



DIPLOMARBEIT

Exact and heuristic methods for solving the Car Sequencing Problem

ausgeführt am
Institut für Computergrafik und Algorithmen
der technischen Universität Wien

unter der Anleitung von
a.o. Univ. Prof. Dipl.-Ing. Dr. Günther Raidl
und
Univ.-Ass. Dipl.-Ing. Bin Hu

durch
Matthias Prandtstetter

Am Neubau 28
A-2100, Korneuburg

August 2005

Abstract

In this thesis, I present several different methods for solving the Car Sequencing Problem. The process of scheduling vehicles along a production line has to take several constraints into account which are defined by the body shop, the paint shop and the assembly shop. I attached importance especially to the paint shop constraints, because no violations are allowed here. Beside the enhancement of an Integer Linear Program (ILP), I developed an alternative formulation. I give a review about the Variable Neighbourhood Search (VNS) and combine this metaheuristic with the exact methods to achieve good heuristic results in relatively short time. In addition, a newly developed heuristic using exact methods is presented which is also used as starting heuristic for VNS. I compare the results obtained by my algorithms with results published in the literature using two sets of benchmarks. One set is taken from the CSPLIB, an publicly available set of problems. ROADEF and the car manufacturer Renault published the other set of instances for the ROADEF Challenge 2005. In contrast to the CSPLIB instances, the ROADEF instances include constraints defined by the paint shop. In some cases, I was able to half the computation time for the CSPLIB instances using exact methods. For the instances defined by ROADEF some new best solutions were obtained.

Zusammenfassung

In dieser Diplomarbeit präsentiere ich einige verschiedene Methoden um das Car Sequencing Problem zu lösen. Die Reihung der Fahrzeuge für die Produktion muss verschiedenste Bedingungen erfüllen, die von der Karosseriefertigung, der Lackierstation und dem Fließband vorgegeben werden. Besonderes Augenmerk lege ich auf die Bedingungen der Lackierstation, da hier keine Verletzungen auftreten dürfen. Zusätzlich zur Verbesserung eines ganzzahligen linearen Programms (ILP), das in [10] vorgestellt wurde, entwickle ich einen neuen Ansatz. Es wird ein Überblick über variable Nachbarschaftsuche (VNS) gegeben, die dann, um gute Lösungen zu erhalten, mit den exakten Ansätzen kombiniert wird. Weiters wird eine neue Heuristik, die exakte Methoden mitverwendet, vorgestellt, die auch als Startheuristik für VNS verwendet wird. Die Ergebnisse, die meine Algorithmen liefern, werden mit Resultaten aus der Literatur mit Hilfe von zwei verschiedenen Instanzbibliotheken verglichen. Die eine Gruppe von Instanzen wird von der CSPLIB – eine öffentliche Sammlung von Benchmarks – zur Verfügung gestellt. ROADEF und der Autohersteller Renault stellten die andere Instanzsammlung für die ROADEF Challenge 2005 zur Verfügung. Im Gegensatz zu den Instanzen der CSPLIB verfügen die Instanzen von ROADEF über Bedingungen, die von der Lackierstation definiert werden. In einigen Fällen konnte ich die benötigte Rechenzeit zum Lösen der CSPLIB-Instanzen mit exakten Methoden halbieren, sowie neue, bessere Lösungen zu manchen ROADEF-Instanzen präsentieren.

Danksagung

An dieser Stelle möchte ich mich bei all jenen bedanken, die in direkter oder indirekter Weise zum Gelingen dieser Arbeit beigetragen haben:

Gunnar Klau danke ich für die erste Einführung in das Thema. Bin Hu und Günther Raidl danke ich für die darauf folgende umfassende, ausgezeichnete Betreuung während der letzten Monate. Allen übrigen Mitarbeitern in der Abteilung für Algorithmen und Datenstrukturen bin ich ebenso zu Dank verpflichtet: Jeder einzelne von ihnen hat seinen Beitrag zum Gelingen dieser Arbeit geliefert. Weiters gilt mein Dank Dr. Rudolf Wild, der mir die Besuche bei der MAN AG in Wien und Steyr ermöglichte.

Meine Studienkollegen trugen ebenfalls viel zu dieser Arbeit bei – auch schon lange bevor ich überhaupt damit begonnen habe. Ausgiebige Lernsessions, Interessante Gespräche und sinnlose Blödeleien begleiteten mich durch mein Studium – nichts davon würde ich missen wollen.

Natürlich gilt mein Dank auch meiner Familie. Besonders meinen Eltern und meiner Schwester, die mir ein sorgloses Studium ermöglichten, und Niki, der sich die Zeit nahm, die Arbeit mehrmals Korrektur zu lesen. Ursula möchte ich für die moralische Unterstützung und die schöne gemeinsame Zeit danken.

*Bildung ist das, was übrig bleibt, wenn man alles,
was man in der Schule gelernt hat, vergisst.*
Albert Einstein

Contents

1	Introduction	1
2	The Car Sequencing Problem	2
2.1	Real world problem	2
2.2	Problem reduction	5
2.3	ROADEF Challenge 2005	6
2.3.1	Counting the violations	6
2.3.2	Paint shop constraints	7
2.3.3	Priority of constraints	7
2.4	Formalisation	7
2.5	Suggestions	8
2.6	Setting used for this thesis	9
3	Related Work	10
3.1	Greedy heuristics	10
3.2	Local Search	10
3.3	Ant Colony Optimisation (ACO)	11
3.4	Exact methods	11
3.5	The paint shop only	11
4	Integer Linear Programming Formulations	12
4.1	Component ILP	13
4.1.1	ILP formulation	16
4.1.2	Example for the formulation	18
4.1.3	Correctness of the formulation	19
4.2	Configuration ILP	19
4.2.1	ILP formulation	21
4.2.2	Example for the formulation	23
4.2.3	Correctness of the formulation	23
4.3	Equivalence of the formulations	24
4.4	Implementation and runtime	25

5	Variable Neighbourhood Search (VNS)	27
5.1	Local search	27
5.2	Shaking	29
5.3	General VNS	30
6	Combination of exact methods and metaheuristics	32
6.1	Combination VNS - ILP	32
6.1.1	CarSPShaking	32
6.1.2	Neighbourhoods	33
6.1.2.1	Swapping	33
6.1.2.2	Inserting	34
6.1.2.3	Selection by Random	36
6.1.2.4	Selection by Costs	36
6.1.2.5	Order of neighbourhoods	37
6.1.3	Initial solution	37
6.2	Partitioning	38
6.2.1	ILP for part one	39
6.2.2	ILP for part two	39
6.2.3	First Incumbent strategy	40
7	Tests and results	41
7.1	Results on CSPlib instances	41
7.1.1	ILPs	42
7.1.2	VNS	43
7.1.3	Partitioning	43
7.2	Results on ROADEF instances	43
7.2.1	Set A	44
7.2.2	Set B	46
7.2.3	Set X	48
7.3	Results on new instances	50
8	Conclusion and future work	52
8.1	Conclusion	52
8.2	Future work	52
8.2.1	Reimplementing Swapping and Inserting	53
8.2.2	Combination of best and next improvement	53
8.2.3	Additional neighbourhoods	54
8.2.4	Modified Variable Neighbourhood Search	55
8.2.5	Expansion of Partitioning	56
8.2.6	Relative vs. absolute	56

Contents

Appendix	57
A Original ILP formulation by Hu	57
B Modified ILP by Gravel et al.	58
C New test instances	60
C.1 Instance 1	60
C.2 Instance 2	62
D Curriculum Vitae	63

List of Figures

2.1	Stages along the production line	2
2.2	Different buffer strategies	4
2.3	A detailed view of an assembly line	5
4.1	An ILP in stadard form	12
5.1	Basic local search strategy.	27
5.2	Different optima are found using different strategies	28
5.3	An exemplary objective function	30
6.1	A swap move	34
6.2	An insertion move	34
6.3	A move using a neighbourhood examined by an ILP	36
7.1	Results for the instances in ROADEF set A	45
7.2	Results for the first group of instances in ROADEF set B	47
7.3	Results for the second group of instances in ROADEF set B	47
7.4	Results for the first group of instances in ROADEF set X	49
7.5	Results for the second group of instances in ROADEF set X	49
7.6	Solution process for instance 1 using the ILPs.	50
7.7	The relation between the time needed to reach an optimum and the time to prove it for instance 2.	51

List of Tables

2.1	Difference of counting violations	6
2.2	Symbols used in the formalisation	8
2.3	Error in counting violations	9
4.1	Used symbols	13
4.2	A test instance	18
6.1	Symbols used for definition of the neighbourhoods	33
6.2	Apply an insertion move	35
7.1	Average times for solving the first set of CSPLIB instances to optimality in seconds.	42
7.2	The standard deviation from the average times presented in table 7.1.	42
7.3	The average time and the standard deviation for finding the best obtained solution.	43
7.4	Denotation of names to the six different setups.	44
7.5	Results obtained using set A with best improvement.	45
7.6	Results obtained using set A with next improvement.	45
7.7	Results obtained using set B with best improvement.	46
7.8	Results obtained using set B with next improvement.	46
7.9	Results obtained using set X with best improvement.	48
7.10	Results obtained using set X with next improvement.	48
C.1	The configurations for the current day and their demand.	60
C.2	The production of day $N - 1$ for instance 1.	61
C.3	The configurations for the current day and their demand.	62
C.4	The production of day $N - 1$ for instance 2.	62

1 Introduction

The production of cars involves several steps that are performed in sequence. Although the vehicles are similar to each other, each car has particular components installed which are assembled by different working bays. The workload for these stations has to be smoothed. As shown in [10], this problem is known to be NP-hard. This means that there exists no polynomial deterministic algorithm at present, which leads to a proveable optimal solution for this so called Car Sequencing Problem.

In 2003, the French Operations Research society ROADEF pronounced a challenge with the Car Sequencing Problem as subject. The contest is called ROADEF Challenge 2005, since the evaluation procedure ended in February 2005.

Bin Hu, who participated in this challenge, proposed an exact approach for solving the Car Sequencing Problem using integer linear programming methods inter alia in his master's thesis [10]. I enhanced his Integer Linear Program.

In addition to this enhancement, I present heuristics for solving the Car Sequencing Problem. One combines the well known metaheuristic Variable Neighbourhood Search with newly developed exact algorithms. The other divides the problem into two sub-problems which are solved using exact methods.

Thesis overview

In the next chapter, I will give a detailed description of the Car Sequencing Problem including a real world problem and its reduction for partly academic development of different algorithms. Further, a formal description and some suggestions for evaluating the objective function are presented. Chapter 3 describes different strategies for solving the Car Sequencing Problem and other related topics, which can be found in literature.

In chapter 4, I discuss two different Integer Linear Programming formulations. I also prove that these two formulations describe the same polyhedron of integer feasible solutions. Afterwards, in chapters 5 and 6, I present the Variable Neighbourhoods Search and its combination with the exact methods described in chapter 4. Furthermore, a new heuristic for solving the Car Sequencing Problem is discussed.

The two last chapters present test results with various instances and describe some new methods which might be promising.

2 The Car Sequencing Problem

This section describes the Car Sequencing Problem. First the real world problem is presented, then the problem is reduced to a formulation which can also be found in the literature. Differences between this description and the description defined by ROADEF Challenge 2005, which I used for this thesis, will be highlighted. At the end of the chapter a formal description of the problem is given.

2.1 Real world problem

During my visit at the company MAN AG, Vienna and Steyr, Austria, I was introduced to the task of scheduling a set of trucks with (slightly) different configurations along a production line so that the workload for the workers is smoothed. This is due to the fact that workers with too much load get tired and make mistakes whereas underemployed workers unnecessarily raise costs.

For the Car Sequencing Problem a sequence is searched which takes the constraints defined by the working bays into account. In addition, the number of colour changes within this sequence has to be minimised.



Figure 2.1: Stages along the production line

The production line itself consists of three stages: the body shop, the paint shop and the assembly shop (see figure 2.1 and [8, 13]). Each of these stages has its own set of constraints which have to be met when arranging the cars along the production line. Because various working bays along the production line install different components and options, the assembly shop and the body shop define similar constraints. While the body shop produces different chassis, e.g. with three, four or five doors, the assembly shop installs components like air conditions or different gear boxes.

There are several constraints defined by the assembly and body shop:

- No more than l cars are allowed to require the component c in a sliding window of m vehicles.

2 The Car Sequencing Problem

- Exactly l cars requiring component c have to be spread over each subsequence with length m .
- No vehicle with component c_1 may be following a car with component c_2 .
- The number of cars requiring component c has to fall into $[l_{min}, l_{max}]$ for each day.
- At least l_{min} cars without component c have to follow a vehicle with component c installed.

In contrary, the paint shop defines only one constraint:

- At most s cars with the same colour are allowed to be arranged consecutively.

In addition to this constraint, the number of colour changes has to be minimised. There are two reasons for this. First, colour changes are very expensive in time and money, since the injector has to be cleaned. In addition, this pollutes the environment. Second, the injector has to be cleaned after a given number of cars for preserving good performance. If the injector is not cleaned frequently, the colour would agglutinate. This would lead to improper painting results. Since cleaning the injector is a displeasing and exhausting work, the staff would get imprecise if the same colour would be used afterwards. Therefore the colour has to be changed after cleaning the injector.

An arrangement with low costs and smoothed workload for the body shop can be expensive in respect to the paint shop and vice versa. To cope with these circumstances, stocks and buffers are interposed between the stages, because they provide the possibility of rearranging the cars during the production process.

Figure 2.2 (a) shows a setup with multiple parallel lines. At the end of the incoming line, it has to be decided to which buffer line the next car is put. The outgoing line is filled with cars taken from the parallel buffer lines. They operate with *First In, First Out* (FIFO) strategy. The loop in figure 2.2 (b) enables the recirculation of cars. This method is often used if the cars have to be repaired. If a car is not properly assembled, it is taken out of the arrangement. After the problem has been corrected, the car is inserted at a new position. The layout shown in figure 2.2 (c) grants random access to all cars currently in the buffer. Therefore it is guaranteed that a buffered car can be reinserted at each position. Figure 2.3 shows an exemplary assembly line layout with different buffers during the production process.

Since buffers and stocks raise additional costs, a layout with as few buffers as possible is preferred. Therefore scheduling applications have to consider all different stages of the production process to meet as many constraints as possible.

In big car factories there are several production lines in parallel which merge into each other. At the first production line the chassis is produced. The second production line provides the gear boxes, whereas on the third production line the motor is assembled. Each of these production lines defines its own restrictions and each time a motor is installed, the production line providing the engines must already have produced the

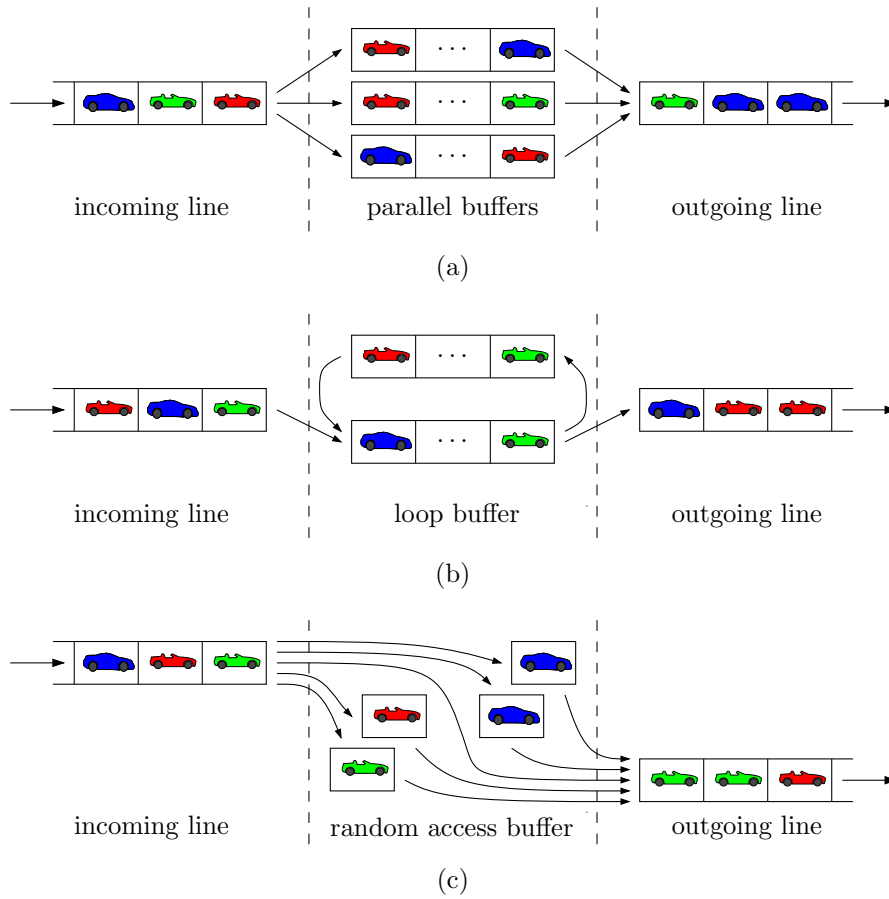


Figure 2.2: Different buffer strategies: rearrangement through merging of parallel lines (a); recirculation (b) and random access stock (c).

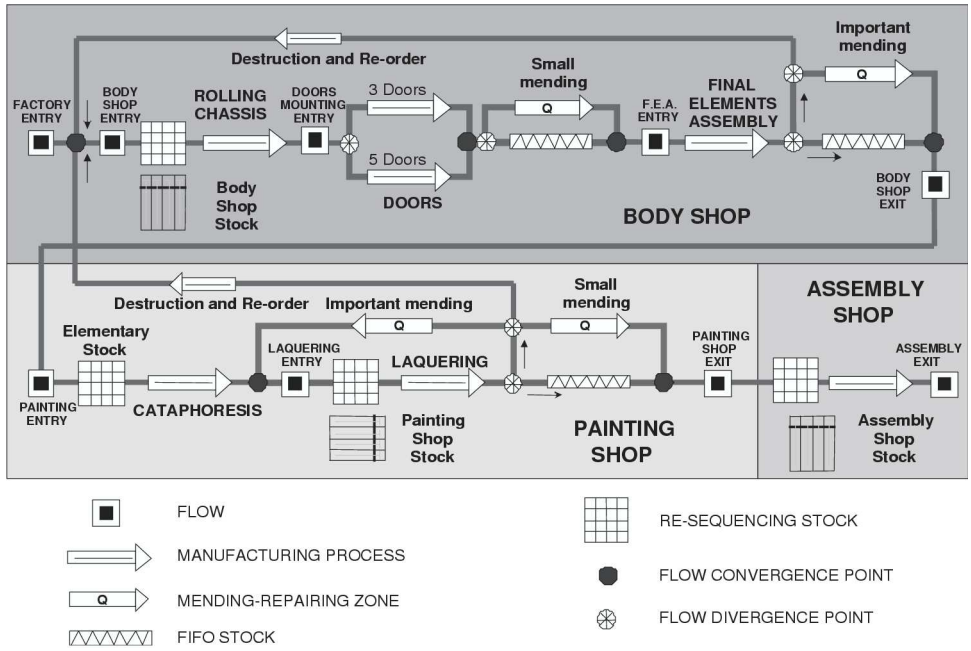


Figure 2.3: A detailed view of an assembly line [13].

appropriate motor. Therefore the scheduling process has to take several production lines into account.

