A Variable Neighborhood Search Approach for Solving the Car Sequencing Problem*

Matthias Prandtstetter[1], Günther R. Raidl[1]

[1]Institute of Computer Graphics and Algorithms,
Vienna University of Technology, Vienna, Austria
{prandtstetter|raidl}@ads.tuwien.ac.at

Abstract

In this paper we present a new method for solving large instances of the Car Sequencing Problem (CarSP) including constraints defined by the assembly shop and the paint shop. Especially the latter are of greater significance, since they allow no violations. Our approach combines general Variable Neighborhood Search with Integer Linear Programming (ILP) methods and benefits from the advantages of both techniques. While two neighborhoods—Swapping and Inserting—are adopted from previous work, two others are based on a new ILP formulation for CarSP, and CPLEX is used as general purpose ILP solver for identifying best solutions within these neighborhoods. The comparison with results obtained during the ROADEF Challenge 2005 shows that this approach is promising and competitive. In particular, we were able to obtain some new, so far unknown best solutions for some of the ROADEF instances.

Keywords: Car Sequencing Problem, Integer Linear Programming, Variable Neighborhood Search

1 Introduction

The production of cars involves several steps that are performed in sequence. Although the vehicles are similar to each other, each car requires particular components which are assembled by different working bays. The workload for these stations has to be smoothed. This is due to the fact that workers with too much load get tired and make mistakes whereas underemployed workers only raise

*This work is supported by the RTN ADONET under grant 504438 and the Austrian Science Fund (FWF) under grant P16263-N04.

costs. In the Car Sequencing Problem (CarSP) a sequence is searched which takes the constraints defined by the working bays into account. This sequence can be described as a permutation of all cars to be produced at the current day. In addition, the number of color changes within this sequence has to be minimized.

The production line itself consists of three stages: the body shop, the paint shop, and the assembly shop. Each of these stages has its own set of constraints which have to be met when arranging the cars. There are different possibilities for constraints defined by the assembly shop and body shop. We consider those which can be expressed as "*No more than l_c cars are allowed to require component c in a sequence of m_c consecutive cars.*" For the paint shop we consider the constraint: "*At most s cars with the same color are allowed to be arranged consecutively.*" As shown in [5], this problem is known to be NP-hard.

In this paper we propose a new Integer Linear Programming (ILP) formulation, which is used to examine two types of large neighborhoods used within a general Variable Neighborhood Search (VNS). Two other types of neighborhoods are adopted from previous work. Experimental results of the new approach are presented and compared. Conclusions complete this paper.

Formalization of the Car Sequencing Problem

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

$$K = \{k : k = k_{comp} \cup \{k_{col}\} \text{ with } k_{comp} \subseteq C \backslash F \wedge k_{col} \in F\}.$$

Each configuration 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 variable a_{ck} is set to 1, otherwise to 0. All configurations are pairwise different, and for each $k \in K$, there is a demand $\delta_k \geq 1$ which indicates how many vehicles with configuration k have to be produced. Exactly one configuration is assigned to each car in the resulting arrangement. We denote the total demand of any component $c \in C$ by $d_c = \sum_{k \in K} a_{ck} \cdot \delta_k$; $n = \sum_{k \in K} \delta_k$ is the total number of commissioned cars. For each component $c \in C$ we are given a length m_c and a quota l_c. Only l_c cars are allowed to require component c in any sequence of consecutive vehicles with length m_c. For all colors $f \in F$, l_f is equal to s and m_f is equal to $s + 1$, where s is the maximum color block size allowed. We formulate the CarSP as an optimization problem in which the number of constraint violations has to be minimized. If x_i denotes the i-th car of the sequence and $conf(i)$ the configuration of the car

x_i, we search a permutation $X = (x_1, \ldots, x_n) : \{1, \ldots, n\} \rightarrow \{1, \ldots, n\}$ of the commissioned cars which minimizes the objective function $obj(X)$:

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

$$costs(i) = change(i) + \sum_{c=1}^{|C \setminus F|} viol(i, c) \cdot cost_c \qquad \forall i \in \{1, \ldots, n\}$$

$$change(i) = \begin{cases} cost_f & \text{if a colour change occurred at position } i \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in \{1, \ldots, n\}$$

$$viol(i, c) = \begin{cases} \max(0, \sum_{j=i-m_c+1}^{i} a_{c,conf(i)} - l_c) & \text{if } i \geq m_c \\ \max(0, \sum_{j=1}^{i} a_{c,conf(i)} + \sum_{j=1}^{m_c - i} e_{cj} - l_c) & \text{otherwise} \end{cases} \quad \forall i \in \{1, \ldots, n\}, \ \forall c \in C \setminus F$$

