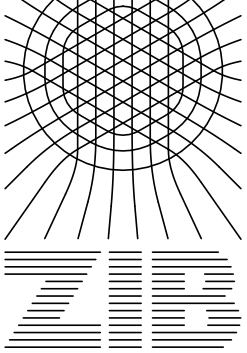
Konrad-Zuse-Zentrum für Informationstechnik Berlin
Heilbronner Str. 10, D-10711 Berlin - Wilmersdorf



Norbert Ascheuer

Hamiltonian Path Problems in the On–line Optimization of Flexible Manufacturing Systems

Technical Report TR 96–3 (February 1996)

Abstract

In this thesis we describe a practical problem that we encountered in the on–line optimization of a complex Flexible Manufacturing System. In the considered system a stacker crane has to fulfill all transportation tasks (jobs) in a single aisled automatic storage system. The jobs have to be sequenced in such a way, that the time needed for the unloaded moves is minimized. The modelling of this question leads to a mathematical problem, that has not been studied in that form in the literature so far, namely the so–called on–line Hamiltonian path problem.

We computationally compare several on–line heuristics and describe an optimization package, that we developed and implemented. This software package is now in use in everyday production on six automatic storage systems, resulting in an average reduction of 30% of the time needed for the unloaded moves. We derive lower bounds on the value obtained by an optimal on–line strategy by analyzing two off–line Combinatorial Optimization problems: the asymmetric Hamiltonian path problem with precedence constraints, also called sequential ordering problem (SOP), and the asymmetric Hamiltonian path problem with time windows (AHPPTW).

In contrast to many other publications for the AHPPTW we are not interested in minimizing the overall completion time, but in minimizing the total sum of intermediate set–up costs between the jobs. Thus, we are not restricted to use the well known model based on a generalization of the Miller–Tucker–Zemlin subtour elimination constraints but can define a TSP–like model. Here the time window restrictions are modeled by infeasible path constraints, forbidding path that violate the given time windows. The two models are compared from a computational point of view on real–life problem instances of up to 50 nodes. This comparison shows that the TSP–like model seems to be superior. For the SOP we work on a model that has been proposed earlier in the literature.

We study the SOP and AHPPTW from a polyhedral point of view and derive several new classes of valid inequalities. Based on the polyhedral investigations we develop branch&cut algorithms for both problems and can achieve encouraging results on solving problem instances from real–world examples of the practical application. For the SOP instances of up to 100 nodes and a varying precedence structure can be solved to optimality. For the AHPPTW instances up to 30–50 nodes and a varying time window width can be attacked by the branch&cut code.

As a by–product we obtained a branch&cut code for the asymmetric travelling salesman problem, whose computational performance on hard problem instances from TSPLIB is comparable to the codes published in the literature.

Acknowledgements

The research presented in this thesis was motivated by a joint project with the *Siemens Nixdorf Informationssysteme AG (SNI)*. It has begun at the *Universität Augsburg* and was completed at the *Konrad–Zuse–Zentrum für Informationstechnik Berlin (ZIB)*. I appreciate all the support these institutions offered.

A number of persons contributed to this thesis and without their support its completion would not have been possible.

The first to be thanked is *Martin Grötschel* for supervising the thesis in an excellent way. He gave me the impression that I could work independently, but always kept an overview on the progress being made. Whenever it was necessary, I could benefit from his broad research experience. I never made a request for his spare time in vain and I always left his office wiser than I entered. The pleasant discussions with him, his stimulating remarks and fruitful suggestions were the basis for the achieved results.

The industry project was carried out together with *Atef Abdel–Hamid*. Thanks to him for joining the laborious way of working in the project and for sharing the interesting and sometimes frustrating experiences such industry projects offer. We were confronted with problems of practical relevance leading to mathematically interesting problems, but also had to recognize that at least half of the problems that had to be faced were of a non–mathematical nature. The combination of Arabian and European mentality was the right mixture to overcome all obstacles involved in such a project.

I am also very grateful to our support team at SNI especially to *Dipl.–Ing. Herbert Schorer* and *Dipl.-Ing. Christof Gerstenäcker*. Without their support during the project, their persistence in collecting all necessary information and data, and their ability of creating a warm and friendly atmosphere, the successful completion of the joint project would not have been possible.

I would like to thank the members of the research group *"Combinatorial Optimization"* at the ZIB for the stimulating discussions during talks at the institute and at lunch time. In particular, I want to thank *Mechthild Stoer* for being a never ending source of good ideas concerning the topics presented in this thesis.

Furthermore, *Ralf Borndörfer, Nicola Kamin, Sybille Mattrisch, Michel Schaffers, Andreas Schulz, and Mechthild Stoer* are thanked for the "painful" work of proof reading different parts of this thesis. The final version of this thesis improved due to their critical remarks.

The implementation of the branch&cut code for the Sequential Ordering Problem was carried out jointly with *Michael Jünger, Stefan Thienel* (Universität zu Köln) and *Gerd Reinelt* (Universität Heidelberg) and was partially supported by the Science Program SC1–CT91–620 of the EEC. Thanks to Mike, Stefan, and Gerd for implementing a general purpose

branch&cut framework, primal heuristics, and for supplying separation routines that were originally designed to be used for the TSP. Beside this technical support, their friendship and encouragement always was a motivation to continue the debugging of the branch&cut code.

The research on the asymmetric Hamiltonian path problem with time windows was done jointly with *Matteo Fischetti* (University of Padova). I feel indebted to him for giving me the opportunity to benefit from his knowledge about Polyhedral Combinatorics. He never gave up explaining things to me and I learned a lot in the discussions with him. I appreciate the comments and suggestions *Andreas Schulz* made on the content of this chapter. Furthermore, I would like to thank Matteo and *Paolo Toth* for providing their branch&bound code for the ATSP.

Last but not least, I express my gratitude to *Karola* and *Katja* who gave me the necessary moral support to finish this thesis and had to tolerate all my moods during the last months. The time I spent with them was the right way to forget all the setbacks and disappointments that are typically involved in research.

Berlin, December 1994 Norbert Ascheuer

Contents

Introduction

In the past decade many enterprises have successfully responded to rapidly changing market demands and increasing competition ability by adopting concepts of flexible manufacturing. There is no doubt that problems from the field of flexible manufacturing belong to the topics of major interest of our time. This new technology, often described as the "third industrial revolution", has as well resulted in a continuous research in development and application of high technology in manufacturing.

Mathematically based optimization tools can be applied to many of the problems arising in the design and control of **Flexible Manufacturing Systems (FMSs)**. Despite this fact, the decision rules that are used in practice in order to control modern FMSs are often based on strategies that are recommended chiefly by their simplicity, such as priority rules or "First–In–First–Out" strategies. Still, the industry seems to trust more in experience–based knowledge than in theory–based knowledge. But due to the rapid technological changes (e.g., in computer industry) and an intensified international competition *"the era of trial–and–error learning belongs to the past because changes occur faster than lessons can be learned."*[FGL92]

The problems arising in flexible manufacturing are interesting from a mathematical point of view and have received considerable attention during the last years. A rapid increase in the number of publications on such problems manifests this trend. But so far, very few publications deal with real applications. Scanning through proceeding volumes of recent conferences on Operations Research or Manufacturing (see, among others, [SS89, IEE90, FGJ92]) demonstrates that most of the considered problems are "scientific problems". The applicability of the models and algorithms is very often not evident, as they are tested on randomly generated data or the considered FMS is not of such high complexity as real systems typically are.

The gap that exists between mathematical theory and industrial applications in manufacturing has been observed by several researchers. We give two quotations that can be found in the literature and that try to provide an analysis of this fact. The first is by R.H.F. Jackson, deputy director of the Manufacturing Engineering Laboratory at the National Institute of Standards and Technology, Gaithersburg, USA. He stated at the plenary address of the "First Joint US/German Conference on New Directions for OR in Manufacturing" held at his institute in July 1991 :

> *"... We believe this is a direct result of the declining number of operations research practitioners who are not content simply to develop a model of a problem, but insist on following through with implementation; who do not accept blindly a decision maker's description of a problem, but will expend considerable effort to understand all the underlying assumptions and question the validity of each before proceeding; who are not afraid to go out into the field or onto the factory floor to*

> *gather data ... and perhaps most importantly 'selling' the solution to the decision makers..." [FGJ92]*

The second is by M. Grötschel who stated during an invited lecture on the Second International Conference on Industrial and Applied Mathematics, 1991:

> *"... there is still a huge gap between what could be done, and what is actually done ... It is my opinion that there are at least two reasons for this phenomenon. In my experience many of the talented engineers who build and operate complicated manufacturing systems so ingeniously, simply don't have the background in the rather new mathematical techniques ... Secondly only a few mathematicians are willing to go through the laborious and occasionally painful process of understanding, analyzing and modeling complex manufacturing systems and then discussing their findings with the practitioners. Both parties suffer as a result. Companies in particular miss opportunities for more efficient and cost-effective production, and mathematicians opportunities to identify and solve challenging problems, problems that arise in connection with one of the most fascinating technical developments of our time..." [Grö92a]*

To our point of view the mentioned gap between theoretical results and practical applications has to be closed. This thesis is regarded as a step towards that direction.

The main motivation for the research, which is partly presented in this thesis, has been derived from a joint project with the Siemens Nixdorf Informationssysteme AG (SNI). In 1987 SNI erected a factory in Augsburg (Germany) where all their personal computers (PCs) and related products such as monitors, data terminals, keyboards, multi–user systems, etc. are manufactured. But soon it turned out that this FMS was designed for a lower amount of production as actually took place. As a result several components in this system turned out to be bottlenecks in the flow of the material and the management was looking for possible ways to improve the production process without having to carry out expensive technical changes.

After a thorough analysis of the system several bottleneck components have been detected and the question of deactivating them has often led to Combinatorial Optimization problems. In this thesis we focus on one of these problems, namely the question of sequencing transportation tasks (jobs) of the stacker crane of an automatic storage system such that the overall time needed for the unloaded moves is minimized. We developed and implemented an **optimization tool** that is now running since more than two years in everyday production on six automatic storage systems at SNI. Its use under heavy load conditions resulted in a reduction of approx. 30% of the time needed for unloaded moves.

As there is a close relation between the application and the theoretical studies, we do not restrict ourselves to describe only the interesting mathematical topics. We also outline the typical Operations Research questions that have to be addressed, i.e., we discuss modeling questions and briefly describe an implemented simulation package.

For a given set of jobs the problem of minimizing the overall time for the unloaded moves of the stacker crane naturally leads to an **asymmetric Hamiltonian path problem (AHPP)**. But in contrast to this well known Combinatorial Optimization problem, the problem we consider is of a dynamic nature. Suppose the stacker crane has a certain set of jobs to be performed and that we calculated a sequence minimizing the time for the unloaded moves. As soon as a new job is generated or a job is cancelled, this sequence has to be reoptimized.

This means that a sequence that was optimal at a certain point in time is not necessarily performed in the calculated order, and that it need not be optimal for the overall time period. Problems with the property that not all input data (here the generated and cancelled jobs) is known in advance is said to be an on–line problem. If in contrast all data is known, we say that it is an off–line problem. Thus, we are actually confronted with an **on–line asymmetric Hamiltonian path problem**.

Up to the present, not much attention has been paid to the study of on–line problems. In general, an algorithm for an on–line AHPP will produce a non–optimal solution. Therefore, a good measure for the quality of the found solutions is needed. So far, the concept of **competitiveness** has been applied to compare the computational performance of on–line algorithms, but leading to unduly pessimistic results.

As in the FMS that we consider, generated jobs cannot wait an unlimited amount of time until they are performed, there exists an implicit due date for all jobs. This allows us to derive lower bounds on the value obtained by an optimal on–line strategy by analyzing two off–line problems: the **asymmetric Hamiltonian path problem with time windows (AHPPTW)** and the **asymmetric Hamiltonian path problem with precedence constraints**, also known as sequential ordering problem (SOP). We show that the value of an optimal solution to one of these problems is a lower bound to the value obtained by an optimal on–line strategy. Thus, by solving these off–line problems we derive instance and application dependent lower bounds to the on–line AHPP.

The SOP and AHPPTW are hard problems from the point of view of complexity theory. Therefore, we cannot expect to find efficient algorithms solving all problem instances to optimality. But lower bounds to the value of the optimal solutions to these problem instances are lower bounds for the on–line AHPP as well. For the AHPP, which is trivially equivalent to the ATSP, **polyhedral approaches** have turned out to be the appropriate method to solve problem instances to optimality or to derive good lower bounds to the value of the optimal solution. The key concept is to associate to each feasible solution a point in an Euclidean space and to describe the polyhedron defined as the convex hull of these points via linear inequalities and equations. If the combinatorial problem can be stated as a minimization, resp. maximization, problem over this polytope we can apply cutting plane and/or branch&bound methods to solve it. It is known that such descriptions exist, but it is unlikely to find complete descriptions of the polytopes associated to the AHPPTW and SOP. Even a partial description may already contain an exponential number of inequalities.

For the TSP and many other Combinatorial Optimization problems it has been shown that even a partial description can be exploited to formulate an algorithm for the solution of these problems. The so–called **branch&cut algorithm** is a variant of the well known branch&bound algorithm where the lower bound calculations are performed by means of a cutting plane algorithm, exploiting the knowledge about the facial structure of the polyhedron.

So far, additional side constraints to the TSP and ATSP have been considered very seldomly in the literature. Due to these additional constraints (in form of precedences, time windows) the problem gets substantially more difficult. For example, it is already $\mathcal{NP}$–complete to find a feasible solution to the TSP with time windows.

By the time of writing, no implementations are known that solve both the SOP and the AHPPTW to optimality by means of a branch&cut algorithm. So far, these problems have been mainly attacked by means of heuristics or by applying concepts of implicit enumeration, such as dynamic programming or branch&bound. We show that from a computational point of view (quality of the found solutions, variety of problem instances that can be solved, etc.)

branch&cut algorithms outperform or are at least comparable to the algorithms published so far.

Outline of the thesis

This thesis contains 6 chapters and is divided into three parts. After an introductory part (Chapters 1–2) we address some modeling and Operations Research questions (Chapters 3–4). In the last part (Chapters 5–6) interesting mathematical problems are discussed in more detail.

In Chapter 1 some basic mathematical definitions and results from graph theory, complexity theory, and polyhedral theory are surveyed. Polyhedral results for the asymmetric travelling salesman problem are summarized as far as they are of interest for this thesis. Shortly Flexible Manufacturing Systems are introduced and the role of Discrete Mathematics in Flexible Manufacturing is discussed. Finally, on–line problems and on–line algorithms as far as they are discussed in the literature are surveyed. This chapter is not meant to be comprehensive, but to provide the reader with the basic concepts and notations.

Chapter 2 contains a description of the considered Flexible Manufacturing System at SNI and addresses several other optimization approaches that are not discussed in this thesis.

Chapter 3 is dedicated to a description of the concepts of event–oriented simulation and a description of the implemented simulation package. This simulation program turned out to be an important tool to study the on–line behaviour of the developed optimization packages.

In Chapter 4 we describe in detail the modeling of the problem of minimizing the unloaded travel time of the stacker crane and perform a computational study on several on–line heuristics. Lower bounds on the value obtained by using a best possible on–line strategy are derived.

In Chapters 5 we analyze the asymmetric Hamiltonian path problem with precedence constraints. Classes of inequalities are summarized that are known in the literature. Furthermore, several new classes of valid inequalities are given that are strengthenings of known ATSP facet defining inequalities. A branch&cut algorithm is described that solves real–world problem instances of up to 100 nodes and a varying number of precedences to optimality. One result of the computational experiments was that the obtained lower bound is relatively good but that additional effort has to be invested in the design of better heuristics. Computational results on the unconstrained ATSP are given as well. It was possible to solve all six TSPLIB instances to optimality in a short amount of computing time.

Chapter 6 is dedicated to the discussion of the asymmetric Hamiltonian path problem with time windows. In contrast to many other publications we are not interested in minimizing the overall completion time but in minimizing the total sum of intermediate set–up costs. Therefore, we are not restricted to use the well known model based on a generalization of the Miller–Tucker–Zemlin subtour elimination constraints but can define a TSP–like model. Here the time window restrictions are modeled by **infeasible path constraints**, forbidding paths that violate the given time windows. Several classes of inequalities of this type are derived. Furthermore, known classes of inequalities are summarized. The implementation of a branch&cut algorithm is described. The two models are compared from a computational point of view on real–life problem instances of up to 50 nodes. This comparison shows that the TSP–like model seems to be superior.
We close by giving some concluding remarks and sketching possible future research topics.

Chapter 1

Preliminaries

1.1 Notation

For completeness we summarize some basic definitions and results from graph theory, linear algebra and polyhedral theory that will be of interest in this thesis. The aim is not to be comprehensive but to provide the basic concepts and notations. Most of the definitions in this section can be found in any introductory textbook. For a more detailed treatment of these topics we refer, e.g., to Bondy and Murty [BM76], Chartrand and Lesniak [CL86], Schrijver [Sch86], and Nemhauser and Wolsey [NW88].

1.1.1 Graph theory

A **digraph** (or **directed graph**) $D = (V, A)$ consists of a nonempty, finite node set V and a finite set of arcs A. An **arc** $a \in A$ is an ordered pair of nodes $i, j \in V$ and is denoted by $a = (i, j)$. We say that i is the **tail** and j is the **head** of arc $a = (i, j)$, and i is said to be the **predecessor** of j, whereas j is the **successor** of i. The arc $a = (i, j)$ is said to be directed from i to j, incident from i and incident to j. In the sequel we only consider loop free digraphs, i.e., $i \neq j$ for all $a \in A$.

A digraph $D = (V, A)$ is called **complete**, if each two nodes $i, j \in V, i \neq j$ are connected by the arcs $(i, j) \in A$ and $(j, i) \in A$. In the sequel a complete digraph on $n := |V|$ nodes is denoted by $D_n = (V, A_n)$. A digraph $D = (V, A)$ is called **transitively closed**, if for all $i, j, k \in V, i \neq j \neq k \neq i$, $(i, j) \in A$ and $(j, k) \in A$ implies $(i, k) \in A$.

An **undirected graph** (or **graph**) $G = (V, E)$ consists of a nonempty, finite set of nodes V and a finite set of edges E. To each **edge** $e \in E$ two nodes $u, v \in V$ are associated and it is denoted with $e = uv$. In the sequel we only consider loop free graphs, i.e., $u \neq v$ for all $e \in E$. A graph $G = (V, E)$ is called **bipartite**, if V can be separated into two sets V_1 and V_2, such that $V = V_1 \cup V_2$, $V_1 \cap V_2 = \emptyset$, and for all $e = uv$ $u \in V_i$, $v \in V_j$, $i \neq j$ holds.

In this thesis we will mainly work on directed graphs. If not stated explicitly, the definitions listed below translate directly to undirected graphs.

Let $D_1 = (V_1, A_1)$ and $D_2 = (V_2, A_2)$ be two digraphs, such that $V_2 \subseteq V_1$ and $A_2 \subseteq A_1$, then D_2 is called a **subdigraph** of D_1.

Suppose we are given a digraph $D = (V, A)$. If W is a subset of V, then

$$A(W) := \{(i, j) \in A \mid i, j \in W\}$$

denotes the set of all edges with tail and head in W. For given node sets $U, W \subset V$ with $U \cap W = \emptyset$,

$$(U : W) := \{(i, j) \in A \mid i \in U, j \in W\},$$

denotes the set of arcs with tail in U and head in W. To simplify notation, we write $(W : j)$ and $(j : W)$ instead of $(W : \{j\})$ and $(\{j\} : W)$. If $U = \emptyset$ or $W = \emptyset$, then $(U : W) = \emptyset$.
Given a node set $W \subset V, W \neq \emptyset$, we set

$$\begin{array}{lcl} \delta^-(W) & := & \{(i, j) \in A \mid i \in V \setminus W, j \in W\}, \\ \delta^+(W) & := & \{(i, j) \in A \mid i \in W, j \in V \setminus W\}, \\ \delta(W) & := & \delta^-(W) \cup \delta^+(W). \end{array}$$

The arc set $\delta(W)$ is called a **cut**. We know that $\delta^-(W) = \delta^+(V \setminus W)$. To simplify notation we write $\delta^-(v), \delta^+(v), \delta(v)$, instead of $\delta^-(\{v\}), \delta^+(\{v\}), \delta(\{v\})$. The numbers $|\delta^-(v)|, |\delta^+(v)|$, and $|\delta(v)|$ are called the **indegree**, **outdegree**, and **degree** of node $v \in V$. A node $v \in V$ is called **isolated**, if it has degree zero, i.e., $|\delta(v)| = 0$.

If in a given digraph $D = (V, A)$ arc weights c_a are associated to each $a \in A$, we say that D is a **weighted digraph**, and the weight of an arc set $B \subseteq A$ is given by

$$c(B) := \sum_{a \in B} c_a.$$

Suppose we are given a digraph $D = (V, A)$, then the arc set

$$P = \{(v_1, v_2), (v_2, v_3), .., (v_{k-1}, v_k)\}, k \geq 2, v_i \in V \ \forall \ i = 1, ..., k$$

is called a **walk**, or more precisely a $[v_1, v_k]$–**walk**. v_1 is the **starting node**, v_k is the **end node** of the walk. The **length** of the walk equals the number of arcs in the path, is denoted by $|P|$ and equals $k - 1$. Each arc that is not in the walk but connects two nodes of the walk is called a **chord**. A walk consisting of nodes that are all pairwise different, i.e., $v_i \neq v_j$ for all $i \neq j$, is called a **path**. A directed path of length $|V| - 1$ is called a directed **Hamiltonian path**. A walk

$$C = \{(v_1, v_2), (v_2, v_3), ..., (v_{k-1}, v_k), (v_k, v_1)\}$$

of length k with a starting and end node that coincide is called a **circuit**. Without risking confusion we will also use the expression **cycle** instead of circuit. A cycle is called **simple**, if all nodes are pairwise different, i.e., $v_i \neq v_j$ for all $i, j = 1, ..., k, i \neq j$. A simple cycle of length $|V|$ is called a **Hamiltonian cycle** or a **tour**. A simple cycle of length less than $|V|$ is called a **subtour**. We say that a digraph $D = (V, A)$ is **acyclic**, if its arc set A does not contain a cycle.
For notational convenience we often abbreviate the path

$$P = \{(v_1, v_2), (v_2, v_3), ..., (v_{k-1}, v_k)\}$$

by stating only the nodes forming the path, i.e.,

$$P = (v_1, v_2, v_3, ..., v_{k-1}, v_k).$$

But if doing so, we will always consider P to be a set of arcs.

Given two paths $P_1 = (v_1, v_2, ..., v_k)$ and $P_2 = (u_1, u_2, ..., u_m)$, we say that P_2 is a **subpath** of P_1, if there exists a $l, 0 \leq l \leq k - m$, such that $v_{i+l} = u_i$ for all $i = 1, ..., m$.

Two nodes $u, v \in G = (V, E)$ are said to be **connected** , if there exists an $[u, v]$–path in G. If each pair of nodes $u, v \in V$ is connected, we say that G is connected. A digraph $D = (V, A)$ is said to be **connected**, if for each pair $i, j \in V$ either a (i, j)–path or a (j, i)–path exists. If both paths exist, D is said to be **strongly connected**. The **components** of D are the maximal strongly connected subdigraphs of D with respect to arc inclusion. A node $v \in V$ is called **articulation node**, if $G - v$ consists of more components than G.

A **clique** in a digraph $D = (V, A)$ is a node set $V_C \subseteq V$ such that $D_C = (V_C, A(V_C))$ is a complete subdigraph of D.

For a given graph $G = (V, E)$ a **tree** is a connected edge set of G that contains no cycle. The tree is said to be **spanning** it it contains all nodes of the graph.
For a given digraph $D = (V, A)$ a **branching** B is an acyclic arc set, such that every node in D is the endpoint of at most one arc in B. A connected branching is called an **arborescence**. Note, that in an arborescence there exists one node r of indegree $|\delta^-(r)| = 0$, called the **root** of the arborescence, from which there exists a unique path to every other node in the arborescence.

1.1.2 Polyhedral Theory

The vector $y \in \mathbb{R}^n$ is called a **linear combination** of the vectors $x_1, ..., x_m \in \mathbb{R}^n$, if there exists $a_1, ..., a_m \in \mathbb{R}$ such that $y = \sum_{i=1}^m a_i x_i$. If, in addition, the a_i satisfy

- $\sum_{i=1}^m a_i = 1$, then y is said to be an **affine combination**,
- $\sum_{i=1}^m a_i = 1$ and $a_i \geq 0, i = 1, ..., m$, then y is said to be a **convex combination**.

If $S \subseteq \mathbb{R}^n$, then the **linear hull** of S is defined as

$$lin(S) := \{y \in \mathbb{R}^n \mid \exists m \in \mathbb{N}, a_1, ..., a_m \in \mathbb{R}, x_1, ..., x_m \in S, \text{ such that } y = \sum_{i=1}^m a_i x_i\}.$$

Similarly, the **convex hull** $conv(S)$ and **affine hull** *aff(S)* can be defined.

A nonempty set $S \subseteq \mathbb{R}^n$ is called **linearly independent**, if for every subset $\{x_1, ..., x_k\} \subseteq S$ the equation $\sum_{i=1}^k a_i x_i = 0$ implies $a_i = 0$ for all $i = 1, ..., k$. The set S is called **affinely independent**, if for every subset $\{x_1, ..., x_k\} \subseteq S$ the equations $\sum_{i=1}^k a_i x_i = 0$ and $\sum_{i=1}^k a_i = 0$ imply $a_i = 0$ for all $i = 1, ..., k$. Otherwise, S is called linearly (affinely) dependent. Every linear (resp. affinely) independent set in $\mathbb{R}^n$ contains at most n (resp. $n + 1$) elements.

The **rank (affine rank)** of a set $S \in \mathbb{R}^n$ is the cardinality of the largest linearly (affinely) independent subset of S, and the dimension of S, denoted by $dim(S)$, is the affine rank minus one. A set $S \subseteq \mathbb{R}^n$ is called full–dimensional, if $dim(S) = n$.

If $0 \notin$ *aff(S)*, i.e., S is contained in the hyperplane $\{x \mid ax = a_0\}$ with $a_0 \neq 0$, then $dim(S)$ is the maximum cardinality of a linearly independent set in S minus one.

The **rank of a matrix** is the rank of the set of the column vectors of the matrix. This is the same as the rank of the set of its row vectors. An (m, n)–matrix is said to have full rank, if its rank equals $min\{m, n\}$.

If $S \subseteq \mathbb{R}^n$, then $Ax = b$ is called a **minimal equation system** for S, if *aff(S)* $= \{x \in R^n \mid Ax = b\}$ and A has full rank.

To simplify notation, we write an inequality $a_1x_1 + a_2x_2 + ... + a_nx_n \leq a_0$ in the form $ax \leq a_0$ instead of $a^Tx \leq a_0$, i.e., a is considered to be a row vector, x to be a column vector. For $a, x \in \mathbb{R}^n, a_0 \in \mathbb{R}$ an **inequality** $ax \leq a_0$ defines a **halfspace** in $\mathbb{R}^n$, i.e., the set $\{x \in \mathbb{R}^n \mid ax \leq a_0\}$, and a **hyperplane** in $\mathbb{R}^n$, i.e., the set $\{x \in \mathbb{R}^n \mid ax = a_0\}$.

A **polyhedron** is the intersection of finitely many halfspaces, i.e., every polyhedron P can be represented in the form $\{x \in \mathbb{R}^n \mid Ax \leq b\}$. A bounded polyhedron is called a **polytope**. In analogy to the definition given above, the dimension of the polyhedron is the maximum number of affinely independent points in the polyhedron minus one. A polyhedron $P \subseteq \mathbb{R}^n$ is said to be of full dimension, if $dim(P) = n$.

Given any polyhedron P, we say that an inequality $ax \leq a_0$ is **valid** for P, if it holds for all points $x' \in P$. If $ax \leq a_0$ is valid for P, then the set $\{x \in P | ax = a_0\}$ determines a **face** of P. It defines a **facet** of P, if it is valid and if there are $dim(P)$ affinely independent points in the hyperplane induced by $ax \leq a_0$. A valid inequality $ax \leq a_0$ for P is called **supporting**, if $P \cap \{x \in \mathbb{R}^n \mid ax = a_0\} \neq \emptyset$.

Given two inequalities $ax \leq a_0$ and $bx \leq b_0$ we say that $ax \leq a_0$ is **dominated** by $bx \leq b_0$, if for all $x' \in P$ with $ax' = a_0$ we also have that $bx' = b_0$. The two inequalities are **equivalent** with respect to P, if one can be obtained by multiplying the other with a positive scalar and adding a linear combination of the equality system for P.

In this thesis inequalities are often "visualized" by means of a digraph. Therefore, let $ax \leq a_0$ be any inequality, with $a, x \in \mathbb{R}^{|A|}$. The **support graph** of this inequality on a node set $S \subseteq V$ is given by $G_a(S) = (S, A_a)$, where $A_a := \{(i, j) \in A(S) : a_{ij} \neq 0\}$.

1.1.3 Complexity theory

We briefly review in an informal manner the basic concepts of complexity theory, as far as they are used in this thesis. For a thorough description of the concepts and a comprehensive survey we refer to Garey and Johnson [GJ79].

We distinguish between two types of problems:

- **decision problems** requiring a *yes* or *no* answer and
- **optimization problems**, that aim to detect a solution minimizing (or maximizing) a certain objective function.

A decision problem is said to be in the class $\mathcal{P}$, if there exists an algorithm with polynomial time complexity that solves this problem. The class $\mathcal{NP}$ contains all problems such that any instance of them that has a *yes* answer can be certified in polynomial time. Obviously, $\mathcal{P} \subset \mathcal{NP}$. It is conjectured and commonly accepted that this inclusion is strict, i.e., $\mathcal{P} \neq \mathcal{NP}$.

An important subclass in $\mathcal{NP}$ are the so–called $\mathcal{NP}$–**complete** problems, i.e., problems such that every other problem in this class can be transformed to it in polynomial time. The problems of this class are considered to be the hardest problems, since if for one of them a polynomial time algorithm is detected, it is shown that $\mathcal{P} = \mathcal{NP}$.

Note, that Combinatorial Optimization problems can be transformed into decision problems. An optimization problem is said to be $\mathcal{NP}$–**hard**, if it has the property that the existence of a polynomial time algorithm for the solution of the associated decision problem implies the solvability of an $\mathcal{NP}$–complete problem.

1.2 Branch&Cut Algorithms

A combinatorial optimization problem $(E, \mathcal{I}, c)$ can be described as follows:
We are given a finite set E (e.g., the arcs of a digraph), a set of feasible solutions $\mathcal{I} \subseteq 2^{|E|}$, and a cost function $c : E \to \mathbb{K}$, assigning a certain weight to each $e \in E$. The "value" of each subset $F \subseteq E$ is given by $c(F) := \sum_{e \in F} c(e)$. The problem consists of finding an element $I^* \in \mathcal{I}$, such that $c(I^*)$ is minimal (or maximal). For the sake of simplicity we always refer to minimization problems in the sequel. For maximization problems the statements are derived in an analogous way.

Although we considered the ground set E to be finite, the "interesting" combinatorial optimization problems have a number of feasible solutions that is exponential in the size n of the ground set E, (e.g., 2^n or $n!$). Therefore, it is impossible to simply enumerate all feasible solutions. The aim of Combinatorial Optimization is to design algorithms that are faster than simple enumeration techniques.

In the 60s and 70s Polyhedral Theory developed as a theoretical tool to describe combinatorial optimization problems. In the end of the 70s and in the beginning of the 80s it turned out that this approach can successfully be used to solve these problems, namely by the means of **cutting plane algorithms**. Their basic principles are described now.

To apply linear programming techniques to solve combinatorial optimization problems we assign variables x_e to each $e \in E$. The variable x_e is considered to be a component of the vector $x \in \mathbb{K}^{|E|}$. In order to simplify notation we write from now on $\mathbb{K}^E$ instead of $\mathbb{K}^{|E|}$. To each subset $F \subseteq E$ we assign a so–called **incidence vector** $\chi^F \in \mathbb{K}^E, \chi^F = (\chi^F_e)_{e \in E}$ that is defined as

$$\chi^F_e := \begin{cases} 1, & \text{if } e \in F, \\ 0, & \text{else.} \end{cases}$$

Thus, to each $F \subseteq E$ we uniquely associate a 0/1–vector χ^F, and vice versa. We define

$$P_{\mathcal{I}} := conv\{\chi^I \in \mathbb{K}^E \mid I \in \mathcal{I}\} \tag{1.2.1}$$

to be the convex hull of the incidence vectors of the feasible solutions of the considered combinatorial optimization problem. Hence, $P_{\mathcal{I}}$ is a polytope and the vertices of $P_{\mathcal{I}}$ correspond to the incidence vectors of the feasible solutions $I \in \mathcal{I}$. If the function $c : E \to \mathbb{K}$ is considered to be a vector $c \in \mathbb{K}^E$, every optimal vertex of the linear program

$$min\{c^T x \mid x \in P_{\mathcal{I}}\} \tag{1.2.2}$$

is the incidence vector of an optimal solution to the considered combinatorial optimization problem $(E, \mathcal{I}, c)$, and vice versa.

The program (1.2.2) is of a form that is not suitable to be attacked by linear programming techniques. Therefore, it is necessary to find a system of equations and inequalities, such that

$$P_{\mathcal{I}} = \{x \in \mathbb{K}^E \mid Dx = d, Ax \leq b\}.$$

From a theoretical point of view this is possible, but the problem of finding such a description is not always easily solvable, as, e.g., exponentially many inequalities might be necessary to describe $P_{\mathcal{I}}$. Thus, for algorithmic purposes it is often sufficient, to find descriptions that are "sufficiently good enough", i.e.,

$$P_{\mathcal{I}} \subseteq \{x \in \mathbb{K}^E \mid D'x = d', A'x \leq b'\}.$$

If these equations and inequalities are selected properly, solutions to the linear program

$$min\{c^T x \mid D'x = d', A'x \leq b'\}$$

often lead to optimal solutions or at least to good lower bounds to the value of the optimal solution of the considered combinatorial optimization problem.

Polyhedral Theory aims to describe the polytope $P_{\mathcal{I}}$ (1.2.1) as good as possible by nonredundant equations and inequalities. Nonredundant means that the system does not contain any equation or inequality that might be left out without changing the set of feasible solutions. Complete descriptions are known for many polytopes that correspond to combinatorial optimization problems that are solvable in polynomial time.

Very often the polytope $P_{\mathcal{I}}$ might be described as

$$P_{\mathcal{I}} = conv\{x \in \mathbb{K}^E \mid Dx = d, Ax \leq b, x_i \in \{0,1\}, i = 1, ...|E|\},$$

i.e., each vertex has just 0/1–components. The typical way of relaxing the problem is to consider a polytope P that is obtained by transforming the conditions $x_i \in \{0,1\}$ into bounds on the variables, i.e.,

$$P = conv\{x \in \mathbb{K}^E \mid Dx = d, Ax \leq b, 0 \leq x_i \leq 1, i = 1, ...|E|\}.$$

This polytope satisfies

$$P_{\mathcal{I}} \subseteq P$$

and therefore we know

$$min\{c^T x \mid x \in P\} \leq min\{c^T x \mid x \in P_{\mathcal{I}}\}$$

Furthermore, we know that each extreme point of $P_{\mathcal{I}}$ is an extreme point of P. If

$$min\{c^T x \mid x \in P\} \tag{1.2.3}$$

leads to a 0/1–solution, we have found a solution to the combinatorial optimization problem $(E, \mathcal{I}, c)$. If not, we have to find inequalities that are valid with respect to $P_{\mathcal{I}}$ and that are violated by the actual solution of (1.2.3). Here the knowledge of facet defining inequalities is very important, as these inequalities are supposed to provide the "deepest cut" into the polytope.

The problem of finding such a valid violated inequality is known as the **separation problem**. We know state this more formally.

Separation problem:
Given a polytope $P_{\mathcal{I}} \subseteq \mathbb{K}^E$ and $y \in \mathbb{K}^E$.
Decide whether $y \in \mathbb{K}^E$, and if not determine a violated inequality $a^T x \leq a_0$ with $a^T y > a_0$.
□

Grötschel, Lovász, and Schrijver [GLS88] showed that the optimization problem can be solved in polynomial time, if the separation problem can be solved in polynomial time.

Suppose we are given a class of inequalities $Ax \leq b$. A procedure that determines for a given point $y \in \mathbb{K}^E$ such a violated valid inequality, i.e., $A_i y > b_i$, is called a **separation routine** for the inequality system $Ax \leq b$. We have to distinguish between exact and heuristic

separation routines. A separation routine for a class of inequalities $Ax \leq b$ and a given point $y \in \mathbb{K}^E$ is called **exact**, if it either finds a violated inequality $A_i y > b_i$ or gives the answer that no such inequality exists. If the routine might just find a violated inequality, but does not terminate with the answer that no such inequality exists, it is called to be a **heuristic separation procedure**. Thus, it might happen that a heuristic separation procedure does not detect a violated inequality of a certain class, although there exists such an inequality.

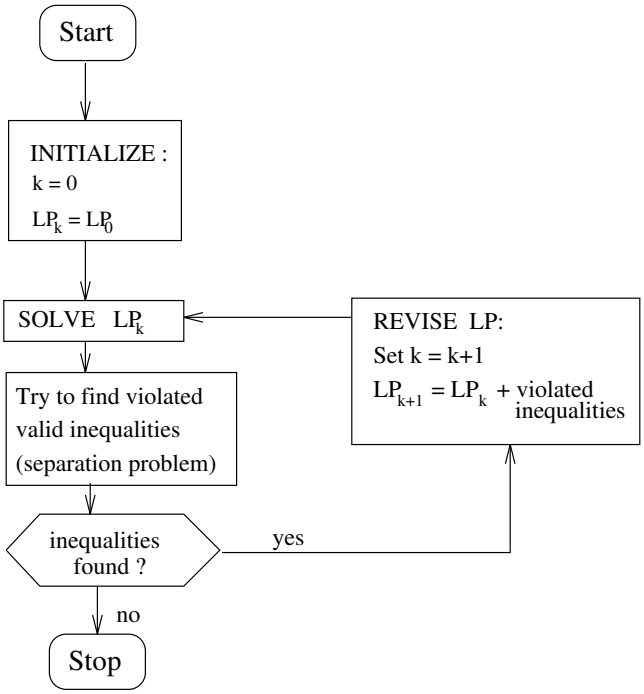


Figure 1.2.1 Flowchart of a cutting plane algorithm

Now we are able to give the standard version of a **cutting plane algorithm** (see Figure 1.2.1). We solve an initial linear program, try to find violated valid inequalities, add them to the LP, resolve it, etc. This is certainly a very brief outline of a cutting plane algorithm and a lot of details that are important for an efficient implementation are left out. In case that this approach results in a 0/1–solution we have found an optimal solution to our problem. If this is not the case, we can either stop and output the value of the current LP as a lower bound on the value of the optimal solution or embed this cutting plane algorithm in a enumerative branch&bound framework. Therefore, we create two new subproblems by fixing a certain variable either to its lower bound 0 or its upper bound 1, and proceed with applying the cutting plane algorithm to each of the subproblems, and so on. If doing so, this approach is called a **branch&cut algorithm**.

Figure 1.2.2 gives a simplified flowchart of a branch&cut algorithm, as it was given in [JRT92]. As already mentioned, the branch&cut algorithm is a variant of the well known branch&bound algorithm where the lower bounds are obtained by cutting plane algorithms. The main task of the branch&cut algorithm is to maintain a list of active subproblems. At the beginning this list is initialized by the problem itself. At each major iteration of the algorithm a global upper bound (feasible solution) and a local lower bound is calculated.

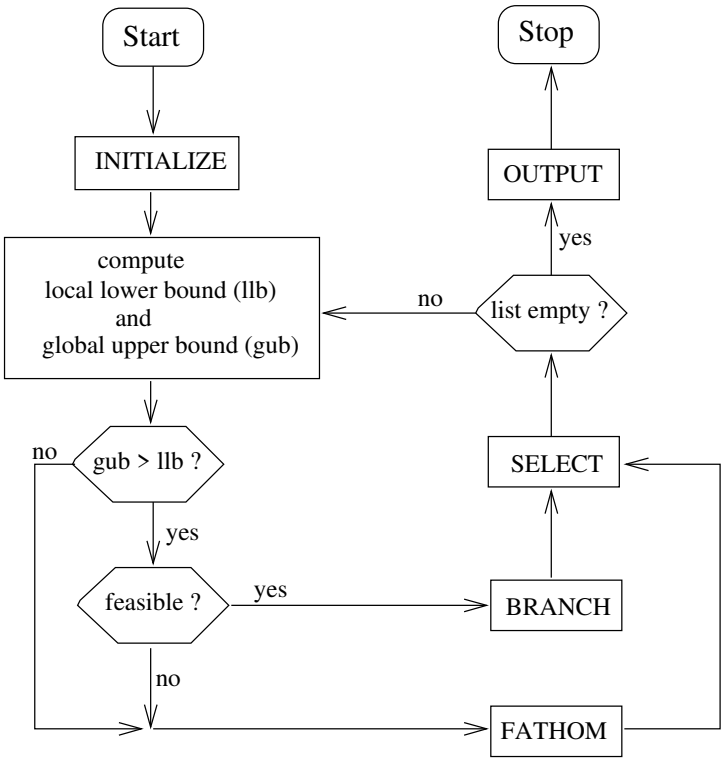


Figure 1.2.2 Flowchart of a branch&cut algorithm

If the local lower bound equals or exceeds the global upper bound, we know that we can fathom that subproblem (node), as in that branch of the branch&cut tree we will never find a feasible solution better than the one we already have. We than select another subproblem and continue until there are no subproblems left (list is empty). In case that the local lower bound is lower than gub, we check if the subproblem contains a feasible solution. If no, we can fathom this node, else we create two new subproblems, add them to the list of all subproblems and proceed.