Since the car factory operates 24 hours a day, the production of the last day has to be borne in mind, because the last cars produced at day $N - 1$ influence the workload in the morning of day N , i.e. the current day. The final arrangement of vehicles along the production line for day N results of two major steps. First of all the cars to be produced are chosen from a pool of commissioned vehicles. Secondly this set of cars is arranged to meet as many constraints as possible.

2.2 Problem reduction

Since the real world problem as stated above is too hard to solve, the problem is reduced. The first part of the selection can be neglected for two reasons. The procedure of extracting a set of vehicles is very extensive and some constraints cannot be formulated by exact methods since they include imprecise information. These constraints include restrictions like: "Client A is a better customer than client B. Therefore vehicles ordered by client A have to be produced faster than the other ones." Since expression like "is a better customer" cannot be measured, it is difficult to express these constraints in a mathematical sense. On the other hand, the arrangement of vehicles along the production line is on its own hard to solve. Furthermore only one assembly line without any buffers and stocks is considered.

Many assembly line restrictions are disregarded in problem descriptions which can be found in literature ([5, 6, 7]). Altogether only one kind of constraints is left. These are constraints which can be expressed as ratio l_c/m_c ("No more than l_c cars are allowed to require the component c in a sliding window of m_c vehicles"). In addition, constraints defined by the paint shop are neglected completely and constraints violations are counted regardless of the workloads for the working bays.








2.3 ROADEF Challenge 2005

Since the ROADEF Challenge 2005 [1] was supported by the automobile manufacturer Renault, the problem description used for the challenge included the constraints defined by the paint shop and the working bay constraints which can be expressed as ratios l/m .








In contrast to instances produced for academic testing (see [5, 7]), instances published by Renault [1] also take the production of the day before into account.

2.3.1 Counting the violations

ROADEF introduced an advanced method for counting violations of constraints defined by the assembly shop. In the literature ([5, 6, 7]) only the positions where a violation occurs are counted, whereas for the ROADEF Challenge 2005, the number of violations is calculated. Table 2.1 shows an example for different countings. Let us consider a component with the constraint that at most one car within three cars is allowed to require the component. Therefore the constraint can be expressed as $1/3$. If counting as proposed in literature is applied, this setup leads to 3 violations ($\not\leq$). If the method defined by ROADEF is used, we count 4 violations.

...								...	total
				$\not\leq$	$\not\leq$	$\not\leq$			3

(a)

...								...	total
	0	0	0	1	2	1	0		4

(b)

Table 2.1: Difference of counting violations: used in literature (a) and by ROADEF (b). Cars marked with + indicate cars requiring the component.

2.3.2 Paint shop constraints

So far I only described the constraints defined by the assembly shop in detail. Although the paint shop constraints differ in verbal formulation, they can be expressed in a similar way to the assembly shop constraints. Since at most s consecutive cars can be painted with the same colour, at least one car in $s + 1$ cars has to be painted with another colour (s denotes the *colour block* or *paint batch limit*). Therefore the constraints can be reformulated as

No more than s cars are allowed to require the colour f in a sliding window of $s + 1$ vehicles.

This formulation is equivalent to the formulation stated above for assembly line constraints, whereby this constraint can be expressed as ratio $s/(s+1)$.

2.3.3 Priority of constraints

ROADEF defined three different priority levels of constraints to be considered: paint shop constraints, important assembly line constraints and less important assembly line constraints. This means, that a violation of an important or high priority constraint (HPRC) is worse than a violation of a less important or low priority constraint (LPRC). To differ between these classes of importance, weight factors are assigned to each occurring violation or colour change. These factors are taken from the set $\{1, 10^3, 10^6\}$.

The objective function has to take these costs into account which is simply done by building the weighted sum of all constraint violations and colour changes.

2.4 Formalisation

I will give a formal description of the problem, here. This notation will be used in the whole document.

There is a given set of components C , a set of colours $F \subseteq C$ and a set of configurations

$$K = \{k : k = k_c \cup \{k_f\} \text{ with } k_c \subseteq C \setminus F \wedge k_f \in F\}.$$

This means that each configuration is a subset of components to be installed, but exactly one colour has to be applied to one configuration. For each $k \in K$, there is a demand δ_k which indicates how many vehicles with configuration k have to be produced. $n = \sum_{k \in K} \delta_k$ is the number of commissioned cars. Each component c defines a length m_c and a quota l_c . Only l_c cars are allowed to require component c in a sliding window of length m_c . This can be written as l_c/m_c . For all colours $f \in F$, l_f is equal to s and m_f is equal to $s + 1$, where s is the maximum colour block allowed. $cost_c$ represents the

2 The Car Sequencing Problem

costs raised if a violation of the corresponding constraints occurs. A permutation Π of the commissioned cars is searched which minimises the objective function $\kappa(\Pi)$.

$$\begin{aligned} \kappa(\Pi) &= \sum_{i=1}^n costs(i) \\ costs(i) &= change(i) + \sum_{j=1}^{|C \setminus F|} (viol(i, j) \cdot cost_j) && \forall i \in \{1, \dots, n\} \\ change(i) &= \begin{cases} cost_f & \text{if a colour change occurred at position } i \\ 0 & \text{otherwise} \end{cases} && \forall i \in \{1, \dots, n\} \\ viol(i, c) &= \text{number of violations at position } i \text{ by component } c \end{aligned}$$

Table 2.2 shows the symbols used in the formalisation.

C ... set of components $c \in C$
$F \subseteq C$... set of colours $f \in F$
k ... set of configurations $k \in K$
δ_k ... demand for configuration $k \in K$
l_c ... quota for component $c \in C$
m_c ... sliding window length for component $c \in C$
s ... maximum colour block length
l_f ... quota for colour $f \in F$; set to s
m_f ... sliding window length for colour $f \in F$; set to $s + 1$
$cost_c$... costs raised by a violation of the constraint for component $c \in C$
$cost_f$... costs for a colour change with colour $f \in F$
$costs(i)$... costs raised by the car placed at position i
$viol(c, i)$... number of violations occurring at position i for component $c \in C \setminus F$
n ... number of cars (is equal to $\sum_{k \in K} \delta_k$)
$\kappa(\Pi)$... costs for sequence Π






Table 2.2: Symbols used in the formalisation

2.5 Suggestions








As also stated in [7], this traditional method of counting violations privileges the last positions along the production line, since violations at those positions are counted less than violations in the beginning or middle of the sequence. Table 2.3 illustrates this phenomenon where the last positions of the current day are shown. Let us assume, that the constraint for this component is expressed as $1/3$. In table (a) only 3 violations are

2 The Car Sequencing Problem

counted, whereas in table (b) the "right" number of 4 violations is counted. Although the positions past the last car are not used, they are needed for counting, since violations at the end of a day are less penalised than violations in the middle.

...					
	0	0	0	1	2

(a)

...							
	0	0	0	1	2	1	0

(b)

Table 2.3: The traditional counting (a) and the new one (b). Cars marked with + indicate cars requiring the component.

To avoid this disparity between different positions, the next day $N + 1$ should be considered too. Since the arrangement of cars produced on day $N + 1$ can change, it should be assumed that all cars of day $N + 1$ require no components. Furthermore the colour of the first car of the next day is different to the colour of the last car of the current day. Because of this, no violation of paint shop constraints can occur.

2.6 Setting used for this thesis

For this thesis I use the method of counting the violations of the working bay constraints defined by ROADEF. Furthermore I accounted for the constraints defined by the paint shop and the day $N - 1$ is borne in mind, too. For comparison with test results found in literature the traditional method of counting the violations is used.

3 Related Work

Several different approaches have been made to solve the Car Sequencing Problem or parts of it. The methods used vary from greedy heuristics to Ant Colony Optimisation (ACO), whereas only a few exact algorithms were proposed.

3.1 Greedy heuristics

Gottlieb et al. [6] proposed greedy heuristics using different evaluation strategies. In principle, they build sequences of cars simply by 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 stays there. Some of the proposed evaluation functions take the currently available cars and the already existing sequence into account, whereas others only compute a global value indicating whether a car is hard to arrange or not.

Many other approaches like Local Search or ACO use similar greedy heuristics for computing initial solutions [6, 7, 11].

3.2 Local Search

Many attempts for solving the Car Sequencing Problem use Local Search. These approaches try to (locally) improve intermediate solutions until no further improvements can be achieved. Puchta et al. [6, 17] proposed a Local Search using six different moves. A current solution is rearranged using swap moves (swapping two cars), insert moves (remove one car and insert it at another position), transposition moves (swapping two consecutive cars), similar swap moves (swapping two cars similar in respect to their configuration), Lin2Opt moves (inverting a subsequence of cars) and random moves (rearranging the cars in a subsequence randomly). The type of move and the affected positions are chosen by random. In contrast, Jaskiewicz et al. [11] decide the initial position, where the move is applied to, by using a greedy heuristic first. Afterwards they look for the best move to be applied at this position.

Perron et al. [15] define similar moves, but they apply them to subsequences. Therefore swap moves are defined as swapping two sequences of cars and block insert moves shift a sequence of cars to another position.

3.3 Ant Colony Optimisation (ACO)

The idea behind Ant Colony Optimisation (ACO) is to model a metaheuristic similar to the behaviour of ants searching a optimal path connecting to different places [6, 7]. For the Car Sequencing Problem, a number of ants build an arrangement of cars—each ant on its own. Each time a car is following another car in the sequence, a pheromone trail is layed which indicates the quality of car one following car two. Multiple cycles are performed, whereby good sequences are indicated by trails with a big amount of pheromone.

Gravel et al. [7] and Gottlieb et al. [6] presented algorithms using ACO with different heuristics for deciding which car to consider next.

3.4 Exact methods

There are few approaches solving the Car Sequencing Problem using exact methods. Gravel et al. [7] proposed an Integer Linear Program (ILP) approach. This ILP is able to solve commonly used benchmark instances in acceptable time [7]. In addition, the optimality of the obtained solutions is proven. For test results comparing this ILP with the ILPs propsoed by myself see chapter 7.

Hu proposed an ILP approach in his diploma thesis [10]. His ILP also takes constraints defined by the paint shop into account, whereby the size of solveable instances is limited.

3.5 The paint shop only

Epping et al. [2, 3, 4] concentrated on reducing the colour changes only with respect to a given arrangement of the components. They showed that even this reduction of the problem is NP-complete [4]. Furthermore they proposed a dynamic programming approach with a memory and time complexity of $O(|F| * n^{|F| * |K|})$. Therefore this method is not applicable for large instances with either many colours, configurations or cars to be arranged. Another approach uses a variation of Multiple Sequence Alignment to receive an optimal colouring [2].

4 Integer Linear Programming Formulations

An Integer Linear Program (ILP) represents a mathematical formulation of an optimisation problem. It defines a linear objective function which has to be minimised or maximised subject to linear constraints. The decision variables have to be integer and the solution has to fulfill all constraints. Figure 4.1 gives the standard form of an ILP, with c as n -dimensional cost vector, x as n -dimensional vector of decision variables, A as coefficient matrix and b as vector of scalars.

$$\begin{array}{ll} \text{minimise} & c^T x \\ \text{subject to} & Ax \geq b \\ & x \in Z^n \end{array}$$

Figure 4.1: An ILP in standard form

In general, ILPs are solved using exact algorithms like "Branch and Bound" or "Branch and Cut" [14]. The result of these exact algorithms is an optimal solution. This means, that there exists no other solution with a better objective value. For the Car Sequencing Problem, an optimal solution is an arrangement along the production line that minimises the number of working bay violations and colour changes. If there are multiple optimal solutions, only one needs to be obtained. Although in general it is possible that no solution can be found, for the Car Sequencing Problem only instances with at least one integer solution are taken into account.

In [7], an attempt to solve the Car Sequencing Problem in an exact way is made using an ILP. Hu also proposed an ILP approach in [10]. Since these ILPs are outperformed by other approaches using heuristics, I decided to try to improve them.

Below I will discuss two resulting ILPs: the first was developed by myself, while the other one is an enhancement of the ILP proposed by Hu.

n	...	number of cars
$k \in K$...	configuration k
δ_k	...	demand for configuration $k \in K$
$c \in C$...	component c
$f \in F \subseteq C$...	colour f
s	...	paint batch limit
l_c/m_c	...	constraint for component $c \in C \setminus F$
$s/(s+1)$...	constraint for colour $f \in F$
$cost_c$...	costs for violations by component c
$cost_f$...	costs for colour change to colour $f \in F$

Table 4.1: Used symbols

4.1 Component ILP

Table 4.1 shows the symbols used in this ILP formulation called *Component ILP* (C-ILP). Each configuration $k \in K$ comprises of relevant components $c \in C$. The entries of matrix \mathbf{A} express these relations:

$$\mathbf{a}_{ck} = \begin{cases} 1 & \text{if configuration } k \text{ contains component } c \\ 0 & \text{otherwise} \end{cases} \quad \forall c \in C, \forall k \in K$$

For computation complexity, it is essential that

$$\exists c : \mathbf{a}_{ci} \neq \mathbf{a}_{cj} \quad \forall (i, j) \text{ with } i \neq j.$$

This constraint ensures, that all configurations are pairwise different. Therefore the search space is reduced.

The following expression computes the demand for each component $c \in C$:

$$\mathbf{d}_c = \sum_{k \in K} (\mathbf{a}_{ck} \cdot \delta_k) \quad \forall c \in C$$

For computing the correct number of violations, the production of day $N - 1$ has to be borne in mind. Therefore matrix $\mathbf{E} = \mathbf{e}_{ci}$ is defined as 1 if the car at position $n - i$ required component c . The number of columns of Matrix \mathbf{E} is $\max_{c \in C} \{m_c\} - 1$, because this is the maximum number of cars possibly having an impact on the number of violations of any constraint on day N .

$$\mathbf{e}_{ci} = \begin{cases} 1 & \text{if the car at position } n - i \text{ required } c \\ 0 & \text{otherwise} \end{cases} \quad \forall c \in C, \forall i \in \{0, \dots, \max_{c \in C} \{m_c\} - 1\}$$

4 Integer Linear Programming Formulations

All matrices and vectors defined so far are given constants for each individual instance. For solving the problem by moving each component until all requirements are met, I have to declare binary decision variables which state if the car at position i requires component c or not. Therefore matrix \mathbf{B} has n columns and $|C|$ rows. Possible values of \mathbf{b}_{ci} are 0 and 1.

$$\mathbf{b}_{ci} = \begin{cases} 1 & \text{car at position } i \text{ requires component } c \\ 0 & \text{otherwise} \end{cases} \quad \forall c \in C, \forall i \in \{0, \dots, n-1\}$$

Each car must be painted with exactly one colour. Therefore the sum of all colour-entries has to be 1 for each car.

$$\sum_{f \in F} \mathbf{b}_{fi} = 1 \quad \forall i \in \{0, \dots, n-1\}$$

Furthermore the number of components used must be equal to the number of components available.

$$\sum_{i=0}^{n-1} \mathbf{b}_{ci} = \mathbf{d}_c \quad \forall c \in C$$

At this time it is not guaranteed that the demand for configurations is satisfied. For this reason, some extra computations are required and so I introduce matrix \mathbf{P} with dimension $|K| \times n$. If configuration k is produced at position i , the corresponding entry \mathbf{p}_{ki} is 1 and 0 otherwise. The value of \mathbf{p}_{ki} can only be 1 if all entries for \mathbf{a}_{ck} are equal to \mathbf{b}_{ci} ($\forall c \in C$). If the entries for \mathbf{a}_{ck} and \mathbf{b}_{ci} are 1, the expression

$$\mathbf{a}_{ck} \cdot \mathbf{b}_{ci}$$

yields 1. In all other cases, this expression is equal to 0. Furthermore if both entries are equal to 0, the expression

$$(1 - \mathbf{a}_{ck}) \cdot (1 - \mathbf{b}_{ci})$$

yields 1. Again, all other cases result in 0. Therefore the sum

$$\mathbf{a}_{ck} \cdot \mathbf{b}_{ci} + (1 - \mathbf{a}_{ck}) \cdot (1 - \mathbf{b}_{ci})$$

is 1 only if both entries are equal. These observations lead to following inequations.

$$\begin{aligned} \mathbf{p}_{ki} &\leq \mathbf{a}_{ck} \cdot \mathbf{b}_{ci} + (1 - \mathbf{a}_{ck}) \cdot (1 - \mathbf{b}_{ci}) && \forall k \in K, \forall c \in C, \forall i \in \{0, \dots, n-1\} \\ \mathbf{p}_{ki} &\geq 0 && \forall k \in K, \forall i \in \{0, \dots, n-1\} \\ \sum_{i=0}^{n-1} \mathbf{p}_{ki} &= \delta_k && \forall k \in K \end{aligned}$$

4 Integer Linear Programming Formulations

Since only one configuration can be manufactured at each position along the production line, the sum over each column of matrix \mathbf{P} has to be 1.

$$\sum_{k \in K} \mathbf{p}_{ki} = 1 \quad \forall i \in \{0, \dots, n-1\}$$

A detailed analysis reveals that this constraint can be neglected as it can be derived from the constraints stated above. Since all configurations are pairwise different, at most one entry in matrix \mathbf{P} can be equal to 1. Furthermore the sum over all entries of matrix \mathbf{P} is equal to n . Therefore each column in entry \mathbf{P} sum up to 1. In addition following relationship exists:

$$\mathbf{b}_{ci} = \sum_{k \in K} \mathbf{a}_{ck} \cdot \mathbf{p}_{ki} \quad \forall i \in \{0, \dots, n-1\}, \forall c \in C$$

This is due the fact that each configuration k assigned to position i assigns the components $c \in C$ needed by configuration k to position i too.