The number of violations at position i by component c is computed as the sum of cars requiring component c within the last m_c cars (including x_i) minus the quota l_c. If this difference is less than 0, the number of violations is set to 0. $cost_c$ is a penalization factor of violations occurring for component c. If i is less than m_c, the production of the last day has to be borne in mind. Additional constants $e_{ci},\ c \in C,\ i = 1, \ldots, m_c - 1$, are used for this purpose: e_{ci} is set to 1 iff the i-th last car of the previous day required component c. To give constraints defined by the paint shop precedence, they are penalized by the factor $n \cdot \max_{c \in C} \{cost_c\}$.

2 Previous Work

There were several attempts to solve the CarSP. Few of them used exact methods [5, 3] based on ILP. However, due to the complexity of the problem, the application of them is limited to relatively small instances. For larger instances methods based on metaheuristics were particularly successful. Gottlieb *et al.* [2] proposed greedy heuristics using different decision strategies. Some of these decision functions take the currently available cars and the already existing sequence into account.

Much more attempts use local search. Puchta *et al.* [10, 2] defined six different moves including swap moves, insert moves, transposition moves, and random moves. The type of move and the affected positions are chosen at random. Jaszkiewicz *et al.* [6] use similar neighborhoods, but determine in a first step the initial positions the moves are applied to using greedy heuristics, the second step of the moves follows a best-neighbor strategy. Perron *et al.* [7] define similar moves, but they apply them to subsequences. Swap moves are redefined as swapping two sequences of cars, and block insert

moves shift a sequence of cars to another position.

Gravel *et al.* [3] and Gottlieb *et al.* [2] presented algorithms using Ant Colony Optimization with different heuristics to select the car that is considered next.

3 An Exact Approach Based on Integer Linear Programming

We improved the ILP formulation proposed by Hu [5] by grouping cars with the same configurations. Figure 1 shows the resulting formulation.

Here, we give a rough overview only. A detailed discussion can be found in [9]. Variable g_{ci} represents the number of constraint violations at position i with respect to component c, whereas variable w_{fi} indicates a possible color change. The binary variable p_{ki} is 1 if the car at position i requires configuration k. The value of r_{ci} indicates how often component c is required for the cars at positions 1 to i. Variable b_{fi} represents the color for the car at position i. The objective function (1) tries to seek a minimum of color changes and constraint violations. Constraints (2)–(5) ensure that at each position along the production line exactly one car is manufactured. Constraints (6)–(11) are responsible for counting the number of violations. Constraints (13)–(15) make sure that no violations of the paint shop constraints can occur. Finally, constraints (16)–(18) count the number of color changes. When solving this ILP using a general purpose solver like CPLEX, instances up to 65 cars and about 10 components plus four colors can be handled well.

4 A Combination of VNS and the ILP Approach

Since today's exact methods are not applicable to large instances of the Car Sequencing Problem in practice, all methods with acceptable run times are of heuristic nature [2, 3, 6, 7, 8, 10]. We developed an algorithm based on a combination of general VNS [4] and the ILP approach introduced in the previous section. This hybridization combines advantages of both approaches.

4.1 Neighborhoods

As local improvement inside the VNS we use a Variable Neighborhood Descent (VND) iterating through six different neighborhoods: *Swapping*, *Inserting*, two variations of κ*-Opt with Random Selection* and two variations of κ*-Opt with Greedy Selection*. All but the first two neighborhoods are

objective function

$$\min \quad \sum_{c \in C \setminus F} cost_c \cdot \sum_{i=0}^{n-1} g_{ci} + \sum_{f \in F} cost_f \cdot \sum_{i=0}^{n-1} w_{fi} \tag{1}$$

subject to

$$p_{ki} \in \{0,1\} \qquad \forall i \in \{0,\ldots,n-1\}, \forall k \in K \tag{2}$$