This is just a brief sketch of the general concept of branch&cut algorithms. There are a lot of details that have to be clarified for an efficient implementation of such an algorithm. See [NW88] for an introduction to linear programming and [PR91, JRT92] for more details concerning branch&cut algorithms.

1.3 The asymmetric travelling salesman problem

In Chapters 5 and 6 we study variants of the **asymmetric Hamiltonian path problem (AHPP)** where additional side constraints are present. In graph theoretical terminology the classical AHPP can be defined as follows: Given a complete weighted digraph $D_n = (V, A_n)$ with arc weights c_{ij} for all arcs $(i, j) \in A_n$, find a Hamiltonian path with minimal cost through the nodes in V. The AHPP is equivalent to one of the most studied problems in Combinatorial Optimization: the **asymmetric travelling salesman problem (ATSP)**. In the ATSP one is interested in finding a minimum cost Hamiltonian circuit (tour) through D_n.

The AHPP on n nodes can easily be transformed into an ATSP on $n+1$ nodes as follows. Add a node v_{n+1}, arcs (v_i, v_{n+1}) and (v_{n+1}, v_i) with cost coefficients 0 for all $v_i \in V$, and calculate a cost minimal tour in the digraph $D_{n+1} = (V \cup \{v_{n+1}\}, A_{n+1})$. Now, let s be a vector, such that $s(v_i)$ denotes the successor of $v_i \in V \cup \{v_{n+1}\}$ in the optimal tour. Let $v_k \in V$ be the node, such that $s(v_k) = v_{n+1}$. Thus, with

$$C = (v_{n+1}, v_{s(v_{n+1})}, v_{s(s(v_{n+1}))}, ..., v_k)$$

we denote an optimal tour in $D'_{n+1} = (V \cup \{n+1\}, A_{n+1})$. Obviously, the path

$$P = (v_{s(v_{n+1})}, v_{s(s(v_{n+1}))}, ..., v_k)$$

corresponds to an optimal Hamiltonian path in $D_n = (V, A_n)$.

As we are interested in solving the AHPP with side constraints by polyhedral methods, results about the facial structure of the corresponding polytopes are of particular interest. So far, the AHP–polytope was not considered explicitly in the literature, but most of the published polyhedral results can be found on the ATS–polytope. Therefore, we restrict ourselves in citing the results for the ATSP.

The following binary linear program models the ATSP

$$\begin{array}{llll} & \min c^T x & & \\ \text{s.t. } (1) & x(\delta^-(i)) & = 1 & \forall\, i \in V \\ (2) & x(\delta^+(i)) & = 1 & \forall\, i \in V \\ (3) & x(A(W)) & \leq |W| - 1 & \forall\, W \subset V,\ 2 \leq |W| \\ (4) & x_{ij} & \in \{0,1\} & \forall\, (i,j) \in A_n \end{array} \tag{1.3.4}$$

We follow the notation introduced in Grötschel and Padberg [GP85b] and denote by

$$P_T^n := conv\{x \in \{0,1\}^{A_n} \mid x \text{ satisfies } (1.3.4)(1) - (3)\}$$

the **asymmetric travelling salesman polytope**, i.e., the convex hull of the incidence vectors of all $(n-1)!$ tours in the complete digraph $D_n = (V, A_n)$. A common approach to obtain results about the facial structure of P_T^n is to analyze its **monotone relaxation**. Here it is assumed that each node is visited at most once, i.e., the equations (1.3.4)(1) and (2) are relaxed to

$$\begin{array}{llll} (1') & x(\delta^-(i)) & \leq 1 & \forall\, i \in V, \\ (2') & x(\delta^+(i)) & \leq 1 & \forall\, i \in V. \end{array} \tag{1.3.5}$$

The monotone ATS–polytope is defined by

$$\tilde{P}_T^n := conv\{x \in \{0,1\}^{A_n} \mid x \text{ satisfies } (1.3.5)(1')(2') \text{ and } (1.3.4)(3)\}$$

The symmetric travelling salesman polytope Q_T^n can be defined in an analogous way as P_T^n. The polytopes P_T^n and $\tilde{P}_T^n$ have been studied, among others, by Grötschel [Grö77], Grötschel and Padberg [GP77], Balas [Bal89], Fischetti [Fis89, Fis91, Fis92], Balas and Fischetti [BF92, BF93a], Chopra and Rinaldi [CR90], and Queyranne and Wang [QW94]. In this section we review some of the known polyhedral results that will be of interest in this thesis.
It is well known that

$$dim(P_T^n) = n \cdot (n-1) - 2n + 1$$

and

$$dim(\tilde{P}_T^n) = n \cdot (n-1)$$

The description of P_T^n by means of equations and inequalities caused the interest of researchers for a long period of time. The first results about its facial structure are due to Heller and Kuhn. In 1953 Heller [Hel53] published a (partial) description of P_T^5 consisting of 9 equations and 215 inequalities. In 1955 Kuhn [Kuh55] presented two more classes of inequalities and stated that his description of 9 equations and 390 inequalities is complete and nonredundant. In 1989 Bartels and Bartels [BB89] calculated by means of a computer program a complete and nonredundant description by P_T^5 leading to the same result. Euler and Le Verge [EL92] calculated complete descriptions of $\tilde{P}_T^5$ and P_T^6. The number of inequalities necessary to describe these polytopes is summarized in the following table.

	# equations	# facets	# classes
P_T^5	9	390	6
$\tilde{P}_T^6$	0	7615	51
P_T^6	11	319015	287

Symmetric inequalities

An inequality $ax \leq a_0$ is called **symmetric**, if $a_{ij} = a_{ji}$ for all $(i,j) \in A$, otherwise the inequality is called **asymmetric**.

The symmetric TSP is equivalent to the ATSP, as it can be transformed into an ATSP, and vice versa. An obvious question is, if there is a relation between facet defining inequalities for P_T^n and Q_T^n. Given a valid TSP–inequality $by \leq b_0$, one can derive a symmetric ATSP–inequality $ax \leq a_0$ by replacing y_e by $x_{ij} + x_{ji}$ and setting $a_0 = b_0, a_{ij} = a_{ji} = b_e$ for all $e \in E, e = ij$. Conversely, every symmetric ATSP–inequality corresponds to an TSP–inequality. Examples for symmetric ATSP inequalities are, among others,

- subtour elimination constraints,
- 2–matching constraints,
- comb–inequalities,
- clique tree inequalities.

Unfortunately, the result that a certain facet defining inequality for Q_T^n is facet defining for P_T^n as well, cannot be derived automatically. Suppose the inequality $ay \leq a_0$ is facet defining for Q_T^n. Then there exist $\frac{n \cdot (n-1)}{2} - n$ linearly independent tours satisfying this inequality with equality. By directing the edges of each tour in the two possible directions we obtain $n \cdot (n-1) - 2n$ directed tours. It is not known under which conditions these ATSP solutions are linearly independent. Moreover, if they are linearly independent this is not sufficient for

proving that $ax \leq a_0$ is facet defining for P_T^n, as $dim(P_T^n) = n \cdot (n-1) - 2n + 1$. Thus, one more tour has to be constructed that is linearly independent to the others.

Grötschel [Grö77] (see also [GP85b]) showed that the **subtour elimination constraint**

$$x(A(W)) \leq |W| - 1$$

defines a facet of $P_T^n, n \geq 5$, if $2 \leq |W| \leq n-2$, and for $\tilde{P}_T^n, n \geq 3$, if $2 \leq |W| \leq n-2$. Fischetti showed that

- comb inequalities define facets of P_T^n, except when $n = 6$, and $\tilde{P}_T^n$ [Fis91],
- the **clique tree inequalities**, which have been introduced by Grötschel and Pulleyblank [GP86], define facets of P_T^n and $\tilde{P}_T^n$, $n \geq 6$ [Fis89].

Asymmetric inequalities

For the symmetric and asymmetric travelling salesman problem the cycle inequality

$$x(C) \leq |C| - 1$$

can be lifted to be facet defining for the polytopes P_T^n and $\tilde{P}_T^n$. For the symmetric case no new inequalities are obtained, as the lifted cycle inequalities are the subtour elimination constraints.

But for the asymmetric case new facet defining inequalities can be found. If $C = (i_1, ..., i_k)$ is a directed cycle of length k, these inequalities are the so–called D_k–inequalities (see e.g., [GP85b]). Dependent on the order in that the arcs are lifted different inequalities are obtained. Figure 1.3.3 gives all inequalities derived from the sequential lifting of cycles of length 3 and 4. The bold lines represent variables with coefficient 2 and the numbers on the arcs indicate the order in that the arcs are lifted.

The number of inequalities derived from the sequential lifting of k–cycles grows fast with k. Although there is only one facet defining inequality for the ATS–polytope derived from a cycle of length 3, there are already four obtained from for a cycle of length 4. Until now there is no formula known that describes all the inequalities that can be obtained by applying the sequential lifting procedure to the cycle inequalities. Grötschel [Grö77] discusses several classes of inequalities, among them the so–called D_k^+– and D_k^-–inequalities.

(1.3.6) Theorem.
Let $C = (i_1, ..., i_k), \{i_1, ..., i_k\} \subset V, 3 \leq k \leq n-1$, be a simple cycle, then the D_k^+**–inequalities**

$$\sum_{j=1}^{k-1} x_{i_j i_{j+1}} + x_{i_k i_1} + 2\sum_{j=2}^{k-1} x_{i_j i_1} + \sum_{j=3}^{k-1}\sum_{h=2}^{j-1} x_{i_j i_h} \leq k - 1$$

and D_k^-**–inequalities**

$$\sum_{j=1}^{k-1} x_{i_j i_{j+1}} + x_{i_k i_1} + 2\sum_{j=3}^{k} x_{i_1 i_j} + \sum_{j=4}^{k}\sum_{h=3}^{j-1} x_{i_j i_h} \leq k - 1.$$

define facets of P_T^n and $\tilde{P}_T^n$, for all $3 \leq k \leq n-1$.

Proof. Validity : See Grötschel [Grö77].
Facet defining property for $\tilde{P}_T^n$: See Grötschel [Grö77].
Facet defining property for P_T^n : See Grötschel [Grö77] for the case of $k = 3, 4$ and Fischetti [Fis91] for all other cases. □

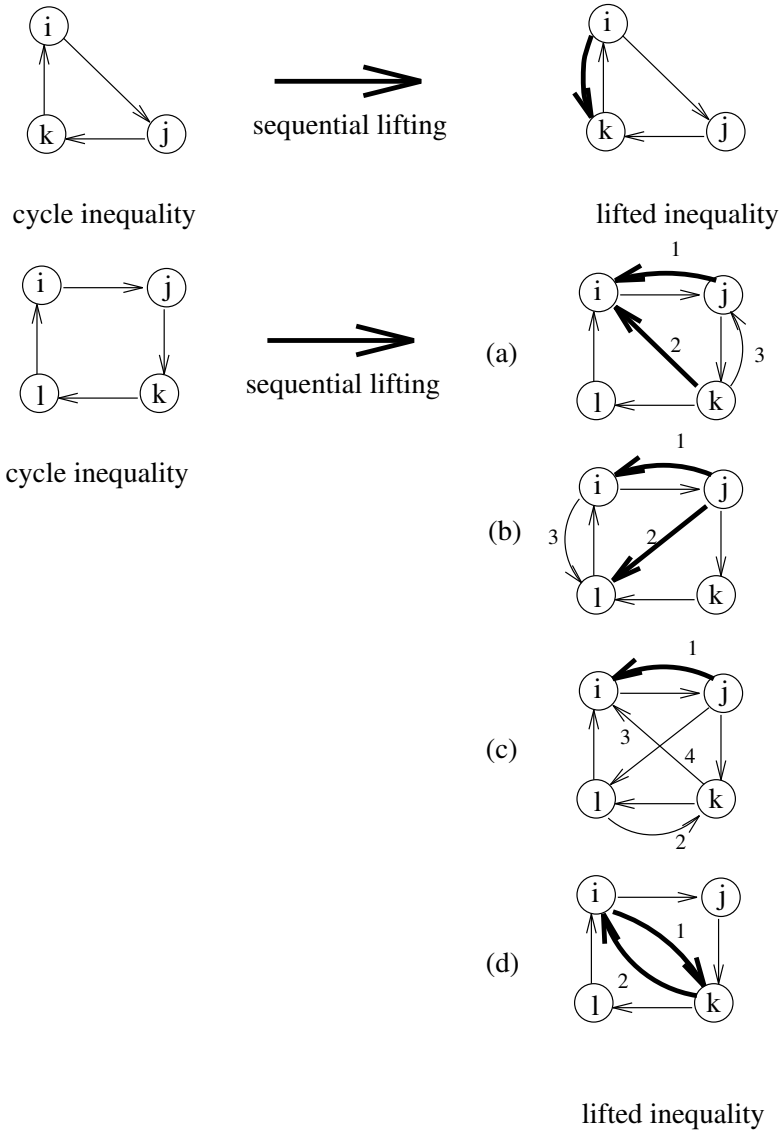


Figure 1.3.3 Lifted 3– and 4–cycle–inequality

The subtour elimination constraint $x(A(W)) \leq |W| - 1$ can be generalized by attaching a source node p and a sink node q. The obtained inequality is known as a T_k–inequality.

(1.3.7) Theorem.
Let $W \subset V$ be a vertex set in $D_n = (V, A_n)$ with $2 \leq |W| = k \leq n-2$, let $w \in W$ and $p, q \in V \setminus W$, then

$$x(A(W)) + x_{pw} + x_{pq} + x_{wq} \leq k$$

is called a **T_k–inequality** and is valid with respect to P_T^n and $\tilde{P}_T^n$.
For $n \geq 4, 2 \leq k \leq n-2$ it defines a facet of $\tilde{P}_T^n$. If, in addition, $k \neq n-3$, it defines a facet of P_T^n.

Proof. See Grötschel [Grö77]. □

This inequality can be further generalized by attaching a source and sink to a comb.

(1.3.8) Theorem.
Let $H \subset V$ be a "handle" and $T_1, ..., T_s$ be pairwise disjoint "teeth" satisfying
(i) $|H \cap T_i| \geq 1$ for all $i = 1, ..., s$,
(ii) $|H \setminus T_i| \geq 1$ for all $i = 1, ..., s$,
(iii) $s \geq 3$ and odd.
For each pair of distinct vertices p and q in $(V \setminus H) \setminus (\cup_{i=1}^s T_i)$, the following **C2–inequality**

$$x(A(H)) + \sum_{i=1}^{s} x(A(T_i)) + \sum_{v \in H} (x_{pv} + x_{vq}) + x_{pq} \leq |H| + \sum_{i=1}^{s} (|T_i| - 1) - \frac{s+1}{2} + 1$$

is valid and facet defining for P_T^n and $\tilde{P}_T^n$.

Proof. Validity: See Grötschel [Grö77].
Facet defining property: See Fischetti [Fis91]. □

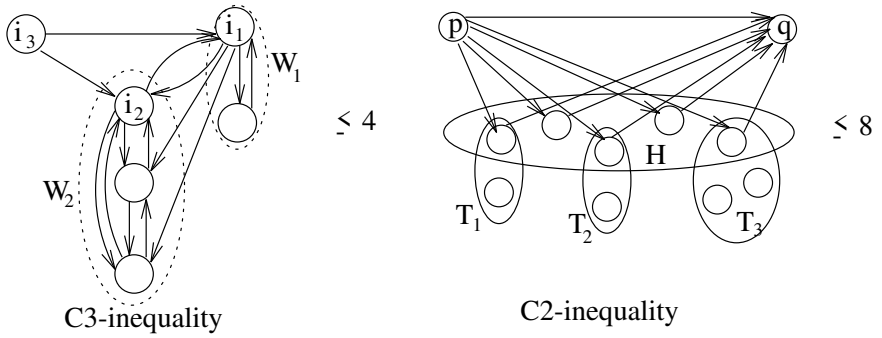


Figure 1.3.4 ATSP facets

(1.3.9) Theorem.
Let i_1, i_2, i_3 be three different vertices, and let $W_1, W_2 \subset V$ be such that
(i) $W_1 \cap W_2 = \emptyset$,
(ii) $W_j \cap \{i_1, i_2, i_3\} = \{i_j\}$,
(iii) $|W_j| \geq 2$ for $j = 1, 2$

then the **C3–inequality**

$$x(A(W_1)) + x(A(W_2)) + \sum_{j \in W_2} x_{i_1 j} + x_{i_2 i_1} + x_{i_3 i_1} + x_{i_3 i_2} \leq |W_1| + |W_2| - 1$$

is valid and facet defining for P_T^n and $\tilde{P}_T^n$

Proof. Validity: See Grötschel [Grö77].
Facet defining property: See Grötschel [Grö77] for the case that $|W_1| + |W_2| = n - 1$, and Fischetti [Fis91] for all other cases. □

Odd CATs

This class of inequalities has been introduced by Balas [Bal89]. We need to set up more notation in order to describe the inequalities of this class.
Two arcs $a_1 = (i_1, j_1)$ and $a_2 = (i_2, j_2)$ are said to be **incompatible**, if
(i) $j_1 = j_2$ or
(ii) $i_1 = i_2$ or
(iii) $i_1 = j_2$ and $j_1 = i_2$.
Otherwise, they are said to be compatible.
Roughly speaking, a **closed alternating trail (CAT)**

$$T = (a_1, a_2, ..., a_t)$$

is defined to be a set of distinct arcs a_k with the property that for $k = 1, ..., t$ the arc a_k is incompatible with arcs a_{k-1} and a_{k+1} and compatible with all other arcs in T (with $a_0 := a_t$ and $a_{t+1} = a_1$). If t is odd, the CAT is said to be **odd**. A node v is said to be a **source**, if it is the common tail of two arcs in the CAT and a **sink**, if it is the common head of two arcs in the CAT. Note, that a node v can either be a sink, a source, both or none. If it is neither a source nor a sink, it is said to be **neutral**.

By definition an alternating trail T is an arc set where the directions of two subsequent arcs a_i and a_{i+1} differ with the exception of pairs that form a 2–cycle. Figure 1.3.5 shows examples of odd closed alternating trails.

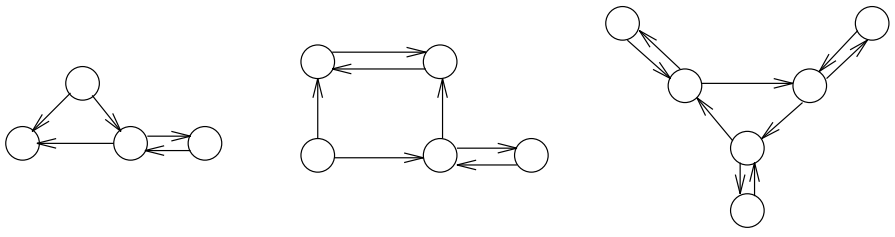


Figure 1.3.5 Odd closed alternating trails

Now, let C be a set of chords $(i, j) \in A \setminus T$ such that i is a source and j is a sink. We can now state the following theorem.

(1.3.10) Theorem. (Balas, 1989)
Let $D_n = (V, A_n)$ be a complete digraph on n nodes. Let T be an odd CAT, and let C be the set of its chords connecting a source to a sink. Then for all $n \geq 6$ the inequality

$$x(T \cup C_1) \leq \frac{t-1}{2}$$

defines a facet of P_T^n (except for two pathological cases for $n = 6$).

Proof. See Balas [Bal89]. □

Euler [Eul89] has generalized this result by "gluing" together several odd CATs at a distinguished 2–cycle and has shown that a new class of facet–defining inequalities for P_T^n is obtained, the so–called **odd CAT–bundles**.

1.4 Flexible Manufacturing and Discrete Mathematics

In the early 80s *"Flexible Manufacturing System (FMS)"* was a well used buzz–word. Huge amounts of money were poured into FMS technology. These systems were described as the automation approach to cure the ills of the manufacturing industry – an industry organized on contemporary technology which could not respond to the new markets requiring flexibility in production. Especially the ever shorter life cycles of products (e.g., in computer industry) and the variety in the product spectrum made it necessary to install flexible production systems.

In Section 1.4.1 we describe recent developments in manufacturing that have led to the modern Flexible Manufacturing Systems that are playing an ever larger role in today's automation of industry. Detailed surveys can be found, among others, in Fine [Fin93], Parish [Par90] or Tetzlaff [Tet90].

During the planning and control of an FMS a lot of decisions have to be made, where models and methods from Discrete Mathematics, embedded in a decision support system, can help to analyze the system and increase the system performance. In Section 1.4.2 we summarize some of these approaches.

1.4.1 Flexible Manufacturing Systems (FMSs)

For a long period of time the market accepted a small product variety for the sake of cheap mass–produced goods. This is changing nowadays, as the richer markets of today demand for flexibility from their industrial suppliers to meet greater individual requirements.

During the last decades two main streams have characterized the developments in the field of manufacturing, namely automation and integration. Automation means the substitution of costly human labour by machines and integration can be described "as the reduction of buffers between physical or organizational entities"[Fin93]. This development is closely related to the advent of computers and electronic data communication, in particular by the introduction of digital computers during the 1950s and integrated circuits (IC) and microcomputers during the 1970s.

During the 50s machines were developed that were equipped with a "computer", known as **numerical controllers (NC)**, and could be run by a computer program. These so–called stand alone **NC–machine** needed an operator to load and unload the workpieces, tools and NC–programs. Although this was not a fully automated machine it required less interference than manually–operated machines. The capacity of the NC–computers increased in the following years and resulted in the ability to control all peripheral equipment and the machine tools. These enhanced controllers, developed in the 1970s, became known as **computer numerical controllers (CNC)** and are still developing. These **CNC–machines** have been equipped with their own small computers and had the advantage over the NC–machines that all NC–programs could be stored locally. Some CNC–machines had the additional feature of automated parts loading and unloading and of automated tool changing. But still for most of these machines the provision of data and material was completely manual, i.e., NC–programs are loaded manually to the CNC, pallets are manually loaded, etc.

The next improvement in order to avoid manual interaction and interference was the development of **Distributed Numerically Controlled (DNC)** systems which normally consist of several CNC machines that are under the control of a larger computer system. This computer automatically downloads the NC–programs via a DNC–link from an NC–program storage computer to the distributed machines.

Next, a group of machines has been connected physically using a transportation system for the workpieces. This automatic material handling system might consist of automatically guided vehicles (AGVs), conveyor belts, rail systems, etc. and has the task to move the material and workpieces between the work stations, storage locations and shipping points. The transportation system is one of the main integrating components, as it connects several points in the system.

Finally, cell systems have been developed where a host computer takes over many of the organizational tasks originally carried out by an operator. This computer is called the "FMS–host". The modern FMSs also include tool flow systems as well as piecepart flows. Nowadays FMSs might also contain robots, which are automated equipment, typically programmable, that can be used for moving material to be worked on (pick and place) or assembling components into a larger device. They also substitute human labour in the use of tools or equipment.

In earlier time, the host computer was a mainframe due to the lower performance of the computers available at that time. The high costs of computers in these days were one of the main reasons for not decentralizing functions on several expensive computers. But this had the disadvantage, that if the central host malfunctioned then the whole FMS stopped. Today with cheaper high–performance hardware and software, the FMS control hierarchy is much more decentralized. Many functions have been off–loaded from the host to the peripheral controllers in the FMS, e.g., to CNCs and Programmable Logical Controllers (PLCs). This decentralization of functions across more localized hardware results in a higher system reliability. If the host breaks down, the FMS can still function, although not with such a high degree of automation and without integration of all components. Alternative emergency strategies have been developed for the peripheral controllers to take over certain functions in such situations.

The FMS is **flexible** in the sense that this system enables to produce a greater variety of products than this is typically possible on standard highly–automated transfer lines. This is possible, as both the operations performed on each machine and the routing between the machines can be influenced by means of a software control system and without costly and time–consuming changeover requirements between product–mixes. Certainly the flexibility of the system is restricted by the technical limits of the FMS (speed of machines, capacities, etc.).

It is not easy to give a formal definition of an FMS. In the literature there were several attempts to point out the main aspects of an FMS. Following this line an FMS is accepted to be

> *a fully integrated manufacturing system consisting of flexible machines or robots, linked by an automated material handling system, where all the system components and the whole system are under computer control.*

Although there is that widely accepted definition of an FMS, all systems certainly differ in their layout and operational rules with respect to their strategic purposes and impact. The integration of automated industry is now recognized by the widely accepted label of **"Computer Integrated Manufacturing" (CIM)**. One aspect of the CIM concept is the application of FMS technology.

1.4.2 The role of Discrete Mathematics in Flexible Manufacturing

In a process that started in the beginning of the 80s industry became aware of the importance of competitive manufacturing and, as a result, there was a remarkable increase in the interest in the contribution Discrete Mathematics could make to the field of modern manufacturing. The rapid increase in the number of mathematically oriented journals, books and papers in the manufacturing field manifests this development. The intention of this section is not to survey all the mathematical problems that arise in that area. Due to the richness of the field of manufacturing this would be out of the scope of this thesis. Rather we like to characterize the ongoing research activities and give an idea of what is done and what could be done.

Most of the FMSs are extremely complex. It is difficult both for researchers and managers to achieve a clear, coherent picture of how such systems work. The result is that it is not possible to predict "with the naked eye" which consequences any layout or operational decision will have on the performance of the whole system. As some sort of help is needed to obtain an understanding of the behaviour of the FMSs, to support in their design and to compare alternative operational decision rules, it will be necessary to model the FMS mathematically. The aim of this modelling phase is to transform the manufacturing problem to some quantitative relations, equations or inequalities and to make them suitable to be attacked by algorithmic methods. The resulting optimization tools, embedded in a decision support system, should help the designer or controller of the system to make decisions to questions of a rather complex structure.

Not only the FMS in itself is rather complex. But, as a result, also the question of modelling the FMS or the problem of "optimizing an FMS" is of an extremely high degree of complexity, typically involving nonlinear system dynamics as well as stochastic elements. With the known methodology it is not possible to solve the problem in its entirety. Therefore, a very common approach to attack that problem is by decomposing it into a **hierarchical structure of subproblems** that are suitable to be attacked by the known methodology. In this approach the solution of one subproblem is accepted, as the starting point for the next subproblem. Certainly this approach does not guarantee to find the global optimum of the original problem, even under the assumption that every subproblem is solved to optimality. As some of the subproblems themselves turn out to be $\mathcal{NP}$–hard, even this cannot be achieved in a reasonable amount of time. It is very common to approximately solve the subproblems by means of heuristics. But nevertheless, this hierarchical approach turned out to give satisfactory results for many practical problems.

The occurring mathematical subproblems may be divided into two classes, namely

(1) layout and design problems of an FMS and

(2) problems arising in the scheduling and in the control of the system.

The first problem class deals with the selection of equipment and with the system layout whereas the second addresses operational questions within an existing FMS. In the following we summarize some of these problems. It is not possible to describe all the mathematical models in detail, since it would be necessary to also describe the technical environment within the FMS to give an "understandable" description of the mathematical models. Thus, for more details the reader is referred to the cited literature. A good survey about related mathematical problems can be found in [GRZ93].

Beside the mathematically based approaches simulation tools are also very common in practice. They are discussed in detail in Chapter 3.

Design problems for FMS

As FMS–technology is very expensive in general, the users are interested in having an "optimal" design and layout of the system. This design and planning phase is prior to the beginning of production and includes among others questions like:

(1.4.11) FMS design problems.

- optimal kind and number of products to be manufactured,
- optimal kind, number and size of the FMS–components (machines, buffers, transportation system, etc.),
- optimal floor layout of the FMS,
- optimal structure of the planning and control system (hard– and software, structure of the computer network, etc.).

□

Two different optimization directions are discussed in the literature. In the first class the problem of minimizing the inventory cost of an FMS configuration while meeting a prescribed throughput rate is discussed. The second deals with the problem of maximizing the throughput while fulfilling some budgetary and capacity restrictions. Some approaches are known that try to combine these two objectives. A major problem is that there is no clear–cut measure of the performance of a proposed production system. The absence of a computable "performance rate" of an FMS makes it difficult to decide between competing system designs.

There are two approaches that are very common in the rough–cut analysis of a system. That are simulation approaches and analytic queueing models. **Simulation models** are very common in practice and are discussed in more detail in Chapter 3. Among the most propagated approaches in the academic world are **analytic queueing models** (open or closed queueing networks) capable of analyzing the steady–state behaviour of a system. They are briefly sketched in the following.

Queueing networks

The research on queueing networks in the context of manufacturing questions was initiated by two publications of Jackson in 1957 and 1963 ([Jac57, Jac63]). It was motivated by the observation that in the modern manufacturing systems arising at that time, products sometimes spend a long time waiting for some resources (material, work stations, etc.). Therefore, one was looking for a methodology that takes waiting times into account, with the aim of quantifying the impact of alternative manufacturing strategies. This was provided by queueing theory.

In that approach each manufacturing cell is considered as a station with a queue and one or more servers, dependent on the number of workstations (CNC–machines, robots, etc.) within this cell. These stations are connected by the underlying transportation system. In

closed queueing networks a fixed number of pallets or work pieces are circulating in the system, whereas in an **open queueing network** pieces are assumed to arrive at the loading station due to a certain probability distribution and they leave the system when they are completed.

Briefly sketched the results of Jackson are the following. Under certain assumptions on the underlying model (e.g., parts enter the system according to a Poisson process, service times are exponentially distributed, service according to a "first–come–first–serve rule", part routing according to a Markov chain) he could show that the steady state joint queue length distribution of the system can be computed relatively easy. It follows a so–called *product–form*, which means that it can be studied by analyzing the length of each server queue independently. In the following years the research activities concentrated on elaborating other conditions under which the product–form holds.

But an observation that has been made is that in most practical applications these assumptions are rather unrealistic. For these networks the assumptions seldomly hold and an exact analysis appears to be impossible. Therefore, a new tendency has gone towards the development of approximate solutions for a general class of problems.

But once being confronted with real problems the results are not yet convincing, as, e.g., R. Suri and S. de Treville state in an article about the future of modelling technologies in the world of competitive manufacturing (see [ST91]):

> *"...the use of queueing models in manufacturing...remains limited. True, queueing models are commonly discussed in academic operations management courses, publications cite their applicability to manufacturing and research proposals assume their wide use. However, we have noticed that queueing models are seldom used for manufacturing decision–making."*

We believe that this is mainly due to the fact that real production systems (not the artificial systems that are studied in the academic literature) seldom reach a steady state. But this approach, as well as a simulation model, is certainly suitable for a rough–cut analysis of a system needed for initial decisions. Bottlenecks of the system can be detected, and an approximate prediction on system variables (throughput, waiting queues in front of workstations, machine utilization, etc.) can be made.

For more details concerning queueing networks the reader is referred to Suri et al. [SSK93]. A survey on problems attacked in the literature by means of queueing networks can be found in Solot and van Vliet [SV93]. A case study where queueing models were used for the long–term design and production planning of a company can be found in de Treville [Tre92].

Other approaches

To our knowledge there are no comprehensive models to determine "optimal" FMS–configurations. The approaches in the literature mainly concentrate on one of the optimization questions mentioned in (1.4.11) and try to find their optimal values. But depending on the objective (linear, nonlinear), the kind of variables (continuous, binary, integer) and the kind of constraints (linear, nonlinear) various models are proposed.

In most cases the resulting mathematical models (e.g., Mixed Integer Programs) are too complex to be solved efficiently to optimality. Therefore, for the most part the problem is attacked by means of heuristics, sometimes with an additional phase of implicit enumeration (branch&bound).

For a more detailed survey the reader is referred, among others, to Graves et al. [GRZ93], Tempelmeier and Kuhn [TK92]. In Tetzlaff [Tet90] several LP–models for FMS design problems can be found. An overview over methods and algorithms arising in that field can be found in Nemhauser at al. [NKT89].

Optimization problems in the control of the FMS

The optimization problems in the control of an FMS occur once we are given a physically installed FMS and want to organize the production in detail, s.t. an "optimal" utilization of the expensive manufacturing technology is achieved. The arising problems might be classified in problems on a strategic (long–term) level and on an operational (short–term) level. The goal of the optimization process on the strategic level is to develop a rough production plan for the coming months, whereas the decisions on the operational level cover the specification of detailed production decisions, typically for time horizons up to a month. In general these problems are too complex to be solved in their entirety. Therefore, they are divided into a hierarchical structure of subproblems, as, e.g., part type selection, routing, scheduling, etc.

All the subproblems are somehow rather system specific and depend very much on the underlying architecture of the FMS. They include, among others, questions such as

(1.4.12) FMS control problems.

- the selection of subsets of jobs to be produced as a batch (part type selection problem),
- the assignment of tools to machines,
- the scheduling of jobs,
- the routing of jobs.

□

Here several modellings with different objectives are known (minimize the makespan, minimize the number of delays, minimize the cost for production and inventory holding, etc.). These often lead to linear binary problems or mixed integer programs. But some system restrictions can lead to models that have to deal with nonlinear constraints or nonlinear objectives.

These problems are mainly attacked by means of heuristic methods or implicit enumeration. In practice there is a tendency towards priority–rules and knowledge based systems. There are also reports on the use of simulation approaches for the prospective valuation of solution variants.

Among the most studied problem classes in the field of FMS–control problems are the **scheduling problems**. A typical scheduling problem is of the form that, for a given set of operations each with a certain processing time and a given set of machines, the starting times of the operations on the machines have to be found. The models in the literature differ in several details, e.g., if it is allowed to interrupt the processing of a job on a certain machine (preemption), if there are identical machines or not, etc.

Although for some special cases it was shown that the problem can be solved in polynomial time for most cases that arise in practice the problems are $\mathcal{NP}$–complete. Some of the problems are very hard from a computational point of view. A job–shop–scheduling problem stated in 1963 by Fisher and Thompson (10 machines, 10 jobs, 100 operations per job)

remained unsolved for more than 25 years, until it was solved to optimality in 1989 by Carlier and Pinson.

Due to the complexity of these problems they are mainly attacked by approximation algorithms based on local enumeration, simulated annealing, taboo search, etc. If the problem needs to be solved to optimality, this is done by a branch&bound approach. Approaches based on linear programming techniques which produce good results for many other combinatorial optimization problems are still at an initial stage and the results are not yet satisfactory.

For a review on scheduling problems the reader is referred to Lawler et al. [LLRS93] and Błazewisz et al. [BCSW86, BESW93]. For a survey on polyhedral approaches to single machine scheduling we refer to Queyranne and Schulz [QS94].

1.5 On–line optimization

Although a great variety of problems (also problems in the control of an FMS) are of an on–line character the theory of on–line optimization has developed just since the mid 1980s, and from our point of view is still in its infancy. There are not many scientific investigations, and only a few practical applications using these concepts are reported. Beginning with the work of Sleator and Tarjan [ST85] a development in Discrete Mathematics and Computer Science started that might be called "theory of on–line algorithms".

The general concepts and notation are introduced in the following sections. Furthermore, we give some examples for the on–line problems studied so far, and explain why these concepts are not useful for the studies that are the topic of the following chapters.

To our knowledge there are no survey articles on on–line optimization but the proceedings volume [MS92] and Albers [Alb93] give a good review on the scope of on–line optimization so far.

1.5.1 On–line problems

The classical theory of algorithms is based on the assumption that the algorithm is assumed to have complete knowledge of the input data. But in many applications this is not realistic. In almost every dynamic system the process of making decisions can be described to be **on–line** because it is necessary to make decisions without having full information about the future and the influences of the decisions on the system. This statement is justified by means of several examples.

Computer systems:
In the management of computers with a 2–level memory system that have a certain number of pages of fast memory, one is confronted with the so–called **paging problem**. A page fault occurs whenever a page which is not in fast memory is needed there. The computer has to decide which page to take out of the fast memory whenever a page fault occurs. This decision has to be taken without knowing what the future requests will be. The goal is to minimize the number of page faults.
The **caching problem** occurs for example while caching fonts into a printer or into the memory of a bitmap display. The printer (or display) can store bitmaps with a fixed number of characters in its font memory. The number of bits required to transmit the bitmap of a character varies according to its complexity. The problem is to decide which font to take out of the font memory when a request for a nonloaded font occurs. This has to be done without knowledge about future requests and with the objective to minimize the overall transmission time.
A similar problem occurs in the management of a **two–headed disk** where you have to plan the motion of the heads that drive along a linear track. Each time that a request occurs, one head has to move to a particular point along the line. The problem is to decide which head should move to that point, such that the total movements of the heads are minimized.
These three problems belong to the class of most studied problems in on–line optimization, the so–called k–server problem, and are discussed in more detail in the following sections.

Communication networks:
An on–line problem occurring in communication networks is that of establishing flexible multipoint connections, e.g., in video broadcasts and multiperson conferences. This problem

is typically a routing problem within an existing communication network and is treated as the problem of finding a minimum cost spanning tree connecting the set of terminal nodes. If all the terminal nodes are known in advance, this is the classical Steiner tree problem. But it is typical for the described application that the nodes are added to or removed from the connection without knowing in advance when and where this will happen. In principle it is possible to reroute the connection when such an event occurs. But as it is time–consuming and may require a significant use of network resources, this has to be avoided, especially if an attempt is made to reroute the entire connection. Thus, you have to solve a so–called dynamic Steiner tree problem. This problem has been introduced and studied by Imase and Waxman [IW91].

Vehicle routing:
An on–line problem in vehicle routing occurs, e.g., in the dispatching of a fleet of vehicles to satisfy multiple demands for service. These vehicles might be taxis, ambulances, bicycles of a messenger service within a city, etc. The demands for these vehicles occur "randomly" in time and site and someone has to decide which vehicle to move to which request in order to optimize a certain objective function, e.g., minimize the waiting time until the demand is served. For further examples and details see, e.g., [Psa88, BR91, BR93a, BR93b].

In these so–called **on–line problems** the input data is supplied to the algorithm only in units, one unit at a time. The algorithm has to take decisions every time that new data occurs and these decisions cannot be changed later. Thus, the algorithm has to make decisions based only on partial information about the whole input. The **on–line optimization problem** is to find an output of the algorithm of minimal (maximal) cost for a given input.

Next,we give some examples of studied on–line problems without having the goal to be encyclopaedic and thus this list should not be regarded as a comprehensive survey of this topic. By giving these examples we just want to give an understanding of the nature of these problems and of the principles underlying the research activities.

***k*–server problem :**
The k–server problem has been introduced by Manasse, McGeoch and Sleator [MMS88, MMS90] as a problem derived from the caching- and paging–problem in computers with different memory units. It can be stated as follows. We are given a finite metric space (e.g., a graph with distances on the edges) and k servers at some initial positions. They have to fulfill sequentially given tasks by moving one of the servers from its current position to the position where the task occurs. The choice of which server is moved must be made without knowledge about the future requests and the tasks have to be worked off in the sequence of their appearance. The objective is to minimize the sum of distances that all servers cover.
The paging problem can be modelled as a k–server problem as follows. You have n nodes in a complete graph which correspond to the n pages of addressed space in the computer. The k servers correspond to the k pages of fast memory. You say that a page is in fast memory, if one of the servers is on the corresponding node. All distances on the edges are equal to 1. When a request occurs and no server is located on the request–node, one server has to move to that node. The objective is to minimize the number of moves (sum of distances).
The caching–problem and the two–headed–disk–problem can be modelled in a similar way.

On–line matching :
Suppose we are given a complete bipartite graph $G = (V, E)$ with $V = V_R \cup V_S$ and $|V_R| = |V_S| = k$. The nodes in V_S correspond to "service nodes" the nodes in V_R to "request nodes". The on–line behaviour is that the weights on the edges are revealed in k different time intervals. In the i–th interval all the weights of the edges incident with the request node i are exposed and an unmatched service node has to be selected to match the request node i. The objective is to find a matching of minimum (maximum) weight (cmp. [KVV90, KP93]).

On–line graph coloring:
In an on–line model for graph coloring the graph is presented one vertex at one time. Each new vertex is given together with all the edges connecting it to already generated vertices. The on–line problem is to assign a color to each vertex as it is received, such that no two vertices of the same color are adjacent. Once a color is assigned it cannot be changed. The algorithm has no information about the order in which the nodes appear or about the chromatic number of the input graph. The objective is to use as less colors as possible (cmp. [LST89, KT92, HS92], among others).