At the moment there is no computation done for counting the violations of the working bay constraints or colour changes. For this purpose matrix \mathbf{Q} is introduced. Its entries indicate the quantity of cars requiring component c in a subsequence of length m_c minus the quota l_c for this component. Because of this definition, only integer values can be entered even though its entries need not to be integers.

$$\begin{aligned} \mathbf{q}_{c0} &= \mathbf{b}_{c0} + \sum_{j=0}^{m_c-2} (\mathbf{e}_{cj} - l_c) && \forall c \in C \\ \mathbf{q}_{ci} &= \begin{cases} \mathbf{q}_{c(i-1)} + \mathbf{b}_{ci} - \mathbf{e}_{c(m_c-i-1)} & \forall i \in \{1, \dots, m_c-1\} \\ \mathbf{q}_{c(i-1)} + \mathbf{b}_{ci} - \mathbf{b}_{c(i-m_c)} & \forall i \in \{m_c, \dots, n-1\} \end{cases} && \forall c \in C \end{aligned}$$

Note that the constraints defined by the paint shop can be expressed as $\frac{s}{s+1}$, since at most s cars might be painted with the same colour. If at most s cars are painted with the same colour in a subsequence of $s+1$ cars there has to be at least one colour change. Therefore no violation of the constraints defined by the paint shop can occur. Because of this equivalence of the constraints defined by the working bays and paint shop, it is very easy to add them to the ILP formulation. Since these constraints are hard constraints which means no violations are allowed, following inequations have to be added.

$$\mathbf{q}_{fi} \leq 0 \quad \forall f \in F, \forall i \in \{0, \dots, n-1\}$$

For computing the actual number of violations, only entries in matrix \mathbf{Q} with values greater than 0 are relevant. Only entries for components have to be considered, since no violations are allowed to occur for paint shop constraints. Thus matrix \mathbf{G} of dimension $|C \setminus F| \times n$ contains non negative (integer) values.

$$\mathbf{g}_{ci} \geq 0 \quad \forall i \in \{0, \dots, n-1\}, \forall c \in C \setminus F$$

4 Integer Linear Programming Formulations

$$\mathbf{g}_{ci} \geq \mathbf{q}_{ci} \quad \forall i \in \{0, \dots, n-1\}, \forall c \in C \setminus F$$

Since this formulation for counting the number of violations requires $2 \cdot |C| \cdot n$ variables, matrices \mathbf{Q} and \mathbf{G} should be reduced to just one matrix. This can be easily achieved by resolving the recursion in matrix \mathbf{Q} and expressing the sums explicitly. Tests revealed that neither of these two formulations is better than the other one using CPLEX for solving this ILP. Matrix \mathbf{G} is defined as

$$\begin{aligned} \mathbf{g}_{ci} &\geq 0 && \forall i \in \{0, \dots, n-1\}, \forall c \in C \setminus F \\ \mathbf{g}_{ci} &\geq \sum_{j=0}^i \mathbf{b}_{cj} + \sum_{j=0}^{m_c-i-2} \mathbf{e}_{cj} - l_c && \forall i \in \{0, \dots, m_c-2\}, \forall c \in C \setminus F \\ \mathbf{g}_{ci} &\geq \sum_{j=i-m_c+1}^i \mathbf{b}_{cj} - l_c && \forall i \in \{m_c-1, \dots, n-1\}, \forall c \in C \setminus F \end{aligned}$$

This formulation uses at most $|C| \cdot n$ variables, but now I have to rework the constraints for the paint shop.

$$\begin{aligned} \sum_{j=0}^i \mathbf{b}_{fj} + \sum_{j=0}^{s-i-1} \mathbf{e}_{fj} &\leq s && \forall i \in \{0, \dots, s-1\}, \forall f \in F \\ \sum_{j=i-s}^i \mathbf{b}_{fj} &\leq s && \forall i \in \{s, \dots, n-1\}, \forall f \in F \end{aligned}$$

Only colour changes are left to be counted. For this purpose, I introduce matrix \mathbf{W} with $|F| \times n$ entries. \mathbf{w}_{fi} is 1 if a change to colour f occurs at position i and 0 otherwise.

$$\begin{aligned} \mathbf{w}_{f0} &\geq 0 && \forall f \in F \\ \mathbf{w}_{f0} &\geq \mathbf{b}_{f0} - \mathbf{e}_{f0} && \forall f \in F \\ \mathbf{w}_{fi} &\geq 0 && \forall i \in \{1, \dots, n-1\}, \forall f \in F \\ \mathbf{w}_{fi} &\geq \mathbf{b}_{fi} - \mathbf{b}_{f(i-1)} && \forall i \in \{1, \dots, n-1\}, \forall f \in F \end{aligned}$$

Now it is possible to define the objective function which has to be minimised.

$$\sum_{f \in F} \left(\text{cost}_f \cdot \sum_{i=0}^{n-1} \mathbf{w}_{fi} \right) + \sum_{c \in C \setminus F} \left(\text{cost}_c \cdot \sum_{i=0}^{n-1} \mathbf{g}_{ci} \right)$$

4.1.1 ILP formulation

At this stage I summarize all considerations and present the resulting formulation:

objective function

$$\min \sum_{f \in F} \left(cost_f \cdot \sum_{i=0}^{n-1} \mathbf{w}_{fi} \right) + \sum_{c \in C \setminus F} \left(cost_c \cdot \sum_{i=0}^{n-1} \mathbf{g}_{ci} \right) \quad (4.1)$$

subject to

$$0 \leq \mathbf{b}_{ci} \leq 1 \quad \forall c \in C, \forall i \in \{0, \dots, n-1\} \quad (4.2)$$

$$\sum_{f \in F} \mathbf{b}_{fi} = 1 \quad \forall i \in \{0, \dots, n-1\} \quad (4.3)$$

$$\sum_{i=0}^{n-1} \mathbf{b}_{ci} = \mathbf{d}_c \quad \forall c \in C \quad (4.4)$$

$$0 \leq \mathbf{p}_{ki} \leq 1 \quad \forall k \in K, \forall i \in \{0, \dots, n-1\} \quad (4.5)$$

$$\mathbf{p}_{ki} \leq \mathbf{a}_{ck} \cdot \mathbf{b}_{ci} + (1 - \mathbf{a}_{ck}) \cdot (1 - \mathbf{b}_{ci}) \quad \forall k \in K, \forall c \in C, \forall i \in \{0, \dots, n-1\} \quad (4.6)$$

$$\sum_{i=0}^{n-1} \mathbf{p}_{ki} = \delta_k \quad \forall k \in K \quad (4.7)$$

$$\mathbf{b}_{ci} = \sum_{k \in K} \mathbf{a}_{ck} \cdot \mathbf{p}_{ki} \quad \forall i \in \{0, \dots, n-1\}, \forall c \in C \quad (4.8)$$

$$0 \leq \mathbf{g}_{ci} \quad \forall i \in \{0, \dots, n-1\}, \forall c \in C \setminus F \quad (4.9)$$

$$\mathbf{g}_{ci} \geq \sum_{j=0}^i \mathbf{b}_{cj} + \sum_{j=0}^{m_c-i-2} \mathbf{e}_{cj} - l_c \quad \forall i \in \{0, \dots, m_c-2\}, \forall c \in C \setminus F \quad (4.10)$$

$$\mathbf{g}_{ci} \geq \sum_{j=i-m_c+1}^i \mathbf{b}_{cj} - l_c \quad \forall i \in \{m_c-1, \dots, n-1\}, \forall c \in C \setminus F \quad (4.11)$$

$$\sum_{j=0}^i \mathbf{b}_{fj} + \sum_{j=0}^{s-1-i} \mathbf{e}_{fj} \leq s \quad \forall i \in \{0, \dots, s-1\}, \forall f \in F \quad (4.12)$$

$$\sum_{j=i-s}^i \mathbf{b}_{fj} \leq s \quad \forall i \in \{s, \dots, n-1\}, \forall f \in F \quad (4.13)$$

$$0 \leq \mathbf{w}_{fi} \leq 1 \quad \forall i \in \{0, \dots, n-1\}, \forall f \in F \quad (4.14)$$

$$\mathbf{w}_{f0} \geq \mathbf{b}_{f0} - \mathbf{e}_{f0} \quad \forall f \in F \quad (4.15)$$

$$\mathbf{w}_{fi} \geq \mathbf{b}_{fi} - \mathbf{b}_{f(i-1)} \quad \forall i \in \{1, \dots, n-1\}, \forall f \in F \quad (4.16)$$

$$\mathbf{b}_{ci}, \mathbf{p}_{ki} \text{ integer} \quad \forall c \in C, \forall k \in K, \forall i \in \{0, \dots, n-1\} \quad (4.17)$$




				constraint	cost
comp #1	1	0	1	1/3	1
comp #2	0	1	1	2/3	10
colour #1	1	0	0	4/5	100
colour #2	0	1	1	4/5	100
demand	3	3	2	—	—

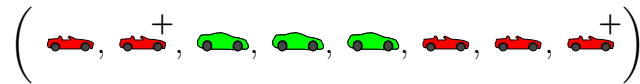
Table 4.2: A test instance

4.1.2 Example for the formulation

I will present a simple example for this formulation. Table 4.2 shows the values defined by a test instance. There are three different configurations – each consisting of two possible components and one of two colours. This specification leads to matrices \mathbf{A} and \mathbf{D} . Additionally matrix \mathbf{E} is presented. Because $\max_{c \in C} \{m_c\} = 5$, only the last $5 - 1 = 4$ cars produced the day before are needed for computing potential violations.

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix} \quad \mathbf{D} = \begin{pmatrix} 5 \\ 5 \\ 3 \\ 5 \end{pmatrix} \quad \mathbf{E} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

Let's examine the following sequence along the production line:



The resulting matrices are

$$\mathbf{B} = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$$

$$\mathbf{Q} = \begin{pmatrix} 0 & 0 & 1 & 2 & 2 & 1 & 0 & 0 \\ 1 & 1 & 0 & -1 & -2 & -1 & 0 & 1 \\ -3 & -3 & -3 & -2 & -1 & -1 & -1 & -2 \\ -1 & 0 & 0 & -1 & -2 & -2 & -2 & -1 \end{pmatrix}$$

$$\mathbf{G} = \begin{pmatrix} 0 & 0 & 1 & 2 & 2 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{H} = (0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0)$$

Matrix \mathbf{G} and vector \mathbf{H} are relevant for evaluation. Since the sum over row one in matrix \mathbf{G} is 6 and the second row sums up to 3, the costs for violating the constraints defined by the production lines are $(6 \cdot 1) + (3 \cdot 10) = 36$. Vector \mathbf{H} sums up to 2 which means that there are two colour changes. Because one colour change costs 100, the objective function yields 236 in total.

4.1.3 Correctness of the formulation

I have to show that this ILP formulation will allways produce an optimal solution to the Car Sequencing Problem. A solution is valid if there is exactly one car at each possible position, which means that exactly one configuration is assigned to every position. Furthermore no violations are tolerated for constraints defined by the paint shop.

Constraint (4.6) ensures that at most one configuration is assigned to each position. Because of constraint (4.7), $\sum_{k \in K} \sum_{i=0}^{n-1} \mathbf{p}_{ki}$ is equal to n . Therefore each position has exactly one configuration assigned which also means that each position has exactly one colour assigned, since each configuration consists exactly one colour.

Constraints (4.12) and (4.13) make sure that no violations of paint shop constraints occur. If there are more than s consecutive cars painted with the same colour, the left sides of these inequations would be greater than s . Thus no violations for constraints defined by the paint shop can occur.

So far I have shown that each solution found by the ILP system is valid. Now optimality has to be proven. Constraints (4.10) and (4.11) in combination with the objective function (4.1) guarantee that each violation is counted. Since the entries of matrix \mathbf{G} appear with positive sign in the objective function, $\sum_{c \in C \setminus F} \sum_{i=0}^{n-1} \mathbf{g}_{ci}$ is minimised. Therefore only occurring violations are counted. In addition all colour changes are taken into account by constraints (4.14) to (4.16). Because the objective function (4.1) is minimised, the ILP formulation must lead to an optimum.

4.2 Configuration ILP

Hu [10] proposed an ILP for solving the Car Sequencing Problem in his diploma thesis (see appendix 8.2.6 for a reprint of his formulation). In praxis his approach can solve instances with up to 20 cars in acceptable time. I tried to strengthen his formulation. In this section I am going to present the results of this effort and show the enhancements. Later on I will show that the changes did not affect the correctness of his formulation. I refer to this formulation as *K-ILP*.

4 Integer Linear Programming Formulations

I use the same symbols as in the preceding formulation (see table 4.1). Furthermore matrices \mathbf{A} , \mathbf{B} and the vector \mathbf{D} are defined in the same way as above.

Exactly one configuration has to be placed at each position along the production line. Hence matrix \mathbf{P} is defined. Its entries \mathbf{p}_{ki} are 1 if configuration k is assigned to position i . Otherwise 0 is entered.

$$\mathbf{p}_{ki} = \begin{cases} 1 & \text{configuration } k \text{ is assigned to position } i \\ 0 & \text{otherwise} \end{cases} \quad \forall k \in K, \forall i \in \{0, \dots, n-1\}$$

In contrast to the original formulation assigning cars to each position, this reformulation of the decision variables forms the major improvement, because cars with same configurations are handled as if they are equal. Normally this reduces the number of variables significantly which also means that the search space is reduced.

To ensure that all demanded cars are produced and exactly one car is allocated for each position following constraints have to be declared.

$$\begin{aligned} \sum_{k \in K} \mathbf{p}_{ki} &= 1 & \forall i \in \{0, \dots, n-1\} \\ \sum_{i=0}^{n-1} \mathbf{p}_{ki} &= \delta_k & \forall k \in K \end{aligned}$$

At any time it has to be guaranteed that the sum of all used components equals the number of available components.

$$\sum_{i=0}^{n-1} \sum_{k \in K} \mathbf{a}_{ck} \cdot \mathbf{p}_{ki} = \mathbf{d}_c \quad \forall c \in C$$

For computing the number of violations, matrix \mathbf{R} is introduced. \mathbf{r}_{ci} specifies the quantity of component c used so far till position i . $\mathbf{r}_{cj} - \mathbf{r}_{c(j-m_c)}$ counts the number of components used in the last subsequence of length m_c . Although it is not required that the entries are integers, this is necessary by the definition of \mathbf{R} .

$$\begin{aligned} \mathbf{r}_{c0} &= 0 & \forall c \in C \setminus F \\ \mathbf{r}_{ci} &= \mathbf{r}_{c(i-1)} + \sum_{k \in K} (\mathbf{a}_{ck} \cdot \mathbf{p}_{k(i-1)}) & \forall i \in \{1, \dots, n\}, \forall c \in C \setminus F \end{aligned}$$

The actual number of violations is stored in the entries of matrix \mathbf{G} . Again, only non negative (integer) values are permitted.

$$\mathbf{g}_{ci} \geq \mathbf{r}_{c(i+1)} - l_c + \sum_{j=0}^{m_c-2-i} \mathbf{e}_{cj} \quad \forall i \in \{0, \dots, m_c-2\}, \forall c \in C \setminus F$$

4 Integer Linear Programming Formulations

$$\mathbf{g}_{ci} \geq \mathbf{r}_{c(i+1)} - \mathbf{r}_{c(i+1-m_c)} - l_c \quad \forall i \in \{m_c - 1, \dots, n - 1\}, \forall c \in C \setminus F$$

Only the number of colour changes is missing. Therefore matrix \mathbf{B} with dimension $|F| \times n$ has to be declared. \mathbf{b}_{fi} is equal to 1 if the car at position i is painted with colour f and 0 in all other cases.

$$\mathbf{b}_{fi} = \sum_{k \in K} \mathbf{a}_{fk} \cdot \mathbf{p}_{ki} \quad \forall f \in F, \forall i \in \{0, \dots, n - 1\}$$

Since only up to s consecutive cars are allowed to be painted with the same colour, two additional constraints have to be added.

$$\begin{aligned} \sum_{j=0}^{s-i-1} \mathbf{e}_{fj} + \sum_{j=0}^i \mathbf{b}_{fj} &\leq s & \forall i \in \{0, \dots, s - 1\}, \forall f \in F \\ \sum_{j=i-s}^i \mathbf{b}_{fj} &\leq s & \forall i \in \{s, \dots, n - 1\}, \forall f \in F \end{aligned}$$

In addition each car must be painted with exactly one colour, but this is already guaranteed by constraints defined so far, because exactly one configuration is assigned to each car. Since each configuration contains only one colour, no car can be painted with two or more colours. If a constraint expressing that each car is painted with exactly one colour is added to the formulation, experiments showed that the runtime increases using CPLEX for solving the ILP.

