with high utilization rates. For two ROADEF instances we were able to provide new optimality proofs for already known solutions and for two other instances we were able to compute tight lower bounds indicating that the currently known best solution is close to the optimum.

Further, we presented a general variable neighborhood search approach to the car sequencing problem, which takes advantage of by searching large neighborhoods by means of integer linear programming. Since the consideration of these is beneficial but computationally expensive, we combined them with traditional neighborhoods that can be quickly searched by enumeration. This approach is competitive compared to the methods presented during the ROADEF Challenge 2005 and the variants of our VNS approach including neighborhoods examined using ILP based methods outperform those variants without these neighborhoods.

For the future, we have plans to add and investigate further large neighborhood structures and other ILP techniques for searching them. For instance, it seems promising to also consider Lagrangian relaxation based methods.

References

[1] I. P. Gent. Two results on car-sequencing problems. Technical report, APES-02-1998, Department of Computer Science, University of Strathclyde, UK, April 1998.

[2] I. P. Gent and T. Walsh. CSPLIB: a benchmark library for constraints. Technical report, APES-09-1999, Department of Computer Science, University of Strathclyde, UK, 1999.

[3] J. Gottlieb, M. Puchta, and C. Solnon. A study of greedy, local search, and ant colony optimization approaches for car sequencing problems. In G. R. Raidl et al., editors, *Proceedings of the Applications of Evolutionary Computing on EvoWorkshops 2003*, volume 2611 of *Lecture Notes in Computer Science*, pages 246–257. Springer-Verlag Berlin Heidelberg, 2003.

[4] M. Gravel, C. Gagné, and W. L. Price. Review and comparison of three methods for the solution of the car sequencing problem. *Journal of the Operational Research Society*, 56(11):1287–1295, November 2005.

[5] P. Hansen and N. Mladenović. A tutorial on variable neighborhood search. Technical Report G-2003-46, Les Cahiers du GERAD, HEC Montréal and GERAD, Canada, 2003.

[6] B. Hu. Interaktive Reihenfolgeplanung für die Automobilindustrie. Master's thesis, Vienna University of Technology, Vienna, Austria, 2004.

[7] A. Jaszkiewicz, P. Kominek, and M. Kubiak. Adaptation of the genetic local search algorithm to a car sequencing problem. *7th National Conference on Evolutionary Algorithms and Global Optimization, Kazimierz Dolny, Poland*, pages 67–74, 2004.

[8] T. Kis. On the complexity of the car sequencing problem. *Operations Research Letters*, 32(4):331–335, July 2004.

[9] A. Nguyen. Challenge ROADEF'2005: Car sequencing problem. online reference at `http://www.prism.uvsq.fr/~vdc/ROADEF/CHALLENGES/2005/`. last visited October 31, 2006.

[10] L. Perron and P. Shaw. Combining forces to solve the car sequencing problem. In J.-C. Régin and M. Rueher, editors, *CPAIOR*, volume 3011 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 2004.

[11] M. Prandtstetter. Exact and heuristic methods for solving the Car Sequencing Problem. Master's thesis, Vienna University of Technology, Vienna, Austria, 2005.

[12] M. Prandtstetter and G. R. Raidl. A variable neighborhood search approach for solving the car sequencing problem. In *Proceedings of the XVIII Mini EURO Conference on VNS*, Tenerife, Spain, 2005.

[13] M. Puchta and J. Gottlieb. Solving car sequencing problems by local optimization. In *Proceedings of the Applications of Evolutionary Computing on EvoWorkshops 2002*, volume 2279 of *Lecture Notes in Computer Science*, pages 132–142, London, UK, 2002. Springer-Verlag.