On–line bin packing:
We are given an infinite number of bins and have to pack an unknown number of objects of different sizes into them. As soon as we get an object, we have to put it into a bin such that it is completely contained in it. This decision cannot be revised afterwards. The aim is to use as few bins as possible. The known "First–Fit–Heuristic", where every object is assigned to the first bin it fits in, is certainly an on–line algorithm because it does not use any information about forthcoming objects. For further information about on–line bin packing see, e.g., [CV93, RT93].

Further examples for on–line problems, e.g., on–line partitioning problems [FKT89], dynamic location problems [CGS89], etc. could be added. Although there is a big variety of off–line combinatorial problems, these problems have not yet been studied such intensively in its on–line versions.

1.5.2 On–line algorithms

In general it is difficult to compute the optimal solution of an on–line problem. Therefore, the research community was looking for a measure for on–line algorithms. First the **probabilistic analysis** was used where the performance of an algorithm is analyzed under the assumption of "typical inputs". But this approach suffers from the criticism that seldom a probability distribution is known that models a typical input and if you input a worst–case sequence the algorithm may perform extremely poor.

Competitiveness

This dilemma resulted in the development of **competitive analysis** which was defined in 1988 in papers by Manasse, McGeoch and Sleator [MMS88] and by Karlin, Manasse, Rudolph and Sleator [KMRS88]. This was based on an idea already used earlier by Sleator and Tarjan in their paper on maintaining linear lists [ST85]. The concept of competitiveness is now commonly used in the literature as a measure of performance of algorithms for on–line problems. For the sake of convenience we describe the concept only for on–line minimization problems. The idea behind this concept is that the performance of an algorithm should be measured

by the ratio of the cost it incurs on a sequence of inputs to the minimum off–line cost for processing the sequence. This has the advantage that one can avoid making assumptions required by probabilistic analysis. We now state it more formally.

(1.5.13) Definition.
Let $\mathcal{P}$ be an on–line problem, $\mathcal{A}$ an on–line algorithm for $\mathcal{P}$. Given an instance I of problem $\mathcal{P}$, $cost_{\mathcal{A}}(I)$ denotes the value (or cost) of the solution calculated by $\mathcal{A}$, whereas $cost_{opt}(I)$ denotes the value of the optimal (off–line) solution. Let $c \geq 1$ and b be constants that do not depend on the input string I. We say that an **on–line algorithm $\mathcal{A}$ for $\mathcal{P}$ is c–competitive**, if for any instance I of problem $\mathcal{P}$

$$cost_{\mathcal{A}}(I) \leq c \cdot cost_{opt}(I) + b$$

holds. We say that a given **on–line problem $\mathcal{P}$ is c–competitive**, if there exists a c–competitive algorithm for $\mathcal{P}$. And we say that it is **not better than c–competitive**, if there exists no c'–competitive algorithm for $\mathcal{P}$ for any $c' < c$. ◇

Using the concept of competitiveness on–line problems can be regarded as a game of two players (the algorithm A and the so–called adversary) who act alternately. The algorithm tries to get the best result as possible, whereas the policy of the adversary, who supplies the algorithm with input, is to avoid that. The adversary has the advantage that it can examine the code for A and will choose the input in the worst possible way. But it is typical for most applications that the input data does not behave that bad. Therefore, the concept of competitiveness is just a first step into the right direction, but many side constraints that are relevant for practical on–line applications (e.g., tasks have to be fulfilled in certain time windows, existence of priorities between the tasks, etc.) are not considered in that concept. This criticism is also mentioned in the preface of the already cited proceedings volume [MS92] where the editors Lyle McGeoch and Daniel Sleator state :

> *"... A high point of the workshop was the panel discussion, which featured Richard Karp, Larry Larmore, Mark Manasse, Prabhakar Raghavan, and Daniel Sleator .... The panel considered the interaction between the theory and practice of on–line algorithms. Competitive analysis, while being a useful mathematical tool, sometimes yields bounds that are unduly pessimistic. It was agreed that it is worthwhile to consider other approaches to evaluating on–line algorithms..."*

Randomized on–line algorithms

Randomized algorithms have turned out to be a powerful tool in the design of algorithms for several problems, covering a wide range of applications from number theory and algebra, pattern matching, sorting and searching, computational geometry, graph theory, data structure maintenance, combinatorial enumeration, distributed computing (see, e.g., Karp [Kar91] for further details). They are often very easy to understand and to implement. Using randomization very simple deterministic algorithms with a poor worst–case behaviour turn out to be an applicable method to give (with high probability) good results on every possible input.

Randomization is also a powerful tool in the design of competitive algorithms. The uncertainty about the future data can be partially compensated by allowing an algorithm to make probabilistic choices. By these random choices the "adversary" has less power, since the moves of the on–line algorithm are no longer predictable. By playing that "mixed" strategy

the adversary has problems in choosing an input that will cause difficulties to the algorithm. The adversary can examine the code for $\mathcal{A}$, but cannot predict the outcome of the random choices within $\mathcal{A}$. Therefore, it cannot predict the output of $\mathcal{A}$.

The concept of competitiveness can be defined in an analogous way for randomized on–line algorithms, but due to the randomness it is inevitable that a description of a randomized algorithm will involve probabilistic statements.

(1.5.14) Definition.
Let $\mathcal{P}$ be an on–line problem, $\mathcal{A}$ a randomized on–line algorithm for $\mathcal{P}$. Given an instance I of problem $\mathcal{P}$, $E[cost_{\mathcal{A}}(I)]$ denotes the expected cost of the solution $\mathcal{A}$ will calculate for a given input I, $cost_{opt}(I)$ denotes the value of the optimal (off–line) solution. Let $c \geq 1$ and b be constants that do not depend on the input string I. We say that a **randomized on–line algorithm $\mathcal{A}$ for $\mathcal{P}$ is c–competitive**, if for any instance I of problem $\mathcal{P}$

$$E[cost_{\mathcal{A}}(I)] \leq c \cdot cost_{opt}(I) + b$$

holds. ◇

For the k–server problem results are known where a randomized algorithm outperforms all deterministic algorithms with respect to the concept of competitiveness (see below). The same holds for the on–line bipartite matching problems where Karp et al. [KVV90] give a simple on–line algorithm that achieves the best possible performance.

For further details about randomized algorithms the reader is referred to [BDBK$^+$90, Kar91].

Results on on–line algorithms

We continue by citing some of the typical results that are known for the on–line problems mentioned in the previous section. As the scientific research in the area of on–line optimization is mainly concentrated on the design of c–competitive algorithms, most of the results deal with this concept. We concentrate on the k–server problem as the most studied problem in on–line optimization. For results on the other problems, that follow the same philosophy, the reader is referred to the cited literature.

***k*–server problem.**
Manasse, McGeoch and Sleator [MMS90] showed that the k-server problem is not better than k–competitive for any metric space with at least $k+1$ points. The problem of whether there is a general on–line algorithm with competitiveness k remains open.

For $k = 2$, the problem was solved by Manasse, McGeoch and Sleator [MMS90] who gave a 2–competitive algorithm for 2 servers. Although for arbitrary k no k–competitive algorithm is known, there are already some partial results in this direction. Fiat et al. [FRR90] have presented a competitive algorithm for k servers whose constant is an exponential function of k. Grove [Gro92b] has proven that for a special case (harmonic k–server problem) the so–called harmonic algorithm is competitive for any k. Most of the other results are typically of the form that it is shown that a certain algorithm is c–competitive, while given a performance ratio measure for c that is as close as possible to k.

As stated above, randomization can be a powerful tool in the design of competitive algorithms. The uncertainty about the future data can partially be compensated by allowing an algorithm to make probabilistic choices. For example, it is known that there is a $2H_k$–competitive randomized algorithm where H_k corresponds to the k–th harmonic number

($H_k = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{k}$; this function is closely approximated by the natural logarithms: $ln(k+1) \leq H_k \leq ln(k) + 1$). This competitive factor of, more or less, $ln(k)$ is far better than the lower bound k for deterministic algorithms. Furthermore, it has been shown that there does not exist a randomized on–line algorithm with competitive factor less than H_k (see [FKL+91, MS91]).

Several other approaches for studying on–line problems have been presented, including *request–answer games* [BDBK+90] or *on–line games* [CL92]. But as these approaches have not yet prevailed, we just mention them without going into details.

Chapter 2

Description of the considered FMS

2.1 Introduction

The Siemens Nixdorf Corporation (SNI) operates a factory in Augsburg (Germany) where all their personal computers (PCs) and related products (monitors, data terminals, keyboards), and multi–user systems are manufactured. This factory was erected in 1987 and designed according to a CIM/CAI–concept, i.e., it has been designed to produce small series with a large product range. The system consists of computer supported production lines and a fully automated transportation system, including conveyor belts and automatic guided vehicles (AGV). The daily production is planned after the customers' order. An automatic repeat order of used parts following a Just–In–Time concept was planned, but was relaxed to a 3–days buffering. This means that with the currently stored material it should be possible to produce for three more days.

The whole system is computer controlled, making use of a central computer and independent software units, connected by a local area network (LAN).

At the start of the project the factory had two production halls. In the main manufacturing hall all PCs and related products are assembled, in the other the multi–user–systems are produced. In the course of the project two more production halls were built, mainly to produce all the software packages that have to be delivered together with the computers and to establish a delivery–center for the whole of Germany. In 1990, when the project began, the daily production of the PC manufacturing hall where the main production process takes place, was approximately

- 60 PC Towers,
- 300 – 400 PCs,
- 400 – 500 data terminals and monitors,
- 700 keyboards.

The system was designed for a smaller scale of production. As a result several components within this hall turned out to be bottlenecks in the flow of the material and the management was looking for possible ways to improve the production process without having to carry out expensive technical changes. We were confronted with the problem of optimizing the flow of material within the main manufacturing hall.

A team of researchers of the University of Augsburg, now with the Konrad–Zuse–Zentrum für Informationstechnik in Berlin, a group of students and a support team from SNI, worked for almost three years together in a project that had the overall aim to develop a software package that optimizes the material flow through the whole factory. This was an ambitious goal and the problems that occur in the modelling and optimization of modern manufacturing systems, as they were described in Section 1.4.2, could also be observed in our case. But this overall goal was kept in mind whenever a new optimization or simulation program was designed, as the output and the performance of the whole system should be improved and not the speed of a few components.

Due to organizational reasons within SNI the project did not reach its ambitious goal in the end. A simulation model as well as several optimization methods were developed and implemented. Some of the software tools (e.g., the one described in Chapter 4) are running with good success in everyday production. The most successful tool as well as all involved mathematical problems will be described in detail. All other approaches are briefly mentioned and it will be referred to the related literature.

Before continuing we would like to make some general remarks. As the developed optimization tools should be tested with real–life production data, one of the central points in the project has been the **collection of data**. An experience everybody who ever worked in such projects can confirm is that this is sometimes as difficult as solving the mathematical problems.

The system is computer controlled and in principle most of the necessary data is available somewhere, in most cases partially distributed on different computers or in different control systems. It is necessary to find somebody having a good general knowledge about all systems, or to write interfaces to filter out all the data that is needed, as one is not interested in obtaining gigabytes of information that is not relevant. Sometimes data could only be obtained by recording the telegrams between some hardware components (e.g., the stacker crane) and their controllers. The software packages that had to be designed have to take this into account. Optimization algorithms that depend on the input of certain data that is not available or that can only be made available with an immense (and sometimes unacceptable) use of network resources will never be used in everyday production.

This chapter is organized as follows. First, a description of the considered FMS is given, including a description of the physical factory layout as well as of the material flow. Next, the bottlenecks, the work–off strategies and possible optimization approaches are presented. As the optimization problems for the automatic storage systems will be of interest in the following chapters, these systems are described in more detail than the other components.

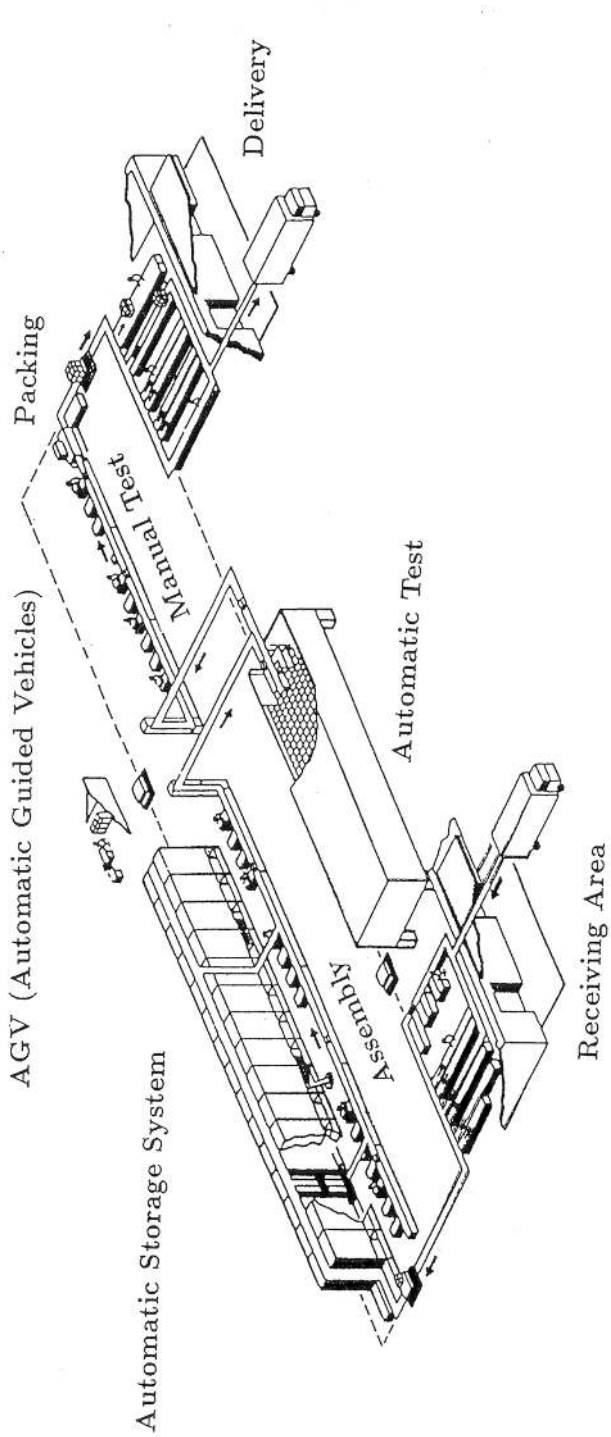


Figure 2.1.1 Layout of the FMS

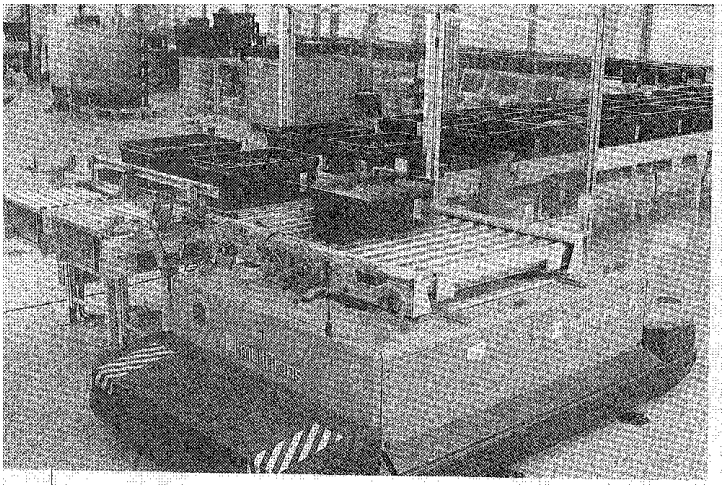


Figure 2.2.2 AGV at the reception area

2.2 Layout of the system

We now briefly describe the main components of the considered production hall (see Figure 2.1.1 for a sketch of the layout). For a more detailed description of technical terms the reader is referred to Sims [Sim91].

Receiving area

All parts necessary for production (floppy disks, chassis, cables, etc.) —some of them already preassembled— are delivered by trucks. The printed circuit boards, e.g., are manufactured in another Siemens plant in Augsburg only a few kilometers away. The parts arrive in containers differing in size and shape (two kinds of tote boxes, pallets, box pallets and special purpose pallets). After detrucking they are supplied with a bar code and booked into the computer system. The bar code and a system of scanners makes an identification of a container throughout the system possible. Together with the registration a storage location in one of the storage systems is reserved. (If no storage location is available, the container is rejected.) One of several conveyor belts transports the container to the loading point for the AGV, where a transportation task is generated. A vehicle picks up the container and transports it to its destination (Figures 2.2.2 and 2.2.3).

Automatically guided vehicles (AGV)

The system of AGV is responsible for all internal transportation between the reception areas (of all halls) and the storage systems and consists of 14 vehicles serving all 4 production halls. For control and guidance a wire is buried in the floor. A small electromagnetic field is radiated from the wire and inductive–type sensors on the vehicles are used as guidance detectors. To interface with the conveyor system at the receiving area and at the loading– and unloading points of the automatic storage systems (AUSS) the vehicles are equipped with a conveyor deck. The vehicles are battery operated and equipped with controllers that take over all

local tasks such as track following, precision stops, loading, unloading, etc. The vehicles are connected to a central AGV host computer via remote radio control. This computer has all information about the current positions of the vehicles, their status (loaded, unloaded, etc.) and all transportation tasks. The computer control system has to decide which vehicle should serve which transportation task.

The system is organized as a so–called block system, which means that it is divided into small sections and only one vehicle is allowed to be in one section. The travel directions are one–way streets although from a technical point of view it would be possible to travel in both directions on the tracks. Although a dynamic rerouting of the vehicles is possible this has not been taken into consideration for the control system. The computer has the necessary information about congestions (vehicles do not leave their section) and could reroute the vehicles, but they always follow a prescribed route (shortest path) to come from point A to point B, unless this is changed manually.

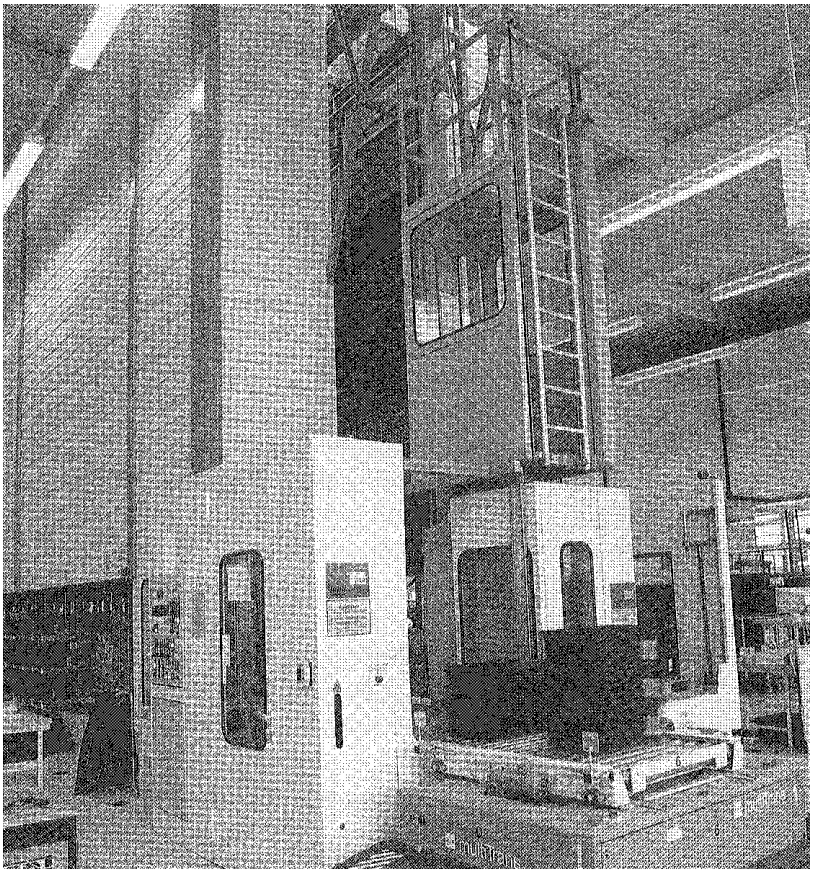


Figure 2.2.3 AGV at the automatic storage system

Automatic storage systems

In the considered manufacturing hall there are three automatic storage systems (AUSSs) that mainly serve as material buffers between the reception area and the assembly lines, located on each side of the AUSS. They are approximately 45 meters long and seven meters high, single aisled, with storage locations on each side of the aisle. The storage system is divided into two levels. In the upper part of the system the **storage locations** are mounted, whereas the lower part serves as a **buffer for the assembly lines**. Here all parts necessary for the manufacturing of the current product mix are buffered. As these assembly line buffers are limited, both in size and capacity, not all parts necessary for the amount of production of one shift can be buffered there. Thus, in case that another product is manufactured or a container is empty a refill of the buffers has to take place.

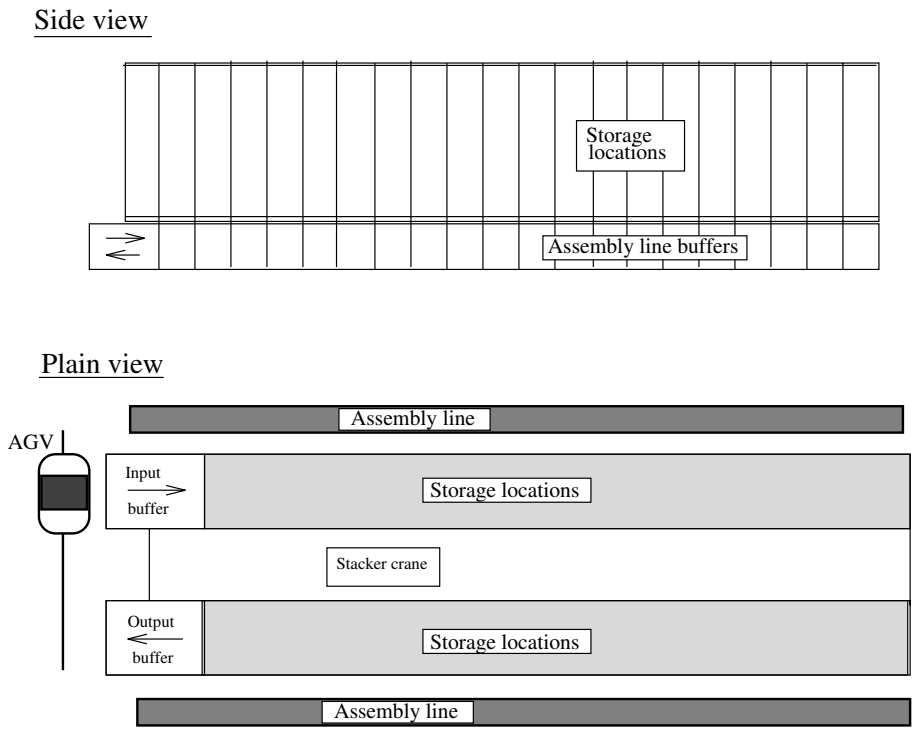


Figure 2.2.4 Sketch of an automatic storage system

As containers of different sizes have to be stored in the system, the storage locations are of different sizes, too (cmp. Figure 2.2.5). The storage locations for tote boxes, so–called **cells**, are of capacity three. Small boxes occupy one unit of capacity, large boxes two. Due to their larger height special purpose pallets can not be stored in locations for pallets and box pallets, but vice versa.

An **input** and an **output buffer** both of capacity two are located at the head of the AUSS. All transportation tasks within the system are performed by a (automatic) **stacker crane** that can move both horizontally and vertically simultaneously. Depending on the input at the reception area, the consumptions at the assembly lines and the degree of utilization of the storage locations the stacker crane has to transport containers

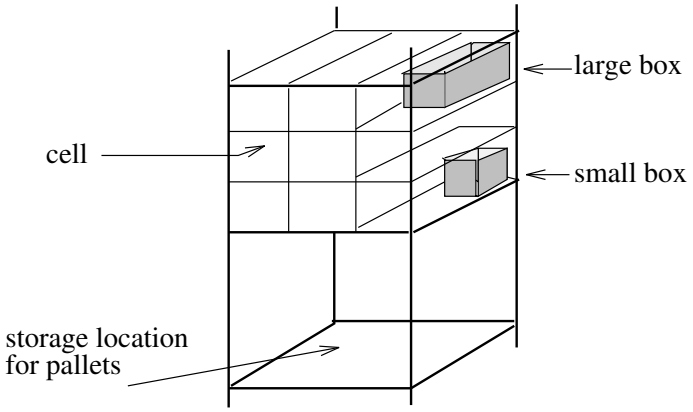


Figure 2.2.5 Storage locations within the AUSS

1. from the input buffer directly to an assembly line buffer (in case that items enter the AUSS that were requested but not stored in the system),
2. from the input buffer to a storage location (storage task),
3. from a storage location to an assembly line buffer (refill task)
4. from an assembly line buffer back to a storage locations (restorage task),
5. from an assembly line buffer to the output buffer (in case that the container is empty – retrieval task),
6. from one storage location to another storage location (in case that a relocation of the system is necessary – relocation task).

Light barriers and scanners recognize that a certain container is entering the input buffer or that some assembly line buffers are empty. In that case a transportation task is generated automatically. Some of the assembly line buffers are half–automatic where the personnel at the assembly lines have to press a button in order to generate a transportation task. Some others are so–called manual buffers that are refilled only if the request for a certain item is generated manually by the help of some software tools.

Assembly lines

An assembly line is located on each side of the automatic storage systems. They are partially equipped with robots (Figure 2.2.6), but the bigger part of the assembly is done manually (Figure 2.2.7). In case that robots assemble the devices the only manual interaction is the refilling of material from the assembly line buffers of the AUSS into the robot buffers. Except for the keyboards that need special machines (e.g., a laser for engraving the keys) each product can be produced on each assembly line. In practice this is not done, as each storage system would then have to be equipped with all parts of all products. Due to the variety in the product mix and the limited storage capacity this would be impossible. Thus, we have one assembly line for PC, Tower PC, monitors, data terminals and keyboards each. One assembly line is reserved for a final test and the packing of the devices.

The parts necessary for production are provided by the assembly line buffers of the AUSS. Assembly line buffers are prefixed and able either to contain tote boxes or the different types of pallets. In the first case, once the last box is pulled out of a buffer, the system detects by

means of light barriers that a new box has to be provided. A corresponding request is sent to the system automatically. As the pallets are too heavy to be removed manually from the buffers, a button has to be pressed in the case that they are empty. Then transportation tasks for the disposal of the container and for the refill of the buffer are generated. The control system tries to achieve that each part is delivered to a buffer that is as close as possible to the point where it is needed for the manufacturing process.

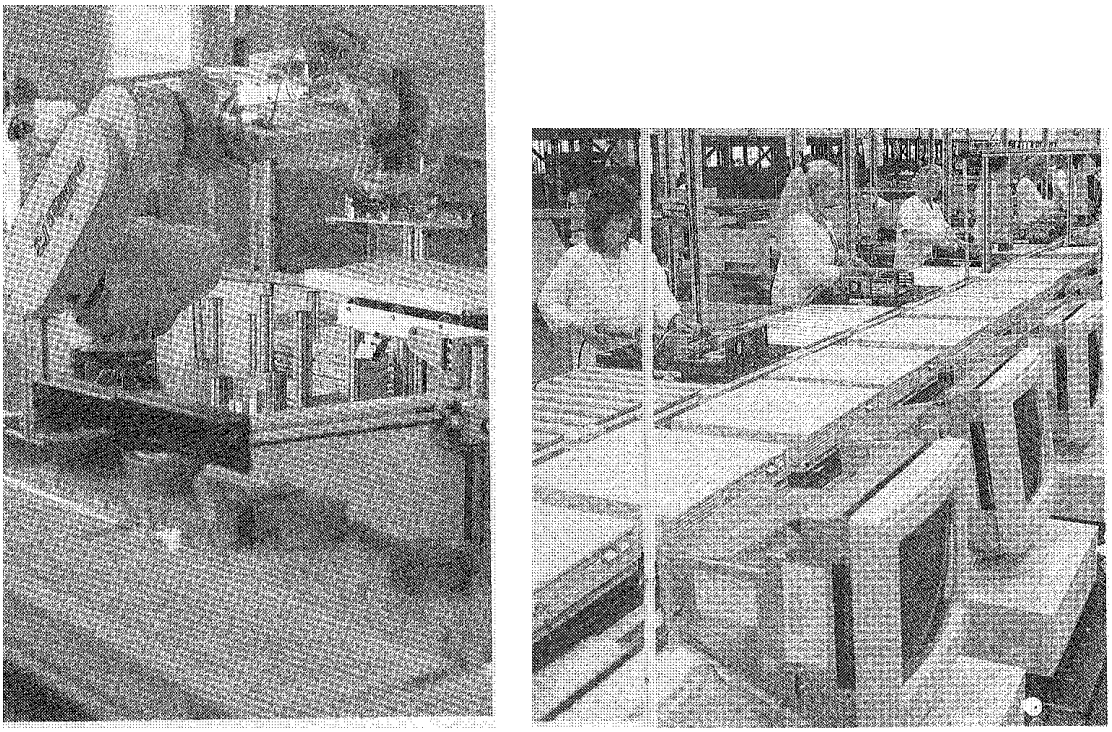


Figure 2.2.6 and 2.2.7 Assembly lines

Testing area and delivery

As soon as the PCs and other products except keyboards are assembled they are brought by conveyor belts to a testing area where at several levels tests concerning the full functionability of the devices are performed. In a first step some simple tests are performed, e.g., it is checked if the power supply is working, or if it is possible to address the floppy disk. If this is not the case, the defective devices are repaired immediately.

After passing this preliminary test the units run into an automatic test area, where for approximately 24 hours test programs are run under aggravated conditions, e.g., high temperature (Figure 2.2.8). If an error occurs during the testing period, it is recorded, e.g., on the floppy disk.

The architecture of this *run–in test* is similar to that of the AUSS. It consists of two aisles, each served by a stacker crane and storage locations on each side of the aisle. An I/O–buffer is located at the head of the run–in test. There are premounted storage locations of two

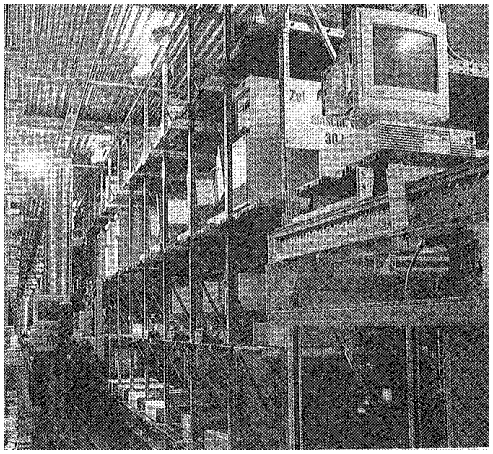


Figure 2.2.8 Run–in test

different sizes, suitable to store the two differently sized products (PC–Tower or PC / data terminals / monitors). The whole test area is divided into several electric circuits, each of them put under current for 115 minutes with a subsequent break of 5 minutes. The testing period of a device starts after its storage location is disconnected for the first time from the power supply.

In 1992 a second test area was built that differs in a few details. The I/O–buffer is not at the head of the system but in the middle and the storage locations are not fixed in size, thus it is possible to store the devices in a more flexible way.

After the completion of the testing period the device is brought to a final test where last checks are performed, including an analysis of the results of the run–in test, a short–circuit test, sharp focussing of the monitors, etc. Afterwards the units are packed and delivered.

2.3 Work–off strategies and optimization approaches

Besides a "good" layout of the system, the way how requests for the production units are worked off is a central aspect for obtaining good performance. If this is done in an unfavourable way, the whole system suffers. After analyzing the system at SNI several bottleneck components were detected and possible optimization approaches were developed. As the layout cannot be changed without expensive reconstructions, the suggested improvements address the work–off strategies.

The optimization approaches can be divided into two levels: decisions on a strategic or operational level. The first might be addressed in non–productive periods (weekends, nights, breaks, etc.) whereas the others have be to solved during everyday production.

Decisions on an operational level often lead to on–line problems as described in Chapter 1.5 because decisions have to be taken within a dynamically changing system and without knowledge about the future. The production process is not completely predictable as manual labour is involved, machines break down, parts are received too early or too late, received parts are defective, etc. Therefore, the control system is often confronted with the problem

of making decisions, without having full knowledge about the future, but which will have an influence on the future behaviour of the system.

The decisions on the strategic level are taken prior to the production process where you only have expectations about the future, but no complete knowledge. But as no dynamic changes of the system are involved during the process of taking decisions they are not considered to be on–line problems. These are problems with stochastic aspects.

In this section we describe the work–off strategies that were used for some components so far, and which were considered to be unfavourable. Furthermore, we sketch the suggested optimization approaches. A more detailed analysis will be given either in the following chapters or in the cited references. These optimization approaches were discussed with the production managers, foremen and computer scientists of SNI. As a result some of the approaches were chosen to be implemented, whereas others could not yet be realized.

2.3.1 Inner architecture of the storage systems

As described in the previous section, articles are received in differently sized containers. In the automatic storage systems (AUSSs) there are premounted storage locations for each type of container. The management has expectations about the products that will be sold during the next planning period. As the product spectrum changes, the parts that have to be stored in the system change too (in quantity and kind). In order to avoid bottlenecks, because storage locations for a certain kind of container are not available or locations are only available far away from the assembly line buffer where this article is needed, it is necessary to rearrange the locations. Approx. every half year a reconstruction of the storage locations is necessary. This is not done more often, as it is necessary to empty the system in order to rearrange the premounted locations.

Out of the expected sales and with the additional information how many containers of each type are needed for that amount of production, approximations on the minimal number of storage locations for each type of container can be given. Furthermore, the production managers have expectations as to which assembly line buffer will have a high throughput. Storage locations of that type should then be mounted close to that buffer in order to minimize the transportation moves of the stacker crane.

The optimization question is where to mount a storage location for a certain kind of container, such that all containers can be stored and the expected utilization of the stacker crane by the forthcoming transportation tasks is as small as possible. Some modelling questions concerning this topic can be found in Abdel–Hamid [Abd94].

2.3.2 Assignment of storage locations to incoming containers

The following strategy was used to assign storage locations to incoming containers.

As soon as a new container arrives in the reception area it is booked into the system and a storage location in one of the automatic storage systems (AUSS) is reserved. The main reason for that reservation is that it is necessary to check if there are capacities available to store the incoming container.

As different products (PC, monitors, keyboards, etc.) are produced on different assembly lines, it is clear in which AUSS the container has to be stored. Furthermore, for most articles it is clear in which assembly line buffer they will be needed. The nearest empty location to that buffer is than searched in a greedy way. For each incoming container the best location

is reserved. Containers with articles that do not have a fixed assembly line buffer a storage location as close as possible to the input buffer of the AUSS is reserved. This reservation cannot be changed.

This work–off strategy has several **disadvantages**. First, the reservation is done too early. Sometimes it takes several hours until the container reaches its destination in the AUSS. In the meanwhile the reserved storage location cannot be used for other purposes. Furthermore, even better storage locations might be available as the container enters the AUSS. But due to the early reservation they are not taken into consideration. Thus, a simple change in the organizational structure could result in an improved performance of the AUSS.

Moreover, it is a greedy strategy where only for one container the best location is found. The "good" reservations for some containers may block locations for others, which have to be stored in a "bad" location.

But as there are up to 50 containers at the reception area, that are not yet brought to the storage systems, it would be better to use a global approach taking all these containers into account. This could result in a better overall performance. We explain that by means of an example (cmp. Fig. 2.3.9).

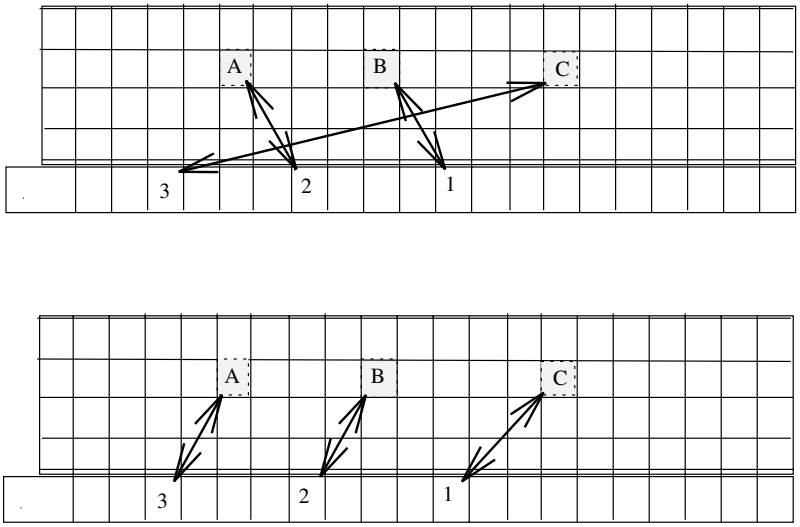


Figure 2.3.9 Optimized assignment of storage locations

Suppose three containers (1,2,3) arrive in that order at the reception area. They are suitable to be stored only in the locations A,B,C. The assembly line buffers where the items in the containers are used are labelled with 1,2,3. The use of the greedy strategy results in a assignment as given in the upper picture, whereas the assignment given in the lower part would be better. An optimization model, a detailed analysis of the problem and the related mathematical background can be found in Abdel–Hamid [Abd94].

Reassignment of storage locations to stored containers

In practice this is done only in the case that some of the tote boxes block a cell, and then only within that cell. But it does make sense to rearrange the assignment of containers to storage locations in case that no other transportation requests are present (e.g., at night).

Containers that are needed within the next production period should be brought as close as possible to the assembly line buffers where they will be needed, and storage locations close to the input buffer of the AUSS should be emptied for incoming containers. This might reduce the utilization of the stacker crane under heavy load conditions.

The problem of reassigning containers to storage locations is similar to the problem of finding optimal storage locations for incoming containers, just by assuming that all stored containers reenter the system. Thus, you have to assign all stored containers to an empty AUSS.

2.3.3 Transportation tasks scheduling for the stacker crane

The work–off strategy for the transportation tasks of the stacker crane was a priority based rule. As described in Section 2.2, the stacker crane has to fulfill certain transportation tasks differing in the location of their source and sink. To each generated transportation task a priority dependent on the type of task is assigned, cmp. Table 2.3.10.

priority	**type**
1	relocation task of boxes within a cell (for boxes),
1	combined storage– and relocation task,
2	retrieval task,
3	storage task,
4	refill task of assembly line buffers for pallets
5	refill task of assembly line buffers for boxes
6	relocation within the storage area,

Table 2.3.10

The task with the lowest priority value is always scheduled next. If there is more than one task with the same priority, the one with the earliest generation time is scheduled next. The priorities of the tasks do not change except this is done manually.

This work–off strategy has several **disadvantages**. First, it implies that tasks with a higher priority value always have to wait until all tasks with a lower priority value are performed, although they might be generated much later in time. This resulted in a lot of manual interferences where the priority of a task that had already been waiting for a long time was changed by the foreman responsible for the AUSS.

Another problem is that this strategy does not allow to schedule transportation tasks of higher priority value in between, although this might result in a shorter overall travel path of the stacker crane. This is described by means of Figure 2.3.11.

Suppose we are given the four transportation tasks depicted by the arrows, where the tail gives the location of the source, the head the location of the sink. The dashed lines in between represent the unloaded moves of the stacker crane, the dot its current position. The stacker crane travels along the arrow (performs a transportation task) and then moves unloaded along the dashed line from the sink of the last task to the source of the next. Once these tasks are given you have no influence on the loaded parts of the moves, as the containers have to be brought from their source to their destination. But you have an influence on the unloaded

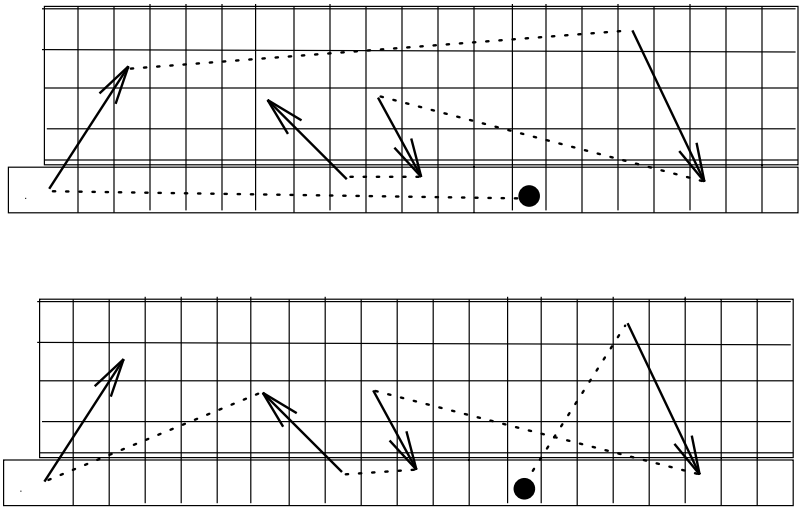


Figure 2.3.11

parts of the moves. A control of the stacker crane due to the priority rule might result in a travel path as depicted in the upper part, whereas an optimized strategy results in a shorter overall travel path (lower part of the picture). The question of minimizing the unloaded travel times in everyday production leads to the question of solving on–line Hamiltonian path problems. The modelling and the related mathematical problems will be discussed in detail in Chapter 4.