Finally, matrix \mathbf{W} is introduced. If a change to colour f occurs at position i , the corresponding entry \mathbf{w}_{fi} is set to 1 and 0 otherwise. For expressing this in an (I)LP relevant form, following constraints are necessary.

$$\begin{aligned} \mathbf{w}_{fi} &\geq 0 & \forall i \in \{0, \dots, n - 1\}, \forall f \in F \\ \mathbf{w}_{f0} &\geq \mathbf{b}_{f0} - \mathbf{e}_{f0} & \forall f \in F \\ \mathbf{w}_{fi} &\geq \mathbf{b}_{fi} - \mathbf{b}_{fi-1} & \forall i \in \{0, \dots, n - 1\}, \forall f \in F \end{aligned}$$

The objective function is composed of the number of constraint violations for components and the number of colour changes. It has to be minimised.

$$\sum_{c \in C \setminus F} \left(cost_c \cdot \sum_{i=0}^{n-1} \mathbf{g}_{ci} \right) + \sum_{f \in F} \left(cost_f \cdot \sum_{i=0}^{n-1} \mathbf{w}_{fi} \right)$$

4.2.1 ILP formulation

In this section the resulting ILP formulation is presented.

objective function

$$\min \sum_{c \in C \setminus F} \left(cost_c \cdot \sum_{i=0}^{n-1} \mathbf{g}_{ci} \right) + \sum_{f \in F} \left(cost_f \cdot \sum_{i=0}^{n-1} \mathbf{w}_{fi} \right) \quad (4.18)$$

subject to

$$0 \leq \mathbf{p}_{ki} \leq 1 \quad \forall i \in \{0, \dots, n-1\}, \forall k \in K \quad (4.19)$$

$$\sum_{k \in K} \mathbf{p}_{ki} = 1 \quad \forall i \in \{0, \dots, n-1\} \quad (4.20)$$

$$\sum_{i=0}^{n-1} \mathbf{p}_{ki} = \delta_k \quad \forall k \in K \quad (4.21)$$

$$\sum_{i=0}^{n-1} \sum_{k \in K} \mathbf{a}_{ck} \cdot \mathbf{p}_{ki} = \mathbf{d}_c \quad \forall c \in C \quad (4.22)$$

$$\mathbf{r}_{c0} = 0 \quad \forall c \in C \setminus F \quad (4.23)$$

$$0 \leq \mathbf{r}_{ci} \quad \forall c \in C, \forall i \in \{1, \dots, n\} \quad (4.24)$$

$$\mathbf{r}_{ci} = \mathbf{r}_{c(i-1)} + \sum_{k \in K} (\mathbf{a}_{ck} \cdot \mathbf{p}_{k(i-1)}) \quad \forall c \in C, \forall i \in \{1, \dots, n\} \quad (4.25)$$

$$0 \leq \mathbf{g}_{ci} \quad \forall c \in C, \forall i \in \{0, \dots, n\} \quad (4.26)$$

$$\mathbf{g}_{ci} \geq \mathbf{r}_{c(i+1)} - l_c + \sum_{j=0}^{m_c-2-i} \mathbf{e}_{cj} \quad \forall i \in \{0, \dots, m_c-2\}, \forall c \in C \setminus F \quad (4.27)$$

$$\mathbf{g}_{ci} \geq \mathbf{r}_{c(i+1)} - \mathbf{r}_{c(i+1-m_c)} - l_c \quad \forall i \in \{m_c-1, \dots, n-1\}, \forall c \in C \setminus F \quad (4.28)$$

$$0 \leq \mathbf{b}_{fi} \leq 1 \quad \forall f \in F, \forall i \in \{0, \dots, n\} \quad (4.29)$$

$$\mathbf{b}_{fi} = \sum_{k \in K} \mathbf{a}_{fk} \cdot \mathbf{p}_{ki} \quad \forall f \in F, \forall i \in \{0, \dots, n-1\} \quad (4.30)$$

$$\sum_{j=0}^{s-i-1} \mathbf{e}_{fj} + \sum_{j=0}^i \mathbf{b}_{fj} \leq s \quad \forall i \in \{0, \dots, s-1\}, \forall f \in F \quad (4.31)$$

$$\sum_{j=i-s}^i \mathbf{b}_{fj} \leq s \quad \forall i \in \{s, \dots, n-1\}, \forall f \in F \quad (4.32)$$

$$0 \leq \mathbf{w}_{fi} \leq 1 \quad \forall f \in F, \forall i \in \{0, \dots, n-1\} \quad (4.33)$$

$$\mathbf{w}_{f0} \geq \mathbf{b}_{f0} - \mathbf{e}_{f0} \quad \forall f \in F \quad (4.34)$$

$$\mathbf{w}_{fi} \geq \mathbf{b}_{fi} - \mathbf{b}_{f(i-1)} \quad \forall f \in F, \forall i \in \{0, \dots, n-1\} \quad (4.35)$$

$$\mathbf{p}_{ki} \text{ integer} \quad \forall k \in K, \forall i \in \{0, \dots, n-1\} \quad (4.36)$$

4.2.2 Example for the formulation

The example of section 4.1.2 is used to demonstrate this formulation. Please recall table 4.2 on page 18 for the example instance. For simplicity, we consider the same sequence as in section 4.1.2

$$\left(\text{red}, \text{red}^+, \text{green}, \text{green}, \text{green}, \text{red}, \text{red}, \text{red}^+ \right).$$

This leads to the following matrices:

$$\begin{aligned} \mathbf{P} &= \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \\ \mathbf{R} &= \begin{pmatrix} 0 & 0 & 1 & 2 & 3 & 4 & 4 & 4 & 5 \\ 0 & 1 & 2 & 2 & 2 & 2 & 3 & 4 & 5 \end{pmatrix} \\ \mathbf{G} &= \begin{pmatrix} 0 & 0 & 1 & 2 & 2 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \\ \mathbf{B} &= \begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \\ \mathbf{W} &= \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{aligned}$$

The objective function yields 236 again.

4.2.3 Correctness of the formulation

Here I will show that this ILP formulation leads to a valid and optimal solution each time. Again I have to ensure that the resulting schedule of cars describes a permutation. Constraint (4.20) makes sure that exactly one configuration is assigned to each car. Because $\sum_{k \in K} \delta_k$ is equal to n , constraint (4.21) assures that each demanded car is produced. Moreover each car along the assembly line has to be painted with exactly one colour. This is guaranteed by constraint (4.20), because each configuration consists only one colour. Therefore the final schedule has to be a valid arrangement.

The argumentation for optimality is the same as presented in section 4.1.3. Since the colour changes are counted in the same way, the arguments can be adopted. Furthermore the entries of matrix \mathbf{G} are minimised again. This leads to the conclusion that only necessary violations are counted. Since matrix \mathbf{R} defines the number of components used up to the current position, the difference $\mathbf{r}_{c(i+1)} - \mathbf{r}_{c(i+1-m_c)}$ is the number of used components during the production of the last m_c cars. Therefore \mathbf{g}_{c_i} shows the number of supernumerous fittings.

4.3 Equivalence of the formulations

Below I will point out that both introduced formulations are equivalent which means that the polyhedra defined by the formulations are equivalent. A polyhedron is the set of all feasible solutions to the LP-relaxation of the ILP formulation. Therefore if both formulations define the same set of solutions, they have to be equivalent.

The objective functions (4.1) and (4.18), respectively, are defined in the same way. It is easy to see that the constraints (4.14)–(4.16) are equal to constraints (4.33)–(4.35). The same can be said about constraints (4.31)–(4.32) and (4.12)–(4.13). Since constraint (4.8)—which is identical to (4.30) ($\forall f \in F$)—states that $\mathbf{b}_{ci} = \sum_{k \in K} \mathbf{a}_{ck} \cdot \mathbf{p}_{ki}$, (4.22) and (4.4) are equal too. (4.23)–(4.28) are equal to (4.9)–(4.11), because $\mathbf{r}_{c(i+1)} - \mathbf{r}_{c(i+1-m_c)} = \sum_{j=i-m_c+1}^i \mathbf{b}_{cj}$. (4.21) is equal to (4.7).

Now, I have to show that constraint (4.20) can be expressed with the help of (4.3) and (4.8). Please recall, that each configuration k contains exactly one colour:

$$\sum_{f \in F} \mathbf{a}_{fk} = 1 \quad \forall k \in K \quad (4.37)$$

This leads to following equations:

$$\begin{aligned} & \sum_{i=0}^{n-1} \mathbf{p}_{ki} = \\ & \stackrel{(4.37)}{=} \sum_{i=0}^{n-1} \sum_{f \in F} \mathbf{a}_{ck} \cdot \mathbf{p}_{ki} = \\ & = \sum_{f \in F} \sum_{i=0}^{n-1} \mathbf{a}_{ck} \cdot \mathbf{p}_{ki} = \\ & \stackrel{(4.8)}{=} \sum_{f \in F} \mathbf{b}_{fi} \stackrel{(4.3)}{=} 1 \quad \forall k \in K \quad (4.38) \end{aligned}$$

Constraint (4.3) can be expressed with the help of (4.20), (4.30) and (4.37):

$$\begin{aligned} & \sum_{f \in F} \mathbf{b}_{fi} = \\ & \stackrel{(4.30)}{=} \sum_{f \in F} \sum_{k \in K} \mathbf{a}_{fk} \cdot \mathbf{p}_{ki} = \\ & = \sum_{k \in K} \sum_{f \in F} \mathbf{a}_{fk} \cdot \mathbf{p}_{ki} = \\ & \stackrel{(4.37)}{=} \sum_{k \in K} \mathbf{p}_{ki} \stackrel{(4.20)}{=} 1 \quad \forall i \in \{0, \dots, n-1\} \quad (4.39) \end{aligned}$$

Finally, I show that the constraint (4.6) is true. This constraint can also be written as

$$\mathbf{p}_{ki} \leq \mathbf{a}_{ck} \cdot \left(\sum_{k' \in K} \mathbf{a}_{ck'} \cdot \mathbf{p}_{k'i} \right) + (1 - \mathbf{a}_{ck}) \cdot \left(1 - \left(\sum_{k' \in K} \mathbf{a}_{ck'} \cdot \mathbf{p}_{k'i} \right) \right) \quad \forall k \in K, \forall c \in C, \forall i \in \{0, \dots, n-1\} \quad (4.40)$$

using equation (4.8). Since \mathbf{a}_{ck} can either be 0 or 1, I differ between these two cases. If $\mathbf{a}_{ck} = 1$:

$$\begin{aligned} \mathbf{p}_{ki} &\leq \sum_{k' \in K} \mathbf{a}_{ck'} \cdot \mathbf{p}_{k'i} = \\ &\mathbf{p}_{ki} + \sum_{k' \in K \setminus \{k\}} \mathbf{a}_{ck'} \cdot \mathbf{p}_{k'i} \quad \forall k \in K \end{aligned} \quad (4.41)$$

Since

$$\sum_{k' \in K \setminus \{k\}} \mathbf{a}_{ck'} \cdot \mathbf{p}_{k'i} \geq 0$$

inequation (4.41) is true. If $\mathbf{a}_{ck} = 0$:

$$\begin{aligned} \mathbf{p}_{ki} &\leq 1 - \sum_{k' \in K} \mathbf{a}_{ck'} \cdot \mathbf{p}_{k'i} = \\ &= 1 - \underbrace{\mathbf{p}_{ki} \cdot \mathbf{a}_{ck}}_0 - \sum_{k' \in K \setminus \{k\}} \mathbf{a}_{ck'} \cdot \mathbf{p}_{k'i} \quad \forall k \in K \end{aligned} \quad (4.42)$$

Since

$$\sum_{k' \in K \setminus \{k\}} \mathbf{a}_{ck'} \cdot \mathbf{p}_{k'i} \leq \sum_{k' \in K \setminus \{k\}} \mathbf{p}_{k'i} = 1 - \mathbf{p}_{ki} \quad \forall k \in K$$

inequation (4.42) leads to

$$\mathbf{p}_{ki} \leq 1 - (1 - \mathbf{p}_{ki}) = \mathbf{p}_{ki} \quad \forall k \in K \quad (4.43)$$

Therefore the two ILP formulations are equivalent.

4.4 Implementation and runtime

Although both formulations are equivalent in mathematical sense, there are significant differences in runtime due to different implementation. I implemented both approaches using CPLEX by ILOG, Inc. CPLEX uses sophisticated "branch and cut"-algorithms for finding feasible solutions [14]. It fixes variables during each iteration. The runtime is strongly influenced by the chosen fixed variables.

Furthermore CPLEX generates additional cuts deduced from the given constraints. Sometimes CPLEX is able to generate better cuts because a redundant information

4 Integer Linear Programming Formulations

is provided to the solver. Setting variables \mathbf{p}_{ki} from the first formulation, for example, to be integer (although this is not necessary) increases the number of iterations done by CPLEX, but reduces the total runtime for finding an optimum. On the other hand, it is possible that the number of iterations increases significantly, if some additional redundant information is provided to CPLEX.

Tests show that the formulation given in section 4.2.1 is faster for solving instances with constraints defined by the paint shop than the one in section 4.1.1. Although instances like the ones defined by CSPLIB without any colours can be solved generally with more ease, the new ILP formulation seems to be better for these instances. Detailed test results, also in comparison with other methods, are shown in chapter 7.

5 Variable Neighbourhood Search (VNS)

Contrary to exact algorithms which always return an optimal solution, heuristics generate solutions which are good, but not necessarily optimal. In return, the average runtime decreases significantly. Metaheuristics are top level heuristics which guide other heuristics. In addition they are independent of the problem descriptions. In [9] a detailed introduction to *Variable Neighbourhood Search* (VNS) can be found. VNS is a metaheuristic which tries to combine the effort to find (local) optima in respect to several neighbourhoods with the attempt to escape the valleys which contain them. To achieve this, two complementary concepts are joined: *Local Search* and *Mutation* (or *Shaking*).

5.1 Local search

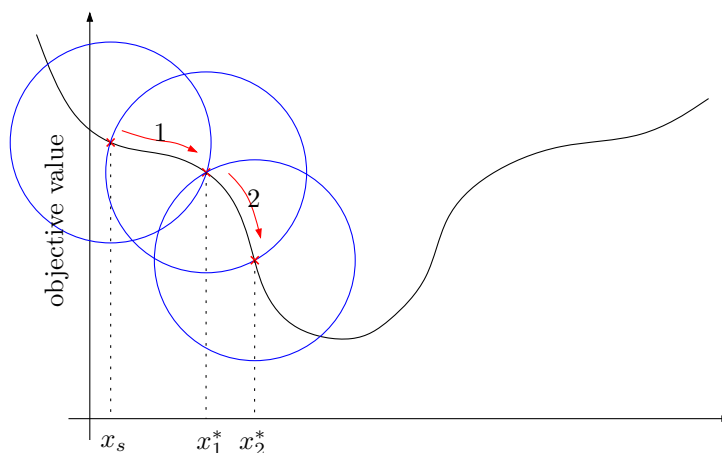


Figure 5.1: Basic local search strategy.

A local search consists of two major steps. First an initial solution x is generated. Secondly, a defined neighbourhood $\mathcal{N}(x)$ is searched. This neighbourhood contains all solutions x' which are similar to x in respect to some criteria. For example, if x denotes a vector of 0-1 variables, x' may denote a vector where at most two variables are complementary. Within $\mathcal{N}(x)$, the goal is to obtain the solution x^* with $f(x^*) \leq$

$f(x')$, $\forall x' \in \mathcal{N}(x)$, if $f(x)$ denotes the objective function for this problem, which has to be minimised. This is called the *best improvement* strategy. This procedure is repeated until no better solutions can be found. As we can see in figure 5.1, x_1^* is found after the neighbourhood of the initial solution x_s was searched.

This best improvement approach might be very time consuming particularly if the neighbourhood $\mathcal{N}(x)$ is large. To speed up this method, one can apply a move, i.e. using x' as new initial solution, as soon as a $x' \in \mathcal{N}(x)$ with $f(x') < f(x)$ was found. Again this procedure is carried out until no further improvements can be achieved. This method is also referred to as *first improvement*.

Although this method looks less time consuming, it is not "better" in general than the best improvement strategy. The local optima which are reached with these two methods might nevertheless differ from each other. Figure 5.2 shows an example where the next improvement strategy leads to solution x_n^* which lies in the valley containing the global optimum, whereas the best improvement strategy leads to solution x_b^* . This solution lies in the valley containing a local minimum different to the global one.

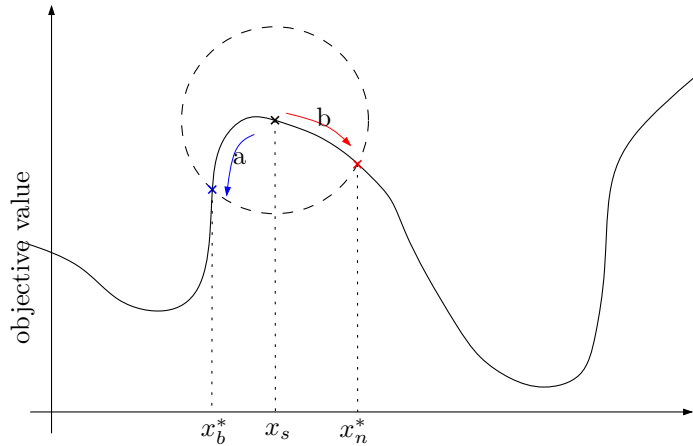


Figure 5.2: Different optima are found using the same initial solution x_s and best improvement (a) or next improvement (b) strategy.

It might be useful to examine more than just one neighbourhood to obtain better results which leads to *Variable Neighbourhood Descent* (VND)—an approach which scans different, perhaps partly overlapping neighbourhoods consecutively. If no improvement can be achieved in the first neighbourhood $\mathcal{N}_1(x)$, the second neighbourhood $\mathcal{N}_2(x)$ is browsed. If an improvement x' in $\mathcal{N}_2(x)$ is found, the search restarts with $\mathcal{N}_1(x')$ again. This procedure is repeated until no improvements can be found in any neighbourhood $\mathcal{N}_i(\tilde{x})$ of the currently best solution \tilde{x} . If $\mathcal{N}_i(x) \subset \mathcal{N}_j(x)$, there is no reason to iterate through $\mathcal{N}_i(x)$ after $\mathcal{N}_j(x)$ was searched. Algorithm 5.1 describes VND in pseudocode.

Algorithm 5.1: VND(x)

Input: x denotes the initial solution**Initialisation:** define neighbourhoods \mathcal{N}_t , for $t = 1, \dots, t_{max}$ **Output:** finds a local optimum in respect to all available neighbourhoods \mathcal{N}_t , for $t = 1, \dots, t_{max}$

```

1 repeat
2   |  $t \leftarrow 1$ 
3   | repeat
4   |   | find  $x^*$  with  $f(x^*) < f(x')$ ,  $\forall x' \in \mathcal{N}_t(x)$ 
5   |   | if  $f(x^*) < f(x)$  then
6   |   |   |  $x \leftarrow x^*$ 
7   |   |   |  $t \leftarrow 1$ 
8   |   |   | else
9   |   |   |   |  $t \leftarrow t + 1$ 
10  |   | until  $t = t_{max}$ 
11 until no improvement is achieved
12 return  $x$ 

```

5.2 Shaking

The major disadvantage of Local Search algorithms is that a valley containing a local optimum which is not equal to the global optimum can never be escaped from, especially if no worse solutions are tolerated during the search. The usage of multiple neighbourhoods as in VNS generally reduces this disadvantage but cannot solve it entirely. To avoid getting stuck on such a local optimum, some new techniques have to be introduced.

In contrast to multistart strategies, where Local Search restarts with completely new initial solutions several times, Shaking changes only a few variables randomly. This leads to slightly different starting points which might yield new local optima when Local Search is applied again. Exemplary code is shown in algorithm 5.2.