$$\sum_{k \in K} p_{ki} = 1 \qquad \forall i \in \{0,\ldots,n-1\} \tag{3}$$

$$\sum_{i=0}^{n-1} p_{ki} = \delta_k \qquad \forall k \in K \tag{4}$$

$$\sum_{i=0}^{n-1} \sum_{k \in K} a_{ck} \cdot p_{ki} = d_c \qquad \forall c \in C \tag{5}$$

$$r_{c0} = 0 \qquad \forall c \in C \setminus F \tag{6}$$

$$0 \leq r_{ci} \qquad \forall c \in C,\ \forall i \in \{1,\ldots,n\} \tag{7}$$

$$r_{ci} = r_{c(i-1)} + \sum_{k \in K} \left(a_{ck} \cdot p_{k(i-1)}\right) \qquad \forall c \in C,\ \forall i \in \{1,\ldots,n\} \tag{8}$$

$$0 \leq g_{ci} \qquad \forall c \in C,\ \forall i \in \{0,\ldots,n\} \tag{9}$$

$$g_{ci} \geq r_{c(i+1)} - l_c + \sum_{j=0}^{m_c-2-i} e_{cj} \qquad \forall i \in \{0,\ldots,m_c-2\},\ \forall c \in C \setminus F \tag{10}$$

$$g_{ci} \geq r_{c(i+1)} - r_{c(i+1-m_c)} - l_c \qquad \forall i \in \{m_c-1,\ldots,n-1\},\ \forall c \in C \setminus F \tag{11}$$

$$0 \leq b_{fi} \leq 1 \qquad \forall f \in F,\ \forall i \in \{0,\ldots,n\} \tag{12}$$

$$b_{fi} = \sum_{k \in K} a_{fk} \cdot p_{ki} \qquad \forall f \in F,\ \forall i \in \{0,\ldots,n-1\} \tag{13}$$

$$\sum_{j=0}^{s-i-1} e_{fj} + \sum_{j=0}^{i} b_{fj} \leq s \qquad \forall i \in \{0,\ldots,s-1\},\ \forall f \in F \tag{14}$$

$$\sum_{j=i-s}^{i} b_{fj} \leq s \qquad \forall i \in \{s,\ldots,n-1\},\ \forall f \in F \tag{15}$$

$$0 \leq w_{fi} \leq 1 \qquad \forall f \in F,\ \forall i \in \{0,\ldots,n-1\} \tag{16}$$

$$w_{f0} \geq b_{f0} - e_{f0} \qquad \forall f \in F \tag{17}$$

$$w_{fi} \geq b_{fi} - b_{f(i-1)} \qquad \forall f \in F,\ \forall i \in \{0,\ldots,n-1\} \tag{18}$$

Figure 1: The ILP formulation.

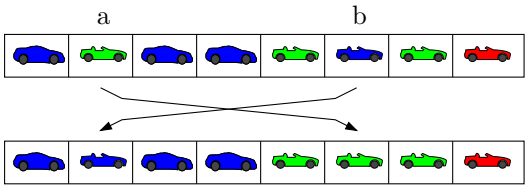

Figure 2: The cars at positions a and b are swapped. All other cars stay at their positions.

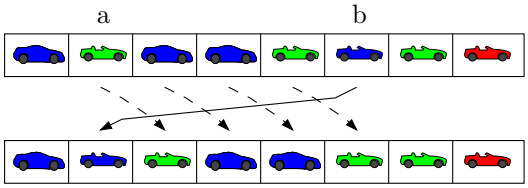

Figure 3: One car is shifted from position b to position a. All cars between these two positions are moved one position backward.

examined using the ILP approach described in section 3. Although four of them are defined similarly, they cover different subsets of all possible (integer) solutions. The neighborhoods Swapping and Inserting utilize moves previously defined by Gottlieb *et al.* in [2]. In the following, we denote by π_i a subsequence of the permutation Π of arbitrary length and by $\langle x_i \rangle$ the subsequence consisting of the single car x_i and "$\cdot$" is the concatenation operator.

4.1.1 Swapping

The Swapping neighborhood $\mathcal{N}_S(x)$ of a current solution x consists of all solutions generated by swapping two cars, see Fig. 2:

$$\mathcal{N}_S(x) = \left\{ x' : x' = \pi_1 \cdot \langle x_i \rangle \cdot \pi_2 \cdot \langle x_j \rangle \cdot \pi_3 \ \wedge \ x = \pi_1 \cdot \langle x_j \rangle \cdot \pi_2 \cdot \langle x_i \rangle \cdot \pi_3 \right\} \tag{19}$$

There exist at most $\frac{n^2-n}{2}$ different neighbors within this neighborhood. To efficiently evaluate all these moves we use an incremental method for computing the objective value with time complexity in $O(m_c)$.