2.3.4 Optimization approaches for the AGV

The automatically guided vehicles (AGVs) turned out to be a critical component in the system as some layout decisions and the selection of the work–off strategies were not carried out in a way to achieve the best possible system performance. Due to organizational reasons within SNI (the layout design could not be changed, and the design and implementation of the control system was carried out by another company) we could not attack the problem, although we considered it to be central. We briefly describe the disadvantages of the current control system.

As there are no alternative tracks, the vehicles are often blocked by other vehicles that stand in front of a station and have to load or unload a container. This process takes approx. 30 seconds (including the precision stop) and during that period all following vehicles have to wait until this action is carried out. Furthermore, not all potential properties of the system that are physically installed are exploited. Several switches are not used because experience showed that their use (under the current control system) often led to heavy congestions.

One of the tasks of the central AGV computer control system is to assign vehicles to generated transportation tasks (jobs). This is done in a greedy way as described in the following.

Strategy for assignment of vehicles to jobs:
Each station where containers have to be picked up has a certain prescribed priority reflecting the relative importance of that station. Each generated transportation task contains, among

others, the following information: time of generation, source and destination and priority of the task. This priority is initialized with the priority of the task's source station. This transportation task is added to the set $T = \{T_1, T_2, ..., T_n\}$ of all generated but not yet worked off tasks. A task is booked off once the corresponding container has been loaded onto some vehicle. Once a task is finished it is deleted from this set and the priority of all remaining tasks in T is increased by one.

From the set T of all tasks only a certain subset $T_A \subseteq T$ is considered for an assignment of tasks to vehicles. In practice the strategy is to assign those tasks whose current priority is exceeding a certain limiting value. Let $Prio_i$ be the priority of task $T_i \in T$, $Prio_{max} = max\{Prio_i \mid T_i \in T\}$ be the highest current priority and let FG be a fixed value that can be changed manually by the system operator. Then the set of assignable tasks $T_A = \{T_i \in T \mid Prio_i > Prio_{max} - FG\}$ contains only those tasks whose priority is "big" enough. Only the tasks in T_A can be assigned to vehicles. The aim of this strategy is to assure that important tasks are scheduled early (high priority for the source) and that unfavourable tasks, i.e., tasks with high cost for each vehicle, are assigned as well. This is assured, since the number of tasks in T_A decreases as the priority of one tasks raises.

The assignment of tasks in T_A to vehicles is done as follows. For each vehicle a cost for moving from its current position to the source of each task in T_A is calculated. This cost coefficient is calculated out of the shortest distance between these points (relative to the one–way system of the AGV), the importance of a certain travel route and the speed of the vehicles. For each disposable vehicle, starting with the vehicle with the lowest internal number, the transportation task in T_A with the lowest cost coefficient is assigned. Certain side constraints (e.g., upper and lower bounds for the number of vehicles in each hall, upper bounds for vehicles with the same destination, etc.) are fulfilled. While the vehicles are moving through the system, or tasks enter T_A, or tasks are deleted, the internal cost matrix is updated. As long as a vehicle has not loaded the container the assignment can be changed in case a more favourable task is generated.

The aim of an *optimization strategy* should be to guarantee a smooth run of the system, such that the generated tasks have to wait for a period as short as possible until they are finished. This results in an "optimal" assignment of tasks to the vehicles (to avoid long waiting times of the containers at the stations) and an "optimal" routing of those (to avoid long transportation times). Up to now possible congestions have not been taken into account in both operational decisions. Furthermore, it should be possible that tasks of blocked vehicles are assigned to other vehicles, the one–way system should be abolished, and a variable rule on right–of–way (now "First–Come–First–Served") should be introduced.

2.3.5 Optimization questions in the test area

As soon as a device enters the input buffer of the test area it is necessary to find a storage location. The control system always chooses among all available locations the one that is closest to the input buffer. Retrievals are performed following a "First–In–First–Out (FIFO)" strategy. This results in the fact that storage and retrieval tasks are not combined in a favourable way to avoid unloaded moves of the stacker crane. It could happen that the crane is moving unloaded from one end to the other in order to pick up a device, although it would have been possible to retrieve a device just next to the location where the incoming device was stored.

The shelves in the test area are divided into power circuits that are put under current for 115 minutes followed by a five minutes' break. The times when a circuit is switched off circulate in the system in a fixed order. Every one and a half minutes another circuit is switched off. The test period of a device starts as soon as it is switched on for the first time in its storage location. The current storage strategy does not take these circuits into account. The "best" storage location is that one closest to the input buffer, or equivalently the one resulting in the shortest move of the stacker crane. In the worst case the device has to spend almost two additional hours in the test area. A better strategy, that can easily be realized, is to store the device in a circuit that will soon be switched off. Just by this simple organizational change it should be possible to reduce the period the devices stay in the test area by approx. one hour, thereby increasing the throughput and improving the utilization of existing capacities.

In the management of the new test area an additional optimization question occurs. The incoming devices have different heights. As there are no premounted storage locations, it is possible to store them at any possible combination. In addition to the question of finding a storage location in a circuit that is soon switched off, the devices have to be stored in such a way that there is as little space wasted (no devices can be stored there) as possible.

So far, the strategies have been looking for storage locations for one device. But similar to the storage question occurring in the management of the AUSS all incoming but not yet stored devices should be taken into consideration. But doing so a severe problem is that one has to deal with "time–dependent" cost functions as the "quality" of a storage location has to be modified as soon as a another circuit is put under current. A certain location can turn from "good" to "bad" within seconds. Therefore, it seems difficult to define a proper model to take these effects into account.
A detailed analysis of these questions and a comparison of several heuristics can be found in [KM93].

2.3.6 Other approaches

Optimized reception of containers

Another strategic decision that is closely related to the topic mentioned in Section 2.3.1 is how many items should be stored in a container as this influences the minimal number of locations needed for that type of container. As only a limited number of products contains a special article, e.g., a certain CPU, a half–emptied container may block a storage location for a long period of time, until it is needed again for production. If there are fewer items in a container, a larger number of containers is needed in order to achieve the same amount of production. Thus, the system will be overloaded.

Another question is when a certain container should be received. If they arrive too early, storage locations within the AUSS are blocked for a certain period of time, if they arrive too late production might be delayed.

Thus, the question is to find an "optimal" load for each container and an "optimal" receiving date.

Strategies at the assembly lines

Another aspect that influences the overall performance of the system concerns one of the central parts of the manufacturing system – the assembly lines. The production of the PCs,

etc. influences all other parts of the system. Used parts are re–ordered and the received containers pass through the components receiving area, AGV, and automatic storage systems. The assembled devices are the input for the test area and are afterwards brought to the delivery zone.

Therefore, it seems central to optimize the behaviour of the assembly lines. There are two possible parameters that are suitable to optimization approaches: the choice when and where to start a certain product on the assembly lines and the distribution of workers on the different lines.

But the production managers did not consider these questions to be that important, as there are not that many possibilities to vary in the load–in of the assembly lines or in the distribution of workers. Furthermore, the experience based decisions taken by the production managers and foremen were considered to be good enough. Therefore, only a simulation–based analysis tool was developed which will not be described in detail.

Master optimization process

Suppose that all the optimization tools mentioned in this section are realized. Then there exist a dozen of optimization programs which are either competing against each other or are hierarchically organized. A major point of future research is to find a "master optimization–tool" controlling all the local optimization programs. This master process has to tune all the local optimizers, to feed them with the appropriate parameters, in order to optimize the overall production system. In some cases it might be better to accept a suboptimal solution because it does not cause that much problems to neighboring components (e.g., it might be better to accept longer travel paths for the stacker crane of the AUSS in order to avoid a blocking of the input buffer, which might cause congestions of the AGV, as vehicles cannot unload their containers). In our approach this control function is realized by a simulation program that checks the influences of decisions taken locally on the whole system. But this is just a "trial–and–error"–method. You observe that some decision taken locally influences the behaviour of the whole system in an unintentional way, and then you change the input parameters for the local optimizer and rerun the procedure.

Chapter 3

A simulation model for the FMS

3.1 Introduction

The necessity of simulation

In most problems of flexible manufacturing we are confronted with rather complex systems consisting of several components that interact in a nontrivial way. In case that strategic or operational decisions have to be taken, it is necessary to "predict" the influence of these decisions on the behaviour of the system. From a theoretical point of view this could be done by performing experiments within the real system. But these experiments are often too risky, too time–consuming, involving immense costs, or even not possible – especially if the system is not yet realized and alternative layouts have to be analysed. Therefore, it is necessary to model the whole system in order to forecast the effects that a change of one component will have on the whole system or to obtain quantitative results about the behaviour of the system.

It would be good to have an overall mathematical model for the system. But with today's insight into these systems and today's mathematical methodology (cmp. Chapter 1.4.2) it seems illusory to develop a mathematical model that captures all the important aspects of a complex FMS (as the one described in Chapter 2) and that can be attacked by algorithmic methods. Due to their high complexity and their dynamic nature, these systems cannot be fully studied by analytic models, such as mathematical programming or queueing network models. Therefore, most analytic models are based on simplifying assumptions and often do not capture the complexity of reality, or cannot be treated with algorithmic tools.

As no mathematical model is available, we restricted ourselves to something else, namely simulation techniques. These techniques are very common in industrial applications and are widely used, e.g., as a design aid, to validate designs, to troubleshoot, or to gain insight into operations.

In general, **simulation** is the reproduction of a real system by means of a computer program. Although no mathematical optimization techniques can directly be applied in the simulation model it has the advantage that it is an (almost exact) copy of the real system. Within this model it is possible to perform the experiments that are not possible within the real system, and therefore to verify the quality of the solutions found by the optimization programs, measure the achieved improvements, and study the on–line behaviour of the system.

Simulation models

There are several types of simulation models that might be classified into time–dependent (dynamic) or time–independent (static or "Monte–Carlo") simulation models. In the **static models** it is assumed that the states of the system are the (time–independent) results of the input of certain values to an "algorithm". These models are, e.g., used for the evaluation of complex integrals.

In the **time–dependent models** we assume that the states of the system develop independently within the considered period of time. Dependent on the transitions of the states of the system we distinguish between discrete and continuous simulation models. If the transitions take place only at discrete points in time, we talk about **discrete simulation models**, if they take place continuously they are considered as **continuous simulation models**. In the latter models time is considered to change continuously and they can often be described by means of differential equations. This is, e.g., the case in models for crystal growing, crash tests in automobile industry, or aircraft models for experiments in a wind tunnel.

Within discrete simulation models two main approaches are widely used, namely **event–oriented** and **process–oriented simulation**. In the second the system is represented as a set of processes through which a set of objects passes in the simulation model. This is somehow a general modelling technique, but if detailed models are required this approach often fails, as it produces large scale models when all interdependencies have to be taken into account. Beside that, debugging and tracing is rather difficult within these models. Therefore, the event–oriented approach is often used to model a complex system in detail.

In many applications in manufacturing the state of the system does not change continuously but only at some distinguished points of time (events). Therefore, it is only necessary to observe and update the state of the system at these discrete points in time. This method is called **(time) discrete event simulation** and is described in detail in Section 3.2.

Another distinction can be made in the way the transitions can be described. If the outcome of the activities is completely predictable, we call it **deterministic model**. If the effects of a certain activity vary over a set of various outcomes, we say that it is a **stochastic model**. In any case it is necessary to determine either the rules or probability distributions causing the transitions.

Advantages of simulation models

If modelling is done in a proper way, simulation models can be a helpful tool in the treatment of problems in flexible manufacturing. As from a theoretical point of view it is possible to model exactly all knowledge about the behaviour of a possibly rather complex system, these models can be considered as an almost exact copy of the system. Certainly it is not possible to exactly reconstruct a complex system by means of a computer model. But as the model only has to reflect the most important aspects of the system it should be possible to construct a suitable model for the special application you have in mind just by modelling the aspects that you consider to be important.

Once we are given a computer model we are in the situation that we can forecast the consequences of changes in the system by performing experiments within the model — experiments that would have been impossible, too expensive or too risky in the real system.
The advantages of a simulation model may be summarized as follows:

- it is an almost exact copy of the real system,

- one might obtain a system wide view of the effect of changes within the system,
- it is an experimental laboratory to answer "what–if" questions in the design and control of a manufacturing system,
- modifications in the model are easier to perform, less expensive and less time–consuming than modifications of the real system,
- results are obtained faster as model time runs faster than real time,
- it is easy to take stochastic aspects (e.g., machine breakdowns) into account,
- it is possible to learn something about the behaviour of a dynamic real–life system,
- a tool for studying the effects of on–line optimization of the system is provided,
- if the model is as exact as possible the understanding of the system is improved and the influence of parameters on system behaviour is illustrated,
- a validation tool to analyse the solutions of the analytic models is provided.

And as a positive side effect it is easier to convince the management with the help of a simulation model. The simulation model looks like the real system and is not such an abstract representation as a mathematical model. Therefore, simulation models may be easier to accept and understand than their mathematically–based counterparts.

Limitations to simulation models

But certainly there are not only advantages of simulation models. This approach of modelling also has his limitations and restrictions. We list some of them.

The first is that a simulation model just leads to a "trial–and–error" optimization method. You just input some parameters, see how the system reacts, and if you are not convinced with the results you change the input parameters and rerun the procedure. A goal would be to develop a combined "simulation–optimization" approach where the process of adjusting the input parameters in order to optimize a certain output parameter is automatized. To our knowledge this has not been done so far.

Beside that, the process of building a "good" simulation model is in general very time–consuming and difficult as a good compromise between accuracy and computing time has to be found. This has to be done with respect to the specific application you have in mind. If the model considers too many parameters and facts that are not important for the conclusions, computing times will be too high. Some authors take the view that more time is spent on building and debugging the model than on running it and that therefore the running time is unimportant (e.g., Swan [Swa91]). But we do not completely agree with that statement. It depends on the specific goal one pursues. If the simulation model is used in the design stage, investments in shortening the simulation's CPU–time are often economically not justified. But if the simulation program is used on the operational level in the on–line control of a manufacturing system a short CPU–time is essential.

An important but time–consuming step is the verification of the model. Here it is tested if the model reflects the real system exact enough. This decision has to be taken with respect to the questions you want to analyse. This can be done, e.g., by a comparison of simulation

results with measurements in the real system. If the model does not reflect the system sufficiently, as parameters or facts are missing or modelled in the wrong way, it is not possible to take the conclusion you want to take. Furthermore, the model should only be used for conclusions that are within the verified and validated range.

3.2 Discrete event simulation

In this section we describe in more detail the steps of building a discrete event simulation model. This is done together with the description of the simulation control as it is difficult to separate them. The concepts described in this section are also the basis for the simulation package AMSEL that was developed and implemented by the author. The main aim will be to explain the principles in an easily understandable and clear way without being too formal.

Basic idea of the simulation method

For the simulation of the material flow within a system consisting of components that are linked in a fixed way (discrete) event oriented simulation seems to be the proper method. The run within the model is dependent on the transitions of the states of the real system. These mainly occur when a part enters or leaves a component.

We introduce some notation. The components that are permanently at a fixed location in the system (work stations etc.) will be called **modules**. The elements that are only temporarily in the system or are moving through the system (e.g., automatic guided vehicles, workpiece carriers, containers) will be called **objects**.

The **event points** are all the points in the system where the state of the system changes in a way that is important for the special application, whenever an object passes that point. If an object passes that event point, we say that an an **event** takes place. Events always take place when an object passes an event point — these events will be called **route events**— but also other incidents, such as breakdowns, cause an event. These will be called **system events**.

The control of the simulation run is performed by a so–called **event list** that contains all events that are known at a certain point in time, but that have not yet occurred. The content of that list is time–dependent and events are sorted by the time they occurred.
The transitions between the states of the system are performed by so–called **event routines**.

A **simulation step** or **iteration** is performed in the following way:

(1) Get the next event from the top of the event list.

(2) Adjust the simulation time to the time when this event takes place.

(3) Call an event routine that initiates (starts) all activities that this event causes (update of the state of the system, update of the event list etc.).

(4) Check if the desired simulation time is reached or if the event list is empty.
If yes, then STOP, else GOTO (1).

To simulate a material flow system it is necessary to describe the three main elements of the system, namely the physical layout, the parts moving through the system (objects) and the logical control in a way that they are suitable to an algorithmic realization. The

physical layout is represented by the event points and the connections between them, the logical control is performed by the event routines, and the objects are described by a data record that can be identified by an internal object number.

Model building

The first decision in the model building process is to decide with which precision the real system has to be modelled. This implies the decision which components of the physically existing system should be mapped to modules and event points. Furthermore, all the control elements that are important for the application have to be identified and have to be included into the model. Identification of event points and control elements cannot be done separately but have to be complementary to each other.

Modules and event points:
In a first step the real system is divided into single components (modules). The event points are derived from the decomposition into the modules. But they are not yet exactly determined by the modules but differ in the desired accuracy. E.g., the number of event points is dependent on the same proportion as congestions in front of the modules (e.g., work stations) are modelled.
In general, the following different **types of route event points** occur:

- points where an object enters a module,
- points where an object leaves a module,
- points where possible congestions can occur,
- points where objects enter or leave the system.

Each module has at least one **entry event** and one **exit event**. More entry or exit events can be possible, e.g., in the case of switches. In case that there are points within the modules that are important for the control of waiting queues we have additional **neutral event points**.

To each event point a **waiting queue** is associated that is identified by the internal number of the event point. The waiting queue contains at each point in time the sequence of all objects that are waiting between this event point and its predecessor. These waiting queues are activated if an object passes a certain event point.
Each event point is described by a set of attributes, such as

event number:	Internal number of the event point.
event type:	To each event point a type is associated, indicating if it is an entry–point, exit–point, neutral–point, or any other type of event point.
module:	Whenever an object passes a certain event point (enters or leaves a module) it might be necessary to increase or decrease the actual load of modules. Which modules are affected is an attribute of that event point.
successors:	Each event point has predecessors and successors. They are defined in a natural way, namely as a subset of all event-points, that an object could have passed immediately before coming to that event point or as the event points an object can move to from this event point.

capacity: Each event point has a certain capacity that is nothing but the number of objects that can be in the section between this event point and its predecessor at the same time.

waiting queues: If an event point is passed by an object, it might be necessary to activate an object in a waiting queue.

special procedures: There are event points where application–specific procedures have to be called. A pointer to a specific function and the parameters are further attributes of an event point.

The above list only contained **constant attributes** of an event point. These attributes are initialized at the start of the simulation and do not change while the simulation program is running. But the data record also contains **variable attributes** that are changing during the simulation run. These are, e.g., the number of objects registered at an event point, a parameter of the identification for the waiting queue whose head has to be activated next, pointers to the head and tail of the waiting queue associated to this event point.

Event types and event routines:
Basically, two different types of events take place. On the one hand an event takes place whenever an object passes an event point (route events). On the other hand all other incidents that influence the state of the system are treated as events, e.g., the begin and the end of a breakdown. They will be called system events. The logical control of the events is performed by event routines that might call special procedures that are dependent on the application. The event types given in Table 3.2.1 are necessary for the simulation of manufacturing systems.

route events :	
entry event	an object enters a module
exit event	an object leaves a module
neutral–event	an object reaches a point where it might block other objects
generation–event	an object enters the system (is generated)
quit–event	an object leaves the system
system events :	
S–event	beginning of a breakdown
s–event	end of a breakdown
request event	a request for some system component was generated (e.g., transportation tasks)

Table 3.2.1 Event types

For each event type an event routine exists that performs all the transitions in the state of the system, the manipulations on the objects etc. that are caused by that certain event point. This is in particular the update of the state of the simulated system, the update of the object data, the insertion, deletion or delay of other planned events, and the actualization of the model time. If it is a route event, it also has to handle the waiting queues (activate objects in a queue, or insert an object into a queue).

The application–dependent special procedures are controlled by a single interface, which is fed by the event routines with a pointer to the special routine and parameters for that procedure.

Objects:
The elements moving in the system are called **objects** and are identified by an object number. When an object enters the system a data record containing all necessary information about that object is generated.

Event list:
The simulation run is controlled by a so–called event list that contains so–called **event notes**. An event note contains the times of occurrence of an event, event type, event number, and the number of the object causing that event. By successively deleting the event notes the activation of the event routines associated to a certain type of event is performed. This routines then perform all necessary updates of the data structures.

3.3 Simulating the FMS with AMSEL

Software survey

In 1988 there were more than 150 commercial software tools available, varying from general purpose simulation programs (such as to SIMULA, GPSS, and SIMSCRIPT) to specialized simulation tools. More than 20 of them were oriented towards manufacturing systems (e.g., Siman, See_Why, Simfactory, XCell, Witness2) (see [AG]).

In these "simulation languages" either already existing high level programming languages are extended with a set of procedures in order to realize the simulation tasks, or independent languages were developed with interfaces to high level programming languages. E.g., SIMULA was developed out of Algol 60 and extended with language elements for the simulation. GPSS (General Purpose Simulation System) was developed in 1962 by IBM with interfaces in the form of FORTRAN procedures. For a detailed review on the most popular commercial packages see [AG, Swa91].

These tools are user–friendly. Most codes are equipped with graphical user interfaces. It is often possible to graphically describe the model ("draw the model") or graphical output is provided beyond character–based output. This is especially true among the specialized products, e.g., for the simulation of manufacturing systems. The graphical output may consist of block diagrams, charts, output statistics, and time plots of selected statistics. A growing number of tools support animation what is valuable for selling the system, debugging, verification, and trouble shooting. The trends in commercial codes go towards an increased specialization in simulation software and an increased integration between simulation software and other programs. Increasingly simulation languages will be able to accept input from a variety of sources (CAD, spreadsheet files) and write output to other programs (databases, spreadsheets) to facilitate analysis and report writing. Up to now expert systems are used as a part of a simulation very seldom and little attention is paid to interpreting the outputs in a statistical sense.

But for the convenience in input and output you have to pay in other aspects. The first is computing time, the other the possibility to model exactly large and complex systems. Due to the graphical overhead, e.g., for the animation, the computing times are relatively close to real–life and to be effective the run length has to be limited.

All the commercial tools did not allow to model the manufacturing system that was described in Section 2.2 as exact as it was necessary for our purpose, at least not within an acceptable computing time. Large complex systems are hard to be modelled with any of the

general purpose simulation tools, e.g., [Kle92] reports on a simulation model for a (not such complex) manufacturing system developed in SIMULA that took a computing time of six hours on a mainframe. As far as we know the specialized tools do not offer all the possibilities that were needed for our application either.

What is AMSEL ?

As we intend to use the simulation program as an on–line control instrument for a complex manufacturing system, we need to have a simulation program that is **fast, exact**, and **flexible**. It should be possible to simulate two shifts of the whole system within a few minutes. As decisions have to be taken that are based on the results of the simulation model, it has to be as exact as possible. As alternative operational strategies (as well as layouts) have to be compared, the system should be flexible enough to take changes of the real system into account easily.

Therefore, we developed the software package AMSEL (A Modelling and Simulation Environment Library). Here data structures and procedures are provided that take over all the simulation tasks (update of the event list, waiting queues etc.). The AMSEL–procedures are implemented in C and are provided in form of a callable library. The user has to write additional procedures that take the special features of the modelled system into account.

A preliminary version was developed by P. Bauer (now University of Cologne) and was used in a joint project with industry on the simulation and optimization of an assembly line of printed circuit boards (see [Bau90]). The concepts developed there were further elaborated and a simulation software package was designed and implemented that differs in some aspects from the commercial codes.

On the one hand simulation programs developed under AMSEL require less computing time. Therefore, it is possible to model bigger systems than is possible with the commercial codes. This enables us to use these simulation programs as a validation tool for the optimization procedures, and as a planning instrument for everyday production. On the other hand the programs use as much as possible exact and realistic parts instead of stochastic components. Therefore, a high accuracy in the modelling of the real system can be achieved.

But this certainly requires some insight into the system that has to be modelled. A careful study of the system is inevitable. As all overhead in the input and output of the data is avoided, some insight into the simulation technique is needed to set up all necessary input data. But the simulation programs implemented with AMSEL turn out to have a good performance when they are used for the simulation of material flow systems with work stations linked together in a fixed way. This enables the direct exploitation of the specific structure of such systems and to support the demands of speed and accuracy. But certainly the use as a general purpose simulation tool suffers due to these aspects.

We believe that our goals concerning speed, accuracy and flexibility of the simulation program are fulfilled by AMSEL. The computational results in the next section show that the programs are fast and exact. Flexibility was achieved by a clear distinction between the data area, containing the layout and all input parameters, and the program area, containing all the logical control elements of the system. For more details see [Asc92].

Example: A simulation model for an Automatic Storage System

To illustrate the theoretical concept given in this section we explain in detail a simulation model for the automatic storage system (AUSS) that was described in Section 2.2. The aim of this simulation approach should be to model the movements of the stacker crane in order to compare different operational strategies for performing the transportation tasks.

Although the movement of the stacker crane is continuous the states of the system AUSS change only at discrete points in time (stacker crane loads or unloads a container, a container enters or leaves the system etc.). Therefore, discrete event simulation is a proper technique for modelling the system. As the system cannot be mapped one–to–one, certain simplifications have to be performed that do not destroy the validity of results based on that model. They will be described in the following.

The elements that are moving through the system are the containers. Thus in the following a container is synonymous with an object. The stacker crane is assumed to be fixed in its position. The time between the loading and unloading of the container is just the time that the stacker crane needs to travel to its destination.

The system AUSS is divided into different modules in such a way that the desired accuracy in which you want to model the system is achieved. In our application these modules are input buffer, output buffer, stacker crane, and storage locations. The whole system is not closed but connected to the environment as it receives input from it and outputs to it. Each module has a certain capacity as given in Table 3.3.2.

module	**capacity**
input buffer	2
output buffer	2
stacker crane	1
storage locations	∞
environment	∞

Table 3.3.2

The input buffer for example has capacity two, which means that two objects (containers) can be in that module at the same time. A third object has to wait in front of the buffer until there is free capacity again, i.e., it is inserted into the corresponding waiting queue. The storage locations are assumed to have infinite capacity as only containers enter the system that can be stored. Therefore, there will never be capacity problems in storing incoming containers. The environment is assumed to be capable of accepting all containers leaving the system at any time. Therefore, it also has infinite capacity. Each module (except the environment) has one entry event (E) and one exit event (A). Each time an object passes one of these events the state of the system changes and has to be updated.

Figure 3.3.3 gives a sketch of the linkage of the event points and a simplified description of the interactions between the modules. Solid lines indicate that there is a uniquely determined successor of an event point. Dashed lines indicate that a certain event point has more than one successor, and a special procedure has to determine the one that will be scheduled next.

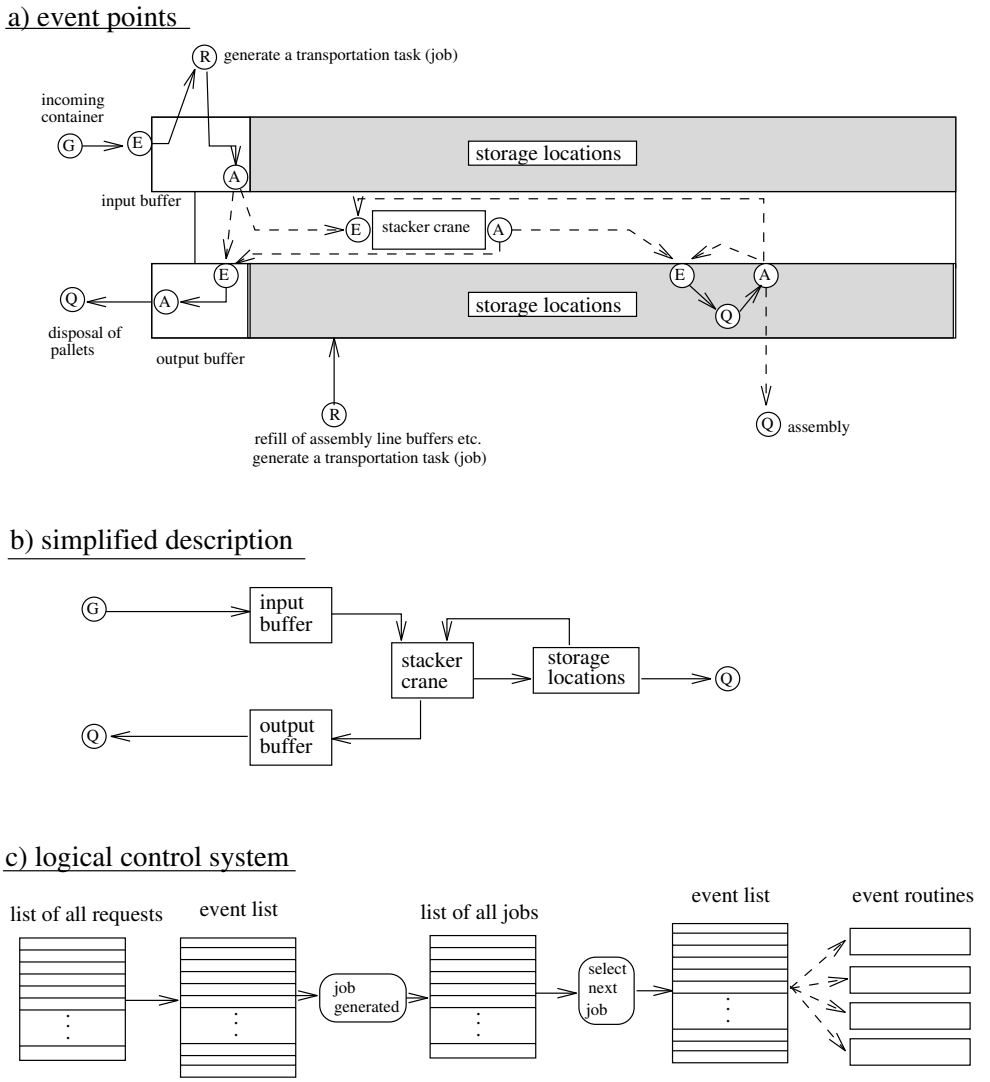


Figure 3.3.3 Simulation model for the AUSS

The system receives its input via the input buffer. All incoming containers pass first a generate–event (G) where a data record is generated that contains all necessary information about the object, e.g., the exact storage location. After entering the input buffer a transportation task for the stacker crane is generated (request event (R)). In case that there are some problems with the container (the contour exceeds some limiting values, it is too heavy etc.) the stacker crane just passes the container to the output buffer. In all other cases the module "stacker crane" loads the object (E–event), transports it to its destination, and delivers it there (A–event followed by the E–event of the module "storage location"). The travel time of the stacker crane, or equivalently the time difference between the E–event and A–event of the module "stacker crane", is dependent on the length of the theoretical move of the stacker crane and is calculated by a special procedure. All other durations, e.g., the loading of a container, or equivalently the time difference between the A–event of the input

buffer (or storage location) and the E–event of the stacker crane are fixed values that were measured in the real system.

Once a container enters a storage location it will remain there for an unpredictable period of time and it is booked out of the simulation system (quit event (Q)). In case that it is needed again it is reactivated by the generation of a transportation task (request event (R)).

If an object has to leave the system, it either leaves it via the assembly line buffers or via the output buffer. In any case a Q–event is performed that, e.g., deletes the corresponding data record.

The logical control of the system is performed in the following way (see Figure 3.3.3.c). The simulation run is triggered by the generation of the transportation tasks. In a list (or file) all requests and generation tasks are stored that take place during the simulated period. In our case these tasks were recorded in the real system. In case that they are deterministic this list also contains all breakdowns of system components. This is, e.g., the case if you want to reproduce the behaviour of the real system with recorded data from everyday production. Always the next request in time is added to the event list. If that job is generated (the corresponding event routine is called), it is added to the list of all generated but not yet performed jobs and the next request is loaded into the event list. Among all available jobs the next one that has to be performed is selected, using, e.g., the priority strategy described in Section 2.3.3 or the optimized strategy described in Chapter 4. The corresponding event (e.g., A–event of the storage location) is added to the event list. If this job is performed, the next job has to be selected, and so on.

Computational results using this model will be given in the following section.

3.4 Computational results

Validation of a simulation model

Once a simulation model is developed it has to be decided if the model is a good representation of the real system. It is not possible to exactly reproduce the system, as some technical data is simply not available, was measured in the system with the help of a stop watch, or was estimated by specialists. Furthermore, not all breakdowns and all manual interferences were recorded. Therefore, the goal of the validation process is a fine adjustment of all the technical parameters that were not determined exactly and verification if the layout and all the logical interactions between the technical components have been modelled and implemented correctly. The system should be modelled in such a way that for any production day the behaviour of the real system is reproduced in the simulation model in the best possible way.

This process of validation is a lot of work and very time–consuming. The most accepted approach is the so–called **trace–driven simulation** where you feed the system with real–life data, run the simulation with that data, and compare the simulation output to the real–life output. Certainly the more real–life data you input the more powerful the test will be.

In principle there are two possible strategies to perform the validation. The first is to simply “eyeball” the behaviour by comparing parts of the systems one by one, or to apply methods of mathematical statistics. But to obtain the full potential of statistical methods they often require that the observations are identically independently distributed. As this is not the case with most of the data, we restricted ourselves to comparing just the mean values and taking the deviation into account. This strategy was accepted to give results that were good enough.

What we did exactly was the following: First we performed so–called **individual validation runs** where during non–productive periods, objects were fed into the system, and their running time was measured with the help of stop watches. This was done for the AGV, e.g., when during weekends engineers of the SNI support team ran the system just for that purpose. This gave us approximations of the times when no congestions, breakdowns and manual interferences were involved. Then **global validation runs** where performed where during everyday production data was protocolled and measured. This data was inputed to the simulation model and the simulation results were compared with the recorded production data.

But the amount of data that is produced in that way is immense, and therefore it can not be documented here in its entirety. We restrict ourselves to give just some of the results. We concentrate on the model of the automatic storage system as described in Section 3.3.

Validation results for the AUSS model

For one week all requests and movements of the stacker crane in everyday production were recorded and the times of the moves in the simulation model were compared with the times of the moves in the real system. Table 3.4.4 summarizes the validation results for this real–life data of one week.

	# jobs	ØSNI	ØSIM	ØDEV	max DEV	% DEV	Time
1	416	76.27	76.83	2.42	20	0.73	11.47
2	423	75.45	76.34	1.96	11	1.17	11.60
3	403	78.81	78.95	2.01	9	0.18	11.17
4	399	76.19	76.58	2.09	9	0.51	11.08
5	450	77.11	76.73	2.06	20	0.49	12.30

Table 3.4.4 Validation results for the AUSS–simulation–model

One job consists of the unloaded positioning move, the pick–up of a container, the loaded move with the container, and the delivery of the container. The average time that was needed for a job on day 1 was 76.27 seconds (see column ØSNI in Table 3.4.4). This time was obtained by measuring all operations of the day and averaging them out. In total 416 transportation tasks were recorded during day 1. The simulation model produced an average job length of 76.83 seconds (column ØSIM) for this day resulting in 0.73 % deviation (column % DEV). Deviations are bound to occur since always some interruptions and manual interactions occur that are not taken into account by the simulation model; see also column ØDEV for the average and column max DEV for the maximum deviation that occurred throughout a day. The CPU–time needed for running the simulation model for the whole day on a SIEMENS PC MX300 is given in column Time. It is about 12 seconds.

One can observe that the system behaviour was reproduced relatively exactly and the results were considered to be very satisfactory by our industry partners and it was decided to accept proposals that are based on the use of this simulation model.

Some further figures

The simulation model for the FMS covering the reception zone, AGV, and all three AUSS was constructed in a similar way. Simulation models for each component were developed

independently and were than “glued” together. This model consists of approx. 300 event points (route events). Thus it is too large to be described in detail within this thesis. A simulation run for two shifts (17 hours) leads to approx. 40,000 iterations and a CPU–time of 5 minutes on a SIEMENS PC MX2 (including I/O–operations).

Using AMSEL a simulation program for the whole system of AGV, serving all four manufacturing halls, was developed. This model contains approx. 300 modules and 850 event points. The simulation of one working day (14 vehicles serving 1400 tasks) leads to approx. 100,000 iterations and took a CPU–time of 1 minute on a SUN SPARC station (including I/O–operations).

Chapter 4

Optimizing the movements of the stacker crane

4.1 The problem

As described in Section 2.3.3, the work–off strategy for the transportation tasks of the stacker crane was a priority–based FIFO–rule. This led to a high proportion of time needed for the unloaded moves of the stacker crane. This was acceptable for normal working conditions but after breakdowns of the stacker crane the number of jobs to be performed accumulated and this work–off strategy often led to delays. Furthermore, an increase in production was expected and the stacker crane was supposed to be a potential bottleneck in the flow of the material. The management was looking for ways to speed up the stacker crane without performing expensive technical changes.

The problem was discussed with our industry partners and they agreed that an optimization package minimizing the unloaded moves would increase the system performance considerably. But they also wished to see certain side constraints taken into account, namely that storage tasks should receive privileged treatment and that it is guaranteed that a generated job is not waiting too long until it is performed.

Thus, the problem was: Given a set of jobs, sequence them in such a way that the sum of unloaded moves between these jobs is minimized and the waiting times of the jobs until they are performed do not exceed certain limits.

4.2 Modelling

The question of minimizing unloaded moves between the transportation tasks leads to the solution of an **asymmetric Hamiltonian path problem (AHPP)**, as will be described in the following (see also Figure 4.2.1).

Suppose we are given $n-1$ jobs. With each job and with the current position of the stacker crane we identify a node in a complete digraph $D_n = (V, A_n)$. The current position of the stacker crane is considered to be a job (node) where the start and end coordinates coincide. Let node 1 represent the current position of the stacker crane. Each arc $(i, j) \in A_n$ represents the possibility to sequence job j immediately after job i.

With each arc $(i, j) \in A_n$ we associate a cost coefficient $c_{ij} \in \mathbb{R}$ which is nothing but the

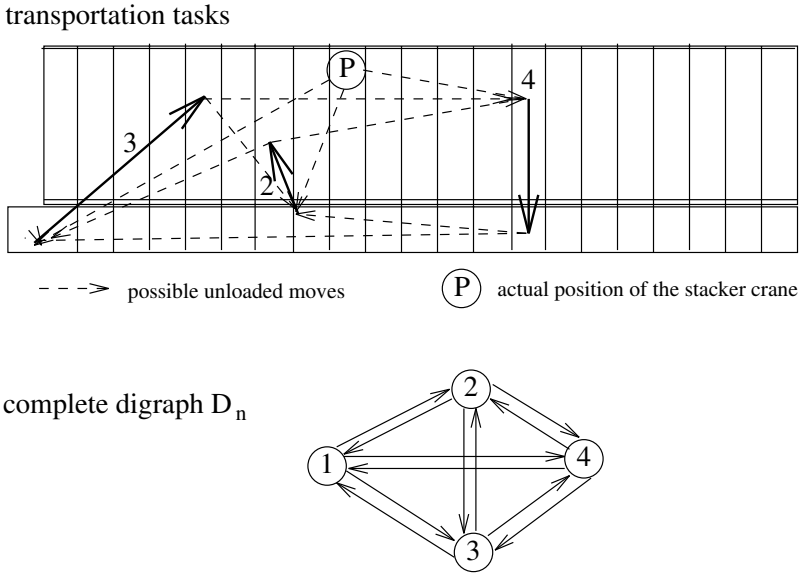


Figure 4.2.1 Modelling

unloaded travel time between jobs i and j, denoted by $time(i,j)$. This coefficient is calculated from the speed of the stacker crane, the coordinates of the endpoint of job i and the starting point of job j. As the current position has to be the first node in any sequence, we have to guarantee that no arc entering node 1 will be in the optimal solution. Therefore, we set:

$$c_{ij} = \begin{cases} time(i,j) & , \quad i,j \in V, i \neq j, j \neq 1, \\ \infty & , \quad j = 1, i \in V \setminus \{1\}. \end{cases} \tag{4.2.1}$$

The value $time(i,j)$ is, of course, only an approximation to the real travel time of the stacker crane, as e.g., acceleration and deceleration are not taken into account. Furthermore, it was not possible to take delays into account that resulted from an overload of the data nets. But the computational results within the simulation model (cmp. Section 3.4) showed that this approximation is satisfactory.

Furthermore, we have logical 0/1-variables x_{ij} describing possible sequences of jobs. To state this more formally we set

$$x_{ij} = \begin{cases} 1, & \text{if } j \text{ is sequenced immediately after } i, \\ 0, & \text{else.} \end{cases}$$

Given a set V of jobs one can formulate the problem of sequencing these jobs, such that the overall unloaded travel time is minimized, as a linear 0/1-program.