Algorithm 5.2: Shaking(u, x)

Input: u has to be smaller than the total number of variables, x denotes the current best solution**Output:** a slightly mutated solution

```

1 for  $i \leftarrow 1$  to  $u$  do
2   | change a—yet unchanged—variable  $x_i$  randomly
3 return  $x$ 

```

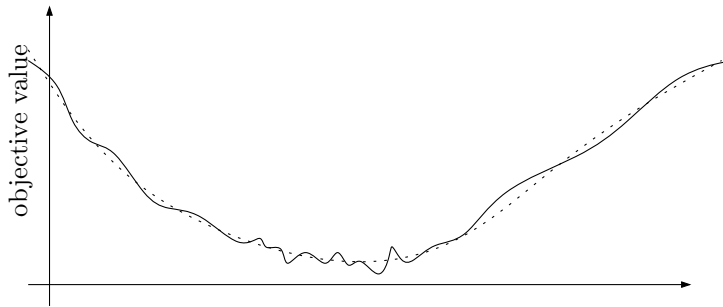


Figure 5.3: An exemplary objective function (solid line) with the general tendency (dotted line).

This approach is often better than multistart strategies, because often good local optima are related to each other, which means that at least some of the variables are set to the same values for two (local) optima [9]. For example, it can be said that certain subsequences of cars are better than other sequences because in general they lead to lower values of the objective function. Therefore it is better to retain good subsequences and replace only cars not included in good sequences. Figure 5.3 shows an exemplary objective function (solid line). The dotted line marks the general tendency of the objective function. It is clear that in this exemplary case this local rearranging is more efficient than multistart.

5.3 General VNS

To take advantage of VND and Shaking, the *General Variable Neighbourhood Search* scheme has been introduced. This strategy starts with an initial solution x and improves it by using VND and Shaking. First VND is used to find a local optimum x^* for all given neighbourhoods $\mathcal{N}_i(x)$. If no better solution can be achieved by VND, a mutation move is applied to the current best solution \tilde{x} which leads to a new starting solution for VND. If the new computed local optimum x^* is better than \tilde{x} , x^* is used as new best solution \tilde{x} . This procedure is repeated until x^* obtained by VND is as good as \tilde{x} (or worse).

If no better solution can be obtained, the number of variables affected by a mutation move is increased. After this the procedure starts again using the original number of variables for mutation moves. See algorithm 5.3 for a pseudocode of VNS.

VNS stops if some conditions are met, e.g. a given amount of computation time has been reached, or the number of iterations exceed a certain threshold. Other more sophisticated stopping conditions can be thought of like the number of iterations since the last improvement, etc.

Algorithm 5.3: VNS()

Initialisation: define u_{max} , the maximum number of variables possibly shaken; define neighbourhoods \mathcal{N}_t , for $t = 1, \dots, t_{max}$ used by VND; generate initial solution x

Output: an arrangement along the production line

```

1 repeat
2   |  $u \leftarrow 1$ 
3   | repeat
4   |   |  $x \leftarrow \text{Shaking}(u, x)$ 
5   |   |  $x' \leftarrow \text{VND}(x)$ 
6   |   | if  $f(x') < f(x)$  then
7   |   |   |  $x \leftarrow x'$ 
8   |   |   |  $u \leftarrow 1$ 
9   |   | else
10  |   |   |  $u \leftarrow u + 1$ 
11  |   until  $u = u_{max}$ 
12 until stopping conditions are met

```

6 Combination of exact methods and metaheuristics

Since exact methods are not applicable for large instances of the Car Sequencing Problem, i.e. instances with a large number of cars to be arranged, all methods with acceptable runtime are heuristics [6, 7, 11, 15, 16, 17].

Additional to the improvement of exact methods, I implemented two algorithms based on a combination of metaheuristics and exact methods. Commonly known or newly developed metaheuristics are used to guide exact algorithms. This hybridisation combines the advantages of both approaches. The runtime to a feasible solution is reduced and at least local optima can definitely be reached.

6.1 Combination VNS - ILP

In chapter 5 the basic principle of Variable Neighbourhood Search (VNS) was presented. To apply VNS to the Car Sequencing Problem, I have to provide the Shaking algorithm and I must define the different neighbourhoods \mathcal{N}_t examined by VND. Some of the neighbourhoods use ILP approaches. This means that each time a neighbourhood is examined, a solution x^* with $f(x^*) \leq f(x')$, $\forall x' \in \mathcal{N}(x)$ is definitely found (if enough runtime was provided).

6.1.1 CarSPShaking

The Shaking algorithm used for my thesis is a simple one. VNS defines u_{max} as upper bound for the number of variables to be shaken. CarSPShaking performs up to u_{max} random swap moves. For my thesis u_{max} is set to $3/4 \cdot n$. Since this move only has to ensure that valleys containing local minima can be escaped, no complex computations need to be done. Speed is much more important, because this random move can be applied many times. Algorithm 6.1 shows this simple shake move.

Algorithm 6.1: CarSPShaking(u, x)

Input: u has to be smaller than the total number of variables, x denotes the current best solution**Output:** a slightly mutated solution

```

1 for  $i \leftarrow 1$  to  $u$  do
2   | select two positions  $j, k$  randomly
3   | swap cars at positions  $j, k$ 
4 return  $x$ 

```

6.1.2 Neighbourhoods

My algorithm iterates through six different neighbourhoods: *Swapping*, *Inserting*, two variations of *Selection by Random* and two variations of *Selection by Costs*. All but the first two neighbourhoods are examined using the ILP approach described in section 4.2. Although four of them are defined similarly, they cover different subsets of all possible (integer) solutions. The neighbourhoods Swapping and Inserting utilise moves defined by Gottlieb et al. in [6].

Random moves like Shaking might violate the hard constraints defined by the paint shop. Additional to the violations of constraints defined by the working bays and colour changes, the violations of hard constraints are counted when evaluating the objective function. Each violation of hard constraints is penalised with $n * cost_f$ added to the objective. Therefore it is better to change the colour at each position than violating a hard constraint once.

x	... sequence of cars of length ≥ 1
x_i	... car at position i
π_i	... subsequence of x of length ≥ 0
$\langle x_i \rangle$... subsequence of x of length = 1
$cost(i)$... costs of violations at position i
\cdot	... concatenates two subsequences

Table 6.1: Symbols used for definition of the neighbourhoods

In the subsections specifying the different neighbourhoods, I will present a formal description for each of them. Table 6.1 shows the used symbols. $\pi_i \cdot \pi_j$ denotes the subsequence of cars in π_i followed by the subsequence π_j .

6.1.2.1 Swapping

The neighbourhood \mathcal{N}_S consists of all solutions generated by swapping two cars (see figure 6.1). There exist $\mathcal{O}(\frac{n^2-n}{2})$ different neighbours within this neighbourhood. Although

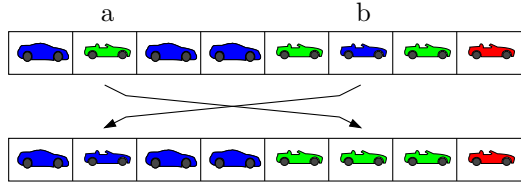


Figure 6.1: The cars at positions a and b are swapped. All other cars stay at their position.

the move itself is very easy, the evaluation of the objective function is a bit tricky. To reduce computation time, the objective function is recalculated for positions affected by the move only. In addition to the positions swapped, these are all positions in the interval $[i, i + m_c - 1]$ and $[j, j + m_c - 1]$, $\forall c \in C$. If these two intervals are overlapping, only positions i to $j - 1$ and $i + m_c$ to $j + m_c - 1$ have to be reevaluated. The evaluation time for this move is in $\mathcal{O}(2 \cdot m_c)$.

Formal \mathcal{N}_S can be written as:

$$\mathcal{N}_S(x) = \{ x' : x' = \pi_1 \cdot \langle x_i \rangle \cdot \pi_2 \cdot \langle x_j \rangle \cdot \pi_3 \wedge x = \pi_1 \cdot \langle x_j \rangle \cdot \pi_2 \cdot \langle x_i \rangle \cdot \pi_3 \} \quad (6.1)$$

6.1.2.2 Inserting

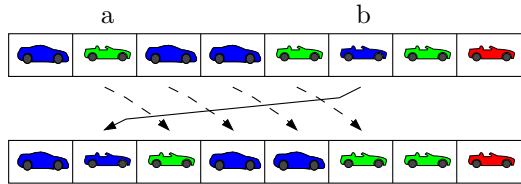


Figure 6.2: One car is shifted from position b to position a. All cars between these two positions are moved one position back.

The neighbourhood Inserting (\mathcal{N}_I) is similar to \mathcal{N}_S but only one car is rearranged. This car is removed from position i and inserted again at position j . Therefore all cars placed between positions i and j are moved one position forward or backward (see figure 6.2). There are at most $n^2 - n$ different neighbours. Again, the reevaluation of the objective function can be tricky. This time all costs raised by cars between positions i and j have to be reevaluated. Additionally the violations for the positions in the interval $[\max\{i, j\}, \max\{i, j\} + s]$ have to be counted.









Table 6.2 (a) presents an example arrangement. The components of the cars are neglected for simplicity. This example shows why the evaluation is more complex, although Swapping is very similar.

If the maximum colour block s is set to 3, it is clear that the arrangement in table 6.2 (a) has 2 violations of paint shop constraints. After removing the car at position 6 and









inserting it again at position 1, there is one additional colour change. In return one paint shop constraint violation at position 8 is eliminated. Table 6.2 (b) shows the resulting arrangement if the car at position 6 is inserted at position 1.

If only the positions in the intervals $[1, 4]$ and $[6, 9]$, $\forall c \in C$ are reevaluated, as it is done for swap moves, the violation at position 4 in the original arrangement is corrected but the new violation at position 5 would be ignored. Additionally the colour change at position 5 would be counted two times: once at position 5 (although there is no colour change left) and once at its new position 6. Using 1 as costs for colour changes and 100 as costs for violations, the original arrangement would evaluate to 201 (2 violations plus 1 colour change). Recalculating intervals $[1, 4]$ and $[6, 9]$ would yield a change of -198 , which would result in 3 as costs for the new arrangement. The actual costs for the new arrangement are 102.

Although the current evaluation of an insert move requires $\mathcal{O}(n)$ time, it can be achieved in constant time $\mathcal{O}(m_c)$.

position	1	2	3	4	5	6	7	8
car								
blue					X	X	X	X
green	X	X	X	X				
violation				⚡				⚡
colour change					•			

(a)

position	1	2	3	4	5	6	7	8
car								
blue	X					X	X	X
green		X	X	X	X			
violation				-⚡	⚡			-⚡
colour change		•			◦	•		

(b)

Table 6.2: The arrangement before the insertion move was applied (a); and afterwards (b). ⚡ indicates a violation, • is a colour change, ◦ is a removed colour change and -⚡ is a removed violation.

The formal description of this neighbourhood is

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

$$\{ x' : x' = \pi_1 \cdot \pi_2 \cdot \langle x_i \rangle \cdot \pi_3 \wedge x = \pi_1 \cdot \langle x_i \rangle \cdot \pi_2 \cdot \pi_3 \} \quad \text{(backward)} \quad (6.2)$$

If $\pi_2 = \langle x_j \rangle$, an insertion move is equal to a swap move swapping two consecutive cars.

6.1.2.3 Selection by Random

A swap move rearranges only two cars and An insertion move rearranges only one car. Especially if the current solution is close to a local optimum, the number of favourable moves is low. This means that in general only small improvements can be achieved.

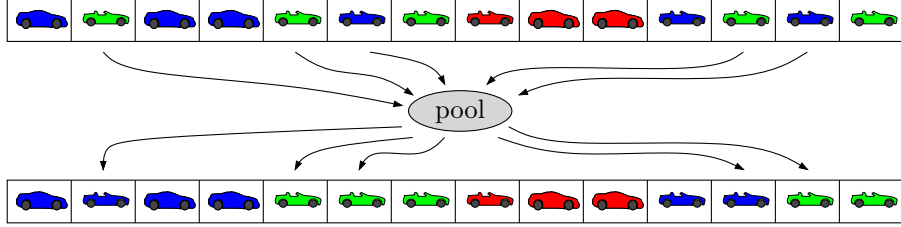


Figure 6.3: Cars are set free randomly, which means that they are put into a pool. Afterwards they are rearranged to the free positions.

Within this neighbourhood \mathcal{N}_{R_z} all solutions with z free positions and $n - z$ fixed positions are located. These z free positions are chosen by random (see figure 6.3). To scan this neighbourhood, the ILP defined in section 4.2.1 is used. If position i is fixed with configuration k , the variable \mathbf{p}_{ki} is set to 1 and all other variables \mathbf{p}_{ji} with $j \in K$ and $j \neq k$ are set to 0. If the position i is set free, all variables \mathbf{p}_{ki} with $k \in K$ are set free too. Examining this neighbourhood always leads to x^* with $f(x^*) \leq f(x')$, $\forall x' \in \mathcal{N}_{R_z}$ if enough computation time is available. If CPLEX consumes too much time solving the ILP, it is stopped and I use the best solution found so far as x^* .

Within this neighbourhood, there are $\binom{n}{z}$ different possibilities to select z positions. Once the free positions are chosen, there are $\mathcal{O}(n!)$ different arrangements taking the z free cars into account.

Below the formal description is presented:

$$\begin{aligned} \mathcal{N}_{R_z}(x) = \{ & x' : x' = \pi_1 \cdot \langle x_{j_1} \rangle \cdot \pi_2 \cdot \dots \cdot \langle x_{j_{z-1}} \rangle \cdot \pi_z \cdot \langle x_{j_z} \rangle \cdot \pi_{z+1} \wedge \\ & \wedge x = \pi_1 \cdot \langle x_{i_1} \rangle \cdot \pi_2 \cdot \dots \cdot \langle x_{i_{z-1}} \rangle \cdot \pi_z \cdot \langle x_{i_z} \rangle \cdot \pi_{z+1} \} \\ & \text{with } \{j_1, \dots, j_{z-1}, j_z\} = \{i_1, \dots, i_{z-1}, i_z\} \subseteq \{1, \dots, n\} \end{aligned} \quad (6.3)$$

6.1.2.4 Selection by Costs

The neighbourhood \mathcal{N}_{C_z} is equal to neighbourhood \mathcal{N}_{R_z} except for the strategy for selecting the free variables. In contrast to \mathcal{N}_{R_z} , I chose the positions which cause the most costs. Since the cars at these positions violate more constraints than the other cars, it is promising to rearrange these cars first. If there are more than z cars causing the maximum costs, cars at the beginning of the sequence are favoured. Altogether this neighbourhood consists of up to $n!$ different arrangements.

Neighbourhood \mathcal{N}_{C_z} is defined as

$$\begin{aligned} \mathcal{N}_{C_z}(x) = & \left\{ x' : x' = \pi_1 \cdot \langle x_{j_1} \rangle \cdot \pi_2 \cdot \dots \cdot \langle x_{j_{z-1}} \rangle \cdot \pi_z \cdot \langle x_{j_z} \rangle \cdot \pi_{z+1} \wedge \right. \\ & \left. \wedge x = \pi_1 \cdot \langle x_{i_1} \rangle \cdot \pi_2 \cdot \dots \cdot \langle x_{i_{z-1}} \rangle \cdot \pi_z \cdot \langle x_{i_z} \rangle \cdot \pi_{z+1} \right\} \\ & \text{with } I = \{j_1, \dots, j_{z-1}, j_z\} = \{i_1, \dots, i_{z-1}, i_z\} \subseteq \{1, \dots, n\} \end{aligned} \quad (6.4)$$

$$\text{and } \sum_{k \in I} cost(i_k) = \max_{S \subseteq \{1, \dots, n\} \wedge |S|=z} \left\{ \sum_{k \in S} cost(i_k) \right\} \quad (6.5)$$

6.1.2.5 Order of neighbourhoods

VND (see section 5.1) needs t_{max} different neighbourhoods. t_{max} is set to 6 for my thesis.

First swap moves are applied to the current best solution, because Swapping is the smallest neighborhood of all. Then Inserting is considered. Afterwards Selection by Random is applied with $n/7$ free variables followed by Selection by Costs also with $n/7$ free variables. Neighbourhoods 5 and 6 are Selection by Random and Selection by Costs too, but this time with $2n/7$ free variables.

To summarise:

$$\begin{array}{lll} \mathcal{N}_1 = \mathcal{N}_S & \mathcal{N}_2 = \mathcal{N}_I & \mathcal{N}_3 = \mathcal{N}_{R_{n/7}} \\ \mathcal{N}_4 = \mathcal{N}_{C_{n/7}} & \mathcal{N}_5 = \mathcal{N}_{R_{2n/7}} & \mathcal{N}_6 = \mathcal{N}_{C_{2n/7}} \end{array}$$

6.1.3 Initial solution

There are three different heuristics for generating the initial solution used by VNS. The first places all cars with configuration 1 to the first positions along the assembly line. Then the cars with configuration 2 are placed following them and so on. This is done until all commissioned cars are placed along the production line. I call this heuristic *Naive Arrangement*.

The second heuristic—called *Radnom Arrangement*—places the cars along the production line like the first heuristic. Afterwards the cars are randomly swapped. Altogether n random swap moves are applied.

The last method for computing an initial solution simply performs Partitioning with *First Incumbent* strategy (see section 6.2.3). If no solution is found until a given amount of computation time has elapsed, the same initial solution as generated by the first heuristic is used.

In chapter 7 test results using different heuristics for computing the start solution are shown.

6.2 Partitioning

In addition to the implementation of VNS, I developed *Partitioning*. This is a heuristic which partitions the Car Sequencing Problem in two subproblems: first reducing the violations for working bay constraints and then minimising the number of colour changes.

Although these two subproblems are very similar, they differ in some situations. As explained in section 2.3.2 the demands defined by the paint shop and the constraints defined by the working bay can be expressed as $\frac{l}{m}$. But the domains for l differ significantly. For paint shop constraints l has to be equal to $m - 1$, whereas for working bay constraints l can consist of values from 1 to $m - 1$. Furthermore paint shop constraints in general are longer than constraints defined by the working bays, i.e. $s + 1 > \max_{c \in C \setminus F} \{m_c\}$. Last but not least paint shop constraints are hard constraints, thus no violations are allowed for these constraints.

To achieve good results the solutions to both above mentioned subproblems must satisfy following requirements:

Condition 1 Unnecessary violations of the currently considered constraints have to be prevented.

Condition 2 Although only a part of the constraints are used for evaluating the objective function, the solution found must be feasible to the entire problem.

Especially condition 6.2 is of particular importance, because if the currently available solution is not feasible to the entire problem, the second part of Partitioning probably cannot find any solution at all.