4.1.2 Inserting

In the Inserting neighborhood $\mathcal{N}_I(x)$ one car is removed from its position and inserted elsewhere. All cars placed between these positions are moved one position forward or backward, see Fig. 3:

$$\mathcal{N}_I(x) = \left\{ x' : x' = \pi_1 \cdot \langle x_i \rangle \cdot \pi_2 \cdot \pi_3 \ \wedge \ x = \pi_1 \cdot \pi_2 \cdot \langle x_i \rangle \cdot \pi_3 \right\} \cup \qquad \text{(backward)}$$

$$\left\{\ x' : x' = \pi_1 \cdot \pi_2 \cdot \langle x_i \rangle \cdot \pi_3\ \wedge\ x = \pi_1 \cdot \langle x_i \rangle \cdot \pi_2 \cdot \pi_3\ \right\} \qquad \text{(forward)} \qquad (20)$$

There are at most $n^2 - n$ different neighbors. An incremental evaluation of one insert move can be implemented in time $O(m_c)$.

4.1.3 κ-Opt with Random Selection

A swap move rearranges only two cars and an insertion move rearranges only one car and shifts a subsequence by one position. To allow larger changes and better local optima the neighborhood $\mathcal{N}_{R_\kappa}(x)$ is defined. It is a reduced form of a general κ-Opt neighborhood: κ positions are selected at random and the associated cars may arbitrarily exchange their positions while the other $n - \kappa$ positions remain fixed:

$$\begin{aligned}\mathcal{N}_{R_\kappa}(x) = \big\{\ & x' : x' = \pi_1 \cdot \langle x_{j_1} \rangle \cdot \pi_2 \cdot \ldots \cdot \langle x_{j_{\kappa-1}} \rangle \cdot \pi_\kappa \cdot \langle x_{j_\kappa} \rangle \cdot \pi_{\kappa+1}\ \wedge \\ & \wedge\ x = \pi_1 \cdot \langle x_{i_1} \rangle \cdot \pi_2 \cdot \ldots \cdot \langle x_{i_{\kappa-1}} \rangle \cdot \pi_\kappa \cdot \langle x_{i_\kappa} \rangle \cdot \pi_{\kappa+1}\ \big\} \\ & \text{with } (j_1, \ldots, j_\kappa) \text{ being a permutation of } \{i_1, \ldots, i_\kappa\} \subseteq \{1, \ldots, n\}\end{aligned} \qquad (21)$$

To examine this neighborhood, the ILP defined in Section 3 is used. If position i is fixed with car of configuration k, the variable p_{ki} is set to 1 and all other variables p_{ji} with $j \in K \setminus \{k\}$ are set to 0. If the position i is set free, all variables $\mathbf{p}_{ki}$ with $k \in K$ are set free, too. Since for large κ running times for completely solving this subproblem may be too long, we limit the allowed CPU-time and use the best solution found so far in case of an early termination. Within this neighborhood, there are $\binom{n}{\kappa}$ different possibilities to select κ positions. Once the free positions are chosen, there are $O(\kappa!)$ different arrangements.

4.1.4 κ-Opt with Greedy Selection

This neighborhood $\mathcal{N}_{C_\kappa(x)}$ is similar to neighborhood $\mathcal{N}_{R_\kappa(x)}$ except for the strategy of selecting the free variables. In contrast to $\mathcal{N}_{R_\kappa}$, we choose the positions which cause the most costs. Since the cars at these positions violate more (important) constraints than the other cars, it is promising to rearrange these cars first. If there are more than κ cars causing the maximum costs, cars at the beginning of the sequence are favored:

$$\begin{aligned}\mathcal{N}_{C_\kappa}(x) = \big\{\ & x' : x' = \pi_1 \cdot \langle x_{j_1} \rangle \cdot \pi_2 \cdot \ldots \cdot \langle x_{j_{\kappa-1}} \rangle \cdot \pi_\kappa \cdot \langle x_{j_\kappa} \rangle \cdot \pi_{\kappa+1}\ \wedge \\ & \wedge\ x = \pi_1 \cdot \langle x_{i_1} \rangle \cdot \pi_2 \cdot \ldots \cdot \langle x_{i_{\kappa-1}} \rangle \cdot \pi_\kappa \cdot \langle x_{i_\kappa} \rangle \cdot \pi_{\kappa+1}\ \big\}\end{aligned}$$

$$\text{with } (j_1, \ldots, j_\kappa) \text{ being a permutation of } I = \{i_1, \ldots, i_\kappa\} \subseteq \{1, \ldots, n\} \tag{22}$$

$$\text{and } \sum_{k \in I} cost(i_k) = \max_{S \subset \{1,\ldots,n\} \wedge |S| = \kappa} \left\{ \sum_{k \in S} cost(i_k) \right\} \tag{23}$$

Altogether this neighborhood consists of up to $\kappa!$ different arrangements.

4.2 The VNS Framework for the Car Sequencing Problem

4.2.1 Initialization

Our approach uses two alternative heuristics to generate initial solutions required by VNS: *Naive Arrangement* (NA) and *Random Arrangement* (RA). NA builds a sequence so that all cars with configuration $k_1 \in K$ are placed at the beginning, followed by cars with configuration $k_2 \in K$ and so on. RA randomly shuffles the sequence obtained with NA.

4.2.2 Shaking

The shaking algorithm we use is straight-forward. In the k-th shaking neighborhood, k random swap moves are performed. If no better solution is identified, k is increased up to $k_{max} = 3/4 \cdot n$ and then reset to 1.

4.2.3 Order of Neighborhoods

First, swap moves are applied to the current best solution, because Swapping is the smallest neighborhood and its evaluation is fastest. Then, Inserting is considered. Afterwards κ-Opt with Random Selection is applied with $n/7$ free variables followed by κ-Opt with Greedy Selection also with $n/7$ free variables. Neighborhoods five and six are κ-Opt with Random Selection and κ-Opt with Greedy Selection too, but this time with $2n/7$ free variables.

4.2.4 Examining the Neighborhoods

For examining the neighborhoods Swapping and Inserting, we use two alternative strategies: *best improvement* and *next improvement*. In the case of the ILP-based neighborhoods, we define a *limited next improvement* and a *limited best improvement* strategy. Using limited next improvement the evaluation process terminates as soon as a better integer feasible solution was found or a time limit

inst. #cars	#comp #col		NA-best	NA-next	RA-best	RA-next	ROADEF
(1)	16	obj	1476069 (14643)	**1083690** (0)	1278579 (44647)	1087460 (18945)	3912479
1161	19	time	600.59 (0.37)	600.84 (0.6)	600.7 (0.3)	601.12 (0.94)	4096795
(2)	16	obj	310009 (18)	313226 (0)	352265 (4628)	312917.8 (9938)	**172180**
1161	19	time	600.67 (0.36)	600.58 (0.68)	600.71 (0.32)	602.05 (2.85)	317852
(3)	20	obj	53972412 (300711)	53656104 (435090)	**53066483** (11069)	53261107 (394380)	54003076
365	20	time	558.93 (125.13)	525.26 (120.1)	546.91 (82.09)	495.43 (138.65)	72687159
(4)	20	obj	**53070948** (300711)	53976425 (299744)	53188023 (314732)	57601641 (29631439)	54049124
365	20	time	388.34 (123.08)	326.66 (161.78)	328.75 (187.72)	360.48 (211.53)	54078415
(5)	4	obj	**61057057** (0)	62056061 (0)	61857959 (599801)	62057357 (631982)	67052049
128	7	time	1.36 (0.01)	1.24 (0.03)	1.37 (0.08)	1.24 (0.1)	67061061
(6)	4	obj	62047063 (0)	**61046065** (0)	61448566 (490924)	61747565 (640549)	67036061
128	7	time	1.32 (0.01)	1.21 (0.01)	1.39 (0.09)	1.19 (0.07)	67053060
(7)	11	obj	208922 (1198)	216566 (2970)	334637 (11350)	293388 (10900)	**189103**
1000	17	time	601.18 (0.72)	529.86 (50.82)	601.39 (1.63)	514.87 (95.93)	211317
(8)	25	obj	201417 (1418)	212096 (2457)	228550 (4890)	221359 (6261)	**161378**
591	14	time	545.98 (43.52)	451.94 (101.8)	589.14 (23.54)	491.97 (115.8)	233461
(9)	14	obj	204455 (902)	213500 (6516)	261083 (8162)	233987 (11471)	**130187**
825	14	time	602.69 (4.66)	521.5 (80.94)	601.16 (1.18)	558.84 (53.35)	234094