(4.2.2) Linear 0/1–program for the AHPP.

$$\begin{array}{llll} & \min c^T x & & \\ \text{s. t. } (1) & x(A) & = n-1 & \\ (2) & x(\delta^-(i)) & \leq 1 & \forall\ i \in V \\ (3) & x(\delta^+(i)) & \leq 1 & \forall\ i \in V \\ (4) & x(A(W)) & \leq |W|-1 & \forall\ W \subset V,\ 2 \leq |W| \\ (5) & x_{ij} & \in \{0,1\} & \forall\ (i,j) \in A_n \end{array}$$

Equality (1) assures that every feasible solution consists of $n-1$ arcs, (2) and (3) guarantee that at most one arc is entering and leaving each node, and the inequality system (4) avoids subtours. The AHPP, which is trivially equivalent to the Asymmetric Travelling Salesman Problem (ATSP), is known to be a $\mathcal{NP}$-hard problem, but it is also one of the most intensively studied problems in Combinatorial Optimization and for the instance sizes that arise in our application (up to 50 - 60 nodes) several codes are known that solve the resulting problem instances to optimality in a reasonable amount of computing time (e.g., Fischetti and Toth [FT92]).

4.3 The optimization program

The optimization program had to be designed to be used under heavy load conditions. It is obvious that there is no necessity and no potential for optimization approaches, if there is only a low number of jobs. But a feature that was needed was a quick recovery from "catastrophes".

The situation is the following: At a general point in time the control program of the stacker crane has to decide which job is scheduled next. The stacker crane is either executing some job or is idle because there is no job to perform. Due to calls from the assembly lines or deliveries from the AGV new jobs are generated.

In case that the stacker crane is idle the generated job is performed immediately. The management would not accept waiting times of the stacker crane. If there are other jobs present, the newly generated job is inserted into a list of all jobs and an optimization process is called minimizing the unloaded moves between them. A situation that should never occur is that the stacker crane is idle because the optimization process has not yet finished its calculations. Therefore, a 3–phases process was implemented that can be interrupted during its execution and still guarantees to supply a feasible solution.

(4.3.3) Optimization process.
Suppose that we are given a sequence $S = (j_1, ..., j_n)$ of n jobs that have to be performed. The job that is presently being performed is job j_1. The job that is newly generated is j_{new}. Assume $j_{new} \notin S$.

Phase 1: Run a **simple insertion heuristic**
Job j_{new} is inserted between k and $k+1$, where
$k = min_{l=1,..,n}\{c_{j_l j_{new}} + c_{j_{new} j_{l+1}} - c_{j_l j_{l+1}}\}$
(Assume that $c_{(i,n+1)} = 0 \; \forall \; i = 1, ..., n$).
Here we just scan through the current sequence S and try to insert job j_{new} as cheaply as possible.

Phase 2: Run a more sophisticated **heuristic**
Here we have the possibility to use any of the available heuristics for the ATSP, e.g., the farthest insertion heuristic.

Phase 3: Solve the problem to **optimality**
This is done using the branch&bound code of Fischetti and Toth [FT92] that solves the instances arising here in a reasonable amount of time.

It can happen that a new job is created while that process is still computing. In that case we stop the optimization process and turn to the new enlarged problem. As the process

is stopped only after the completion of phase 1, it is guaranteed that a solution (enlarged sequence) is supplied with almost no time delay. This sequence will be improved by phases 2 and 3, if there is still computing time available.

As mentioned in the introduction, we had to take technical side constraints into account. We shortly outline how this was done. To give preference to certain jobs we defined three classes of jobs. The first are *urgent* jobs, the second *normal* jobs, and the third jobs that can wait any amount of time until they are performed (e.g., rearrangements within the system). In the overall sequence the jobs of class one always precede those of class two, and so on. The coordinates of the end node of the jobs in class i are considered to be the coordinates of the starting node (position of the stacker crane) in the partial sequence in class $i+1$.

Each job in class two is initialized with a certain system–dependent counter that is decreased by one as soon as another job is finished. If the counter of such a job is zero it is inserted into class one. If the waiting time of a job exceeds some limiting value, it is scheduled next. With that strategy it is assured that jobs do not wait too long until they are performed. To give preference to storage tasks, they are always inserted into class one.

This is just a brief outline of the optimization module that was developed for Siemens Nixdorf. There are certainly further details that are important for an efficient implementation within a production system, e.g., interfaces to the software environment, behaviour in case of system errors. But a detailed description would be beyond the scope of this thesis.

4.3.1 Computational results

The optimization process was embedded into the simulation model and we used the results obtained within the simulation model to compare our approach with the old strategy. As there is an unrestricted amount of computing time for the optimization process within the simulation model, all problem instances of the AHPP were solved with phase 3 of the optimization process. Table 4.3.2 shows the results for the real–life data of the week that was used for the simulation model validation (cmp. Section 3.4).

	# jobs	uTr–P	uTr–O	Imp. %	max# jobs	Ø# jobs
1	416	8599	8325	3.18	6	2.31
2	421	8655	8141	5.93	8	1.94
3	405	8238	7956	3.42	6	2.27
4	398	8017	7634	4.77	8	1.93
5	447	9411	8951	4.88	8	2.13

Table 4.3.2 Minimizing the unloaded travel times (normal load conditions)

Key to Table 4.3.2:

# jobs	:	Number of transportation tasks (jobs).
uTr–P	:	Unloaded travel time of the stacker crane with the old priority based rule (in seconds).
uTr–O	:	Unloaded travel time of the stacker crane with the optimization process (in seconds).
Imp. %	:	Improvement in %, calculated by $\frac{(\text{uTr–P})-(\text{uTr–O})}{(\text{uTr–P})} \cdot 100$.
max# jobs	:	Maximal number of jobs at the same time.
Ø# jobs	:	Average number of jobs at the same time.

The results in the improvements of the unloaded moves (cmp. column Imp. %) of 3% to 6% were rather disappointing. By analyzing the data it turned out that the week considered was a week of low production volume, furthermore no major breakdowns of the stacker crane occurred. On the average there were only two jobs at the same time (see column Ø# jobs) and thus nothing to optimize.

As the process of gathering data in the real system is quite difficult, we and our industry partners decided to perform experiments with the available data. Therefore, we just added major breakdowns of the stacker crane to the input data and artificially created heavy load conditions. This was easy to perform within the simulation model. Some of the computational results achieved under heavy load conditions are reported in the following.

Experiment 1

We inserted a breakdown of one hour and collected all the jobs that were generated during that time period. We only take these jobs into consideration for our optimization process. All other jobs are ignored, namely those generated before the breakdown and those during the execution of the jobs. The results are reported in Table 4.3.3.

	# jobs	uTr–P	uTr–O	Imp. %
1	18	344	239	30.52
2	10	194	143	26.28
3	18	347	211	39.19
4	26	507	330	34.91
5	20	388	283	27.06

Table 4.3.3 Minimizing the unloaded travel times
(heavy load conditions – artificial breakdown)

This is a rather static experiment, since we do not take into account previously generated tasks or jobs that are generated during the performance of these jobs. Thus, this experiment does not reflect the dynamic nature of the system, but it shows us that there is an immense potential for optimization approaches.

Experiment 2

Next we performed a more realistic experiment to construct heavy load conditions in a dynamically changing environment.

The simulation started at 6 a.m. with the first deliveries at the receiving area and the assembly line buffers are supplied with the articles for the first production lot. The first shift started at 7 a.m. The stacker crane is working without any problems until 7 a.m., but then we inserted a breakdown from 7 a.m. until 9 a.m. All jobs generated during that time period were collected and sequenced optimally. Starting at 9 a.m. the transportation tasks were performed again and all newly generated jobs were also taken into consideration, i.e., the sequence is updated each time a new job is generated. The simulation run was stopped at 10 a.m. The results are shown in Table 4.3.4.

	# jobs	uTr–P	uTr–O	Imp. %	max# jobs	Ø# jobs
1	50	917	693	24.42	29	13.32
2	49	974	749	23.10	20	8.32
3	50	1007	783	22.24	26	12.23
4	49	889	662	25.53	31	15.24
5	50	985	839	14.82	25	13.08

Table 4.3.4 Minimizing the unloaded travel times
(heavy load conditions – artificial breakdown)

The computational results with that process (see Section 4.3.1) were very satisfactory and this process was put into use at the SNI plant. The computational experience in the real system showed that at almost all times there was enough time to finish phase 3.

4.4 On-line phenomena

The model described in Section 4.2 that was the basis for the optimization tool described in Section 4.3 is a *static model*. That means that it does not take the dynamic changes of the system into account. In the model it was assumed that for a given set V of transportation tasks an optimal sequence is calculated and that this sequence is performed in that order.

But this is not exactly what we needed for our application. The practical application we have in mind requires a *dynamic model* taking all the changes in the system into account. The modelling problem caused by the dynamic nature will be described in the following.

Assume that for a certain point in time i we are given a set of jobs V_i that are sequenced optimally and that are started to be performed. As soon as a job is finished, it is deleted and it is not necessary to consider that job any longer. But as soon as a new job is generated a new sequence containing the remainder of the old sequence and the new node has to be calculated.

(4.4.4) Notation.
We say that a Hamiltonian path problem is **off–line** or **static**, if the input to the problem does not change, neither during the execution of the algorithm, nor during the execution of the route.
On the other hand we call a Hamiltonian path problem **on–line**, if the input may (and will) change or be updated during the execution of the algorithm, or during the execution of the route.
The problem of (re–) calculating the current sequence that has to be performed at a certain point in time, will be addressed as a **subproblem** of the on–line HPP. □

The modelling problem is to find a local criterion for the solution of the subproblems leading to a global optimum of the on–line HPP. This will be illustrated by the following example:

(4.4.5) Example.

Suppose that we are given the nine nodes (eight jobs and current position of the stacker crane) as given in Figure 4.4.5 that have to be sequenced starting with node 1 (current position of the stacker crane). The shortest sequence is given in Figure 4.4.5 (a). Now we start performing the jobs. After t units of time nodes 2, 3 and 4 have already been visited

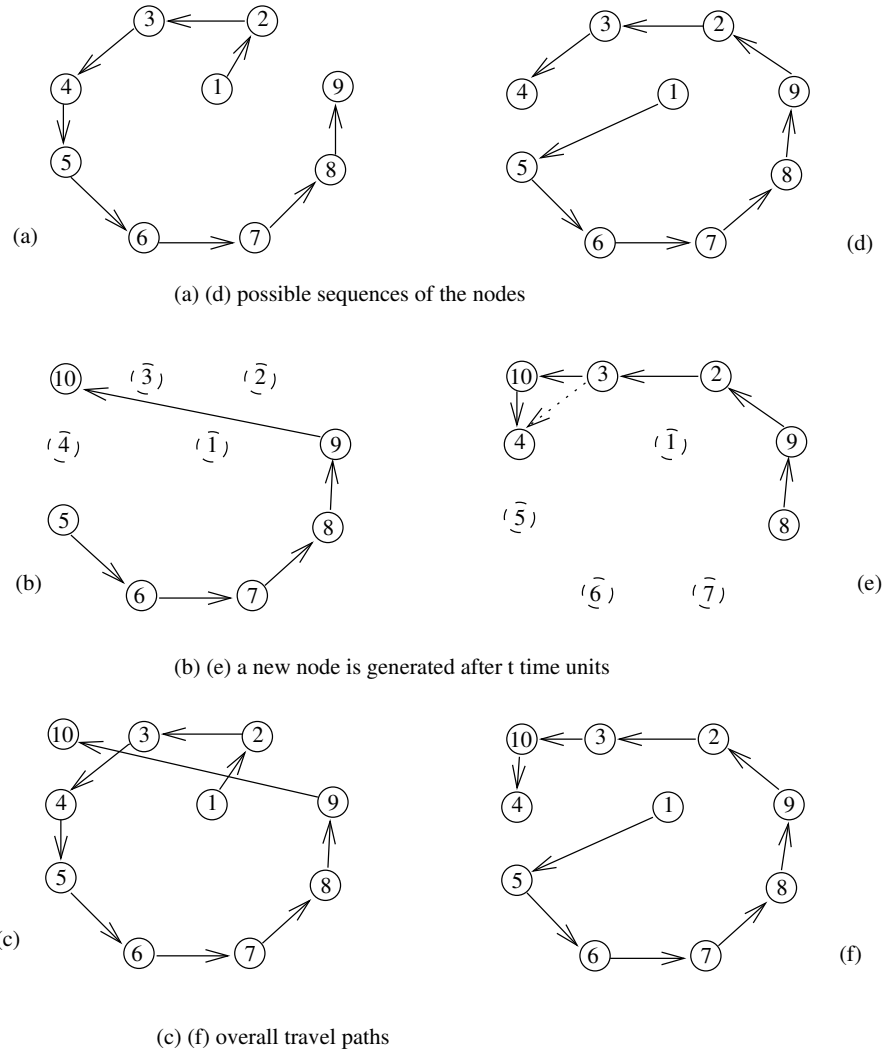


Figure 4.4.5

and a new node 10 is generated. A new sequence starting at node 5 and visiting nodes 6 – 10 has to be calculated. Again the optimal sequence is calculated. The resulting overall travel path is given in Figure 4.4.5(c).

Now suppose that in the initial step you choose a path that is not locally optimal (Figure 4.4.5(d)). Again after t time units a new job is generated that has to be visited. If the new node is inserted best possible (Figure 4.4.5(e)), the overall travel path is the one depicted in Figure 4.4.5(f). This one is shorter than the one you obtain if you solve every subproblem to optimality. □

Up to now it is not clear how such dynamic behaviour can be modelled correctly to guarantee that the overall travel path of minimal length is found. An important step might be to find objective functions taking these dynamic changes into account. But today's theory (and practice) of on–line optimization is still far away from giving satisfactory answers. That

is why we restricted ourselves to implement an optimization package based on the static model only. The computational results showed that this approach gives satisfactory results for the application we have in mind.

Studies on the on–line behaviour

An obvious question is how the problem of optimizing the movements of the stacker crane – that is modelled as an (on–line) Hamiltonian path problem – should be attacked. How should every subproblem be solved? Should one use a greedy heuristic with the argument that the sequence will change anyway and that it is good to have the "cheap" nodes in front of the sequence? Should one apply one of the heuristics that are known to work well for the ATSP? Should every subproblem be solved to optimality, as optimality cannot be that bad?

We were in the lucky situation that we had available several heuristics (implemented by Michael Jünger and Gerd Reinelt) and a branch&bound code (implemented by Matteo Fischetti and Paolo Toth [FT92]) for the ATSP. Furthermore, we have the simulation program for the AUSS (cmp. Section 3.3). Thus, we could simply test several strategies with real–life data within a (simulation) model that represents the real system exactly enough to take qualitative conclusions. We tested the following strategies:

priority	:	SNI-priority rule (old strategy run in FMS).
random	:	Generation of random sequences.
optimal	:	Each subproblem is solved to optimality.
greedy	:	Greedy heuristic.
greedy+2opt	:	Greedy heuristic with additional improvement heuristic (2–opt heuristic).
greedy+3opt	:	Greedy heuristic with additional improvement heuristic (3–opt heuristic).
farins	:	Farthest insertion heuristic.
listins	:	List insertion heuristic: The nodes to be inserted best possible are taken in the order $1, 2, ..., n$.
randins	:	Random insertion heuristic: The nodes to be inserted best possible are chosen randomly.
bestins	:	Best insertion heuristic: Choose always the best possible insertion.
shuffle	:	Shuffle heuristic: Start with sequences of length 1, "shuffle" them together to sequences of length 2, "shuffle" these together to sequences of length 4, etc.

Table 4.4.7 summarizes the results we achieved for 5 different real–life data sets. Further information on that data can be gathered from of Table 4.4.6.

	1	2	3	4	5
# jobs	50	49	50	48	50
Ø# jobs	12–15	7– 9	10–12	15–19	13–15
max. #jobs	30	21	27	32	26

Table 4.4.6: Data information

# jobs	:	number of transportation tasks (jobs)
max# jobs	:	maximal number of jobs at the same time
Ø# jobs	:	average number of jobs at the same time

strategy	1	2	3	4	5	$\sum$
priority	17.98	19.48	19.74	18.14	19.31	94.65
random	19.16	19.02	18.07	17.92	19.03	93.20
optimal	**11.50**	14.72	**11.33**	10.34	13.77	61.66
greedy	12.07	15.42	11.98	11.83	**11.92**	63.22
greedy + 2opt	11.87	15.34	11.87	11.72	12.69	63.49
greedy + 3opt	11.81	15.32	12.22	11.42	12.85	63.62
farins	13.00	15.16	12.64	**10.00**	13.87	64.67
listins	12.50	15.12	11.87	10.92	12.63	63.04
randins	11.64	15.28	12.17	11.20	12.74	63.03
bestins	11.92	**14.66**	12.29	11.85	12.94	63.66
shuffle	11.94	15.36	11.94	11.27	12.65	63.16
improvement	36.0%	24.7%	42.6%	44.9%	38.2%	34.85%

Table 4.4.7: Average unloaded travel time

The numbers in rows 2–12 give the average unloaded travel time in seconds between the jobs, if every subproblem is solved with the strategy given in the first column. Using the priority rule for the first set of data results in an average unloaded travel time of 17.98 seconds whereas the use of the greedy strategy results in an average unloaded travel time of 12.07 seconds. The row *"improvement"* gives the improvement of the strategy giving the best results compared with the old priority rule (*optimal* for data set 1, *bestins* for data set 2 etc.). This improvement factor is calculated by $100 \cdot \frac{prio-best}{prio}$, where *prio* stands for the value achieved by the priority rule and *best* for the value achieved by the strategy producing the best results.

In the following we derive an order of the strategies with the aim to distinguish between good and bad on–line heuristics. It is certainly not possible to take final conclusions based on these five data sets only but we give possible orders of the eleven on–line heuristics. Let $res(i,k)$ denote the average unloaded travel time obtained by using strategy i and applying it to data k. Table 4.4.8 gives three possible orders that are obtained as follows:

1.) Calculate $s_i := \sum_{k=1}^{5} res(i,k)$ for all heuristics i and sort them in increasing order (*order 1*).

2.) Perform a pairwise comparison: Set $\alpha_{ij}^k := 1$, if $res(i,k) > res(j,k)$, and 0 otherwise. Set up a matrix $A = (a_{ij})$ with $a_{ij} := \sum_{k=1}^{5} \alpha_{ij}^k$, i.e., a_{ij} gives the number of data sets on which i achieves a better result than j. With the cost matrix A solve a linear ordering problem (*order 2*).

3.) Perform a weighted pairwise comparison: Set up a matrix $B = (b_{ij})$ with $b_{ij} := \sum_{k=1}^{5} \max\{0, res(j,k) - res(i,k)\}$, and solve a linear ordering problem on B(*order 3*).

	Order 1	$(\sum)$	Order 2	Order 3
1.	optimal	(61.66)	optimal	optimal
2.	randins	(63.03)	listins	randins
3.	listins	(63.04)	randins	listins
4.	shuffle	(63.16)	greedy + 2opt	shuffle
5.	greedy	(63.22)	shuffle	greedy
6.	greedy + 2opt	(63.49)	greedy + 3opt	greedy + 2opt
7.	greedy + 3opt	(63.62)	greedy	greedy + 3opt
8.	bestins	(63.66)	bestins	bestins
9.	farins	(64.67)	farins	farins
10.	random	(93.20)	random	random
11.	priority	(94.65)	priority	priority

Table 4.4.8: Orders of the on–line heuristics

(4.4.6) Remarks.
Some interesting observations can be derived from of Tables 4.4.7 and 4.4.8.

(a) First, note that orders 1 and 3 give the same sequence of the on–line heuristics.

(b) There is almost no difference between the old priority rule and the generation of random sequences. The generation of random sequences gives slightly better results. Compared with the other strategies these results are significantly worse.

(c) Once you apply any optimization strategy a considerable improvement can be achieved. The best possible improvement ranges from 25% to 45%, on the average an improvement of 34 % was achieved.

(d) *Optimal* seems to be the best strategy under all orders. Although *farins* once gave the best result (on data 4) it seems to give worse results than the other heuristics. There is almost no difference in the results obtained by applying one of the other insertion heuristics or one of the *greedy* versions.

(e) *Randins* and *listins* tend to give slightly better results, although they never achieved the best result on any of the data sets.

(f) Surprisingly, it seems not worth to apply an exchange heuristic after the *greedy* heuristic has constructed a sequence.

(g) All the tested strategies are static strategies that do not take the dynamic changes of the system into account.

4.5 Related problems in the literature

In this section, we review some problems that are related to the solution of the on–line Hamiltonian path problem and that can be found in the literature. These are either dynamic or probabilistic versions of the travelling salesman problem or case studies on practical problems.

In 1985 the so–called **probabilistic travelling salesman problem (PTSP)** was introduced by Jaillet [Jai85] (see also [JO88, Ber92, BH93]). In the PTSP a complete weighted graph on n nodes is given. At each node i a demand occurs with probability p_i and does not occur with probability $1 - p_i$. Thus, in a given instance of the problem only a subset S of the nodes will be present. The goal is to find an a priori tour through all nodes that has minimal expected length for a particular instance. The convention is that the nodes in the instance are visited in the same order as in the a priori tour and that nonactive nodes are skipped.

Some interesting facts about optimal PTSP–solutions were elaborated by Jaillet [Jai85], e.g., that in the Euclidean plane the optimal PTSP–tour may intersect itself. This is in sharp contrast to the TSP. A couple of asymptotic results for the PTSP are known, e.g., that a PTSP strategy is asymptotically equivalent to a reoptimization strategy (calculate the optimal TSP-tour through the nodes in S), if the nodes are uniformly distributed in the unit square [BH93]. Furthermore, probabilistic results for heuristics, such as the space–filling curve heuristic and the nearest neighbour heuristic, are known. For the nearest neighbour heuristic, for instance, it was shown that on the average it produces poor solutions for the PTSP. But all results are based on the stochastic assumption that the points are uniformly and independently distributed in the unit square.

A remarkable amount of work was done on **dynamic vehicle routing** mainly derived out of the necessity to solve problems that arise in practice. Dynamic vehicle routing problems address the movement of vehicles over a given planning horizon, where the demands for vehicles to carry loads between the locations arrive sequentially over time and are uncertain. This is, for example, the case in the management of rental cars, or in the distribution of empty railroad freight cars, see [Pow86, FP90] among others. The scheduling of the vehicles has to be done in such a way that it is economical, e.g., such that the company's profit is maximized, that the deadhead miles driven by the vehicles (i.e., empty miles from the current location to a pickup point) are minimized, or an acceptable service is provided (minimize waiting times for delivery or service).

A study for one of North American's largest truckload motor carriers which must supply empty trucks to different cities is described in [ABN+88]. The optimization approach is based on the construction of a stochastic time–space network, where each node represents a particular region on a given day. The known loads are represented by "deterministic" links in that network, whereas the expected loads are represented by "stochastic" links. The uncertainties of the future (expected loads) are captured by a data–base, recording all loads during the last months. A flow in that model corresponds to a truck driving (loaded or unloaded) through the different regions. The benefit of this approach is measured within a Monte–Carlo simulation model.

In a survey article Psaraftis [Psa88] reviews the dynamic aspects of vehicle routing and compares it to static approaches. In the chapter *"Directions for further research"* the **dynamic travelling salesman problem** (DTSP) is introduced. To our knowledge this was the first time a dynamic version of the TSP was mentioned. It is defined as follows: Given a complete graph $G = (V, E)$ on n nodes with cost t_{ij} (travel time from i to j) on each arc. The

demands at each node occur independently according to a Poisson process with parameter λ. These demands have to be served by a salesman who is initially located on node 1 and spends a service time t on each demand.

The question is which "optimal" policy the (dynamic) salesman should follow. It is mentioned that the measure for optimality may vary dependent on the objective. It is possible to apply either a throughput measure or a delay measure. The maximization of throughput means to maximize the average expected number of demands served per time unit. The minimization of delays is equivalent to the minimization of the average (or maximal) expected time from the appearance of the demand until its service is completed. Furthermore, a combination of both measures might also be taken into consideration. One of the main questions that is posed is under which circumstances a policy is optimal that is based on the known demands only. Unfortunately no results are presented.

The main difference between the DTSP and the PTSP is that the PTSP, in the strong sense, is a static problem although the active nodes depend on the (probabilistic) demands. This is due to the fact that the calculation of the route R is made before the actual start of the salesman's tour. This tour cannot be changed later on. In the DTSP, however, the salesman reacts to the dynamically changing environment, characterized by the generated demands. The salesman's decisions are based on the current and, generally, on the future states of the system as well, which are given by a stochastic process.

In a series of papers Bertsimas and van Ryzin [BR91, BR93a, BR93b] studied the DTSP which they call the **dynamic travelling repairman problem (DTRP)**: a routing problem in a stochastic and dynamically changing environment. The research was motivated by an application, namely the scheduling of a repair crew to service geographically dispersed failures. Their objective is to minimize the sum of waiting times until a demand is satisfied. As the request for demands is uncertain, they consider it to be a stochastic process. First they analyzed the problem for the case of uniformly distributed demand locations and Poisson arrivals. Their aim was to find stationary policies for routing one (or more) vehicles. They proposed several strategies and proved that, under the stochastic assumptions, they are optimal in light traffic and provably within a constant factor in heavy traffic. The results were later on extended to the case where demand locations have an arbitrary continuous distribution and arrivals follow a general renewal process. Computational results are presented only with random data but not with real–life data.

Burkard et al. [BFR91] present a study on the design of an operating system for vehicles in an automated warehouse with a given layout. The considered warehouse consists of seven double–sided aisles being served by three stacker cranes. Each stacker crane is capable of moving into each aisle. The delivery and arrival area is being served by an additional stacker crane each. One of the optimization problems that have to be attacked is to minimize the stacker cranes' idle movements and waiting times. A difference to the system considered in Section 2.2 is that the system has more stacker cranes, which might block each other. Therefore, a strategy is needed avoiding such conflicts. The solution approach that is presented in the report is to build up a decision tree for a restricted time horizon. In that tree branches correspond to taken decisions and are weighted by idle stacker crane seconds caused by that decision. The solution corresponding to a shortest path from the root to a leaf is accepted as the current control for the next time period.

Besides the question of optimizing the stacker crane movements other optimization approaches are discussed. The overall control of the optimization packages is also performed

with the help of a decision tree. Although the study was made for a company and the optimization tool is now running in that factory no computational results (e.g., a comparison of different strategies) are presented.

4.6 Bounds for on–line strategies

Remark (4.4.6)(g) leads to the question how good the found solutions summarized in Table 4.4.7 are. One could surely imagine designing a more sophisticated on–line heuristic that takes the dynamic changes better into account and that produces even better results. That question can be reformulated as:

> *Are there lower bounds for a "good" or even "optimal" on–line strategy for the Hamiltonian path problem?*

There are two possible ways of analyzing the problem, namely to perform an on–line analysis or an off–line analysis. In the **on–line analysis** you still deal with the uncertainties of the system due to its unpredictable dynamic nature. You either have to apply a theoretical concept of on-line optimization (cmp. Chapter 1.5) or model that behaviour with the help of probability distributions in order to apply methods from mathematical statistics or stochastics. In the **off–line analysis** you eliminate the uncertainties by collecting all necessary data and information over a certain time period and perform the analysis with the now deterministic data.

As the use of the concepts of on–line optimization is rather limited and no probability distribution is available that models the dynamic behaviour satisfactorily, we decided to choose the second approach. We solved three different problems, namely the

(1) off–line Hamiltonian path problem (HPP),

(2) off–line HPP with time windows (HPPTW),

(3) off–line HPP with precedence constraints (SOP).

The asymmetric Hamiltonian path problem with precedence constraints is also known as Sequential Ordering Problem (SOP) (see Chapter 5).

Some notation:
In the following we assume that over a considered time period (e.g., one manufacturing shift) $n-1$ jobs are generated that have to be performed. As described in Section 4.2, with each job and with the current position of the stacker crane a node in a complete digraph $D = (V, A)$ is associated. Node 1 is assumed to be the current position of the stacker crane. Thus, in total we have $|V| = n$. In the following the expressions "node" and "job" will be used synonymously. The cost coefficients c_{ij} of each arc $(i, j) \in A$ are calculated as described in 4.2.1.

Any *optimal* Hamiltonian path (or sequence of jobs) starts with node 1. Let S_a denote the optimal sequence obtained by solving problem a ($a \in \{\text{HPP, HPPTW, SOP}\}$), let S_{online} be a sequence generated by an on–line algorithm for the HPP (e.g., one of the heuristics in Section 4.4), and let S^{opt}_{online} be the best possible sequence generated by an on–line algorithm for the HPP. By $c(S)$ we denote the cost of sequence S. □

In the following sections, we show that the inequalities

$$c(S_{HPP}) \leq c(S_{SOP}) \leq c(S_{HPPTW}) \leq c(S^{opt}_{online}) \leq c(S_{online}) \tag{4.6.7}$$

hold which will be essential for the on–line studies that are presented. They imply that the optimal solutions of problems (1)–(3) and even lower bounds on these values are lower bounds for the "optimal" on–line strategy for the problem of minimizing the unloaded moves of the stacker crane. It cannot be expected that the inequalities in (4.6.7) will be tight. The relative error of these approaches compared to the on–line heuristic will always be calculated by the formula

$$GAP = \frac{c(S_{online}) - c(S_a)}{c(S_a)} \cdot 100. \tag{4.6.8}$$

In the following sections we briefly sketch the models and summarize the results achieved by the use of these approaches. The mathematical background and details can be found in Chapters 5 and 6.

4.6.1 Offline-HPP

A first approach is to solve the so–called off–line Hamiltonian path problem. Here you just calculate the shortest Hamiltonian path S_{HPP} among the nodes $1, ..., n$.

The value of the optimal solution to the off–line HPP is a lower bound for the optimal on–line strategy for the considered time period, since it is trivially true that

$$c(S_{HPP}) \leq c(S^{opt}_{online})$$

otherwise S_{HPP} would not be an optimal Hamiltonian path.

We solved the off–line HPP resulting out of the same sets of real–life data as were used in Table 4.4.7. The Hamiltonian path problems were solved with the help of the branch&bound code of Fischetti and Toth [FT92]. Unfortunately the achieved results, which are summarized in Table 4.6.9, are relatively poor.

	1	2	3	4	5
on–line HPP	11.50	14.66	11.33	10.00	11.92
off–line HPP	7.36	6.64	6.38	5.89	8.14
GAP	56.2%	120.7%	77.5%	69.7%	46.4%

Table 4.6.9: Off–line HPP

The numbers in the row *off–line HPP* give the average unloaded travel time between the jobs in the optimal sequence S_{HPP} for the Hamiltonian path problem, whereas the numbers in the row *on–line HPP* are the average unloaded moves of the best on–line strategy so far (cmp. Table 4.4.7). The row *GAP* gives the relative error between these two values (cmp. (4.6.8)).

This approach leads to rather poor bounds. The GAP is between 50 % and 120 %. But this is not very surprising, if you retranslate this model to the practical application we have in mind. This means that during the considered time period the stacker crane does not move at all. All generated jobs are collected and at the end of the time period the optimal sequence among them is calculated and the jobs are then performed in that order. Thus, to achieve tighter bounds more realistic models are needed.

4.6.2 The construction of time windows for all jobs

The first approach of solving the off–line HPP suffers from the criticism that it does not take the dynamic behaviour of the system into account. But it is possible to model the dynamic aspects by time windows $[r_i, d_i]$. Here, r_i denotes the release date of job i (i.e., the earliest possible time when job i can start) and d_i denotes its due date (i.e., the latest possible time job i can start). The problem is to find the appropriate values for r_i and d_i. This has to be done in such a way that it is guaranteed that a solution of the off–line HPP with time windows is a lower bound for an "optimal" on–line strategy.

These time windows are already indirectly given in the manufacturing system we consider. A job cannot start before it is generated and thus there is no problem in finding the release dates r_i. The problem reduces to constructing appropriate due dates d_i for each job $i \in V$. Not every on–line strategy would produce acceptable solutions for our practical problem. Thus, we can make use of some system specifics.

During a production period without breakdowns all jobs should be performed within a given time period (e.g., one hour). If a job waits too long, it is given preference. Making use of that fact one can construct the time windows. Let g_i be the generation time, f_i^{prio} the completion time of job i with the use of the priority strategy, and let p be the time–period within the real system in which each job has to be finished. Then we set

$$\begin{aligned} r_i &= g_i \\ d_i &= f_i^{prio} + p. \end{aligned} \tag{4.6.9}$$

By these settings we have guaranteed that a feasible solution to the problem exists and furthermore that the lower bound is feasible for our problem. This is true as every feasible on–line solution is a feasible solution to the AHPP with time windows as constructed above. In the manufacturing system we consider the stacker crane is not allowed to be idle as long as there are jobs to perform. Therefore, an optimal solution of the AHPP with time windows is not necessarily a feasible on–line solution, as it can imply waiting times.

For the general on–line HPP this setting of the due dates d_i is certainly not feasible as it is possible that a job can wait as long as possible. Therefore, it would be necessary to set $d_i = \infty$. But the wider the time windows are the closer the result will be to the result obtained by the solution of the off–line HPP. With the settings in (4.6.9) we have chosen time windows that are tighter than the ones for the general case. Thus, they will produce tighter bounds.

4.6.3 Off–line HPP with precedence constraints

Once we are given the time windows $[r_i, d_i]$ for each node $i \in V$ it is possible to relax them to precedence constraints.

Suppose that for each job $i \in V$ we are given a time window $[r_i, d_i]$ where r_i gives the release date of node i (earliest time when the job can be performed) and d_i the due date (latest possible starting time) of job i. With s_i we denote the service time (time needed to perform the job) of job $i \in V$. If there are two time windows $[r_i, d_i]$ and $[r_j, d_j], i \neq j$, with

$$d_i < r_j + s_j, \tag{4.6.10}$$

then we know that i has to precede j, because if we perform j at the earliest possible time r_j and then try to perform job i we would violate the due date of job i. If $d_i \geq r_j + s_j$ and

$d_j \geq r_i + s_i$ holds for two jobs i and j, then the jobs can be sequenced in any order $i \to j$ or $j \to i$ and therefore no precedence relationship can be derived. Applying the described procedure it is possible to relax all time windows to precedence relationships. For the case of the general on–line HPP where we have $d_i = \infty$ for all $i \in V$ no precedences can be derived out of the time windows and we have $c(S_{HPP}) = c(S_{SOP})$.

As the HPP is a special case of the HPP with precedence constraints and every solution of the SOP is feasible for the HPP, we have

$$c(S_{HPP}) \leq c(S_{SOP}).$$

Furthermore, we have

$$c(S_{SOP}) \leq c(S^{opt}_{online})$$

as every on–line solution of the considered HPP is a feasible solution to the SOP, but not vice versa.

The resulting problem instances of the SOP were solved using a branch&cut code that was implemented jointly with M. Jünger, G. Reinelt, and S. Thienel in the SCIENCE-project of the European Community. The results that are slightly better than the ones for the off–line HPP are summarized in the following table 4.6.10. They are read in the same manner as the results in Table 4.6.9.

	1	2	3	4	5
on–line HPP	11.50	14.66	11.33	10.00	11.92
off–line SOP	8.18	7.40	8.10	7.46	9.59
GAP	40.6%	98.1%	39.5%	34.0%	24.3%

Table 4.6.10: Off–line HPP with precedence constraints

4.6.4 Off–line HPP with time windows

The SOP as considered in the last section is a relaxation of the asymmetric Hamiltonian path problem with time windows (HPPTW). Therefore, we have

$$c(S_{SOP}) \leq c(S_{HPPTW}).$$

If we can solve the HPPTW directly, we expect to obtain better bounds. Since from the computational point of view the HPPTW belongs to the difficult problems in combinatorial optimization (cmp. Chapter 6), we cannot expect to solve every problem instance to optimality. But as the inequality

$$c(S_{HPPTW}) \leq c(S^{opt}_{online})$$

holds, we know that any lower bound on the value of $c(S_{HPPTW})$ will be a lower bound on the value of $c(S^{opt}_{online})$. The time windows for nodes $i \in V$ are constructed as described in Section 4.6.2.

We now calculate a shortest Hamiltonian path S_{HPPTW} that satisfies all the time windows with the help of a branch&cut code that was implemented jointly with M. Fischetti. In case that we cannot solve the problem instance to optimality, this polyhedral approach guarantees to find at least good lower bounds on the value $c(S_{HPPTW})$. The results are summarized in

Table 4.6.11. If the problem was not solved to optimality, this is indicated in the table by giving the upper and lower bounds ([lower,upper]). In that case, for the calculation of the gap with formula (4.6.8) the lower bound is used instead of the optimal value.

	1	2	3	4	5
on–line HPP	11.50	14.66	11.33	10.00	11.92
off–line HPPTW	8.45	[8.54,19.17]	[9.14,19.18]	[9.68,18.21]	[10.37,18.12]
GAP	36.1%	71.7%	24.0%	3.3%	14.9%

Table 4.6.11: Off–line HPP with time windows

The large gaps for the off–line HPPTW are due to the fact that the heuristics used so far do not produce satisfactory results. We believe that the lower bound is far closer to the value of the optimal solution than the upper bound is. The use of better primal heuristics will close this gap.

4.7 Summary

In the last sections we derived lower bounds on the value of an optimal on–line solution. The results are summarized in Table 4.7.12.

	1	2	3	4	5
on–line HPP	11.50	14.66	11.33	10.00	11.92
off–line HPP	7.36	6.64	6.38	5.89	8.14
off–line SOP	8.18	7.40	8.10	7.46	9.59
off–line HPPTW	8.45	8.54*	9.14*	9.68*	10.37*
GAP HPP	56.2%	120.7%	77.5%	69.7%	46.4%
GAP SOP	40.6%	98.1%	39.5%	34.0%	24.3%
GAP HPPTW	36.1%	71.7%	24.0%	3.3%	14.9%

* lower bound, as instance not solved to optimality

Table 4.7.12: lower bounds for on–line HPP

For data sets 2–5 the problem instances for the HPPTW could not be solved to optimality. Therefore, the corresponding bounds are supposed to improve in case that optimal solutions are calculated.

First, note that the gap reduces the tighter the off–line model is to the real application. For the HPPTW it varies between 3% and 70%. We believe that these are instances, where we had "good luck", resp. "bad luck", in the calculations. In the average a gap of approx. 30% can be observed. Although this gap seems relatively big, it is likely to occur, since the on–line problem is attacked by means of "off-line" heuristics and the lower bound might be improved by solving the time constrained AHPP where no idle time is allowed when a node can be processed.

Chapter 5

Hamiltonian path problems with precedence constraints (Sequential ordering problem)

5.1 The problem

The **asymmetric Hamiltonian path problem with precedence constraints (AHPP-PC)**, also called **sequential ordering problem (SOP)**, can be phrased in graph theoretical terminology in the following way.

We are given a directed, complete graph $D_n = (V, A_n)$ on n nodes and cost coefficients $c_{ij} \in \mathbb{R}, c_{ij} \geq 0$, associated with each arc $(i, j) \in A_n$. The precedences that have to be satisfied are given by an additional **precedence digraph** $P = (V, R)$ that is defined on the same node set V as D_n. An arc $(i, j) \in R$ represents a precedence relationship $i \prec j$, i.e., i has to precede j. In this chapter we consider the case where each Hamiltonian path has a distinct starting node, say node 1, and a distinct final node, say node n. Therefore, $(1, i) \in R \ \ \forall i \in V \setminus \{1\}$ and $(i, n) \in R \ \ \forall i \in V \setminus \{n\}$. If this condition is not satisfied, it can always be achieved by adding additional nodes.

Let $pos_H(i)$ denote the position of node i in the Hamiltonian path H. The path H is called a **feasible Hamiltonian path** (with respect to P), if it does not violate the precedence relationships given by P, or to state it more formally, if $pos_H(i) < pos_H(j)$ is satisfied for every given precedence relationship $i \prec j$. Obviously, the precedence digraph P has to be acyclic, otherwise no feasible solution exists. Moreover, P can be assumed to be transitively closed, because if $i \prec j$ and $j \prec k$ we can conclude that $i \prec k$.

The problem is to find a feasible Hamiltonian path with minimal cost in D_n.

The **asymmetric travelling salesman problem with precedence constraints** (ATSP-PC) is defined in a similar way. In order to achieve that the concept of precedences does make sense we need to have a designated starting node, say node 1. The problem is to find a Hamiltonian cycle "starting" and "ending" at node 1 of minimal length that satisfies all given precedence relationships among the nodes in $V \setminus \{1\}$.

Note that the AHPP-PC (or SOP) reduces to the asymmetric Hamiltonian path problem (AHPP) in the case that the precedence digraph $P = (V, R)$ has empty arc set R. Thus the AHPP is a special case of the SOP. As the AHPP is an $\mathcal{NP}$-hard problem, the same holds for

the SOP.

There is a great variety of real–world problems that can be modelled as a SOP. They occur, e.g., in routing applications where a pick–up has to precede the delivery, or in scheduling applications where a certain job has to be completed before other jobs can start.

Although the travelling salesman problem is one of the most investigated problems in Combinatorial Optimization, very little attention has been paid to the precedence constrained version.