I combined this strategy with the ILP presented in section 4.2.1. I decided to reduce the number of violations of the assembly line constraints first, because the constraints defined by the paint shop are harder to solve using ILPs. After the first step, one colour is assigned to each position. In the second step, the components already assigned to the positions along the production line are considered. This makes the problem of placing the colours more complicated, as some combinations of components along the assembly line do not get along with certain colours. With this strategy, the general problem becomes easier, because we do not have to consider all constraints at the same time. Algorithm 6.2 shows the basic principle of Partitioning written in pseudocode.

Algorithm 6.2: Partitioning()

Data: x current solution

Output: an arrangement along the assembly line

- 1 $x \leftarrow$ solve CarSP using the ILP in respect to constraints defined by the working bays
 - 2 solve CarSP using the ILP with components fixed regarding to x
-

For the combination I had to adapt the ILP formulation twice: for part one neglecting the colour changes and for part two setting variables to previously computed values, respectively.

6.2.1 ILP for part one

For the first part, i.e. reducing the number of violations of the working bay constraints, constraints (4.33) to (4.35) have to be removed. In addition, the objective function (4.18) must be reformulated to

$$\min \sum_{c \in C \setminus F} \left(cost_c * \sum_{i=0}^{n-1} \mathbf{g}_{ci} \right) \quad (6.6)$$

Now the ILP is solved producing a solution with a minimum objective subject to the constraints defined by the working bays. Since constraint (4.31) and (4.32) are still considered, the provided solution conforms condition 6.2. Furthermore the output of this part meets condition 6.2 too.

6.2.2 ILP for part two

For the second part of Partitioning, i.e. minimising the number of colour changes, the ILP presented in section 4.2.1 has to be adapted once again. This time additional constraints are added to the original formulation. These constraints make sure that the distribution of components computed so far remains unchanged in the final solution. Therefore matrix \mathbf{B}' is introduced, which is the final output of part one. The entries \mathbf{b}'_{ci} are binary variables with

$$\mathbf{b}'_{ci} = \begin{cases} 1 & \text{if car at position } i \text{ contains component } c \\ 0 & \text{otherwise} \end{cases} \quad (6.7)$$

Now the additional constraints can be defined as

$$\mathbf{p}_{ki} \leq \mathbf{a}_{ck} * \mathbf{b}'_{ci} + (1 - \mathbf{a}_{ck}) * (1 - \mathbf{b}'_{ci}) \quad \forall c \in C \setminus F, \forall k \in K, \forall i \in \{0, \dots, n - 1\} \quad (6.8)$$

As a result of this formulation, the output of this extended ILP forms a valid solution to the entire problem. Because of the additional constraints defined by the inequation (6.8), this part meets requirement 6.2. Furthermore the ILP leads to an optimum in respect to the previously computed preconditions which is consistent with condition 6.2.

6.2.3 First Incumbent strategy

Since solving the ILPs for part one and part two on their own can consume much time, a *next improvement strategy* can be applied. In contrary to solving the ILPs exact, this strategy only delivers the first feasible solution to the subproblems. Although this approach reduces the runtime of each part, it is possible that no solution can be found in a given amount of time.

This strategy is used for computing initial solutions for the VNS algorithm (see chapter 5).

7 Tests and results

In this chapter, I will present the results of my programs in comparison to other solutions, which can be found in literature. All results were computed on a Intel Pentium 4 with 2 GB RAM and 2.8 GHz. All programs are written in C++ using the library EALib 2.0, which is written in C++ too and is actively developed by the Institute of Computer Graphics and Algorithms at the Vienna University of Technology, Austria. For solving ILPs the commercial general purpose solver CPLEX 9.0 by Ilog, Inc., is used.

For testing I used instances defined by Renault for the ROADEF Challenge 2005 [1] and instances of CSPLIB [5] provided by Lee et al. and Gent et al. While the instances from Renault contain up to 1300 cars to be arranged in respect to constraints defined by the working bays and the paint shop, instances in the CSPLIB consist of up to 200 cars without any paint shop constraints.

As stated in section 2.3.1 there are differences in counting the number of violations between ROADEF and CSPLIB. I decided to use the method defined by ROADEF.

7.1 Results on CSPlib instances

CSPLIB [5] itself is a library of instances and usefull tools for problems modeled with constraints. Beside problems derived from the area of bioinformatics, bin packing, combinatorial mathematics, etc., there are also problems from the field of scheduling included. The instances for the Car Sequencing Problem in CSPLIB are divided in two sets. The first set was proposed by Lee et al. [12] and consists of 70 instances with 200 cars. There are 5 components and 17 to 30 configurations. For all of the instances at least one solution without any violations is prooven to exist. The utilisation rate, i.e. the percentage of cars requiring a component in regard to all cars of the instance, is 60 to 90. The instances are named `<utile-rate>-<problem-number>`, e.g. 60-01 indicates instance number one with utilisation rate 60%. There are 10 instances for each utilisation rate. I refer to them as `<utile-rate>-*`.

The second set of nine instances was presented by Gent et al. [5]. They consist of only 100 cars, again 5 components and 19 to 26 different configurations. Although these parameters indicate easier instances, they are harder to solve than the first set. For four of these instances, it is known that there is at least one solution without any violation. Four other instances were proved to be unsolvable which means that at least

	60-*	65-*	70-*	75-*	80-*	85-*	90-*
C-ILP	7.98	10.93	10.74	20.86	29.63	29.84	54.13
K-ILP	15.92	15.48	25.30	36.07	50.63	59.65	99.34
GRAVEL	5.74	12.43	21.48	27.80	47.70	68.42	127.78

Table 7.1: Average times for solving the first set of CSPLIB instances to optimality in seconds.

	60-*	65-*	70-*	75-*	80-*	85-*	90-*
C-ILP	9.2	9.06	2.78	11.8	15.71	13.62	24.24
K-ILP	9.5	7.72	5.57	19.93	18.09	22.32	42.5
GRAVEL	4.17	6.58	10.05	13.04	27.22	37.47	65.22

Table 7.2: The standard deviation from the average times presented in table 7.1.

one violation is unavoidable. For the ninth instance there is no optimal solution known. No new results have been achieved during my test runs for this set of instances.

7.1.1 ILPs

Since the ILP formulation proposed by Gravel et al. in [7] counts the number of positions but not the amount of violations, I adopted their ILP, see appendix B for the modified formulation. This adoption also takes the day before into account. Although this is not necessary for instances of the CSPLIB, this allows comparison using other instances (see section 7.3).

Table 7.1 and 7.2 displays the runtime results obtained solving the first set of the CSPLIB instances. All instances with the same utilisation rates are grouped for compensation of outliers. *C-ILP* indicates the ILP defined in section 4.1. *K-ILP* stands for the formulation presented in section 4.2. *GRAVEL* indicates the adopted ILP proposed by Gravel et al. Since at least one solution without any violation exists for this set of instances and each of the ILP delivers the optimal solution in acceptable time, I present the average time over all instances to one utilisation rate for finding optimal solutions and proving their optimality.

The formulation by Gravel et al. yields the best results for the group of instances with utilisation rate 60. For all other groups of instances my approach is the fastest one. For groups 85-* and 90-*, even the enhanced formulation is better than the approach by Gravel et al.

	60-*	65-*	70-*	75-*	80-*	85-*	90-*
best solution	0.77	0.86	0.9	1.07	1.38	56.31	2.01
deviation	0.1	0.1	0.08	0.28	0.36	163.94	0.34
optimality	10	10	10	9	7	3	1

Table 7.3: The average time and the standard deviation for finding the best obtained solution.

7.1.2 VNS

Although the CSPLIB instances can be solved by the ILP formulations in acceptable time, I tested VNS on them too. VNS did not always yield the optimal solution, which is 0 for the CSPLIB instances. Therefore table 7.3 shows the average time for finding the best achieved solution. Further it presents the number of instances for which the optimal solution was found.

7.1.3 Partitioning

Since Partitioning is developed for instances with working bay and paint shop constraints, it makes no sense to test Partitioning on the CSPLIB instances. Partitioning would yield the same results as K-ILP.

7.2 Results on ROADEF instances

ROADEF published three sets of test instances: set A, set B and set X [1]. The last set was used for the final evaluation procedure and ranking. I performed tests with all three sets for comparison with the results obtained by the ROADEF Challenge.

Set A consists of 16 instances with 334 to 1314 cars, 11 to 24 colours, 6 to 22 components and 36 to 287 configurations. Set B provides a wide range of 45 instances with 65 to 1270 cars, 4 to 20 colours, 4 to 25 components and 11 to 339 configurations. There are 19 instances with 65 to 1319 cars, 5 to 20 colours, 5 to 26 components and 10 to 328 configurations in set X.

For all ROADEF instances with more than 500 cars, no useable results could be obtained using the ILP formulations. Therefore I developed the VNS approach with six different setups. I tested three different heuristics for providing an initial solution: Naive Arrangement, Random Arrangement, Partitioning with First Incumbent strategy.

Naive Arrangement simply places the configurations along the production line without any additional computation. Therefore this method is very fast and the most computation time is used for examining the different neighbourhoods.

Random Arrangement places the cars along the production line like the first method. Afterwards, it applies n random swap moves to the arrangement.

The last method uses Partitioning with First Incumbent strategy as start heuristic, but the computation time is limited. If there is no solution found after 300 seconds, the initial solution computed by the first method is used.

det_best	...	Naive Arrangement in combination with best improvement
det_next	...	Naive Arrangement in combination with next improvement
part_best	...	Partitioning in combination with best improvement
part_next	...	Partitioning in combination with next improvement
rand_best	...	Random Arrangement in combination with best improvement
rand_next	...	Random Arrangement in combination with next improvement

Table 7.4: Denotation of names to the six different setups.

Once an initial solution is found, VNS gets started using either best or next improvement strategy. Table 7.4 shows the denotation of the six different setups and their names used in further tables and figures. ROADEF_best and ROADEF_worst refer to the best and worst solutions among all candidates obtained during the ROADEF Challenge. These two values are indicated in diagrams using dashed lines. If only one line is visible in a diagram, these two solution values are very close to each other. Furthermore bold values indicate the best solution value for the corresponding instance. For rand_best and rand_next the values present the average over 11 runs. The values in brackets indicate the standard deviation.

I limited the runtime of VNS to 600 seconds for all instances published by ROADEF. This is the same time limit as used at the ROADEF Challenge.

7.2.1 Set A

Tables 7.5 and 7.6 show the absolute values of the objective function obtained for a few instances contained in set A. Figure 7.1 presents a graphic representation. As we can see, the best improvement strategy is generally worse than the next improvement strategy. This is because the neighbourhoods Swapping and Inserting consist of too many possible solutions. Therefore VND is not able to complete the search in these neighbourhoods using best improvement. Furthermore we see that VNS using a random initial solution is usually worse than using one of the other two heuristics.

7 Tests and results

	det_best	part_best	rand_best	ROADEF_worst
(1)	97159817	93106632	102989604 (9688221)	71118491
(2)	81372363	72365174	104255096 (5752215)	63376290
(3)	113070558	114070587	129986772 (9309780)	82165438
(4)	128399127	128399161	131554928 (9287529)	75477143
(5)	31102368	27102368	31632003 (3914997)	27135374
(6)	23207713	24200774	30585611 (2867805)	27202853

Table 7.5: Results obtained using set A with best improvement.

	det_next	part_next	rand_next	ROADEF_best
(1)	69078468	66083470	67613095 (7686433)	4000302
(2)	72365174	72365174	62207252 (6080618)	4000302
(3)	67082428	67082428	95618711 (10163553)	4034309
(4)	49411121	49411121	95683561 (7160459)	4280079
(5)	25085363	26082354	29989630 (2642313)	61290
(6)	25209729	25209729	27027781 (1403191)	174612

Table 7.6: Results obtained using set A with next improvement.

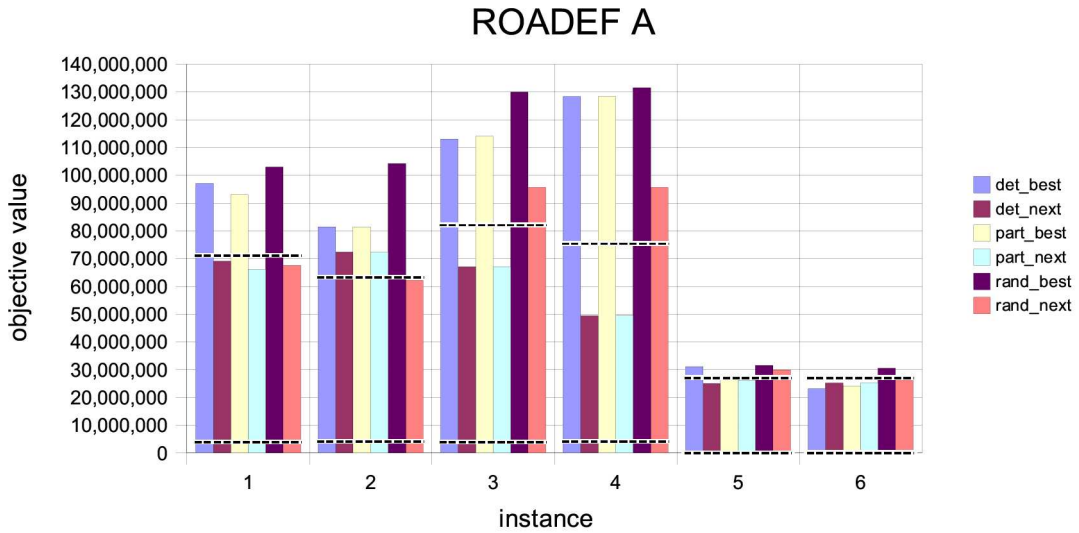


Figure 7.1: Results for the instances: 024_38_3_EP_ENP_RAF (1), 024_38_3_EP_RAF_ENP (2), 024_38_5_EP_ENP_RAF (3), 024_38_5_EP_RAF_ENP (4), 048_39_1_EP_ENP_RAF (5) and 048_39_1_EP_RAF_ENP (6).

7.2.2 Set B

The results obtained for set B are shown in tables 7.7 and 7.8 and figures 7.2 and 7.3. For instances (1)–(5) VNS was better than the best solution obtained during the ROADEF Challenge. Furthermore it is interesting that there is little difference between the various strategies used for solving these instances. For instances (6)–(9) there is more variance between the methods I proposed, but the results are acceptable for all instances.

	det_best	part_best	rand_best	ROADEF_worst
(1)	1327780	1327776	1197090 (28414)	4096795
(2)	53073107	53057109	53611015 (503403)	72687159
(3)	53074155	53074155	53350130 (446628)	54078415
(4)	61057057	61057057	61694513 (771196)	67061061
(5)	62047063	62047063	61410339 (643172)	67053060
(6)	207182	207182	270651(6636)	234094
(7)	222389	220356	236544 (5264)	233461
(8)	319422	319407	347203 (5498)	317852
(9)	222389	209315	351764 (7152)	211317

Table 7.7: Results obtained using set B with best improvement.

	det_next	part_next	rand_next	ROADEF_best
(1)	1104672	1103671	1161685 (42187)	3912479
(2)	53068109	53061107	5324683 (386487)	54003076
(3)	53082133	53082133	53077040 (2540)	54049124
(4)	61057061	61057061	61875421 (574385)	67052049
(5)	61044067	61044067	62137973 (513909)	67036061
(6)	219112	207128	242901 (7391)	130187
(7)	229299	207300	225166 (5646)	161378
(8)	304837	304836	326311 (5642)	172180
(9)	220189	220189	294703 (8640)	189103

Table 7.8: Results obtained using set B with next improvement.

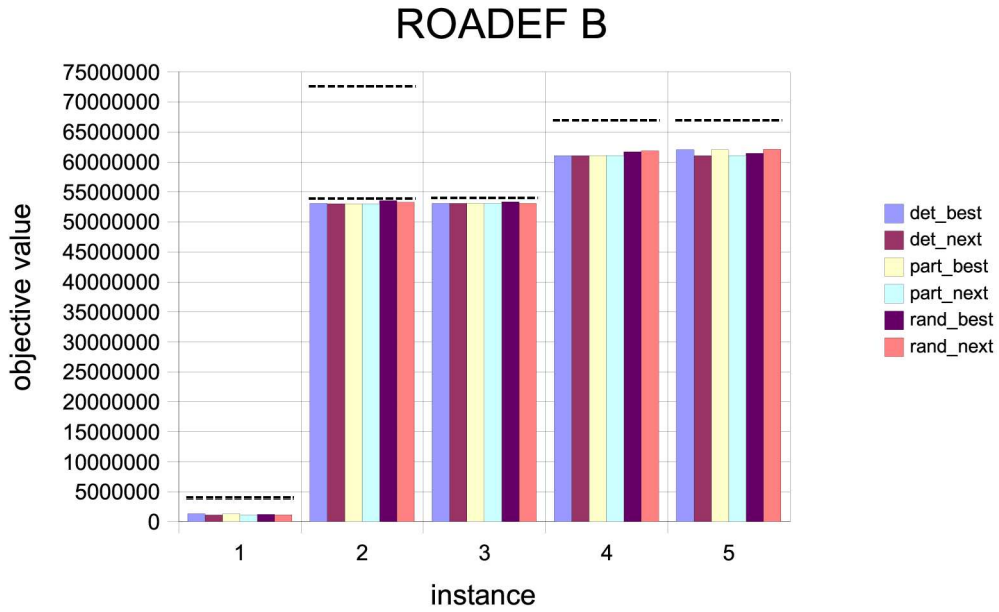


Figure 7.2: Results for the instances 025_EP_ENP_RAF_S22_J3 (1), 028_ch1_EP_ENP_RAF_S22_J2 (2), 028_ch1_EP_RAF_ENP_S22_J2 (3), 035_ch1_EP_ENP_RAF_S22_J3 (4) and 035_ch1_EP_RAF_ENP_S22_J3 (5).