Table 1: Results for some instances defined by ROADEF. 025_EP_ENP_RAF_S22_J3 (1), 025_EP_RAF_ENP_S22_J3 (2), 028_ch1_EP_ENP_RAF_S22_J2 (3), 028_ch1_EP_RAF_ENP_S22_J2 (4), 035_ch1_EP_ENP_RAF_S22_J3 (5), 035_ch1_EP_RAF_ENP_S22_J3 (6), 039_ch3_EP_RAF_ENP_S22_J4 (7), 048_ch1_EP_RAF_ENP_S22_J3 (8), and 064_ch1_EP_RAF_ENP_S22_J3 (9).

has been reached. Limited best improvement tries to find the best improvement within the given time limit. If this is not possible, the best so far found solution is returned.

5 Tests and Results

For comparison with other methods, we decided to use instances proposed by the French Operations Research Society ROADEF and the automobile manufacturer Renault for the ROADEF Challenge 2005 [1]. These 80 instances include constraints defined by the assembly line and the paint shop. They consist of up to 1319 cars, 4 to 24 colors, 4 to 26 components, and up to 339 different configurations.

All experiments were performed on a Pentium 4 2.40 GHz PC with 1 GB RAM. Our algorithm has been implemented in C++. For solving the ILPs the general purpose ILP solver CPLEX 9.0 by

ILOG, Inc., is used. Table 1 summarizes some of the obtained results. On the one hand, we decided to display results for instances we obtained new, so far unknown solutions. On the other hand, we chose to present typical performance of our algorithm. We used 4 different setups: NA-best, NA-next, RA-best, and RA-next. NA and RA refer to the method providing the initial solution. *best* and *next* indicate (limited) best improvement and limited next improvement strategies for examining the neighborhoods, respectively. The results shown in the table are the average over 10 runs. The values in parentheses indicate corresponding standard deviations. The column labeled with ROADEF shows the best and worst results obtained among the candidates of the ROADEF Challenge 2005. Additional to the objective value, we provide the time until the best solution was found. All test runs were limited to 600 seconds CPU-time—the same time as allowed by ROADEF. We used 60 seconds as time limit for the examination of one neighborhood.

The solutions obtained indicate that our approach is promising and competitive. Although RA provides the best so far found solution for instance 3, NA performs in general better. There is no wide difference between best and next, except for large instances. For these, VNS was not able to provide a local optimum even in respect to the first two neighborhoods Swapping and Inserting. In this case, next outperforms best.

6 Conclusion and Future Work

We combined a newly developed ILP formulation with VNS to benefit from the advantages of both techniques. Our approach consists of six different neighborhoods. Two of them—Swapping and Inserting—are defined using swap and insert moves, respectively. The remaining neighborhoods are special cases of κ-Opt: one with Random Selection and one with Greedy Selection. They are examined using the ILP formulation and CPLEX as general purpose solver.

Test revealed that this hybridization performs well on real world instances provided by the car manufacturer Renault for the ROADEF Challenge 2005. In general, we achieved results comparable to the results obtained during the ROADEF Challenge 2005, and for some instances we even found so far unknown best solutions.

Future work will focus on the incorporation of additional neighborhoods and a more intelligent, dynamic strategy for switching between them. In addition, it would be interesting, if a better initial solution improve the overall performance of our VNS. Therefore, new heuristics for quickly generating initial solutions will be added to the framework.

References

[1] Roadef challenge 2005. http://www.prism.uvsq.fr/~vdc/roadef/challenges/2005/. last verified on 31st of August 2005.

[2] 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, 2003.

[3] 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*, to appear.

[4] 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.

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

[6] 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.

[7] 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.

[8] L. Perron, P. Shaw, and V. Furnon. Propagation guided large neighborhood search. In M. Wallace, editor, *CP*, volume 3258 of *Lecture Notes in Computer Science*, pages 468–481. Springer, 2004.

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

[10] 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.