The sequential ordering problem (SOP) as stated in the form here, seems to have been mentioned first by Escudero [Esc88a, Esc88b]. The aim of his investigations has been to design heuristics for a production planning system that perform well in practice with respect to solution guarantee and running time.

Ascheuer [Asc89] and Ascheuer, Escudero, Grötschel, and Stoer [AEGS90, AEGS93] describe a cutting–plane approach to obtain lower bounds on the value of the optimal solution. These lower bounds have been used to measure the quality of the solutions obtained by the heuristics developed by Escudero. The authors compare three different models and give computational results on real–life data of IBM that was provided by Escudero. The model that will be presented in Section 5.2 turned out to give satisfactory bounds with a gap of approx. 5% and to be superior to the other models from a computational point of view.

Around the same time a similar model has been developed independently by Balas, Pulleyblank, and Timlin [Tim89, PT91]. They considered the symmetric case and the aim of their research has been to design heuristics to schedule helicopters that have to visit offshore oil–platforms in a certain order. It was desired to find a route for each daily set of stops that satisfies all the requirements (precedence constraints, helicopter capacity, etc.) and that minimizes the total distance flown.

Savelsbergh [Sav90] describes a k–exchange heuristic for the *single–vehicle dial–a–ride problem* where a vehicle has to pick up and deliver n customers. Each customer has a pick up and delivery location and the pick up must precede the delivery. He outlines a procedure where the feasibility check can be done in constant time. In the problem he describes the precedences are very structured, as any node has one uniquely determined predecessor or successor.

Bianco et al. [BMRS] attack the problem with the help of a dynamic programming algorithm. Their algorithm has been tested with randomly generated data. As it can be expected, the dynamic programming algorithm performs the better the more precedences are given, because the number of possible states can be reduced dramatically. They have been able to solve very special dial–a–ride problems up to 105 nodes in approx. 5 minutes.

Very recently the first polyhedral investigations have been carried out by Balas, Fischetti, and Pulleyblank [BFP92]. We refer to their results in the following sections.

5.2 Linear Programming Model

In the literature several models for the SOP are known (cf. [Esc88a, Esc88b], [Asc89, AEGS90], [BFP92]). Ascheuer et al. [AEGS93] performed computational experiments showing that the model described in the following is superior to the others from a computational point of view. This model is based on a linear programming formulation of the asymmetric Hamiltonian path problem. The precedence relationships are modelled by an exponentially large class of inequalities.

For each arc $(i,j) \in A_n$, we introduce a binary variable $x_{ij} \in \{0,1\}$ with the interpretation that

$$x_{ij} = \begin{cases} 1, & (i,j) \in A_n \text{ is in the Hamiltonian path,} \\ 0, & \text{else.} \end{cases}$$

With each arc $(i,j) \in A_n$, a cost coefficient $c_{ij} \geq 0$ is associated. Immediately, some reductions can be performed.

(5.2.1) Lemma.
Given $D_n = (V, A_n)$ and $P = (V, R)$, then
(i) $x_{ji} = 0 \quad \forall\ (i,j) \in R$
(ii) $x_{ik} = 0 \quad \forall\ (i,k) \in R$ s.t. $\exists (i,j) \in R$ and $(j,k) \in R$.

Proof. If $i \prec j$, we know that j can never precede i in a feasible Hamiltonian path. Therefore, x_{ji} can be fixed permanently to 0.
If $i \prec j$ and $j \prec k$, we know that we can fix x_{ik} permanently to 0, as at least node j has to be scheduled in between. □

Balas, Fischetti, and Pulleyblank [BFP92] proved that this lemma yields a complete characterization of all variables that can a priori be fixed to zero. The variable set (resp. arc set) is reduced by fixing these variables to 0 (resp. deleting the arcs). To state this more formally, we set

$$\begin{aligned} R_1 = &\ \{(j,i) \in A_n \mid (i,j) \in R\} \\ R_2 = &\ \{(i,k) \in A_n \mid (i,j) \in R \text{ and } (j,k) \in R\}. \end{aligned}$$

By setting

$$A = A_n \setminus (R_1 \cup R_2)$$

we end up with the digraph $D = (V, A)$ in which we look for feasible Hamiltonian paths. We only associate variables with the **feasible arc set** A.

The linear 0/1-model can now be stated as follows.

(5.2.2) Linear 0/1–Programming Formulation for SOP.

$$\begin{array}{llll} & \min c^T x & & \\ \text{s. t. } (1) & x(\delta^-(i)) & = 1 & \forall\ i \in V \setminus \{1\} \\ (2) & x(\delta^+(i)) & = 1 & \forall\ i \in V \setminus \{n\} \\ (3) & x(A(W)) & \leq |W| - 1 & \forall\ W \subset V,\ 2 \leq |W| \\ (4) & x(j:W) + x(A(W)) + X(W:i) & \leq |W| & \forall\ (i,j) \in R \text{ and} \\ & & & \forall\ W \subseteq V \setminus \{i,j\}, W \neq \emptyset \\ (5) & x_{ij} & \in \{0,1\} & \forall\ (i,j) \in A \end{array}$$

(1)–(3),(5) is the standard formulation for finding an asymmetric Hamiltonian path from 1 to n. (1) and (2) guarantee that one arc is entering and leaving each node **(in– and out–degree constraints)**. Inequalities (3) avoid subtours **(subtour elimination constraints)**. The inequality system (4) assures that all precedences are satisfied. These inequalities are called **precedence forcing constraints**. Note that the arcs in $\delta^-(1)$ and $\delta^+(n)$ are eliminated due to Lemma (5.2.1). Therefore, the equations $x(\delta^-(1)) = 0$ and $x(\delta^+(n)) = 0$ are trivially valid and need not be considered.

The linear programming model for the ATSP-PC is more or less the same but equations (1) and (2) have to be substituted by

$$\begin{array}{lll} (1') & x(\delta^-(i)) & = 1 \quad \forall\, i \in V \\ (2') & x(\delta^+(i)) & = 1 \quad \forall\, i \in V \end{array}$$

It is obvious that every feasible solution of (1)–(5) is the incidence vector of a feasible Hamiltonian path and vice–versa.

In the sequel we investigate the **sequential ordering polytope**

$$SOP(n, P) = conv\{x \in \mathbb{R}^A \mid x \text{ is a feasible Hamiltonian path from 1 to } n\}$$

whose vertices are the characteristic vectors of the feasible Hamiltonian path of D. In a similar way the precedence constrained ATSP polytope

$$P_{PC}(n, P) = conv\{x \in \mathbb{R}^A \mid x \text{ is satisfies } (1')(2'), (3) - (5)\}$$

is defined. The study of the structure of the sequential ordering polytope (dimension, facets, etc.) is of particular interest for the solution of the SOP with the help of a branch&cut approach and is done in the following sections. This is closely related to the study of the ATSP polytope (cf. Section 1.3).

5.3 Dimension of the polytope

Generally, the first step in establishing facial results about a certain polytope P is to determine the dimension of P. For the SOP this turns out to be a difficult problem involving a lot of technical details. Partial results have been proven by Balas, Fischetti and Pulleyblank [BFP92]. In this section we derive a dimension formula for a large class of instances of the sequential ordering problem. This is a more general result as the one by Balas et al. and has been developed jointly with Mechthild Stoer.

(5.3.3) Definition.
Given $D = (V, A)$ and the precedence digraph $P = (V, R)$, the set $\pi(j)$ of predecessors of $j \in V$ and the set $\sigma(i)$ of successors of i is defined by

$$\begin{aligned} \pi(j) &:= \{i \in V | (i,j) \in R\}, \\ \sigma(i) &:= \{j \in V | (i,j) \in R\}. \end{aligned}$$

V^- (resp. V^+) is defined to be the set of all nodes having at least one predecessor (resp. successor), i.e.,

$$\begin{aligned} V^- &:= \{i \in V | \pi(i) \neq \emptyset\}, \\ V^+ &:= \{i \in V | \sigma(i) \neq \emptyset\}. \end{aligned}$$

A node $i \in \pi(j)$ is called a **direct predecessor** of j, if the precedence relationship is not implied by a transitivity relation, i.e., $(i,j) \in R$ and $(i,j) \in A$.
A node $k \in V$ is called **free**, if $\pi(k) = \{1\}$ and $\sigma(k) = \{n\}$, i.e., k is not involved in any precedence relationship, except for the starting node 1 and the ending node n. ◇

In the sequel we will consider some special instances of the SOP. Therefore, we need the following definitions.

(5.3.4) Definition.
With (V, A, R) we denote an **instance of the SOP** given by the digraphs $D = (V, A)$ and $P = (V, R)$. As it was done in the previous section, let $n := |V|$ be the ending node of every feasible Hamiltonian path.
An instance (V, A, R) of the SOP is called **regular**, if one of the following conditions is satisfied:

(a) there exists a free node, or
(b) n has more than two direct predecessors, or
(c) if n has exactly two direct predecessors, say i and j, then $\pi(i) \neq \pi(j)$ holds

Otherwise, the instance is called **nonregular**. ◇

Results on the dimension of the polytope are known for the case that a free node exists. Balas et al. [BFP92] showed that for this case all but one of the degree constraints (1') and (2') form a minimal equation system for $P_{PC}(n, P)$, i.e.,

$$dim(P_{PC}(n, P)) = |A| - 2n + 1.$$

Therefore, we obtain

$$dim(SOP(n, P)) = |A| - 2n + 3$$

if a free node exists. The assumption of the free node is essential for the proof, because it is based on interchange arguments for constructing feasible tours. This can be done relatively easy with a free node, as this node can be reinserted anywhere in the remaining sequence

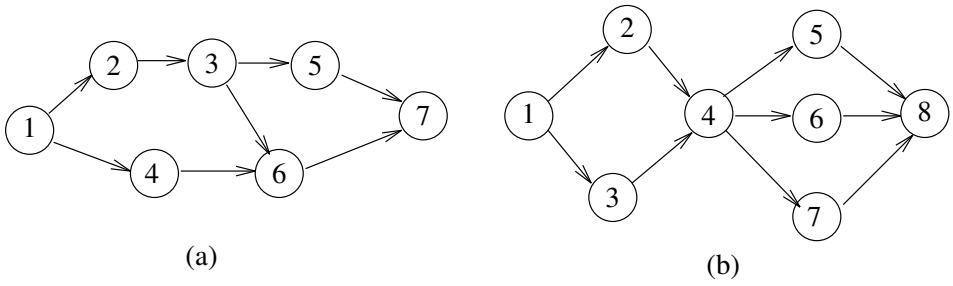


Figure 5.3.1

without constructing an infeasibility. But this is certainly a rather restrictive assumption, since it does not cover precedence digraphs as given in Figures 5.3.1(a) and (b).

Having a look at Figure 5.3.1(b) one immediately recognizes that the problem can be decomposed into two subproblems, namely to find shortest feasible Hamiltonian paths through the node sets $V_1 = \{1, 2, 3, 4\}$ and $V_2 = \{4, 5, 6, 7, 8\}$.

To state this more formally, let $m \in V$ be a fixed node. Set $V_1 := \{v \in V \mid v \in \pi(m)\} \cup \{m\}$ and $V_2 := V \setminus \pi(m)$. We now solve the Sequential Ordering Problem on the smaller digraphs $D_1 = (V_1, A(V_1))$ and $D_2 = (V_2, A(V_2))$. Observe that m is the ending node of the optimal path on D_1 and the starting node on the optimal path on D_2 and that the concatenation of these two paths at node m is an optimal path on the original instance (V, A, R).

If we split the original instance at a fixed node in the way described above, we also say that the instance (D, A, R) **decomposes** into the two instances $(V_1, A(V_1), R(V_1))$ and $(V_2, A(V_2), R(V_2))$.

It is easy to see that in each of the subproblems (or in the problem instance itself, if it is not decomposable) one of the degree constraints is linearly dependent. Therefore, a correction term in the dimension formula is necessary. Set

$$F := \{i \in V \setminus \{1, n\} \mid \exists\ k, 0 < k < n, \text{ s.t. } |\pi(i)| = k \text{ and } |\sigma(i)| = n - k - 1\} \tag{5.3.5}$$

to be the set of nodes that are fixed in their position. If $i \in F$, it will be called **fixed**. For example, in Figure 5.3.1(b) node 4 is fixed. In the following we show that

$$dim(SOP(n, P)) = |A| - 2n + 3 + |F|$$

holds for regular instances. This will be done by first discussing the case where $|F| = \emptyset$ and then prove the formula for the more general case where $|F| \neq \emptyset$. We conjecture that this dimension formula holds for all classes of instances, but a proof remains open.

(5.3.6) Theorem.
Suppose we are given a regular instance (V, A, R) of the SOP with $n \geq 4$. If $|F| = \emptyset$, then

$$dim(SOP(n, P)) = |A| - 2n + 3.$$

Proof. If there exists a free node, the result has already been established by Balas et al. [BFP92]. Therefore, assume in the following that no free node exists. We are going to prove the remaining cases by induction over $|R|$.

It is easy to see that the result holds for the easiest case where $n = 4$ and $R = \{(1,2), (1,3), (2,4), (3,4)\}$.

Now suppose we are given an instance on $D = (V, A)$ on $n := |V|$ nodes with a precedence digraph $P = (V, R)$.

In case that n has more than two direct predecessors, choose any of these predecessor. In case that n has exactly two direct predecessors, say i and j, choose a predecessor i such that $\pi(i) \not\subseteq \pi(j)$. In the sequel this selected predecessor is denoted with v_{n-1}.

We now shrink n and v_{n-1} to a single node n'. Let (V', A', R') denote the shrunken instance. Through this shrinking operation the precedences not involved with n and v_{n-1} are not influenced. We set $i \prec n'$, if

$$\begin{aligned} &i \neq v_{n-1} \text{ and} \\ &i \prec n \text{ or } i \prec v_{n-1}. \end{aligned}$$

holds. As the instance is regular, a direct predecessor of n can be found such that the shrunken instance does not contain a fixed node.

Note that through this shrinking operation precedences might occur that are implied by transitivity relations (e.g., arcs e and g in Figure 5.3.2) and that this operation has an influence on the feasible arc set A', as well.

If we identify the arcs $(i, n') \in A'$ with the arcs $(i, v_{n-1}) \in A$, the arc sets A' and A differ by the following arcs (dashed lines in Figure 5.3.2(c)):

(i, n)	such that $(i, n) \in R$ and $(i, n) \in A$,
(i, v_{n-1})	such that i is a direct predecessor of v_{n-1} and there exists an $[i, n]$–path in R not using v_{n-1} (e.g., arcs e, g in Figure 5.3.2(b)),
(j, v_{n-1})	such that $j \notin \pi(v_{n-1})$, and j is not a direct predecessor of n,
(v_{n-1}, i)	with $i \notin \pi(v_{n-1})$ (e.g., arc (v_{n-1}, v_{n-2}) in Figure 5.3.2).

Let $a'x' = a'_0$ be an equation that is satisfied by all feasible solutions of the shrunken instance (V', A', P'). By induction we know that this equation is a linear combination of the degree constraints, i.e., these equations can be combined such that $a'_{ij} = 0$ for all $(i, j) \in A'$.

Now, let $ax = a_0$ be an equation that is satisfied by all feasible solutions of the original instance (V, A, P). The same combinations that led to $a'_{ij} = 0$ for all $(i, j) \in A'$ can be used in the original instance to set the coefficients of the following arcs $(i, j) \in A$ to 0:

(i, v_{n-1})	if $(i, n) \in R, (i, n') \in A', i \neq v_{n-1}$,
(i, v_{n-1})	if $(i, v_{n-1}) \in R$, i is a direct predecessor of v_{n-1} and $\nexists$ a $[i, n]$–path in R not using v_{n-1},
(i, j)	for all $i, j \in V \setminus \{v_{n-1}, n\}$.

Assume that in the shrunken instance the equation $x'(\delta^-(n')) = 1$ was not used. Therefore, two of the three equations

$$\begin{aligned} &\text{(i)} && x(\delta^-(n)) && = && 1, \\ &\text{(ii)} && x(\delta^+(v_{n-1})) && = && 1, \\ &\text{(iii)} && x(\delta^-(v_{n-1})) && = && 1, \end{aligned}$$

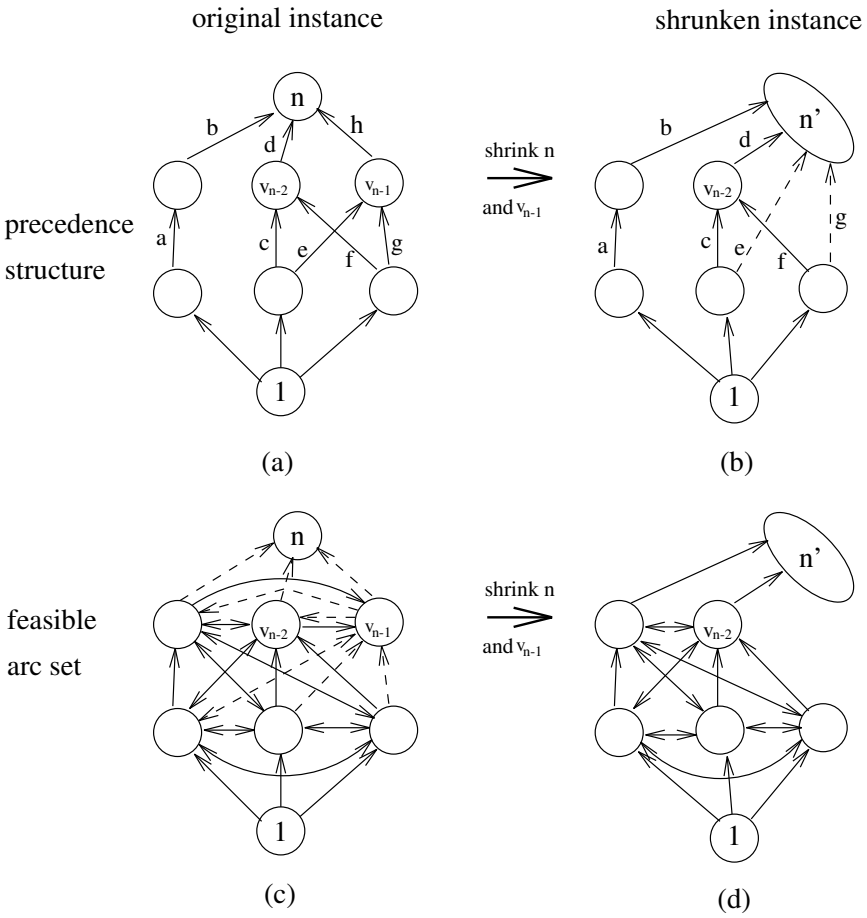


Figure 5.3.2 Influence of shrinking operation

can be used to set further coefficients to 0. We use (i) to fix $a_{kn} = 0$ for one direct predecessor k of n, $k \neq v_{n-1}$, and (iii) to fix $a_{v_{n-1},n} = 0$. Such a k exists, since v_{n-1} is not fixed. Figure 5.3.3 shows all arcs whose coefficients are not yet fixed to 0.
To simplify notation we introduce the following node sets:

$$\begin{aligned}
V_1 &:= \{l \in \pi(v_{n-1}) \mid (l, v_{n-1}) \in A \text{ and } \exists \text{ a } [i,n]\text{–path in } R \text{ not using } v_{n-1}\},\\
V_2 &:= \{k \in V \setminus \{v_{n-1}\} \mid (k,n) \in A \text{ and } (k,n) \in R\},\\
V_3 &:= \{j \in V \mid j \notin \pi(v_{n-1}), j \neq n\},\\
V_4 &:= \{j \in \pi(v_{n-1}) \mid (j, v_{n-1}) \in A, \text{ and } \nexists \text{ a } [i,n]\text{–path in } R \text{ not using } v_{n-1}\},\\
\tilde{V} &:= V \setminus \{1, V_1, V_2, V_3, v_{n-1}, n\},
\end{aligned}$$

i.e., the sets $V_i, i = 1, ..., 3$, correspond to nodes that are incident to arcs whose coefficient is not yet fixed to 0. As v_{n-1} is not a fixed node, we know that $V_2 \neq \emptyset$.

Let $S_1, ..., S_k$ be any disjoint partition of $V \setminus \{1, n\}$. If in the sequel a Hamiltonian path is given in the form $(1, S_1, S_2, ..., S_k, n)$, this means that the nodes of $S_i, i = 1, ..., k-1$, are visited before any node of $S_j, i < j$, is visited and that the nodes in $S_i, i = 1, ..., k$, may be sequenced in any (uninterrupted) feasible order.

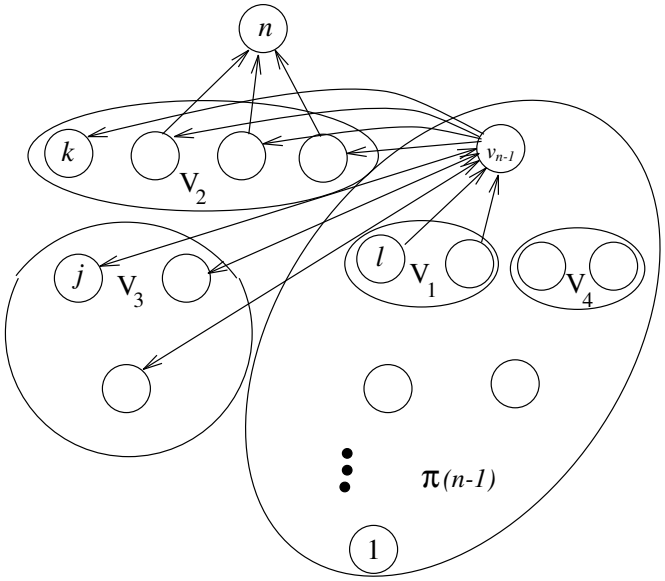


Figure 5.3.3

In the following we construct Hamiltonian paths and claim that they are feasible. By definition we know that

v_{n-1}	can be sequenced before any node in $V_2 \cup V_3$,
$l \in V_1$	can be sequenced before any node in $V_2 \cup V_3 \cup \{v_{n-1}\}$,
$j \in V_3$	can be sequenced before any node in $V_1 \cup V_2 \cup \{v_{n-1}\}$,
$k \in V_2$	can be sequenced before $\{v_{n-1}\}$.

We now construct feasible Hamiltonian paths to fix the remaining coefficients of $ax = a_0$ to 0. Let $k \in V_2$ be the arbitrary but fixed node that was previously used to set a coefficient a_{kn} to 0.

First, consider the path

$$(1, \tilde{V} \setminus V_4, \{V_4 \cup V_1 \cup V_3\}, V_2, v_{n-1}, n).$$

Since at least one of the sets V_1, V_3, V_4 is nonempty and v_{n-1} cannot be a predecessor of any node in that union, a path of this form exists. As all coefficients are 0, we can conclude that $a_0 = 0$.

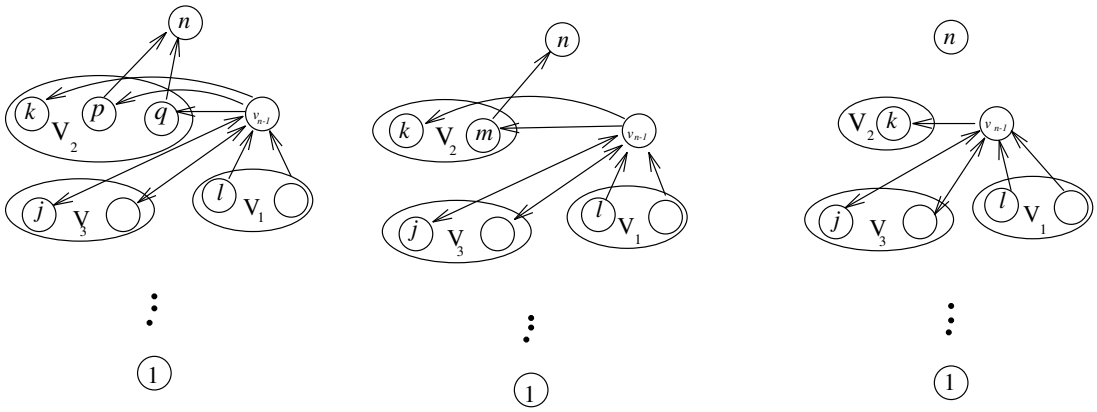
In the following we discuss several possible cases for V_2. Note, that if $V_3 = \emptyset$, not all of the constructed paths have to be considered.

Case 1: $|V_2| \geq 3$

1.) Consider the path $(1, \tilde{V}, V_1, V_3, q, v_{n-1}, p, k, n)$ for any distinct $p, q \in V_2 \setminus \{k\}$. We can conclude that $a_{v_{n-1},p} = 0$ for all $p \in V_2 \setminus \{k\}$.

2.) Consider the path $(1, \tilde{V}, V_1, V_3, k, v_{n-1}, q, V_2 \setminus \{k, q\}, q, n)$. We can conclude that $a_{qn} = 0$ for all $q \in V_2$.

3.) Consider the path $(1, \tilde{V}, V_1, V_3, V_2 \setminus \{k\}, v_{n-1}, k, n)$. We can conclude that $a_{v_{n-1},k} = 0$.



Case 1 Case 2 Case 3

Figure 5.3.4

4.) Consider the path $(1, \tilde{V}, V_1, V_3, v_{v_{n-1}}, V_2, n)$. We can conclude that $a_{j,v_{n-1}} = 0$ for all $j \in V_3$.

5.) Consider the path $(1, \tilde{V}, V_3, V_1, v_{n-1}, V_2, n)$. We can conclude that $a_{l,v_{n-1}} = 0$ for all $l \in V_1$.

6.) Consider the path $(1, \tilde{V}, V_1, v_{n-1}, V_3, V_2, n)$. We can conclude that $a_{v_{n-1},j} = 0$ for all $j \in V_3$.

Thus, we showed $a_{ij} = 0$ for all $(i, j) \in A$ and the result follows.

Case 2: $|V_2| = 2$
Assume $V_2 = \{k, m\}$.

1.) Consider the path $(1, \tilde{V}, V_1, V_3, m, v_{n-1}, k, n)$. As all coefficients except $a_{v_{n-1},k}$ are 0, we can conclude that $a_{v_{n-1},k} = 0$.

2.) Consider the path $(1, \tilde{V}, V_3, V_1, v_{n-1}, m, k, n)$. We can conclude that $a_{l,v_{n-1}} = \beta$ for all $l \in V_1$ and $a_{v_{n-1},m} = -\beta$.

3.) Consider the path $(1, \tilde{V}, V_3, V_1, v_{n-1}, k, m, n)$. We can conclude that $a_{m,n} = -\beta$.

4.) Consider the path $(1, \tilde{V}, V_1, V_3, k, v_{n-1}, m, n)$. We can conclude that $a_{v_{n-1},m} = -a_{m,n}$.

Therefore, $\beta = 0$. The paths

5.) $(1, \tilde{V}, V_1, v_{n-1}, V_3, V_2, n)$,

6.) $(1, \tilde{V}, V_1, V_3, v_{n-1}, V_2, n)$

complete the proof of this case.

Case 3: $|V_2| = 1$
Assume $V_2 = \{k\}$. First, note that due to the regularity of the instance there exists a node $l \in V_2$ such that k can be sequenced before l in a feasible Hamiltonian path.

1.) Consider the path $(1, \tilde{V}, V_3, V_1, v_{n-1}, k, n)$. We can conclude that $a_{l,v_{n-1}} = \beta$ for all $l \in V_1$ and $a_{v_{n-1},k} = -\beta$.

2.) Consider the path $(1, \tilde{V}, V_1, V_3, v_{n-1}, k, n)$. We can conclude that $a_{j,v_{n-1}} = \beta$ for all $j \in V_2$.

3.) Consider the path $(1, \tilde{V}, V_1, v_{n-1}, V_3, k, n)$. We can conclude that $a_{v_{n-1},j} = -\beta$ for all $j \in V_2$.

4.) Consider the path $(1, \tilde{V}, V_3, k, l, V_1 \setminus \{l\}, v_{n-1}, n)$. We can conclude that $a_{l,v_{n-1}} = 0$.

Therefore, $\beta = 0$, and $a_{ij} = 0$ for all $(i,j) \in A$. The discussion of this case completes the proof. □

We now prove the result for the case where $|F| \neq \emptyset$.

(5.3.7) Theorem.
Suppose we are given an instance (V, A, R) of the SOP on $n \geq 4$ nodes that decomposes into regular instances. Then

$$dim(SOP(n,P)) = |A| - 2n + 3 + |F|.$$

Proof. We prove the result by induction over $|F|$.

For $|F| = 0$ the result holds due to Theorem (5.3.6).

Suppose $|F| \geq 1$. Choose the node $m \in F$, such that $i \prec m$ for all other nodes $i \in F \setminus \{m\}$ and split the original instance at m into two smaller instances (V_1, A_1, R_1) and (V_2, A_2, R_2), where

$$\begin{array}{lcllcl} V_1 & := & \pi(m) \cup m, & V_2 & := & \sigma(m) \cup m, \\ A_1 & := & \{(i,j) \in A \mid i,j \in V_1\}, & A_2 & := & \{(i,j) \in A \mid i,j \in V_2\} = A \setminus A_1, \\ R_1 & := & \{(i,j) \in R \mid i,j \in V_1\}, & R_2 & := & \{(i,j) \in R \mid i,j \in V_2\}. \end{array}$$

Due to the assumption these two instances are regular. Set $n_1 := |V_1|$ and $n_2 := |V_2|$ and let F_1 denote the set of all fixed nodes in the instance (D_1, A_1, R_1). As node m occurs in each instance, we know that $n_1 + n_2 = n + 1$. Furthermore, we know that $|F_1| = |F| - 1$. As a solution to the original instance can be obtained by first solving the problems on the smaller instances and then composing the two solution, we know that

$$\begin{array}{rcl} dim(SOP(n,P)) & = & dim(SOP(n_1,P_1)) + dim(SOP(n_2,P_2)) \\ & = & |A_1| - 2n_1 + 3 + |F_1| + |A_2| - 2n_2 + 3 \\ & = & |A| - 2(n_1 + n_2) + 5 + |F| \\ & = & |A| - 2n + 3 + |F| \end{array}$$

□

We conjecture that the result holds as well for the case of nonregular instances. In the proof of Theorem (5.3.6) the assumption that after shrinking two nodes the shrunken instance does not contain a fixed node is essential for the induction step.

It is easy to see that the argumentation as outlined in the proof can be applied if instead nodes v_{n-1} and n, nodes 1 and one of its direct successors are shrunk. Just the role of predecessors and successors changes. We believe that a similar argumentation can be applied to any two nodes $i, j \in V$ such that the shrunken instance does not contain a cutnode. But here it is necessary to simultaneously maintain the predecessor and successor structure of the instance.

Finally, we would like to mention that the conjecture that the dimension formula of Theorem (5.3.6) holds for general instances has been verified by means of a computer program for small instances ($n \leq 7$ and varying precedence digraphs $P = (V, R)$).

5.4 Valid inequalities known from the literature

Not much attention has been paid to the polyhedral investigation of the precedence constrained travelling salesman problem. In this section we summarize the classes of valid inequalities for the SOP that are known from the literature. These are mainly inequalities very recently developed by Balas, Fischetti, and Pulleyblank [BFP92].

5.4.1 Precedence forcing constraints

Ascheuer [Asc89] and Ascheuer et al. [AEGS93] describe the implementation of a cutting plane approach for the asymmetric case. The inequalities they use are mainly ATSP–like constraints. They use only one class of inequalities that especially have been designed to take precedences into account, the so–called precedence forcing constraints. A similar class of inequalities was independently developed by Balas & Pulleyblank (cmp. [BFP92]). Although these inequalities are not strong enough to be used in a computational frame, we list them for the sake of completeness.

(5.4.8) Lemma.
Let $(i,j) \in R$ be a precedence relationship, then for all $W \subseteq V \setminus \{i,j\}, W \neq \emptyset$, the so–called **precedence forcing constraints**

$$x(j:W) + x(A(W)) + X(W:i) \leq |W|$$

are valid with respect to $SOP(n,P)$. □

Ascheuer et al. [AEGS90] have shown that this class of inequalities can be separated in polynomial time. Roughly speaking their approach is the following. For each precedence relationship $(i,j) \in R$ shrink i and j to a single node v_{ij} and look for a violated subtour elimination constraint (containing v_{ij}) in the shrunken digraph. This requires an overall computing time of $O(|R| \cdot n^3) \sim O(n^5)$.
Balas, Fischetti, and Pulleyblank [BFP92] have carried out polyhedral investigations for the precedence constrained ATSP and developed several classes of valid inequalities. They remark that the precedence forcing constraints can be strengthened to

$$x(j:W) + x(A(W)) + X(W:i) + x(W:j) \leq |W|$$

or

$$x(j:W) + x(A(W)) + X(W:i) + x(i:W) \leq |W|.$$

But several classes of the inequalities Balas et al. [BFP92] present, strictly dominate these strengthened versions of the precedence forcing constraints. Furthermore, they give polynomial time separation algorithms for several of these classes that run in $O(n^4)$ time. Therefore, the precedence forcing constraints are not used in the implementation of the branch&cut algorithm that is described in Section 5.7.

5.4.2 Predecessor Inequalities (π–inequalities)

Recall that $\pi(S)$ was defined to be the set of predecessors of S, i.e.,

$$\pi(S) := \{i \in V \mid \exists j \in S, \text{s.t. } (i,j) \in R\}.$$

Note that $\pi(S)$ and S may intersect.

(5.4.9) Theorem. (Balas, Fischetti, Pulleyblank, '92)
Let $S \subseteq V \setminus \{1, n\}, \bar{S} := V \setminus S$, then the **predecessor inequality (π–inequality)**

$$x(S \setminus \pi(S) : \bar{S} \setminus \pi(S)) \geq 1$$

is valid for $SOP(n, P)$.

Proof. See [BFP92]. □

The π–inequality implies that at least once a feasible Hamiltonian path has to leave S through a node that has no predecessor in S and enter $\bar{S}$ via a non predecessor node of S. Otherwise, a precedence relationship is violated. Balas, Fischetti, and Pulleyblank have shown that the inequalities of this class are facet defining for the precedence constrained ATS–polytope $P_{PC}(n, P)$ if there exits a free node and if $\pi(s) \subset S$ and $\sigma(S) \subset S$.

The π–inequalities can be rewritten as

$$x(A(S)) + x(S : \bar{S} \cap \pi(S)) + x(S \cap \pi(S) : \bar{S} \setminus \pi(S)) \leq |S| - 1. \tag{5.4.10}$$

This demonstrates that the π–inequality is a strengthening of the subtour elimination constraint $x(A(S)) \leq |S| - 1$.

It is not known if the π–inequalities can be separated in polynomial time, but Balas, Fischetti, and Pulleyblank have given a polynomial time separation algorithm for a smaller class of inequalities, the so–called weak π–inequalities. These inequalities are obtained by substituting the node set S by a single node j.

(5.4.11) Definition.
For a given $j \in V$ such that $\pi(j) \neq \emptyset$ and any $S \subset V$ such that $j \in S$, the inequality

$$x(S \setminus \pi(j) : \bar{S} \setminus \pi(j)) \geq 1$$

is called a **weak π–inequality**. ◇

(5.4.12) Exact separation procedure for weak π–inequalities.

Input : $D = (V, A), P = (V, R)$ and a fractional LP–solution $\bar{x}$.
Output : violated weak π–inequality or the answer that no such violated inequality exists.

For all $j \in V \setminus \{n\}$ with $\pi(j) \neq \emptyset$ do :

1. Construct a temporary digraph $D' = (V', A')$ by deleting all nodes in $\pi(j)$ and all incident arcs, i.e.,
   $V' = V \setminus \pi(j)$,
   $A' = \{(i, j) \in A \mid i, j \in V'\}$.
2. Associate arc weights $\bar{c}_{ij} = \bar{x}_{ij} \; \forall \; (i, j) \in A'$.
3. Calculate a minimum (j, n)–cut $\delta^-(S)$ in D'.
4. if $\bar{c}(\delta^-(S)) < 1$ then j and S violate a weak π–inequality (5.4.11), else no weak π–inequality is violated with respect to j.

□

Note that a violated weak π–inequality can be strengthened by replacing $\pi(j)$ by $\pi(S)$. The overall complexity of the algorithm is $O(n^4)$ assuming that the complexity of the maxflow calculation in step 3 is $O(n^3)$. This is better than the complexity for the separation algorithm of the precedence forcing constraints (PFC) presented in [AEGS90], which is more or less $O(n^5)$. Moreover, in [BFP92] it has been shown that the π–inequalities and even the weak π–inequalities strictly dominate the strengthened precedence forcing constraints.

5.4.3 Successor Inequalities (σ-inequalities)

The next class of inequalities is similar to the π–inequalities but the set of predecessors is substituted by the set of successors. Recall that $\sigma(S)$ was defined to be the set of successors of S, i.e.,

$$\sigma(S) := \{j \in V \mid \exists i \in S, \text{s.t. } (i,j) \in R\}.$$

Note that $\sigma(S)$ and S may intersect.

(5.4.13) Theorem. (Balas, Fischetti, Pulleyblank, '92)
Let $S \subseteq V \setminus \{1,n\}, \bar{S} := V \setminus S$, then the **successor inequality (σ–inequality)**

$$x(\bar{S} \setminus \sigma(S) : S \setminus \sigma(S)) \geq 1$$

is valid for $SOP(n,P)$.

Proof. See [BFP92]. □

It was shown that the inequalities of this class are facet defining for $P_{PC}(n,P)$ under the same conditions as for the π–inequalities.

As it was the case for the π–inequalities, the σ–inequalities are a strengthening of the subtour elimination constraints. They can be written as

$$x(A(S)) + x(\bar{S} \cap \sigma(S) : S) + x(\bar{S} \setminus \sigma(S) : S \cap \sigma(S)) \leq |S| - 1. \tag{5.4.14}$$

No exact separation algorithm is known for the σ–inequalities, but a smaller class of inequalities, the weak σ–inequalities, can be separated in polynomial time.

(5.4.15) Definition.
For a given $j \in V$ such that $\sigma(j) \neq \emptyset$ and any $S \subset V$ such that $j \in S$, the inequality

$$x(\bar{S} \setminus \sigma(j) : S \setminus \sigma(j)) \geq 1$$

is called a **weak σ–inequality**. ◇

In [BFP92] it has been shown that even the weak σ–inequalities strictly dominate the strengthened precedence forcing constraints.

As the σ–inequality is the counterpart of the π–inequality by replacing predecessors by successors, it is obvious that the separation procedure for the weak σ–inequalities will follow the same philosophy as described above.

(5.4.16) Exact separation procedure for weak σ–inequalities.

Input : $D = (V, A), P = (V, R)$ and a fractional LP–solution $\bar{x}$
Output : violated weak σ–inequality or the answer that no such inequality is violated.

For all $j \in V \setminus \{1\}$ with $\sigma(j) \neq \emptyset$ do :

1. Construct a temporary digraph $D' = (V', A')$ by deleting all nodes in $\sigma(j)$ and all incident arcs, i.e.,
   $V' = V \setminus \sigma(j)$
   $A' = \{(i, j) \in A \mid i, j \in V'\}$.
2. Associate arc weights $\bar{c}_{ij} = \bar{x}_{ij} \; \forall \; (i, j) \in A'$.
3. Calculate a minimum $(1, j)$–cut $\delta^-(S)$ in D'.
4. If $\bar{c}(\delta^-(S)) < 1$ then j and S violate a weak σ–inequality (5.4.15), else no weak σ–inequality is violated with respect to j.

□

In analogy to the π–inequalities the overall complexity of the separation algorithm is $O(n^4)$ assuming that the complexity of the maxflow calculation in step 3 is $O(n^3)$. Note that a violated weak σ–inequality can be strengthened by replacing $\sigma(j)$ by $\sigma(S)$.

5.4.4 Predecessor-Successor inequalities

This class of inequalities combines the two structures of predecessors and successors in one inequality.

(5.4.17) Theorem. (Balas, Fischetti, Pulleyblank, '92)
Let $X, Y \subseteq V$, s.t. $i \prec j \; \forall$ pairs $i \in X, j \in Y, W := \pi(X) \cup \sigma(Y)$. Then for all $S \subset V$, s.t. $X \subset S, Y \subset \bar{S}$

$$x(S \setminus W : \bar{S} \setminus W) \geq 1$$

is called a **predecessor–successor inequality or (π, σ)–inequality** and is valid with respect to $SOP(n, P)$.

Proof. See [BFP92]. □

In contrast to the π– and σ–inequalities there are no conditions known under which these inequalities are facet–defining. Similar to the π– and σ–inequalities the inequalities of this class are a strengthening of the subtour elimination constraints, as they can be written as

$$x(A(S)) + x(S : \bar{S} \cap W) + x(S \cap W : \bar{S} \setminus W) \leq |S| - 1. \tag{5.4.18}$$

Furthermore, they also strictly dominate the precedence forcing constraints. No polynomial time separation algorithm for the (π, σ)–inequalities is known. But a smaller class of inequalities can be separated in polynomial time. These are the weak (π, σ)–inequalities.