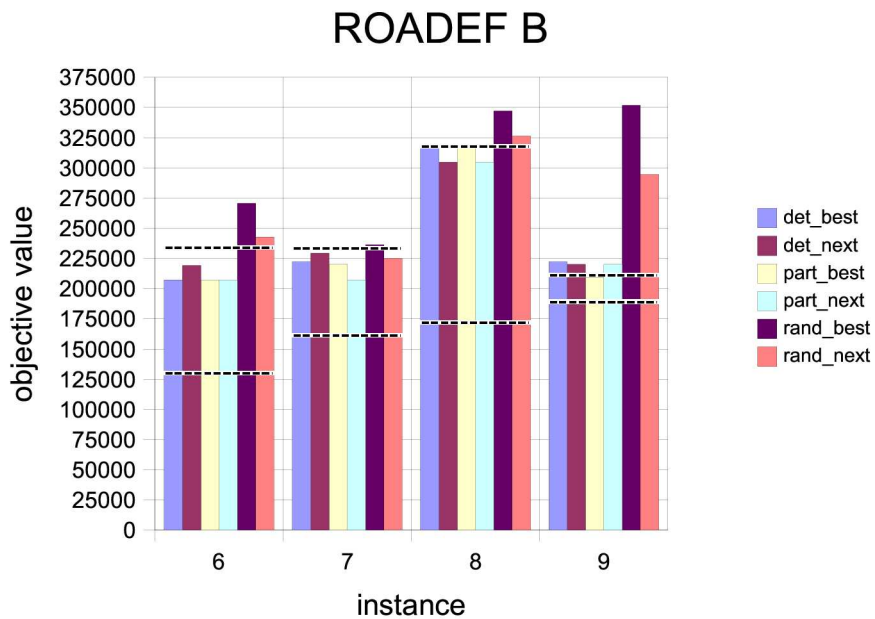


Figure 7.3: Results for the instances 064_ch1_EP_RAF_ENP_S22_J3 (6), 048_ch1_EP_RAF_ENP_S22_J3 (7), 025_EP_RAF_ENP_S22_J3 (8) and 039_ch3_EP_RAF_ENP_S22_J4 (9).

7.2.3 Set X

This set was used by ROADEF to calculate the final ranking for the Challenge. Tables 7.9 and 7.10 show the results obtained by my implementation of VNS. In figures 7.4 and 7.5 diagrams of these values are presented. Again, the next improvement strategy outperforms best improvement for several instances. For instance (7), I was able to find a new best solution. In general, VNS produces acceptable results which compete well with other results obtained during the ROADEF Challenge.

	det_best	part_best	rand_best	ROADEF_worst
(1)	170025	171022	235093 (7537)	181023.8
(2)	134351	134351	364176 (12340)	88237
(3)	263336	265336	362807 (6871)	285101
(4)	229000	229000	246625 (4086)	360000
(5)	54000	61000	70454 (2871)	72400
(6)	39000	40000	41636 (3891)	53000
(7)	7101047	7101047	7106951 (1703740)	12103052
(8)	42395095	42407101	44623736 (1850263)	44567925.8
(9)	39119011	39119011	36587453 (2020568)	34951538
(10)	87253070	87249077	67862427 (2903141)	67263119.6

Table 7.9: Results obtained using set X with best improvement.

	det_next	part_next	rand_next	ROADEF_best
(1)	198013	203024	220911 (9661)	110298.4
(2)	115320	115319	297602 (10788)	69239
(3)	269046	269048	331863 (7613)	231030
(4)	239988	240980	242442 (4327)	197005.6
(5)	59000	54000	69636 (3796)	37000
(6)	41000	41000	38636 (4959)	30000
(7)	8101040	8101040	6374408 (2415365)	8087035.8
(8)	42409110	44420100	44155644 (2801445)	36341495.4
(9)	36113968	39121957	35673470 (2677826)	31077916.2
(10)	65275050	67273056	64286508 (2130224)	61187229.8

Table 7.10: Results obtained using set X with next improvement.

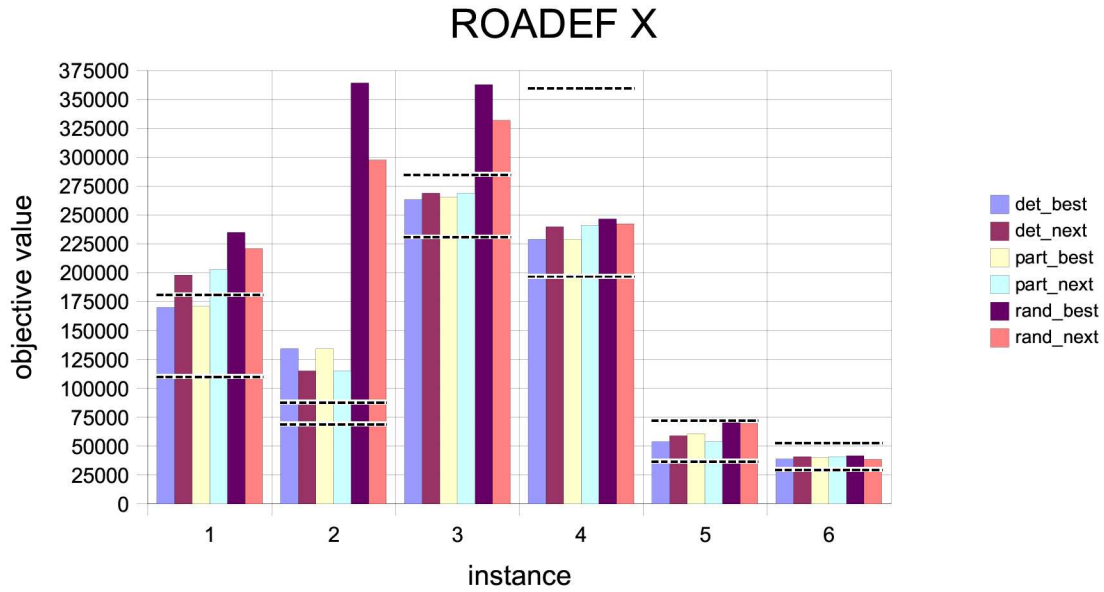


Figure 7.4: Results for the instances 029_EP_RAF_ENP_S49_J5 (1), 039_CH1_EP_RAF_ENP_S49_J1 (2), 039_CH3_EP_RAF_ENP_S49_J1 (3), 048_CH1_EP_RAF_ENP_S50_J4 (4), 064_CH2_EP_RAF_ENP_S49_J4 (5) and 655_CH1_EP_RAF_ENP_S51_J2_J3_J4 (6).

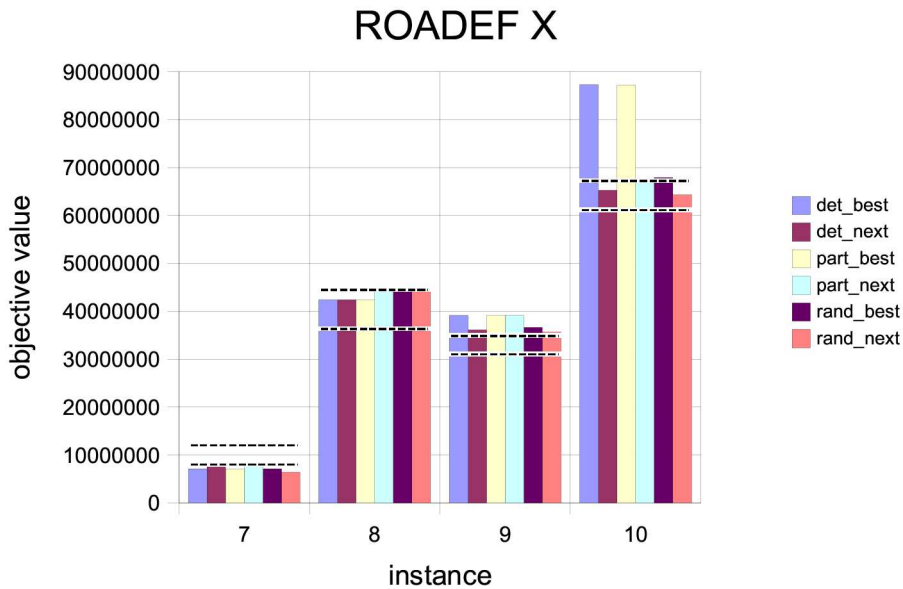


Figure 7.5: Results for the instances 034_VU_EP_RAF_ENP_S51_J1_J2_J3 (7), 028_CH1_EP_ENP_RAF_S50_J4 (8), 048_CH2_EP_RAF_ENP_S49_J5 (9) and 064_CH1_EP_RAF_ENP_S49_J1 (10).

7.3 Results on new instances

To compare the three ILPs discussed in my thesis in more depth, I created new instances. These instances include constraints defined by the working bays, constraints defined by the paint shop and the production of day $N - 1$. They consist of about 70 cars with ca. 10 different configurations with up to 10 components and 5 colours. The instances were produced by dividing ROADEF instances into smaller instances. In appendix C two of these instances are presented. Figure 7.6 shows the typical behaviour of the three different ILP approaches. The enhanced ILP reaches optimality relatively fast. The other two approaches need much more time to find the optimal solution, although all three approaches find good solutions in acceptable time, if the size of the instances is held small (about 70 cars including constraints by the paint shop and the assembly shop). In addition, much time is spend on proving the optimality of an obtained solution. Again, figure 7.7 shows the typical behaviour of these three approaches.

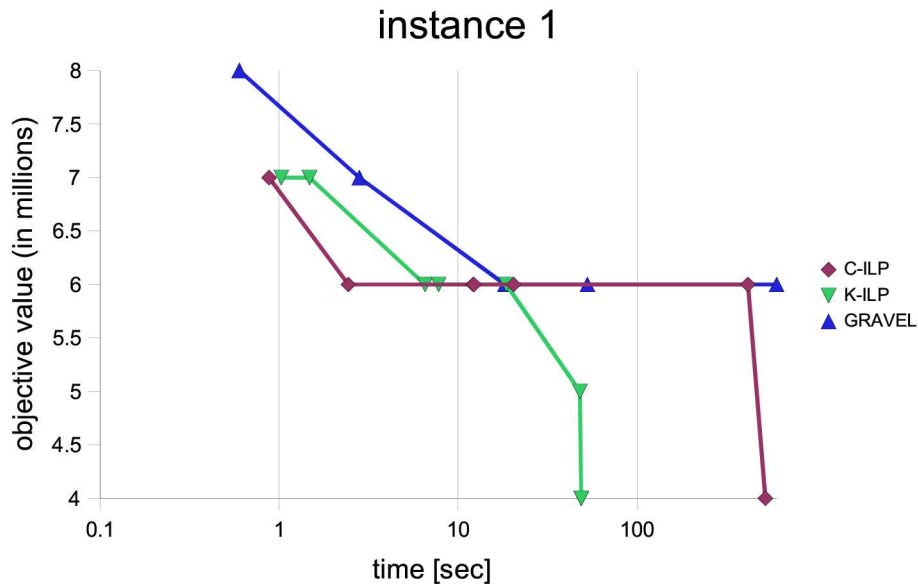


Figure 7.6: Solution process for instance 1 using the ILPs.

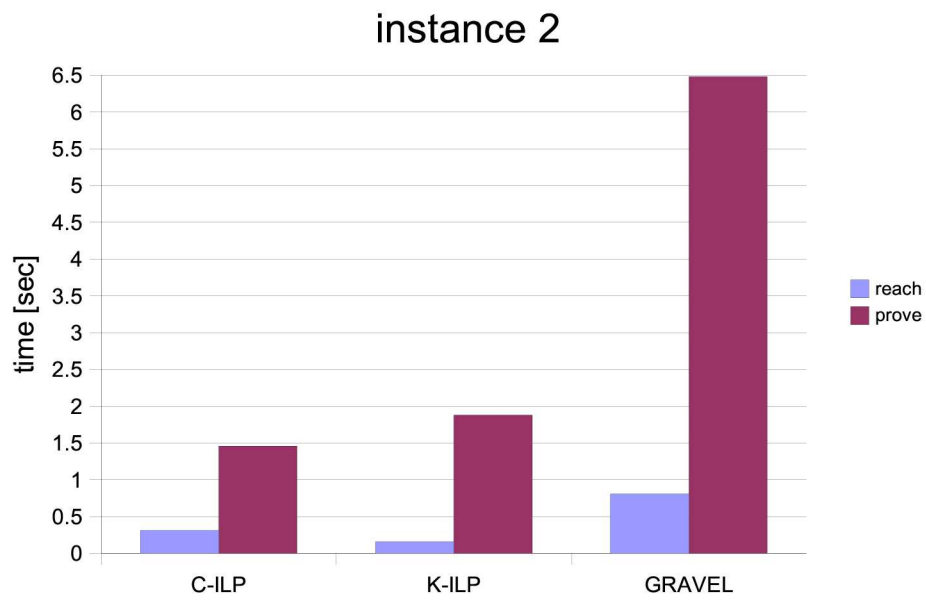


Figure 7.7: The relation between the time needed to reach an optimum and the time to prove it for instance 2.

8 Conclusion and future work

8.1 Conclusion

The tests showed that solving the Car Sequencing Problem exact using ILP approaches works fine for small instances with up to 70 cars including paint shop and assembly shop constraints. I was able to half the required computation time for some instances of the CSPLIB. Test runs using ILPs on ROADEF instances did not yield any useable results so far, since these instances consist in general of 500 cars and more. The ROADEF instances are harder to solve than the instances in the CSPLIB, since there are much more cars and constraints in these instances.

To handle larger instances including paint shop constraints, I defined different neighbourhoods to be used in a Variable Neighbourhood Search approach. Two of these neighbourhoods were proposed by Gottlieb et al. The other neighbourhoods are ILP based, trying to combine exact methods with the metaheuristic. For 6 instances defined by ROADEF, I was able to achieve better results than the best obtained solution during the ROADEF Challenge. Although the Variable Neighbourhood Search performs well, some additional work should be done for achieving even better output. The definition and implementation of the neighbourhoods should be improved especially, since for some test instances the program did not find a local optimum in respect to the first two neighbourhoods.

I also proposed a heuristic called Partitioning, which tries to solve this problem by reducing the search space. Since many of the instances defined by ROADEF consist of more than 300 cars and more than 10 constraints for components only, solving this reduced problem in acceptable time is too hard for my ILP based approaches.

8.2 Future work

To achieve better results, especially on instances including paint shop constraints, some additional improvements should be applied to the methods discussed in this thesis.

8.2.1 Reimplementing Swapping and Inserting

The test runs revealed that VNS is not able to find a local optimum in respect to the neighbourhoods Swapping and Inserting especially for large instances. This is partly due the fact, that the evaluation of these neighbourhoods consumes too much computing time. Therefore the evaluation function has to be reimplemented. At this time for each potential move the change in the objective function is recalculated. Maybe the evaluation is faster, if only the differences between two possible moves are evaluated. Furthermore it should be examined if some moves can be excluded from the set of potential promising moves.

8.2.2 Combination of best and next improvement

For some instances the next improvement strategy yields much better results than best improvement. On the other hand, if the number of promising moves within a neighbourhood is much smaller than the number of all imaginable moves, best improvement strategies can lead to better performances. Therefore I think it is promising to combine these two methods.

The basic principle is presented in algorithm 8.1. The idea is to reduce the size of the neighbourhood $\mathcal{N}(x)$, if there are many good moves in the neighbourhood. A good move is a move decreasing the objective function if a minimum is searched. If the number of such promising moves decreases, the neighbourhood $\mathcal{N}(x)$ is searched more exhaustively, until no better solution can be found.

$\mathcal{N}_{Red}(x)$ denotes the reduced neighbourhood of the current best solution. $\mathcal{N}_{Red}^C(x)$ contains all solutions from $\mathcal{N}(x)$ which are not within $\mathcal{N}_{Red}(x)$. In other words

$$\mathcal{N}_{Red}(x) \cup \mathcal{N}_{Red}^C(x) = \mathcal{N}(x)$$

and

$$\mathcal{N}_{Red}(x) \cap \mathcal{N}_{Red}^C(x) = \{\}$$

p denotes the relative size of $\mathcal{N}_{Red}(x)$ in respect to $\mathcal{N}(x)$:

$$p = \frac{|\mathcal{N}_{Red}(x)|}{|\mathcal{N}(x)|}$$

The two functions $ADV(p)$ and $RED(p)$ advance and reduce the parameter p , respectively.

$$ADV(p) : p \leftarrow \frac{|\mathcal{N}_{Red}(x)| + y_b}{|\mathcal{N}(x)|}$$

$$RED(p) : p \leftarrow \max \left\{ z, \frac{|\mathcal{N}_{Red}(x)| - y_g}{|\mathcal{N}(x)|} \right\}$$

Algorithm 8.1: NextBestImprovement(x)

Input: x denotes the initial solution**Data:** x^* indicates the current best solution; z indicates a threshold for minimum size of $\mathcal{N}_{Red}(x)$ **Initialisation:** set z to any value $\in [0, 1]$ **Output:** finds a local optimum in respect to the neighbourhood $\mathcal{N}(x)$

```

1  $x^* \leftarrow x$ 
2 repeat
3    $x^* \leftarrow$  best improvement within  $\mathcal{N}_{Red}(x)$ 
4    $y_g \leftarrow$  number of good moves within  $\mathcal{N}_{Red}(x)$ 
5   if  $f(x^*) = f(x)$  then
6      $x^* \leftarrow$  next improvement within  $\mathcal{N}_{Red}^C(x)$ 
7      $y_b \leftarrow$  number of tried moves within  $\mathcal{N}_{Red}^C(x)$ 
8     ADV( $p$ )
9   else
10    RED( $p$ )
11 until no improvement was achieved and  $p/n \geq 1$ 
12 return  $x^*$ 

```

y_b denotes the number of moves which are tried out until an improvement was found when examining $\mathcal{N}_{Red}^C(x)$, whereas y_g is the number of good moves in $\mathcal{N}_{Red}(x)$.

8.2.3 Additional neighbourhoods

At the moment there are six different neighbourhoods examined by VNS. Especially the strategy of setting cars free in neighbourhoods \mathcal{N}_3 – \mathcal{N}_6 should be reconsidered. Choosing to randomly rearrange a given amount of cars possibly unties good sequences. On the other hand, choosing cars with many violations favours cars with difficult configurations. Therefore a rearrangement results in very similar solutions. Some new methods for setting cars free need to be thought of.

At present, there are only very large neighbourhoods. Using moves which take only similar cars into account would add some new smaller neighbourhoods. A new type of swap moves can be thought of, which only considers cars which differ in one or two components. Another fast move would be shifting one car only two or three positions, since the violations for only few positions have to be recalculated.

Instead of rearranging single cars only, it might be promising to move complete sequences of cars. These block moves have been used by Perron et al. [15] and Hu [10]. There are

two variants: swap block moves and shift block moves. Similar to moves affecting single cars, these moves swap to sequences of cars or shift one sequence forward or backward.