(5.4.19) Definition.
Let $X = \{i\}$ and $Y = \{j\}$ with $i \prec j$ and $W_{ij} := \pi(i) \cup \sigma(j)$. Then

$$x(S \setminus W_{ij} : \bar{S} \setminus W_{ij}) \geq 1$$

is called a **weak (π, σ)–inequality**. ◇

The (π, σ)–inequality is a combination of the π– and σ–inequalities. Also the separation procedure can be regarded as a combination of the two procedures (5.4.12) and (5.4.16).

(5.4.20) Exact separation procedure for weak (π,σ)–inequalities.

Input : $D = (V, A), P = (V, R)$ and a fractional LP–solution $\bar{x}$
Output : violated weak (π,σ)–inequality or
the answer that no such inequality is violated

For all $(i, j) \in R$ do :

1. Construct a temporary digraph $D' = (V', A')$ by deleting all nodes in $\pi(i)$ and $\sigma(j)$ and all incident arcs, i.e.,
$$V' = V \setminus (\pi(i) \cup \sigma(j))$$
$$A' = \{(i, j) \in A \mid i, j \in V'\}.$$
2. Associate arc weights $\bar{c}_{ij} = \bar{x}_{ij} \; \forall \; (i, j) \in A'$.
3. Calculate a minimum (i, j)–cut $\delta^-(S)$ in D'.
4. If $\bar{c}(\delta^-(S)) < 1$ then i, j and S violate a weak (π,σ)–inequality (5.4.19), else no weak (π,σ)–inequality is violated with respect to $(i, j) \in R$.

□

The complexity of that procedure is $O(|R| \cdot n^3)$, assuming that the complexity of the maxflow-algorithm used in step 3 is $O(n^3)$. Note that it is sufficient to run Procedure (5.4.20) only for $(i, j) \in R$ that are not transitively derived.

5.4.5 Precedence cycle breaking inequalities

The next class of inequalities is derived from the requirement that the precedence graph has to be acyclic.

(5.4.21) Theorem. (Balas, Fischetti, Pulleyblank, '92)
Let $S_1, .., S_m \subseteq V, m \geq 2$, be disjoint node sets such that $\sigma(S_i) \cap S_{i+1} \neq \emptyset$ with $S_{m+1} = S_1$. Then the **precedence cycle breaking inequality** (or **pcb–inequality**)

$$\sum_{i=1}^{m} x(A(S_i)) \leq \sum_{i=1}^{m} |S_i| - m - 1 \tag{5.4.22}$$

is valid for $SOP(n, P)$.

Proof. See [BFP92]. □

If $|S_1| > 3$ the pcb–inequalities strictly dominate the precedence forcing constraints.

(5.4.23) Example.
In the simplest form the precedence cycle breaking inequality for $m = 2$ and $|S_2| = 1$ (see Figure 5.4.5) is of the form

$$x(A(S_1)) \leq |S_1| - 2. \tag{5.4.24}$$

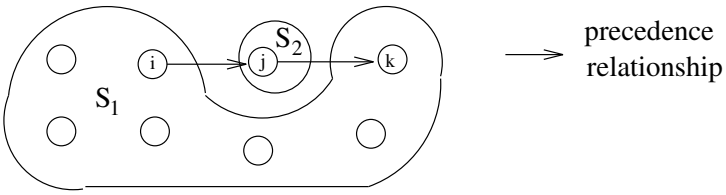


Figure 5.4.5

This inequality will be called a **simple pcb–inequality**.
Due to (5.4.24) this is also a strengthening of the subtour elimination constraint where the right hand side is decreased by 1. This can be done whenever there exists a pair $i, k \in S_1, j \notin S_1$ such that $i \prec j \prec k$. □

5.4.6 A lifting procedure

Balas, Fischetti, and Pulleyblank [BFP92] describe a lifting procedure that is similar to the clique lifting procedure for facet defining inequalities for ATSP(n) as introduced in [BF93b].

Suppose we are given a digraph $D = (V, A)$ and a precedence digraph $P = (V, R)$. Consider a partition $S_1, ..., S_m$ of V into nonempty sets. Let $\tilde{D}_m = (\tilde{V}, \tilde{A}_m)$ be a digraph on m nodes, where each node $v \in \tilde{V}$ corresponds to a node set S_v of the partition. The precedence structure can be extended to $\tilde{D}_m$. Therefore, introduce for two given nodes $u, v \in \tilde{V}$ the precedence relationship $u \prec v$ if there exists $s \in S_u$ and $t \in S_v$ such that $s \prec t$. Let $\tilde{P}$ be the precedence digraph with respect to the digraph $\tilde{D}_m$. We can assume that $\tilde{P}$ is transitively closed.

The digraph $\tilde{D} = (\tilde{V}, \tilde{A})$ is obtained from $\tilde{D}_m$ by deleting all arcs fixed to 0 due to Lemma (5.2.1). You might think of $\tilde{D}$ and $\tilde{P}$ as obtained by replacing the node sets $S_v, v = 1, ..., m$, by a single node v in a such a way that the precedence relations between the sets are "preserved". Let $cl(i)$ denote the cluster (or node set) in $\tilde{D}$ containing node $i \in V$. Set

$$\begin{aligned} B &= \{(i,j) \in A \mid (cl(i), cl(j)) \in \tilde{A}\}, \\ \bar{B} &= \{(i,j) \in A \mid (cl(i), cl(j)) \notin \tilde{A}\}, \end{aligned}$$

i.e., B is the set of arcs of A that are also present in $\tilde{A}$, whereas $\bar{B}$ are the set of arcs that are not (due to the precedence relationships among the clusters S_i). Furthermore, let $I := \{i \in \{1, ..., m\} \mid |S_i| \geq 2\}$.

(5.4.25) Lifting Theorem. (Balas, Fischetti, Pulleyblank, 1992)
Let $\tilde{a}y \leq \tilde{a_0}$ be a valid inequality for $SOP(m, \tilde{P})$ and let σ_{ij} be arbitrary (finite) numbers. There exists $\mu_k^0, k \in I$, such that for any $\mu_k \geq \mu_k^0, k \in I$, the inequality $ax \leq a_0$ is valid for $SOP(n, P)$ with

$$a_{ij} = \begin{cases} \mu_{cl(i)} (= \mu_{cl(j)}), & \text{if } cl(i) = cl(j) \text{ and } (i,j) \in A, \\ \tilde{a}_{cl(i)cl(j)}, & \text{if } cl(i) \neq cl(j) \text{ and } (i,j) \in B, \\ \sigma_{ij}, & \text{if } cl(i) \neq cl(j) \text{ and } (i,j) \in \bar{B}, \end{cases}$$

and

$$a_0 := \tilde{a_0} + \sum_{i \in I} \mu_i (|S_i| - 1).$$

Proof. See [BFP92]. □

In case that no precedence relationships are present this reduces to the lifting procedure given by Balas and Fischetti [BF93b] with

$$\mu_{cl(i)} = \max\{\tilde{a}_{cl(p)cl(i)} + \tilde{a}_{cl(i)cl(q)} - \tilde{a}_{cl(p)cl(q)} \mid \begin{array}{l} cl(p), cl(q) \in \tilde{V} \setminus \{cl(i)\}, \\ cl(p) \neq cl(q) \end{array} \}.$$

This procedure is based on **cloning**, a technique where each node that is added to the graph becomes an indistinguishable copy of some other node with respect to the inequality to be lifted. The difference to the procedure above is that each node of a set S_i may be required to follow distinct sets of nodes, i.e., may have distinct sets of predecessors and/or successors in D.

The lifting procedure may be interpreted as follows. Suppose we are given a valid inequality $\tilde{a}y \leq \tilde{a_0}$ for $SOP(m, \tilde{P})$, we define an inequality $a'x \leq \tilde{a_0}$ by setting

$$a'_{ij} = \begin{cases} 0, & \text{if } cl(i) = cl(j) \text{ and } (i,j) \in A \\ \tilde{a}_{cl(i)cl(j)}, & \text{if } cl(i) \neq cl(j) \text{ and } (i,j) \in B \\ \sigma_{ij}, & \text{if } cl(i) \neq cl(j) \text{ and } (i,j) \in \bar{B}. \end{cases}$$

$a'x \leq \tilde{a_0}$ is valid for $SOP(n, P)$ if every Hamiltonian path is required to enter and exit each set $S_k, k = 1, ..., m$, exactly once, i.e., each subtour elimination constraint $x(A(S_k)) \leq |S_k| - 1$ must hold with equality for each node set S_k with $|S_k| \geq 2$. Hence, $ax \leq a_0$ may now be regarded as a nonnegative linear combination of $a'x \leq \tilde{a_0}$ weighted with 1, and the subtour elimination constraints for clusters S_k, s.t. $|S_k| \geq 2$, weighted with μ_k.

Note that $a'x \leq \tilde{a_0}$ might not be valid for $SOP(n, P)$, but the combined inequality $ax \leq a_0$ becomes valid if the μ_k are chosen sufficiently large enough. In order to obtain a tight inequality the problem is to find the lowest value μ_k^0 for which $ax \leq a_0$ is valid for $SOP(n, P)$.

Inequalities obtained by applying the lifting procedure

We close this section by giving a list of inequalities valid for $SOP(n, P)$ as they were given by Balas, Fischetti, and Pulleyblank [BFP92]. These classes of inequalities are obtained by applying Theorem (5.4.25) to classes of inequalities that are trivially valid for $SOP(m,\tilde{P})$.

(5.4.26) Lemma.
Let $S_1, S_2, S_3 \subset V \setminus \{1, n\}$ be disjoint node sets, with $\sigma(S_1) \cap S_2 \neq \emptyset$, $\sigma(S_2) \cap S_3 \neq \emptyset$. The following inequalities are valid for $SOP(n, P)$:

(a) $x(S : S) + x(1 : S) \leq |S| - 1, \quad S \subset V \setminus \{1\}, \text{ s.t. } \pi(S) \setminus S \neq \emptyset,$

(b) $x(S : S) + x(S : n) \leq |S| - 1, \quad S \subset V \setminus \{n\}, \text{ s.t. } \sigma(S) \setminus S \neq \emptyset,$

(c) $x(S_2 : S_2) + x(S_2 : S_1) + x(S_1 : S_1) \leq |S_2| + |S_1| - 2,$

(d) $\sum_{i=1}^{3} x(S_i : S_i) + x(S_1 : S_3) \leq |S_1| + |S_2| + |S_3| - 3.$

Proof. Apply lifting Theorem (5.4.25) to the following inequalities:

(a) $y_{1i} \leq 0$, for $i \in \tilde{V}$, s.t. $\tilde{\pi}(i) \neq \emptyset$.

(b) $y_{in} \leq 0$, for $i \in \tilde{V}$, s.t. $\tilde{\sigma}(i) \neq \emptyset$.

(c) $y_{ij} \leq 0$ for $i, j \in \tilde{V}$, s.t. $(i, j) \in \tilde{R}_1$.

(d) $y_{ik} \leq 0$ for $i, k \in \tilde{V}$, s.t. $(i, k) \in \tilde{R}_2$.

For more details see [BFP92]. □

5.5 New valid inequalities

In the literature several classes of facet defining inequalities are known for the asymmetric Hamiltonian path polytope or equivalently for the asymmetric travelling salesman polytope (cf. [Grö77], [GP85b], [Fis91] among others). As the SO-polytope is contained in the AHP-polytope these inequalities are at least valid for $SOP(n,P)$. But they can be strengthened as they do not take the precedences at all into account. In the π–inequalities, σ–inequalities and (π,σ)–inequalities we have already seen examples for strengthened subtour elimination constraints. It is not clear if these classes of inequalities cover all possible strengthenings of the subtour elimination constraints. One should note that dependent on the structure of the precedences one obtains different inequalities.

In this section we give more examples of facet defining inequalities for the asymmetric travelling salesman polytope that can be strengthened in case that precedences are involved. This will be done for D_3–inequalities, T_k–inequalities and 2–matching constraints.

5.5.1 Strengthened D_3-inequalities

In Section 1.3 the so–called D_k^-–inequalities

$$\sum_{j=1}^{k-1} x_{i_j i_{j+1}} + x_{i_k i_1} + 2\sum_{j=3}^{k} x_{i_1 i_j} + \sum_{j=4}^{k}\sum_{h=3}^{j-1} x_{i_j i_h} \leq k-1.$$

and D_k^+–inequalities

$$\sum_{j=1}^{k-1} x_{i_j i_{j+1}} + x_{i_k i_1} + 2\sum_{j=2}^{k-1} x_{i_j i_1} + \sum_{j=3}^{k-1}\sum_{h=2}^{j-1} x_{i_j i_h} \leq k-1$$

have been introduced. Note that for $k=3$ they define the same inequality. In this section we discuss how D_3–inequalities can be strengthened in case that the cycle nodes are involved in precedence relationships.

(5.5.27) Lemma.
Let $n \geq 4, i,j,k \subset V, S=\{i,j,k\}$. Then

$$\begin{array}{lll} (a) & x_{ij}+x_{jk}+x_{ki}+2x_{ik}+x(\bar{S}\cap\sigma(\{i,j\}):k)+x(\bar{S}\cap\sigma(k):i) & \leq 2 \\ (b) & x_{ij}+x_{jk}+x_{ki}+2x_{ik}+x(i:\bar{S}\cap\pi(\{j,k\}),k)+x(k:\bar{S}\cap\pi(i)) & \leq 2 \end{array}$$

are valid for $SOP(n,P)$.

Proof.

(a) Let H be any feasible Hamiltonian path. Let $a \in (\bar{S}\cap\sigma(\{i,j\}):k)$ and $b \in (\bar{S}\cap\sigma(k):i)$.
If neither $a \in H$ nor $b \in H$, the inequality reduces to the D_3, that is facet defining for $P_T^(n)$ and is therefore valid for $SOP(n,P)$.
If $a \in H$ and $b \in H$, only arc (i,j) can be used without violating the degree constraints. But this violates some of the precedence relationship, as either j or k is sequenced after his successor.
If only $a \in H$, it follows that (i,k) and $(j,k) \notin H$. If both (k,i) and $(i,j) \in H$ either i or j is sequenced after its successor.
If only $b \in H$, neither $(i,k) \notin H$ nor $(k,i) \notin H$, as either $k \prec \sigma(k)$ or a degree constraint is violated. If both (i,j) and $(j,k) \in H$ the precedence relationship $k \prec \sigma(k)$ is violated.

(b) Similar arguments.

□

Figure 5.5.6 shows the support digraphs of two examples for strengthened D_3–inequalities.

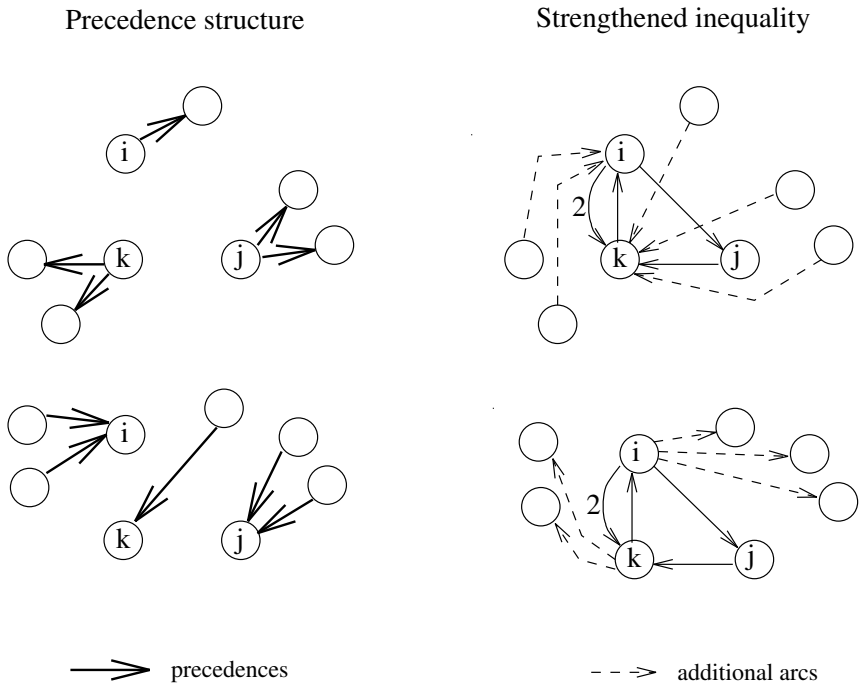


Figure 5.5.6 Strengthened D_3–inequalities

5.5.2 Strengthened T_k–inequalities

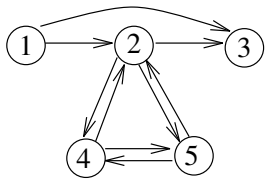
The inequalities in this class are valid for P_T^n and most of these inequalities are known to be facet defining for P_T^n (see [GP85b] for more details). Therefore, we have the following lemma.

(5.5.28) Lemma.
Let $W \subset V, 2 \leq |W| = k \leq n-2$, let $w \in W$ and $p, q \in V \setminus W$, then

$$x(A(W)) + x_{pw} + x_{pq} + x_{wq} \leq k$$

is called a T_k–**inequality** and is valid with respect to $SOP(n, P)$. □

The support graph of a T_3–inequality with $W = \{2, 4, 5\}$ is shown in Figure 5.5.7.



$$\begin{aligned} &x_{12} + x_{13} + x_{23} + x_{24} + x_{42} + \\ &x_{25} + x_{52} + x_{45} + x_{54} \qquad \leq \quad 3 \end{aligned}$$

Figure 5.5.7 T_3–inequality

Note that the T_k–inequality is obtained from the subtour elimination constraint (SEC) $x(A(W)) \leq |W|-1$ by applying the so–called T–lifting procedure (see [Fis92]), i.e., a "source" p and a "sink" q are attached and the right hand side is increased by 1. In Section 5.4 we have seen several strengthenings of SECs in case that precedences are involved. This can also be applied to the "SEC–part" of the T_k–inequalities. But we have to take care of the node w connecting the "source" p and the "sink" q to the SEC. This node has to be excluded from the determination of the predecessor and successor sets. Further classes of inequalities are obtained in case that the tournament (p, w, q) is involved in precedence relationships. The classes of inequalities derived from T_k–inequalities are given in Theorems (5.5.29), (5.5.30), and (5.5.31).

(5.5.29) Theorem.
Let $W \subset V, 2 \leq |W| = k \leq n-2$, $w \in W$, $p, q \in V \setminus W$, $\tilde{W} = W \setminus \{w\}$, and $x(A(T_k)) := x(A(W)) + x_{pw} + x_{pq} + x_{wq}$. Then

$$\begin{aligned} &(a) \quad x(A(T_k)) + x(\bar{W} \cap \sigma(\tilde{W}) : W) + x(\bar{W} \setminus \sigma(\tilde{W}) : W \cap \sigma(\tilde{W})) \quad \leq k \\ &(b) \quad x(A(T_k)) + x(W : \bar{W} \cap \pi(\tilde{W})) + x(W \cap \pi(\tilde{W}) : \bar{W} \setminus \pi(\tilde{W})) \quad \leq k \end{aligned}$$

are valid with respect to $SOP(n, P)$.

Proof.

(a) Let $A_1 := (\bar{W} \cap \sigma(W) : W)$, $A_2 := (\bar{W} \setminus \sigma(\tilde{W}) : W \cap \sigma(W))$, $m := x(A_1) + x(A_2)$ and $\gamma = x_{pw} + x_{wq} + x_{pq}$.
If $m = 0$ we have the standard T_k–inequality, as given in (5.5.28).
Therefore, assume $m > 0$.
We have to show that $x(A(W)) + m + \gamma \leq |W|$.
If $\gamma \in \{0, 1\}$, we know that $x(A(W)) \leq |W| - 1 - m$, otherwise a precedence relationship would be violated.
If $\gamma = 2$ we obtain $x(A(W)) \leq |W| - 2 - m$. The result follows.

(b) Similar arguments. □

Figure 5.5.8 shows the support graph of a strengthened T_3–inequality due to Theorem (5.5.29(a)). (To simplify the figure the arcs corresponding to $A(W)$ are not drawn.)

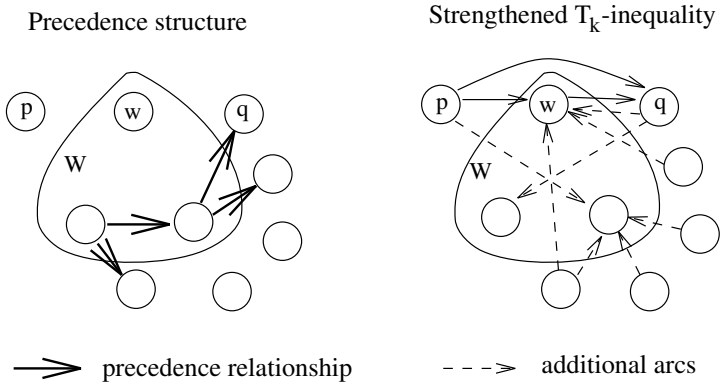


Figure 5.5.8 Strengthened T_3–inequalities

For the SEC we have seen that under certain conditions to the precedence structure it is possible to reduce the ride hand side by 1 (see 5.4.24). This is also true for the T_k–inequalities. Furthermore, if the tournament nodes p, w, q are involved, it is possible to lift the arcs in $(p : \tilde{W})$ or $(\tilde{W} : q)$. This is summarized in the following theorem.

(5.5.30) Theorem.
Let $p, q, w, W, \tilde{W}, x(A(T_k))$ be defined as in Theorem (5.5.29).

(a) If there exist nodes $i, k \in \tilde{W}$ and $j \notin W \cup \{p, q\}$, such that $i \prec j \prec k$, then the inequality

$$x(A(T_k)) \leq k - 1$$

is valid with respect to $SOP(n, P)$.

(b) If there exists an $i \in \tilde{W}$ such that $i \prec q \prec w$, then the inequality

$$x(A(T_k)) + x(p : \tilde{W}) + x_{wp} \leq k - 1$$

is valid with respect to $SOP(n, P)$.

(c) If there exists an $k \in \tilde{W}$ such that $w \prec p \prec k$, then the inequality

$$x(A(T_k)) + x(\tilde{W} : q) + x_{qw} \leq k - 1$$

is valid with respect to $SOP(n, P)$.

Proof.

(a) Let $\delta = x_{pw} + x_{pq} + x_{wq}$.
The validity of the inequalities follows from the fact that $x(A(W)) \leq |W| - 2$ if $\delta \in \{0, 1\}$, and $x(A(W)) \leq |W| - 3$ if $\delta = 2$. Otherwise, a precedence relationship related to $i \prec j \prec k$ would be violated.

(b) As we know that a feasible path has to leave and reenter W at least once in order to satisfy the precedence relationship related to $i \prec q \prec w$, we know that

$$x(A(W)) \leq |W| - 2.$$

Let $\gamma = x_{pw} + x_{pq} + x(p : \tilde{W}) + x_{wq}$. As $\gamma \leq 2$, it remains to show that $x(A(W)) \leq |W| - 3$ whenever $\gamma = 2$.
$\gamma = 2$ implies $x_{wq} = 1$. We can conclude that $x_{pw} + x_{pq} = 0$, otherwise a precedence relationship or a SEC is violated.
On the other hand, if $x(p : \tilde{W}) = 1$ we obtain a subpath $(w, p, k), k \in \tilde{W}, k \neq i$. We entered W twice and in order not to violate the precedence relationship related to $i \prec q \prec w$ is violated. The result follows.

(c) Similar arguments as in (b). □

Figure 5.5.9 shows the support graph of a strengthened T_3–inequality due to Theorem (5.5.30) with ride hand side of 2. The arcs corresponding to $A(W)$ are not drawn.

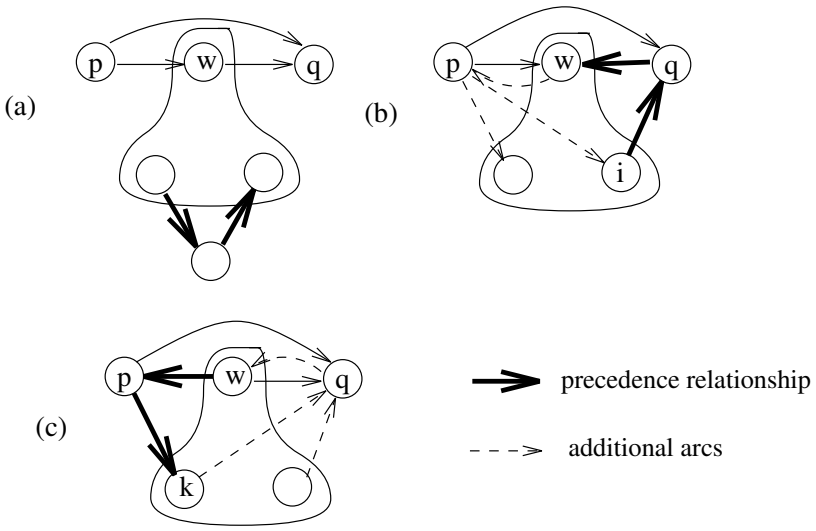


Figure 5.5.9 Strengthened T_3–inequalities

It should be mentioned that no strengthenings are possible if i, j, or k coincides with the intersection node w.
It is possible to obtain further classes of valid inequalities for $SOP(n, P)$ in case that the nodes p, w, q are involved in precedences.

(5.5.31) Theorem.
Let $p, q, w, W, \tilde{W}, x(A(T_k))$ be defined as in Theorem (5.5.29).

(a) If $w \prec p$ or $q \prec p$, then

$$x(A(T_k)) + x(p : \tilde{W}) + x(q : W) \leq k$$

is valid with respect to $SOP(n, P)$.

(b) If $q \prec w$, then

$$x(A(T_k)) + x_{wp} + x(\tilde{W} : q) \leq k$$

is valid with respect to $SOP(n, P)$.

Proof.

(a) We first prove the result for $w \prec p$.
Let $m := x(p : \tilde{W}) + x(q : W)$, and $\delta := x_{pq} + x_{wq}$. Note that $x_{pw} = 0$.
We know that $\delta + x_{qw} \leq 1$, and $x(A(W)) \leq |W| - 1 - m$. Otherwise, the precedence relationship $w \prec p$ would be violated. The result follows.

Now consider the case that $q \prec p$.
Let $m := x(p : \tilde{W}) + x(q : W)$ and $\gamma = x_{pw} + x_{wq} + x_{qw}$. We know that $\gamma \leq 1$. (Note that $x_{pq} = 0$.)
It is easy to see that $x(A(W)) \leq |W| - \gamma - m$, otherwise the precedence relationship $q \prec p$ would be violated. The result follows.

(b) Analogous to the case $w \prec p$, discussed in (a).

□

Figure 5.5.10 shows the support graph of two strengthened T_3–inequalities due to Theorem (5.5.31). Once again the arcs of $A(W)$ are not drawn.

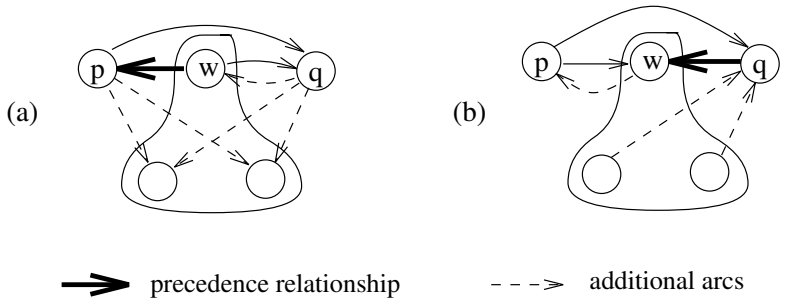


Figure 5.5.10 Strengthened T_3–inequalities

5.5.3 Strengthened 2–Matching constraints

(5.5.32) Lemma.
Let $D_n = (V, A)$ be a complete digraph on $n \geq 6$ nodes, and assume that $H, T_1, T_2, ..., T_k \subset V, k \geq 1$ are vertex sets satisfying

$$\begin{array}{ll} |H \cap T_i| = 1 & i = 1, ..., k, \\ |T_i \setminus H| = 1 & i = 1, ..., k, \\ T_i \cap T_j = \emptyset & 1 \leq i < j \leq k, \\ k \geq 3 \text{ and odd }, \quad \text{or } k = 1 \text{ and } |H| \geq 4 & \end{array}$$

Then

$$x(A(H)) + \sum_{i=1}^{k} x(A(T_i)) \leq |H| + \frac{k-1}{2}$$

is called a **2–matching inequality** and is valid with respect to $SOP(n, P)$. □

Generalizations of the 2–matching constraints are known as comb– and clique–tree inequalities. As only 2–matching constraints are used in the branch&cut code we restrict ourselves to give strengthened versions just for 2–matching constraints. Similar to the inequalities discussed in the last subsections different classes of valid inequalities are obtained dependent on the structure of the precedences. This is summarized in the following Theorems (5.5.33), (5.5.34), and (5.5.35).

(5.5.33) Theorem.
Let $D_n = (V, A)$ be a complete digraph on $n \geq 6$ nodes, $P = (V, R)$ be a given precedence graph, let H and $T_1, ..., T_k$ be defined as in (5.5.32), $T = \cup_{i=1}^{k} T_i$, $S = H \cup T$. Let $s_i := H \cap T_i$, $t_i := T_i \setminus H$, and $x(2M) := x(A(H)) + \sum_{i=1}^{k} x(A(T_i))$.
Then the following inequalities are valid with respect to $SOP(n, P)$:

$$(a) \quad x(2M) \quad + \sum_{i=1}^{k} x(\bar{S} \cap \sigma(t_i) : s_i) + \sum_{i=1}^{k} x(s_i : \bar{S} \cap \pi(t_i)) \qquad \leq |H| + \tfrac{k-1}{2}$$

$$(b) \quad x(2M) \quad + \sum_{i=1}^{k} x((H \setminus T) \cap \sigma(s_i) : t_i) + \sum_{i=1}^{k} x(t_i : (H \setminus T) \cap \pi(s_i)) \quad \leq |H| + \tfrac{k-1}{2}$$

Proof. If none of the additional arcs is used, the inequalities above reduce to the standard 2–matching constraints and are therefore valid.

(a) Note that $x(\bar{S} \cap \sigma(t_i) : s_i) + x(s_i : \bar{S} \cap \pi(t_i)) + x(T_i) \leq 1$. Otherwise, a precedence relationship is violated. The result from the validity proof for the 2–matching constraints.

(b) Similar arguments as in (a) □

Figure 5.5.11 gives examples for strengthened 2–matching constraints due to Theorem (5.5.33) with right hand side of 6. The arcs corresponding to the support graph of the standard 2–matching constraint are not drawn in order to simplify the figure.

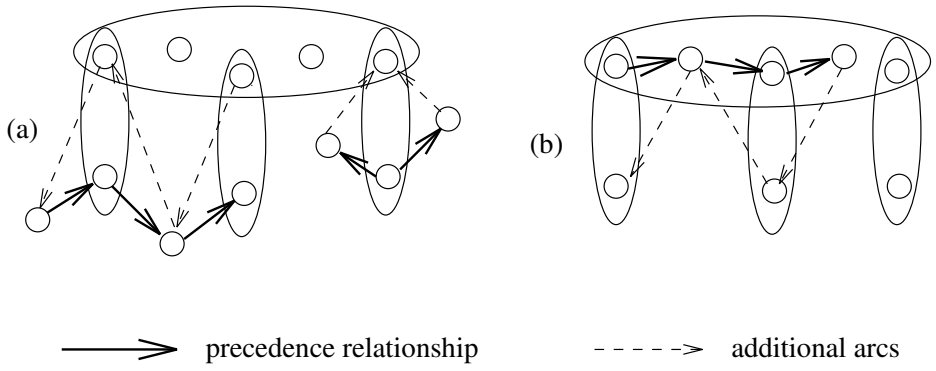


Figure 5.5.11 Strengthened 2–matching constraints

(5.5.34) Theorem.
Let $D_n = (V, A)$ be a complete digraph on $n \geq 6$ nodes, $P = (V, R)$ be a given precedence graph, let H and $T_1, ..., T_k$ be defined as in (5.5.32), $T = \cup_{i=1}^{k} T_i$, $S = H \cup T$. Let $s_i := H \cap T_i$, $t_i := T_i \setminus H$, and $x(2M) := x(A(H)) + \sum_{i=1}^{k} x(A(T_i))$.
Then the following inequalities are valid with respect to $SOP(n, P)$:

$$(a) \quad x(2M) + \sum_{i=1}^{k} x(H \setminus s_i : t_i \cap \sigma(T_i)) \leq |H| + \tfrac{k-1}{2}$$

$$(b) \quad x(2M) + \sum_{i=1}^{k} \sum_{\substack{j=1 \\ i \neq j}}^{k} x(s_i \cap \pi(T_i) : t_j) \leq |H| + \tfrac{k-1}{2}$$

$$(c) \quad x(2M) + \sum_{i=1}^{k} x(t_i \cap \pi(T_i) : H) \leq |H| + \tfrac{k-1}{2}$$

$$(d) \quad x(2M) + \sum_{i=1}^{k} \sum_{\substack{j=1 \\ i \neq j}}^{k} x(t_j : s_i \cap \sigma(T_i)) \leq |H| + \tfrac{k-1}{2}$$

Proof.

(a) Let $a \in (H \setminus s_i : t_i \cap \sigma(T_i))$ and let F be any feasible Hamiltonian path. To simplify notation set $x(A(T)) := \sum_{i=1}^{k} x(A(T_i))$.

If $a \notin F$, the result follows from the validity of the standard 2–matching constraint.

If $a \in F$, we know that no other arc in $(H : t_i \cap \sigma(T_i)) \cup (s_i, t_i)$ can be in F without violating the degree constraint for t_i. Therefore, we have $x(T_i) = 0$ and $x(A(T)) \leq k-1$.

Furthermore, as k is assumed to be odd, we know that

$$x(T) \leq \begin{cases} k-1, & \text{if } x(T) \text{ is even,} \\ k-2, & \text{if } x(T) \text{ is odd.} \end{cases}$$

Note that

$$x(A(H)) \leq \begin{cases} |H| - \frac{1}{2}x(A(T)) - 1, & \text{if } x(T) \text{ is even,} \\ |H| - \frac{x(A(T))-1}{2}, & \text{if } x(T) \text{ is odd.} \end{cases}$$

For the case that $x(A(T))$ is even it follows that

$$\begin{aligned} & x(A(H)) && \leq && |H| - \tfrac{x(A(T))}{2} - 1 \\ \Leftrightarrow \quad & x(A(H)) + x(A(T)) && \leq && |H| + x(A(T)) - \tfrac{x(A(T))}{2} - 1 \\ & && \leq && |H| + \tfrac{x(A(T))}{2} - 1 \\ & && \leq && |H| + \tfrac{k-1}{2} - 1 \\ & && = && |H| + \tfrac{k-3}{2} \end{aligned}$$

As $x(H : t_i \cap \sigma(T_i)) = 1$ we have

$$x(A(H)) + x(A(T)) + x(H : t_i \cap \sigma(T_i)) \leq |H| + \frac{k-1}{2}.$$

The case that $x(A(T))$ is odd is carried out in an analogous way.

(b) Similar arguments as in (a).

(c) Similar arguments as in (a).

(d) Similar arguments as in (a).

□

Figure 5.5.11 gives examples for strengthened 2–matching constraints due to Theorem (5.5.34) with right hand side 6. Note that the reverse arcs of the precedences are not present.

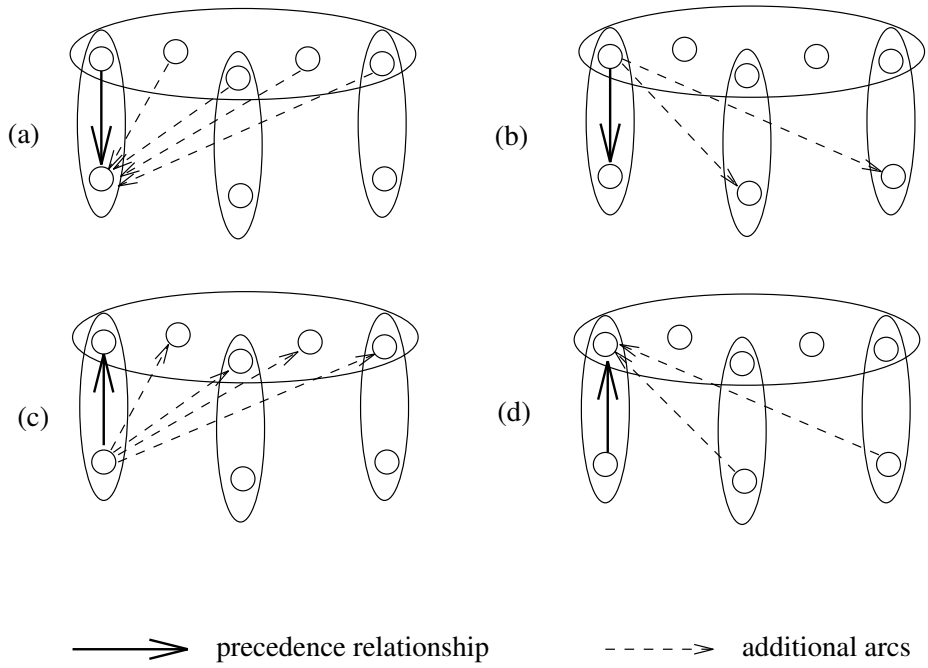


Figure 5.5.12 Strengthened 2–matching constraints

For the SEC it was possible to reduce the right hand side by 1 if a certain structure appeared in the precedences. The same holds for the 2–matching constraints but the structure is more complicated than for the SEC.

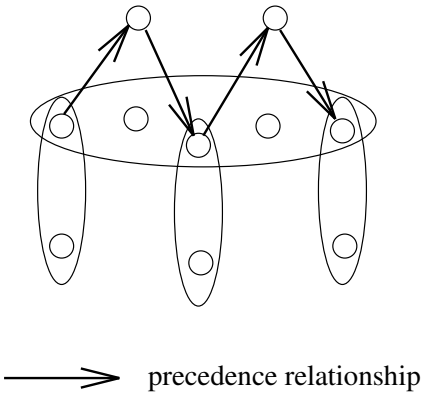


Figure 5.5.13 Strengthened 2–matching constraints

(5.5.35) Theorem.
Let $D_n = (V, A)$ be a complete digraph on $n \geq 6$ nodes, $P = (V, R)$ be a given precedence graph, let H and $T_1, ..., T_k$ be defined as in (5.5.32), $k \geq 3$, $T = \cup_{i=1}^{k} T_i$.
If there exists $m := \frac{k+1}{2}$ triples $(u_i, v_i, w_i), i = 1, ..., m$, such that

$$\begin{array}{ll} \{u_i, w_i\} \in H \cap T & \\ v_i \in \bar{H} \setminus T & \\ u_i \prec v_i \prec w_i & \forall\ i = 1, ..., m \\ v_i \neq v_j & \forall\ i, j = 1, ..., m, i \neq j \\ w_i = u_{i+1} & \forall\ i = 1, ..., m-1, \end{array}$$

then the inequality

$$x(A(H)) + \sum_{i=1}^{k} x(A(T_i)) \leq |H| + \frac{k-1}{2} - 1$$

is valid with respect to $SOP(n, P)$.

Proof. Let F be a feasible Hamiltonian path violating the above inequality. As the 2–matching constraint (5.5.32) is valid, the 2–matching constraint has to be tight for F.

To simplify notation, set $x(T) := \sum_{i=1}^{k} x(A(T_i))$. As $x(A(H)) \leq |H| - \lfloor \frac{x(T)}{2} \rfloor$ we can conclude that $k - 1 \leq x(T) \leq k$. Thus it suffices to consider the following two cases

$$\begin{array}{lll} x(T) = k & \text{and} & x(A(H)) = |H| - \frac{k+1}{2} \\ x(T) = k-1 & \text{and} & x(A(H)) = |H| - \frac{k-1}{2} \end{array}$$

and show that this violates a given precedence relationship $u_i \prec v_i \prec w_i$ for some $i = 1, ..., m$.

Case 1 : Let $x(T) = k$, i.e., all teeth T_i are used by F. In order to achieve $x(A(H)) = |H| - \frac{k+1}{2}$ $\frac{k-1}{2}$ pairs of teeth have to be connected through H, i.e., there are direct paths from s_i to s_j using only nodes in H. Finally, the last tooth is visited and all nodes of H not yet visited. But as we have $\frac{k+1}{2}$ precedence triples (u_i, v_i, w_i) at least one of them is violated.

Case 2 : Let $x(T) = k - 1$. In order to achieve $x(A(H)) = |H| - \frac{k-1}{2}$ the $k - 1$ teeth have to be connected through nodes in H. As $\frac{k+1}{2}$ precedence triples (u_i, v_i, w_i) are given this leads once again to a violation.