8.2.4 Modified Variable Neighbourhood Search

VNS, as described in chapter 5, tries to find a local optimum in respect to all defined neighbourhoods by examining the first neighbourhood until no further improvements can be achieved. Afterwards VNS iterates over the second neighbourhood. As soon as an improvement could be found, VNS restarts with examining the first neighbourhood.

This strategy can be changed into examining the current neighbourhood until no further improvements can be achieved in this neighbourhood. Afterwards, the search process restarts at neighbourhood one if at least one better solution was found. If no new local optimum was reached, the search continues with the next neighbourhood. Although this strategy differs only in detail from the VNS as described above, it might uncover new solutions.

Another approach is to change the strategy for finding a local optimum completely. The method used by either best improvement or next improvement only uses one solution within the current neighbourhood for a step towards the (local) optimum. Many other interim solutions are disregarded.

To take advantage of all neighbours within the current neighbourhood one could apply Depth First Search (DFS) or Breadth First Search (BFS). Since there are potentially thousands of promising neighbours, DFS should be adapted. Algorithm 8.2 shows a modified version of DFS using a best improvement strategy for searching the neighbourhood $\mathcal{N}(x)$. This approach returns the best achievable solution using the defined moves.

Algorithm 8.2: DFSForCSP(x, d)

Input: x denotes the initial solution; d denotes the current depth

Data: x^* indicates the current best solution

Initialisation: set z to any value > 1

Output: finds a local optimum in respect to the neighbourhood $\mathcal{N}(x)$

```

1  $x^* \leftarrow x$ 
2 if  $d \geq z$  then
3   return  $x^*$ 
4 forall  $x' \in \mathcal{N}_G(x)$  do
5    $x^r \leftarrow \text{DFSforCSP}(x, d + 1)$ 
6   if  $x^r$  is better than  $x^*$  then
7      $x^* \leftarrow x^r$ 
8 return  $x^*$ 

```

$\mathcal{N}_G(x)$ denotes all neighbours of x which can be obtained when applying a good move. To expand the search space, $\mathcal{N}_G(x)$ can be replaced with $\mathcal{N}(x)$, for example. In contrast to commonly used DFS, this approach stops after a given depth is reached. As the case may be, z is set to smaller values for larger instances and vice versa.

8.2.5 Expansion of Partitioning

Partitioning, as proposed in this thesis, arranges the cars in respect to the working bay constraints first. After this, Partitioning takes the paint shop constraints into account. Hu showed in his master thesis that arranging the cars in respect to the paint shop constraints only can be done in polynomial time [10]. Therefore it might be interesting to arrange the cars along the production line in respect to the paint shop constraints first. In a second step, an adapted version of the ILP proposed in section 6.2.1 produces an output which minimises working bay constraints in respect to the result of step one.

8.2.6 Relative vs. absolute

Many approaches including the strategies proposed in this thesis place cars or configurations at absolute positions along the production line. Although this absolute placement is necessary in the last step of the scheduling process, it might be promising to pay more attention to the relative position of a car in regard to its preceding and succeeding vehicles.

A Original ILP formulation by Hu

This formulation uses the symbols defined in table 4.1. For a detailed description, I refer to [10].

objective

$$\min \sum_{i \in C \setminus F} \sum_{j=m_i}^n v_{ij} + \sum_{i \in F} \sum_{j=2}^n w_{ij} \quad (\text{A.1})$$

subject to

$$\sum_{i=1}^n x_{ij} = 1 \quad \forall j \in \{1, \dots, n\} \quad (\text{A.2})$$

$$\sum_{j=1}^n x_{ij} = 1 \quad \forall i \in \{1, \dots, n\} \quad (\text{A.3})$$

$$p_{i0} = 0 \quad \forall i \in C \setminus F \quad (\text{A.4})$$

$$p_{ik} = p_{ik-1} + \sum_{j=1}^n b_{ji} \cdot x_{jk} \quad \forall i \in C \setminus F, \forall k \in \{1, \dots, n\} \quad (\text{A.5})$$

$$v_{ij} \geq 0 \quad \forall i \in C \setminus F, \forall j \in \{m_i, \dots, n\} \quad (\text{A.6})$$

$$v_{ij} \geq p_{ij} - p_{ij-m_i} - l_i \quad \forall i \in C \setminus F, \forall j \in \{m_i, \dots, n\} \quad (\text{A.7})$$

$$q_{ik} = \sum_{j=1}^n b_{ji} \cdot x_{jk} \quad \forall i \in F, \forall k \in \{1, \dots, n\} \quad (\text{A.8})$$

$$\sum_{j=0}^s q_{ik+j} \leq s \quad \forall i \in F, \forall k \in \{1, \dots, n-s\} \quad (\text{A.9})$$

$$w_{ij} \geq 0 \quad \forall i \in F, \forall j \in \{2, \dots, n\} \quad (\text{A.10})$$

$$w_{ij} \geq q_{ij} - q_{ij-1} \quad \forall i \in F, \forall j \in \{2, \dots, n\} \quad (\text{A.11})$$

B Modified ILP by Gravel et al.

This formulation uses the symbols defined in table 4.1. Again, matrix \mathbf{A} , vector \mathbf{D} and matrix \mathbf{E} are defined as in section 4.1.

$$\mathbf{a}_{ck} = \begin{cases} 1 & \text{config } k \text{ contains component } c \\ 0 & \text{otherwise} \end{cases} \quad \forall c \in C, \forall k \in K$$

$$\mathbf{d}_c = \sum_{k \in K} (\mathbf{a}_{ck} \cdot \delta_k) \quad \forall c \in C$$

$$\mathbf{e}_{ci} = \begin{cases} 1 & \text{if the last but } i \text{ car required } c \\ 0 & \text{otherwise} \end{cases} \quad \forall c \in C, \forall i \in \{0, \dots, \max_{c \in C} \{m_c\} - 1\}$$

Following matrices are used in this formulation too:

$$\mathbf{x}_{ki} = \begin{cases} 1 & \text{if car at pos } i \text{ receives configuration } k \\ 0 & \text{otherwise} \end{cases}$$

\mathbf{y}_{ci} = number of violations at position i for component c

$$\mathbf{h}_j = \begin{cases} 1 & \text{if a colour change occurred} \\ 0 & \text{otherwise} \end{cases}$$

objective

$$cost_f \cdot \sum_{i=0}^{n-1} (\mathbf{h}_i) + \sum_{c \in C} \left(cost_c \cdot \sum_{i=0}^{n-1} (\mathbf{y}_{ci}) \right) \quad (\text{B.1})$$

subject to

$$\sum_{j=0}^{n-1} \mathbf{x}_{kj} = \delta_k \quad \forall k \in K \quad (\text{B.2})$$

$$\sum_{k \in K} \mathbf{x}_{kj} = 1 \quad \forall j \in \{0, \dots, n-1\} \quad (\text{B.3})$$

$$\sum_{k \in K} \left(\sum_{i=0}^j (\mathbf{a}_{ck} \cdot \mathbf{x}_{ki}) \right) + \sum_{i=0}^{m_c-j-2} \mathbf{e}_{ci} \leq l_c + \mathbf{y}_{cj} \quad \forall c \in C \setminus F, \forall j \in \{0, \dots, m_c-2\} \quad (\text{B.4})$$

$$\sum_{k \in K} \left(\sum_{i=j-m_c+1}^j (\mathbf{a}_{ck} \cdot \mathbf{x}_{ki}) \right) \leq l_c + \mathbf{y}_{cj} \quad \forall c \in C \setminus F, \forall j \in \{m_c-1, \dots, n-1\} \quad (\text{B.5})$$

$$\sum_{k \in K} \left(\sum_{i=0}^j (\mathbf{a}_{fk} \cdot \mathbf{x}_{ki}) \right) + \sum_{i=0}^{s-j-1} \mathbf{e}_{fi} \leq s \quad \forall f \in F, \forall j \in \{0, \dots, s-1\} \quad (\text{B.6})$$

$$\sum_{k \in K} \left(\sum_{i=j-s}^j (\mathbf{a}_{fk} \cdot \mathbf{x}_{ki}) \right) \leq s \quad \forall f \in F, \forall j \in \{0, \dots, s-1\} \quad (\text{B.7})$$

$$\mathbf{h}_0 \geq \sum_{k \in K} (\mathbf{a}_{fk} \cdot \mathbf{x}_{k0}) - \mathbf{e}_{f0} \quad \forall f \in F \quad (\text{B.8})$$

$$\mathbf{h}_j \geq \sum_{k \in K} (\mathbf{a}_{fk} \cdot \mathbf{x}_{kj}) - \sum_{k \in K} (\mathbf{a}_{fk} \cdot \mathbf{x}_{k(j-1)}) \quad \forall j \in \{1, \dots, n-1\}, \forall f \in F \quad (\text{B.9})$$

C New test instances

For testing the different ILP approaches, I defined several new instances. Two of them are presented in this section.

C.1 Instance 1

This instance consists of 65 cars. The maximum colour block is set to 20. The weights for the violations are 1000000 for colour changes, 1000 for important constraints and 1 for less important constraints. The first component of each car (c_1) has an important constraint. Table C.2 shows the production of day $N - 1$. In table C.1 the ratios of the constraints and the demand for the configurations can be seen.

	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	k_9	k_{10}	k_{11}	ratio
c_1	0	0	1	0	0	0	0	0	0	1	1	$1/6$
c_2	0	0	0	0	0	0	0	0	1	0	0	$1/5$
c_3	1	1	1	1	1	1	1	1	1	1	1	$1/2$
c_4	0	1	0	0	0	0	0	1	0	0	0	$1/2$
c_5	0	1	0	1	0	0	1	0	1	0	1	$1/4$
c_6	0	0	0	0	0	0	0	0	0	0	0	$1/5$
c_7	0	1	0	0	0	1	0	0	0	0	0	$1/3$
c_8	0	0	0	0	0	0	0	0	0	0	0	$2/3$
c_9	0	0	0	0	0	0	0	0	1	0	0	$1/28$
colour	1	1	1	2	3	3	3	3	3	3	3	$20/21$
demand	1	1	1	2	32	1	11	7	2	6	1	

Table C.1: The configurations for the current day and their demand.

C New test instances

pos	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	colour
$n - 26$	0	0	1	0	1	0	0	0	0	4
$n - 25$	1	0	1	0	0	0	0	0	0	4
$n - 24$	0	0	1	0	0	0	0	0	0	4
$n - 23$	0	0	1	0	1	0	0	0	0	4
$n - 22$	0	0	1	0	0	0	0	0	0	4
$n - 21$	0	0	1	1	0	0	0	0	0	4
$n - 20$	0	0	1	0	1	0	0	0	0	4
$n - 19$	0	0	1	0	1	0	0	0	0	4
$n - 18$	0	0	1	0	0	0	0	0	0	4
$n - 17$	1	0	1	0	0	0	0	0	0	4
$n - 16$	0	0	1	0	1	0	0	0	0	4
$n - 15$	0	0	1	0	0	0	0	0	0	4
$n - 14$	0	0	1	0	0	0	0	0	0	4
$n - 13$	0	0	1	0	1	0	0	0	0	4
$n - 12$	0	0	1	0	0	0	0	0	0	4
$n - 11$	0	0	1	0	1	0	0	0	0	4
$n - 10$	0	1	1	0	0	0	0	0	1	4
$n - 9$	1	0	1	1	0	0	0	0	0	4
$n - 8$	0	0	1	0	1	0	0	0	0	4
$n - 7$	0	0	1	0	1	0	0	0	0	4
$n - 6$	0	0	1	0	0	0	0	0	0	4
$n - 5$	0	0	1	1	0	0	0	0	0	4
$n - 4$	0	0	1	0	1	0	0	0	0	4
$n - 3$	0	0	1	0	0	0	0	0	0	4
$n - 2$	0	0	1	1	0	0	0	0	0	4
$n - 1$	0	0	1	0	1	0	0	0	0	4
n	0	0	1	0	1	0	0	0	0	3

Table C.2: The production of day $N - 1$ for instance 1.

C.2 Instance 2

This instance contains 65 cars and the maximum colour block is set to 13. This time colour changes only raise 1 costs. The weights for low priority constraints are 1000 and for high priority constraints are 1000000. Again, the first component (c_1) has a high priority constraint. Tables C.3 and C.4 show all relevant informations.

	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	k_9	k_{10}	ratio
c_1	0	0	0	0	0	0	0	0	1	1	$1/6$
c_2	0	0	0	0	0	0	0	1	0	0	$1/5$
c_3	1	0	0	0	0	0	1	0	0	1	$1/2$
c_4	0	0	0	0	0	1	0	0	0	0	$1/5$
c_5	0	0	0	0	1	0	0	0	0	0	$1/3$
c_6	0	0	0	0	0	0	0	1	0	0	$1/14$
colour	1	2	3	4	4	4	4	4	4	4	$13/14$
demand	1	1	2	36	7	1	4	4	7	2	

Table C.3: The configurations for the current day and their demand.

pos	c_1	c_2	c_3	c_4	c_5	c_6	colour
$n - 12$	0	0	0	0	1	0	5
$n - 11$	0	0	1	0	0	0	5
$n - 10$	0	0	0	0	0	0	5
$n - 9$	0	0	0	0	0	0	5
$n - 8$	0	0	0	0	0	0	5
$n - 7$	0	1	0	0	0	1	5
$n - 6$	1	0	0	0	0	0	5
$n - 5$	0	0	1	0	0	0	5
$n - 4$	0	0	0	0	1	0	5
$n - 3$	0	0	0	0	0	0	5
$n - 2$	0	0	0	0	0	0	5
$n - 1$	0	0	0	0	0	0	5
n	1	0	1	0	0	0	4

Table C.4: The production of day $N - 1$ for instance 2.

D Curriculum Vitae

Personal Information

- | | |
|--------------------|--|
| 18th of Sept. 1980 | • Born in Vienna, Austria |
| Nationality | • Austria |
| Languages | • German (native), English (fluent), French (basics) |
| 1987 – 1995 | • Piano lessons |
| since 1987 | • Competitive sports (Swimming) |

Education

- | | |
|-------------------------|---|
| since Oct. 2000 | • Student of Computer Science at the Vienna University of Technology, Austria |
| Oct. 1999 – Oct. 2000 | • Civilian Service in Vienna, Austria |
| Sept. 1991 – Sept. 1999 | • Comprehensive School in Stockerau, Austria |
| Sept. 1987 – Sept. 1991 | • Primary School in Korneuburg, Austria |

Work Experience

- since July 2005
 - Employed in the FWF project *Combining Memetic Algorithms with Branch and Cut and Price for Some Network Design Problem* under grant P16263-N04, Algorithms and Data Structures Group, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Austria
- Mar. 2002 – June 2005
 - Student assistant, Algorithms and Data Structures Group, Institute of Computer Graphics and Algorithms, Vienna University of Technology, Austria
- August 2001
 - Employed as intern, SCHINNER Vermögenstreuhand- und Versicherungsberatungsges.m.b.H.

Software Projects

- July 2004 – July 2005
 - Redesign of code fragments and implementation of a distributed framework for MOST; coded in Java
- May 2004 – June 2005
 - Reimplementing the compiler GAJA for the GANIMAL-framework; coded in Java
- Jan. 2004 – June 2004
 - Improvement and reimplementation of code fragments for MOST (Microvariability and Oscillations of Stars); coded in C
- Feb. 2004 – May 2004
 - Implementing additional functionality for the GANIMAL-framework; coded in Java

Bibliography

- [1] Roadeff challenge 2005. <http://www.prism.uvsq.fr/~vdc/roadeff/challenges/2005/>. last verified on 31st of August 2005.
- [2] Thomas Epping and Winfried Hochstättler. Abuse of multiple sequence alignment in a paint shop. Technical report, Zentrum für Angewandte Informatik Köln, Arbeitsgruppe Faigle/Schrader, July 2001.
- [3] Thomas Epping, Winfried Hochstättler, and Peter Oertel. A paint shop problem for words. Technical Report 395, Zentrum für Angewandte Informatik Köln, 2000.
- [4] Thomas Epping, Winfried Hochstättler, and Peter Oertel. Complexity results on a paint shop problem. *Discrete Applied Mathematics*, May 2002.
- [5] I.P. Gent and T. Walsh. CSPLib: a benchmark library for constraints. Technical report, APES-09-1999, 1999. Available at <http://4c.ucc.ie/~tw/csplib/schedule.html> (last verified on 31st of August 2005), a shorter version appeared in CP99.
- [6] Jens Gottlieb, Markus Puchta, and Christine Solnon. A study of greedy, local search, and ant colony optimization approaches for car sequencing problems. In Günther R. Raidl et al., editors, *EvoWorkshops*, volume 2611 of *Lecture Notes in Computer Science*, pages 246–257. Springer, 2003.
- [7] 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.
- [8] Yong-Hee Han, Chen Zhou, Bert Bras, Leon McGinnis, Carol Carmichael, and PJ Newcomb. Simulation in automotive industries: paint line color change reduction in automobile assembly through simulation. In *WSC '03: Proceedings of the 35th conference on Winter simulation*, pages 1204–1209. Winter Simulation Conference, 2003.
- [9] Pierre Hansen and Nenad Mladenović. A tutorial on variable neighborhood search. Technical Report G-2003-46, Les Cahiers du GERAD, HEC Montréal and GERAD, Canada, 2003.
- [10] B. Hu. Interaktive Reihenfolgeplanung für die Automobilindustrie. Master's thesis, Vienna University of Technology, Vienna, Austria, 2004.

Bibliography

- [11] Andrzej Jaskiewicz, Pawel Kominek, and Marek 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.
- [12] J. Lee, H. Leung, and H. Won. Performance of a comprehensive and efficient constraint library using local search. In G. Antoniou and J.K. Slaney, editors, *11th Australian Joint Conference on Artificial Intelligence*, pages 191–202. Springer Verlag Heidelberg, 1998.
- [13] E. Muhl, P. Charpentier, and F. Chaxel. Optimization of physical flows in an automotive manufacturing plant: some experiments and issues. *Engineering Applications of Artificial Intelligence*, 16(4):293–305, June 2003.
- [14] G. Nemhauser and L. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1988.
- [15] Laurent Perron and Paul Shaw. Combining forces to solve the car sequencing problem. In Jean-Charles Régin and Michel Rueher, editors, *CPAIOR*, volume 3011 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 2004.
- [16] Laurent Perron, Paul Shaw, and Vincent Furnon. Propagation guided large neighborhood search. In Mark Wallace, editor, *CP*, volume 3258 of *Lecture Notes in Computer Science*, pages 468–481. Springer, 2004.
- [17] Markus Puchta and Jens 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.