□

Note that the precedences have to intersect the teeth, i.e., either the teeth node t_i or the handle node s_i have to be involved. In contrast to the SEC no strengthening is possible if just a precedence relationship $i \prec j, \{i, j\} \in H \setminus T$ is given. This is due to the fact that a SEC $x(A(W)) \leq |W| - 1$ is only tight, if the node set W is entered just once. In contrast the 2–matching constraint is tight if the node set $H \cup T$ is entered twice.
An open question is, if it is possible to combine the strengthenings in case that a mixture of the different precedence structures of types in Theorems (5.5.33)(a)(b) and (5.5.34)(a)–(d) occurs. It is not possible to strengthen the "SEC" of the handle of the 2–matching constraint as it was done in the case of some of the inequalities presented in Section 5.4 (e.g., as π–, σ–, or (π,σ)–inequalities). Counterexamples are known for many of these situations.

5.6 Separation procedures

To apply linear programming techniques we need a description of the sequential ordering polytope $SOP(n, P)$ by means of equations and inequalities. The classes of inequalities presented in the last sections are part of that description. As most of theses classes are of exponential size, we cannot consider all of them in the LP we intend to solve. Given an optimal solution $\bar{x}$ of the current linear program we have to decide, if $\bar{x}$ satisfies all inequalities $ax \leq a_0$ that are known to be valid for $SOP(n, P)$, and if not to identify such a violated inequality. This problem is addressed as the **separation problem**.

The separation problems for the **subtour elimination constraints** and **2–matching constraints** are solved by procedures developed for the TSP. This is motivated by the fact that each TSP–inequality $by \leq b_0$ might be transformed into a symmetric ATSP–inequality $ax \leq a_0$ by substituting y_{ij} with $x_{ij} + x_{ji}$ and setting $a_0 = b_0, a_{ij} = a_{ji} = b_{ij}$, and vice versa. Due to that fact, every separation algorithm for TSP–inequalities can be used as a "black box" for the ATSP as well.

Suppose we are given a point $\bar{x}$. First set

$$\bar{y}_e := \bar{x}_{ij} + \bar{x}_{ji} \qquad \text{for all } e = ij \in E$$

and then apply the TSP separation algorithm to $\bar{y}$. If a violated inequality is detected, it is transformed into its ATSP counterpart. In the last years several exact and heuristic separation procedures for TSP–inequalities have been proposed. All of them can be used for the ATSP and for the SOP as well. For a description of these separation routines the reader is referred to the literature (e.g., [GP85a, JRR94]).

In this section we describe heuristic separation procedures that are used in the current implementation of the branch&cut algorithm. These are

- a heuristic procedure to identify strengthened T_k–inequalities,
- a heuristic procedure based on shrinking and clique lifting in order to identify violated pcb–inequalities and inequalities of type (5.4.26)(c)(d).

Furthermore, we describe an exact separation procedure for T_k–inequalities.

5.6.1 A heuristic separation procedure based on shrinking

Based on the lifting Theorem (5.4.25) it is possible to design a heuristic separation procedure for precedence cycle breaking inequalities and inequalities of type (5.4.26)(c)(d). This procedure was already outlined in [BFP92] and is roughly speaking the following. Shrink node sets $S_i \subset V$, s.t. the subtour elimination constraint $x(A(S_i)) \leq |S_i| - 1$ is fulfilled with equality. Within this "shrunk digraph" one checks for violated valid inequalities. These inequalities are then lifted to be valid for $SOP(n, P)$ by applying Theorem (5.4.25).

(5.6.36) Heuristic separation procedure.

Input :	$D = (V, A), P = (V, R)$ and a fractional LP–solution $\bar{x}$
Output :	violated pcb–inequality or violated inequality of type 5.4.26(c) and (d) or the answer that no such inequality is found

1. Initialize: Set $m = n, S_i = \{i\}\ \forall\ i \in V, \tilde{y}_{ij} = \bar{x}_{ij}, \tilde{D} = D, \tilde{P} = P$.
2. Find a set $S \subseteq \tilde{V}$, s.t. $|S| \geq 2$ and $\tilde{y}(S) = |S| - 1$.
   If found : shrink S to a single node, update $\tilde{D} = (\tilde{V}, \tilde{A})$, $\tilde{P} = (\tilde{V}, \tilde{R})$, m and y, else STOP
3. If $\tilde{R}$ contains a cycle, then a violated precedence cycle breaking inequality (5.4.22) is found.
4. If $\tilde{y}_{ij} > 0$, for some $(i,j) \in \tilde{A}$ s.t. $(i,j) \in \tilde{R}_1$ (resp. $(i,j) \in \tilde{R}_2$), then a violated inequality of type (5.4.26)(c) (resp. (5.4.26)(d)) is found.
5. Goto 2.

□

One of the central steps in this procedure is how to detect the saturated SEC in step 2. This can be computational expensive. Therefore, in a first step we shrink all arcs with value $\tilde{y}_{ij} = 1$. If no violated inequality is found, we shrink node sets of size 2, i.e., $\{i,j\} \in \tilde{V}$ with $\tilde{y}_{ij} + \tilde{y}_{ji} = 1$. These nodes are detected by enumeration.

5.6.2 Separation of T_k–inequalities

No polynomial time separation algorithm for T_k–inequalities has been published in the literature so far.

(5.6.37) Lemma.
The T_k–inequalities $x(A(W))+x_{pw}+x_{pq}+x_{wq} \leq k$ for the ATSP, as defined in Theorem (1.3.7), can be separated in polynomial time.

Proof. Let $\bar{x} \in \mathbb{R}^A$ be a given vector satisfying the trivial bounds and the SECs.
For every three distinct nodes $p, q, w \in V$ compute a cut $\delta^-(W)$ of minimum capacity $\bar{x}(\delta^-(W))$ separating w from p and q. Thus we compute $\binom{n}{3}$ minimum cuts, but observe that this procedure can be made more effective.
Set $\alpha := x_{pw} + x_{pq} + x_{wq}$. We claim that, if

$$\bar{x}(\delta^-(W)) < \alpha$$

then $\bar{x}$ violates the T_k–inequality $x(A(W)) + x_{pw} + x_{pq} + x_{wq} \leq k$. If $\bar{x}(\delta^-(W)) \geq \alpha$ holds for all possible choices $p, q, w \in V$, then all T_k–inequalities are satisfied.
Indeed,

$$\begin{array}{lll} & \bar{x}(\delta^-(W)) & < \alpha \\ \Leftrightarrow & -\bar{x}(\delta^-(W)) + \sum_{v \in W} \bar{x}(\delta^-(v)) + \bar{x}_{pw} + \bar{x}_{pq} + \bar{x}_{wq} & > |W| \\ \Leftrightarrow & \bar{x}(A(W)) + \bar{x}_{pw} + \bar{x}_{pq} + \bar{x}_{wq} & > |W| \end{array}$$

□

Note, that the procedure outlined above can be made more effective, as the minimum capacity cut has to be calculated just for choices $p, q, w \in V$ such that

$$1 < \bar{x}_{pw} + \bar{x}_{pq} + \bar{x}_{wq} < 2.$$

A disadvantage of the procedure is that it does not take possible precedence structures into account. Therefore, we suggest to use the following straightforward heuristic separation procedure to detect such violated inequalities. Roughly speaking the idea is that one tries to "guess" nodes p, w, q and a node set W that are likely to violate a (strengthened) T_k–inequality.

(5.6.38) Heuristic separation procedure for strengthened T_k–inequalities.

Input : $D = (V, A), P = (V, R)$ and a fractional LP–solution $\bar{x}, \alpha, \beta, \epsilon$
Output : violated T_k–inequality or
the answer that no such inequality is found

1. Determine node sets S_i, such that $|S_i| \geq 2$ and $\bar{x}(S_i) = |S_i| - 1$.
   Let $\mathcal{S} := \{S_1, ..., S_m\}$ be the set of all such node sets.
2. Determine node triples (p, w, q) such that
   $p \neq w \neq q \neq p$ and $\alpha \leq x_{pw} + x_{pq} + x_{wq} \leq \beta$.
   Let $\mathcal{P} := \{(p_1, q_1, w_1), ..., (p_l, q_l, w_l)\}$ be the set of all such triples.
3. for all (p, w, q) in $\mathcal{P}$ do :
   (a) Determine all $v \in V \setminus \{p, q\}$ such that $\bar{x}_{wv} > \epsilon$ or $\bar{x}_{vw} > \epsilon$.
   (b) For all such v determine the largest set $W \in \mathcal{S}$ such that $v \in W$.
   (c) Check, if (p, w, q) and W violate an inequality of type (5.5.29)(a)(b), (5.5.30)(a)–(c), (5.5.31)(a)(b).

□

In the first step node sets S with $|S| \geq 2$ and $x(A(S)) = |S| - 1$ are determined. This is done by enumerating nodes $i, j \in V$ with $x_{ij} + x_{ji} = 1$, shrinking these nodes to a single node and iteratively applying this enumeration procedure to the resulting shrunken digraph. Each shrinking that was performed is stored in a shrinking digraph $D_s = (V, A_s)$, where an arc $(i, j) \in A_s$ occurs whenever clusters i and j were shrunken to a new cluster j. The iteration in which the arc is generated is stored as the arc weight of (i, j). Note that the outdegree of each node is at most 1, whereas more than one arc can enter a node. For a given node $i \in V$ node sets S_k containing i and fulfilling $x(A(S_k)) = |S_k| - 1$ can easily be detected.

In step 2 promising node triples for the T_k inequality are enumerated. In order to keep computing time moderate we only consider nodes p, w, q, such that $x_{pw} + x_{pq} + x_{wq}$ is between some limiting values. Computational tests showed that it is most likely to find (within a reasonable computing time) violated T_k–inequalities if $1.33 \leq x_{pw} + x_{pq} + x_{wq} \leq 1.75$. Furthermore, we generate just n such triples.

Finally, it is checked, if a combination of the triples (p, w, q) and S_k violates a strengthened T_k–inequality.

5.7 A Branch&Cut Algorithm

The implementation of a branch&cut algorithm from scratch typically leads to a huge overhead for maintaining the list of subproblems, choosing and implementing branching strategies, regenerating the linear programs while switching to another node in the branch&cut tree, etc. Jünger, Reinelt, and Thienel developed and implemented the general branch&cut framework ABACUS, taking over all the maintenance of the branch&cut tree. The user "just" has to implement problem specific routines, such as heuristics, separation procedures, etc.

In this section we discuss the used strategies and the parameter setting within this framework. This will be accomplished by giving computational results for seven hard problem instances we consider to be representative of the problem instances we had available (*ESC47, ESC78, rbg048a, rbg050c, ft53.4, ft70.1, prob.5*). A description of these instances can be found in Section 5.8.

Initial calculation of an upper bounds

Feasible solutions to the SOP instances are calculated at two points of the algorithm. In the initialization step of the branch&cut algorithm an **initial heuristic** is called. Furthermore, the information contained in the solutions of the linear programs is exploited by constructing feasible Hamiltonian paths from the LP–solution obtained after each iteration of the branch&cut algorithm.

The computational experience showed that it does not make sense to spend much computing time for the calculation of the initial solution.

The heuristics we had available were originally designed to be used for the ATSP. They have been modified to take precedences into account, i.e., feasibility checks were implemented and a sequence is simply rejected, if a precedence relationship is violated. In the exchange heuristics nonfeasible "intermediate" sequences are not allowed. The following heuristics have been tested:

topsort() :	The topological ordering $(1, 2, ..., n)$ of the nodes is generated. Nodes are exchanged until the sequence is feasible.
farins() :	Farthest insertion heuristic.
listins() :	List insertion heuristic: The nodes to be inserted best possible are taken in the order $1, 2, ..., n$.
randins() :	Random insertion heuristic: The nodes to be inserted best possible are chosen randomly.
bestins() :	Best insertion heuristic: Choose always the best possible insertion
2opt() :	exchange heuristic on two nodes
3opt() :	exchange heuristic on three nodes
locenu(m) :	local enumeration on sequences of length m

The quality of the heuristic solutions in relation to the computing time needed to construct them is not very satisfactory, as it is indicated in the following Table 5.1.

	ESC47		prob.5		ft70.1		rbg174a	
n	49		42		71		176	
$\|R\|$	10		10		17		1113	
optimal	1288		243		39313		2033	
topsort	3843	0:00.02	477	0:00.02	46060	0:00.05	2478	0:03.42
farins	4149	0:00.18	550	0:00.10	44663	0:00.50	2222	0:02.28
listins	3463	0:00.00	481	0:00.02	46968	0:00.02	2226	0:00.05
randins	4889	0:00.02	721	0:00.00	48193	0:00.02	2200	0:00.05
bestins	4149	0:00.17	550	0:00.10	44663	0:00.50	2222	0:02.28
topsort +2opt	3378	0:00.07	430	0:00.07	45014	0:00.27	2235	0:09.75
farins +2opt	4149	0:00.20	545	0:00.13	44293	0:00.65	2186	0:04.83
listins +2opt	3419	0:00.05	481	0:00.03	45102	0:00.17	2182	0:03.88
randins +2opt	4804	0:00.07	574	0:00.07	45136	0:00.23	2162	0:02.58
bestins +2opt	4149	0:00.18	545	0:00.13	44293	0:00.62	2186	0:04.83
topsort +3opt	2665	0:00.62	310	0:00.17	43058	0:02.22	2204	0:11.97
farins +3opt	2863	0:00.80	328	0:00.32	42911	0:01.97	2172	0:05.72
listins +3opt	2247	0:00.57	346	0:00.17	43441	0:01.43	2122	0:07.22
randins +3opt	2958	0:00.58	385	0:00.20	42462	0:02.88	2144	0:05.20
bestins +3opt	2863	0:00.77	328	0:00.32	42911	0:01.93	2172	0:05.70
topsort +2opt +locenu(8)	2948	0:00.90	381	0:00.28	44044	0:17.60	2055	5:05.05
farins +2opt +locenu(8)	2523	0:00.93	382	0:01.23	43220	0:03.87	2073	4:07.42
listins +2opt +locenu(8)	3127	0:00.30	429	0:00.67	44189	0:04.40	2089	2:43.08
randins +2opt +locenu(8)	3701	0:00.42	436	0:00.60	44059	0:08.17	2081	2:49.35
bestins +2opt +locenu(8)	2523	0:00.92	382	0:01.23	43220	0:03.85	2073	4:07.52
topsort +2opt +locenu(11)	2626	3:45.17	332	0:02.82	43713	33:26.75	2057	1038:16.07
farins +2opt +locenu(11)	2552	5:37.88	339	4:10.98	42829	4:05.83	2036	633:40.12
listins +2opt +locenu(11)	3038	1:24.07	345	5:23.32	43327	11:46.67	2075	490:08.85
randins +2opt +locenu(11)	3463	9:34.80	353	2:17.75	43750	6:35.70	2039	592:48.83
bestins +2opt +locenu(11)	2552	5:37.90	339	4:10.90	42829	4:05.80	2036	633:48.92
topsort +3opt +locenu(8)	2592	0:00.83	310	0:00.22	42913	0:05.35	2122	4:19.18
farins +3opt +locenu(8)	2863	0:00.93	328	0:00.43	42468	0:05.05	2059	3:18.80
listins +3opt +locenu(8)	2247	0:00.67	346	0:00.22	43251	0:05.82	2095	1:46.10
randins +3opt +locenu(8)	2958	0:00.70	364	0:00.33	42288	0:05.03	2076	3:42.75
bestins +3opt +locenu(8)	2863	0:00.93	328	0:00.43	42468	0:05.02	2059	3:19.00
topsort +3opt +locenu(11)	2293	4:04.30	310	0:19.07	42622	7:58.68	2082	1017:31.07
farins +3opt +locenu(11)	2293	2:31.18	320	1:29.85	41875	1:05.93	2039	473:38.47
listins +3opt +locenu(11)	2076	2:21.50	334	2:21.32	43063	9:10.80	2083	326:11.52
randins +3opt +locenu(11)	2580	3:14.80	364	1:00.92	42279	1:16.82	2046	590:38.40
bestins +3opt +locenu(11)	2293	2:31.08	320	1:29.78	41875	1:05.92	2039	472:35.45

Table 5.1. Results of SOP heuristics

We have chosen four instances (*ESC47, prob.5, ft70.1, rbg174a*) for which we know the optimal solution value and have run several combinations of the heuristics. Observe that the results can be rather poor. For example, for problem instance *ESC47* the best solution found after more than two minutes of computing time has a value of 2076 whereas the optimal solution value is 1288. The results for the other instances are in an analogous way not really convincing, also compared to the computing time needed to solve these instances to optimality (cmp. Section 5.8).

In the branch&cut code we are not working on the full variable set A, but only on a subset A' (see also *"Initialization"* in this section). As we want to assure that this subset contains a feasible solution, we run a heuristic in order to obtain after "short" computing time a "relatively good" feasible solution. Variables corresponding to this solution are added to A'.

For our implementation we decided to use the list insertion heuristic with an additional call of the 3–opt heuristic. After a few iterations of the cutting plane algorithm better solutions will be constructed "on the fly" by the LP–exploitation heuristic (see *"LP–exploitation heuristic"* in this section).

Initialization

We start the branch&cut algorithm solving the **initial LP**

$$\begin{array}{lllll} & & \min c^T x & & \\ \text{s. t.} & (1) & x(\delta^-(i)) & = 1 & \forall\, i \in V \setminus \{1\} \\ & (2) & x(\delta^+(i)) & = 1 & \forall\, i \in V \setminus \{n\} \\ & (3) & x_{ij} & \geq 0 & \forall\, (i,j) \in A \\ & (4) & x_{ij} & \leq 1 & \forall\, (i,j) \in A \end{array}$$

As only a small **subset of the variables** will be in an optimal solution, we are not working on the whole set of variables, but choose a promising subset of all variables, from which we think that the optimal solution is among them. We have chosen the variables of the k–nearest neighbour digraph, which we also denote as **sparse digraph** $D_s(V, A_s)$. We run an **initial heuristic** to obtain a feasible solution. The main purpose of this heuristic is to make the sparse digraph Hamiltonian. Thus the arcs of that path are added to the sparse digraph. Only the variables corresponding to the arcs of the sparse digraph are considered in the initial LP. The value of the global upper bound is initialized with the value of this feasible Hamiltonian path.

This keeps the size of the linear program "moderate" but makes it necessary to price out nonactive variables, that are not considered in the actual LP. This **pricing** is performed in a hierarchical order. First the variables in a so–called **reserve digraph** $D_r(V, A_r)$ are considered, before performing a pricing on the complete set of variables. We have chosen D_r to be the m–nearest neighbour digraph for $m > k$. Arcs already contained in the A_s are not considered in A_r. If the reserve–graph–pricing was successful, the complete pricing is omitted (see also *"Variable pricing"* in this section).

The computational experiments showed that in the current implementation the choice of m and k only has a minor influence on the overall computing time. We tested values for $k \in \{2, 3, 4, 5, 6, 7\}$ and $m \in \{5, 6, 7, 8, 9, 10\}$ leading to more or less the same overall computing times. We decided to choose $k = 4$ and $m = 7$ as default values.

Separation strategy

The separation phase is the "core" of the branch&cut algorithm. Here violated valid inequalities are generated that are added as cutting planes to the actual linear program.

Each generated cut is additionally stored in a **pool of inequalities** that is, e.g., used to regenerate the linear programs from scratch. Since not all pool inequalities are active, i.e., are considered in the actual LP, this pool can also be used to check, if any of the cuts generated in an earlier iteration of the algorithm is violated by the actual LP–solution. This can be done by "checking" all nonactive pool–inequalities. As the size of the pool might be very large, this can be very time consuming. Therefore, we do not separate pool–inequalities for which "fast" separation routines exist.
We decided to use the following separation strategy:

(5.7.39) Separation procedure.

Input : Incumbent LP–solution $\bar{x}$.
Output : An inequality $ax \leq a_0$ violated by $\bar{x}$, i.e., $a\bar{x} > a_0$.

1. Call an exact procedure for the separation of subtour elimination constraints.

2. Check the pool for violated π–, σ–, (π,σ)–inequalities, pcb–inequalities, strengthened 2–matching constraints and strengthened T_k–inequalities.
3. Call the shrinking separation procedure (5.6.36).
4. Call the exact procedure for weak π–inequalities (see 5.4.12)
5. Call the exact procedure for weak (π,σ)–inequalities (see 5.4.20)
6. Call the exact procedure for weak σ–inequalities (see 5.4.16)
7. Call a heuristic procedure for 2–matching constraints (see e.g. [PR90]),
8. Call a heuristic procedure for T_k–inequalities.

□

If one of the steps is successful, i.e., a violated inequality is found, no other separation procedure is called. Whenever in step 1 a SEC is found it is checked, if it can be strengthened to a π–, σ, (π,σ), or pcb–inequality. If this is possible, the strengthened inequality is added, otherwise the SEC. If a violated 2–matching constraint is detected, it is checked, if it can be lifted to any of the constraints presented in Section 5.5.3.

LP–exploitation heuristic

As we do not have a complete description of the sequential ordering polytope, it is very unlikely that the branch&cut algorithm will result in an integer optimal solution, i.e., the incidence vector of a feasible Hamiltonian path. But the fractional LP–solutions contain some information about the structure of an optimal solution. We exploit this information by the following LP–exploitation heuristic.

(5.7.40) Exploitation algorithm for fractional LP–solutions.

Input : LP–solution $\bar{x}$,
$2optfreq$ (frequency of 2–opt calls),
$3optfreq$ (frequency of 3–opt calls),
$locenufreq$ (frequency of local enumeration calls),
$lwidth$ (width of local enumeration),
M (big value).
Output : feasible Hamiltonian path H.

1. If the LP–solution is a feasible Hamiltonian path, $H = \bar{x}$, GOTO 5
2. Construct a temporary cost matrix C' by applying one of the following strategies:
   $c'_{ij} = (1.0 - \bar{x}^2_{ij}) \cdot M$;
   $c'_{ij} = (1.0 - \bar{x}_{ij}) \cdot c_{ij}$;
   $c'_{ij} = (1.0 - \bar{x}_{ij}) \cdot M$;
3. Call one of the following heuristics with the cost matrix C':
   $H = \text{topsort}(C')$;
   $H = \text{listins}(C')$;
   $H = \text{farins}(C')$;
   $H = \text{bestins}(C')$;
   $H = \text{randins}(C')$;
4. Try to improve the H :
   Every $2optfreq$ times call: $H = 2\text{opt}(H, C)$;

Every $3optfreq$ times call: $H = 3\text{opt}(H, C)$;
Every $locenufreq$ times call: $H = \text{locenu}(H, C, lwidth)$;

5. RETURN(H)

□

Roughly speaking, this heuristic is the following. Given a fractional LP–solution $\bar{x}$, set up a temporary cost matrix C', which is obtained by combining the original cost matrix C and the LP–solution in a proper way. Call a heuristic with the cost matrix C' and then call an improvement heuristic with the obtained solution and the original cost matrix C. If the branch&cut algorithm was running for some iterations, the LP–solution does not change dramatically from one iteration to the next. In order to avoid that the same solutions are constructed several times, it is necessary to vary in the construction procedures for C' and in the heuristics.

The results obtained by this heuristic outperform the ones obtained by the initial heuristic. The procedure is rather sensitive to the setting of the parameters. By decreasing the values of *2optfreq*, *3optfreq*, and *locenufreq*, and by increasing the value of *lwidth* it is certainly possible to construct better feasible Hamiltonian paths. But while doing so computing time at each iteration will increase dramatically. Table 5.2 summarizes the computational times for different settings of these parameters. See also Table 5.1 for the influence of *lwidth* on the computing time. The entries in the first column of Table 5.2 have to be read like (*2optfreq, 3optfreq, locenufreq, lwidth*).

	ESC47	ESC78	rbg048a	rbg050c	ft53.4	ft70.1	prob.5
(1,4,41,8)	10:47.55	874:38.87	2:13.78	3:05.65	113:01.22	25:45.98	27:54.07
(1,4,41,9)	11:13.82	–*	2:57.03	5:54.38	41:13.05	27:19.70	29:52.23
(1,7,100,8)	9:34.47	52:14.15	2:33.85	1:29.20	15:20.77	29:24.43	21:03.53
(1,7,100,9)	9:42.67	65:26.67	3:01.95	2:16.62	17:10.27	29:54.87	21:52.78
(3,10,201,8)	8:38.18	372:17.65	3:09.30	2:10.18	11:52.52	34:37.07	24:43.30
(3,10,201,9)	8:40.13	–*	3:23.48	2:42.15	15:41.93	36:35.83	25:33.53

–* : time limit of 15 hours CPU exceeded

Table 5.2. Comparison of different values for LP–exploitation heuristic

Comparing the results for ESC78 shows that the setting of these parameters has a dramatical influence on the overall computing time. In this problem instance the lower bound at the root equals the value of the optimal solution and the computing time is mainly spent in finding a feasible solution with this value. The following settings tend to give the best results with respect to computing time:

2optfreq	$= 1$	*3optfreq*	$= 7$
locenufreq	$= 100$	*lwidth*	$= 8$

They are accepted as default values for the branch&cut algorithm.

Enumeration strategy

In order to keep the branch&cut tree small the way in which the subproblems in the tree are processed is very important for an efficient implementation. The branch&cut framework ABACUS provides three possible strategies: depth–first–search (DFS), breadth–first–search

(BRFS), and best–first–search (BEFS). The computational results of the benchmark problems show that DFS is not a good strategy, as the "risk" of spending too much time in a branch of the tree that is useless for the computation of the bounds is too high. BRFS slightly outperforms BEFS (see Table 5.3) as it tends to give more "stable" results. Thus, we decided to use BRFS as a default strategy.

	ESC25	ESC47	ESC78	rbg048a	rbg050c	ft53.4	ft70.1	prob.5
BEFS	0:08.10	8:03.35	494:44.83	1:33.72	2:20.52	20:39.38	45:39.92	24:57.60
BRFS	0:09.33	10:12.48	67:44.30	3:08.82	2:20.10	17:52.43	30:42.08	22:49.78
DFS	0:10.45	8:34.43	50:58.38	1:43.27	0:36.10	34:30.43	602:53.32	16:14.62

Table 5.3. Comparison of enumeration strategies

Variable pricing

As we are working only on a subset of all variables, it is necessary to price out all variables that are not considered in the incumbent LP. The purpose is to check, if the LP solution is valid for the complete graph, i.e., if all nonactive variables "price out" correctly.

As this might be very time consuming, it is done in a hierarchical way. Before considering the whole set of variables, we first check the variables corresponding to the arcs of the reserve digraph (see *"Initialization"* for the choice of the size of the sparse and reserve digraph).

This pricing step has to be done before branching, if an infeasibility was detected, and if a tailing–off phenomena is observed. Furthermore, the pricing step is performed every pricing_freq iterations. This assures that variables needed in an optimal solution enter the LP as early as possible. Table 5.4 gives the computing times for different settings of pricing_freq. We decided to accept pricing_freq = 10 as default value.

	ESC25	ESC47	ESC78	rbg048a	rbg050c	ft53.4	ft70.1	prob.5
5	0:10.37	13:12.17	91:03.67	6:10.63	0:48.63	37:09.98	50:17.08	33:20.60
10	0:09.12	10:03.78	68:04.43	3:12.97	2:18.18	17:33.75	31:47.67	23:29.25
15	0:10.10	10:23.15	71:12.22	1:59.03	2:16.53	19:01.28	39:50.37	27:56.45
20	0:09.08	8:47.30	66:34.57	2:18.62	2:29.48	18:51.75	78:28.60	23:48.33

Table 5.4. Comparison of different pricing frequencies

Branching rules

At the branching step a fractional variable is chosen and two new subproblems are generated by setting the value of the variable to 0, resp. to 1. ABACUS supports the strategy to choose the variable closest to 0.5 with highest cost coefficient, which we accept as default strategy.

Tailing off

If during the last k iterations of the cutting plane phase no significant improvement (of at least $p\%$) was achieved, a pricing step is performed. If this was not successful, i.e., no variables were added, we leave the cutting plane phase and branch. We use the default values of ABACUS, namely $k = 5$ and $p = 2.5 \cdot 10^{-5}$.

5.8 Computational results

The branch&cut code is tested with four classes of problem instances that contain real–life data, randomly generated data, and a combination of both. The instances *ESC..* have been supplied by L. Escudero. In [AEGS90] computational results of a cutting plane approach have been already reported for these problem instances. The problem instances *rbg..* are derived from the stacker crane application as described in Section 4.6. The random instances *prob..* have been supplied by M. Jünger and G. Reinelt. TSPLIB contains six hard ATSP instances for which we randomly generate precedence digraphs $P = (V, R)$ with a varying number of precedences.

For the solution of the linear programs the LP–solver CPLEX2.2 was used. All computational results were obtained on a SUN SPARC IPX with 64 MB main memory under SUN OS 4.1.2. For compilation the GNU C compiler with optimization option O2 was used. For all runs we allowed a maximum CPU–time of $5 \cdot \lceil \frac{n}{100} \rceil$ hours.

Key to the following tables:

problem	:	name of the problem instance.
n	:	number of nodes.
$\|R\|$	:	number of precedence relationships (without transitively derived precedences).
Opt.	:	value of an optimal solution. If this value is not known, the best lower lb and upper bound ub is given in the form $[lb, ub]$.
Heur.	:	value of the solution obtained by the initial heuristic.
BC-root	:	Quality of the solution at the root LP, namely
...bounds	:	lower and upper bound at the root LP (if the problem instance was solved to optimality at the root it is marked with "–").
...GAP	:	optimality gap at the root LP. Let lb_{root} denote the lower before branching. Then the optimality gap at the root is defined as $\frac{opt - lb_{root}}{lb_{root}} \cdot 100$.
BC–tree ...	:	information on the branch&cut tree, namely
... (# N)	:	number of generated nodes in the branch&cut tree (except the root node),
... (level)	:	depth of the branch&cut tree (the level of the root is 0).
#cuts	:	number of generated cutting planes.
#LPs	:	number of linear programs that had to be solved.
CPU	:	CPU time needed to solve the problem. If the problem instance could not be solved to optimality within a certain time limit, this is marked by giving the CPU–time after which the run was stopped. (All computing times for SUN SPARC IPX).

Results on real–life data

In this subsection we summarize the computational results on different sets of real–life data that were all derived from a manufacturing environment. The instances *ESC..* are production data from IBM, the instances *rbg..* have been derived from the stacker crane application as described in Section 4.6. Table 5.5 summarizes the results obtained for small to medium sized real-life instances varying from 9 – 100 nodes. Table 5.6 gives results for large sized real-life instances on more than 100 nodes.

Problem	n	$\|R\|$	Opt.	Heur.	BC–root bounds	GAP	BC–tree #N, Level	# cuts	#LPs	CPU
ESC07	9	6	2125	2125	–	–	0, 0	4	3	0:00.10
ESC11	13	3	2075	2075	–	–	0, 0	12	6	0:00.23
ESC12	14	7	1675	1728	–	–	0, 0	24	12	0:00.52
ESC14	16	12	2125	2125	–	–	0, 0	19	8	0:00.38
ESC25	27	9	1681	2692	[1678, 1684]	0.18	2, 1	67	41	0:04.33
ESC47	49	10	1288	2247	[1247, 1500]	3.29	90, 7	2620	1252	11:48.05
ESC63	65	95	62	64	[62, 64]	0.00	38, 5	2285	120	4:33.22
ESC78	80	77	18230	18475	[18230,18365]	0.00	66, 6	2209	246	19:10.20
ESC98	100	84	2125	2125	–	–	0, 0	35	12	0:29.67
prob.faw1	56	45	210	226	[210, 226]	0.00	6, 2	53	21	0:16.12
rbg016a	18	26	167	172	–	–	0, 0	3	2	0:00.18
rbg016b	18	30	132	145	–	–	0, 0	78	26	0:01.37
rbg017a	17	38	129	135	–	–	0, 0	15	13	0:00.48
rbg019a	21	43	198	203	[197, 199]	0.51	2, 1	13	8	0:00.48
rbg019b	21	57	199	225	–	–	0, 0	4	2	0:00.13
rbg021a	21	68	158	164	–	–	0, 0	31	9	0:00.45
rbg023a	23	79	155	161	–	–	0, 0	19	9	0:00.52
rbg048a	50	192	351	408	[349, 357]	0.57	94,17	2693	315	8:55.53
rbg049a	51	241	355	415	[355, 377]	0.00	14, 3	211	39	0:18.57
rbg050a	52	225	400	455	–	–	0, 0	316	40	0:15.02
rbg050b	52	258	397	418	[397, 405]	0.00	2, 1	150	26	0:09.73
rbg050c	52	256	467	511	[467, 478]	0.00	14, 3	860	106	1:37.37
rbg068a	68	249	609	633	–	–	0, 0	30	9	0:03.92

Table 5.5. Results on small and medium sized real–life instances

All small and medium sized problem instances could be solved to optimality in the given time limit of 5 hours. Half of the instances could be solved to optimality at the root of the branch&cut tree. For all other instances the lower bounds on the root are close to the optimal solution value. They either equal the optimal solution value or lie in a 3% range (see column *GAP*). As outlined in the last section where the heuristics for the SOP are described, they sometimes fail to quickly find good or even optimal feasible solutions. For some problem instances (e.g., *ESC47*), the upper bound on the root is far away from the optimal solution value. Considering instance *ESC78* one recongnizes that the lower bound on the root already equals the value of the optimal solution and that "most" of the computing time is spent to find a feasible solution of that value.

Most of the large scale instances of more than 100 nodes could not be solved to optimality. Due to the computational experience with the small to medium sized instances, we conjecture that also for these instances the lower bound is close to the optimal solution value and that the upper bound is the bound that has to improved.

Problem	n	$\|R\|$	Opt.	Heur.	BC–root bounds	BC–root GAP	BC–tree #N, Level	# cuts	#LPs	CPU
rbg109a	111	622	1038	1134	[1027, 1085]	1.07	752,18	4961	2114	189:51.18
rbg150a	152	952	[1748, 1750]	1807	[1747, 1753]	?	816,25	11290	3112	–*
rbg174a	176	1113	2033	2122	–	–	0, 0	469	57	10:32.02
rbg253a	255	1721	[2943, 2987]	3113	[2943, 2990]	?	218,21	5135	1044	–*
rbg323a	325	2412	[3136, 3221]	3388	[3135, 3263]	?	78, 7	5169	670	–*
rbg341a	343	2542	[2543, 2854]	2963	[2541, 2854]	?	54, 6	4548	594	–*
rbg358a	360	3239	[2518, 2758]	2935	[2517, 2758]	?	14, 3	5613	443	–*
rbg378a	380	3069	[2761, 3142]	3250	[2761, 3142]	?	20, 5	4148	370	–*

–* : time limit exceeded

Table 5.6. Results on large scale real–life instances

In the appendix the interested reader will find further statistics on the branch&cut code, such as generated cuts, distribution of the computing time, etc.

Randomly generated problem instances

Problem	n	$\|R\|$	Opt.	Heur.	BC–root bounds	BC–root GAP	BC–tree #N, Level	# cuts	#LPs	CPU
prob.1	11	6	38	38	–	–	0, 0	6	3	0:00.08
prob.2	11	6	287	287	–	–	0, 0	2	2	0:00.10
prob.3	22	4	218	301	[216, 230]	0.93	6, 3	39	21	0:01.77
prob.5	42	10	243	346	[236, 245]	2.88	480,17	9138	3058	17:50.23
prob.6	82	5	614	915	–	–	0, 0	94	21	0:37.27
prob.7	100	41	[1026,1542]	2531	[1016,1796]	?	638,15	7144	3383	–*

–* : time limit exceeded

Table 5.7. Results on randomly generated problem instances

The results on the randomly generated data are more or less the same as for the real–life instances. Half of them were solved at the root and for two of them the lower bounds on the root are very good. One of the instances (*prob.7*) is still unsolved. We conjecture that also here the problem mainly is in finding a good upper bound. Observe that the feasible solution found after 5 hours of computing time is 1542, whereas the value of the best feasible solution known so far is 1385. We conjecture that also this solution is far away from the optimum.

Note that *prob.5* and *prob.7* are random data in the sense that the precedence structure was generated randomly. The surprising fact is that the branch&cut code has more problems with randomly generated precedence graphs than with precedence graphs that typically occur in the practical applications we considered.

Pure ATSP instances

Note that the Hamiltonian path problem is a special case of the SOP where the set of precedence relationships is empty. As the ATSP is an equivalent problem to the Hamiltonian path problem, the branch&cut code for the SOP can be used to solve ATSP instances to optimality.

TSPLIB [Rei91] contains six (hard) asymmetric instances varying from 17–100 nodes, that are used as benchmark problems for our branch&cut algorithm. One of these instances (*p43.atsp*) contains noninteger values. We create two new instances: one by multiplying each number by 2 (*p43x2*), and one by multiplying by 10 (*p43*). Furthermore, we ran ATSP instances that have been derived from the stacker crane application (problem instances *rbg*.0*).

The results are summarized in the following Tables 5.8 and 5.9. The results in Table 5.8 are obtained without the separation heuristic for T_k–inequalities, whereas the results in Table 5.9 document the computational results with the use of that heuristic. As all *rbg*–instances could be solved just with subtour elimination constraints (SEC), Table 5.9 does not contain these instances. More detailed statistics can be found in the appendix.

Problem	n	Opt.	Heur.	BC–root bounds	BC–root GAP	BC–tree #N, Level	# cuts	#LPs	CPU
br17.0	18	39	39	–	–	0, 0	9	8	0:00.38
p43	44	28100	28135	[28055,28135]	0.16	194,25	172	353	2:07.62
p43x2	44	5620	5627	[5611, 5621]	0.16	440,34	267	642	3:50.53
ry48p	49	14422	15091	[14324,14444]	0.68	28, 7	150	95	0:40.12
ft53.0	54	6905	7481	–	–	0, 0	30	22	0:09.55
ft70.0	71	38673	39342	[38653,38673]	0.05	6, 3	22	24	0:29.05
kro124p	101	36230	38776	[36106,36353]	0.34	28, 5	178	125	6:27.63
rbg323.0	324	1326	1650	[1326,1355]	0.00	2, 1	30	28	33:52.75
rbg341.0	342	1116	1485	–	–	0, 0	18	20	27:56.38
rbg358.0	359	1163	1521	[1163,1164]	0.00	2, 1	16	21	35:04.35
rbg378.0	379	1633	1988	[1633,1635]	0.00	2, 1	15	21	37:29.18
rbg399.0	400	2048	2269	[2048,2058]	0.00	2, 1	22	22	43:07.30
rbg403.0	404	2465	2646	[2465,2491]	0.00	6, 2	15	16	37:41.82
rbg416.0	417	2126	2320	–	–	0, 0	22	28	72:30.37
rbg423.0	424	2065	2318	–	–	0, 0	10	10	35:51.23
rbg443.0	444	2720	2873	[2720,2725]	0.00	6, 2	19	22	62:00.30

Table 5.8. Solution of asymmetric TSP instances

Problem	n	Opt.	Heur.	BC–root bounds	BC–root GAP	BC–tree #N, Level	# cuts	#LPs	CPU
br17	18	39	39	–	–	0, 0	9	8	0:00.42
p43	44	28100	28135	[28055,28105]	0.16	142,17	202	309	2:50.77
p43x2	44	5620	5627	[5611, 5621]	0.16	90,21	202	234	1:39.50
ry48p	49	14422	15091	[14326,14444]	0.67	14, 5	168	87	0:40.80
ft53	54	6905	7481	–	–	0, 0	28	24	0:11.82
ft70	71	38673	39342	[38653,38673]	0.05	6, 3	22	24	0:29.17
kro124p	101	36230	38776	[36177,36807]	0.15	20, 4	185	97	7:05.15

Table 5.9. Solution of asymmetric TSP instances (with T_k–inequalities)

Although the code was not originally designed to solve ATSP instances, the achieved results are comparable to those published in the literature (see [FT94]). All instances could be solved to optimality in a reasonable amount of computing time. It is interesting to observe that the stacker crane problems (*rbg**) are easily solvable. Even large scale problems up to 440 nodes could be solved after a few iteration of the branch&cut algorithm. It was necessary to generate just a few cutting planes, moreover, for all of these instances only subtour elimination constraints were generated. Maybe this is one explanation for the fact that the available real–life instances were easier to solve than some of the randomly generated ones.

The simple heuristic separation procedure for the T_k–inequalities does not really help to solve ATSP-instances. For problem instances *ry48p* and *kro124p* the lower bound at the root increased, in most instances fewer LPs had to be solved, and the branch&cut tree is of smaller size. But the overall computing times are more or less the same with and without this separation heuristic. The additional computing time is not spent in total in the separation routine but also in other parts of the code, such as LP–solver, pricing, etc. A fine–tuning of the parameters of the separation routine will improve the result for the ATSP–instances.

TSPLIB problem instances with random precedences

As the ATSP that are the basis for the real–life problems *rbg** are easy, the precedence constrained instances derived form the stacker crane application are perhaps not really hard benchmark problems for the branch&cut code. Therefore, we performed experiments to see what influence additional precedence constraints will have on hard ATSP instances. For that purpose we randomly generated precedence digraphs $P = (V, R)$ and added them to the TSPLIB problem instances. The results are summarized in Table 5.10.

The random precedence digraph $P = (V, R)$ are constructed in the following way: Initially set $R = \emptyset$. For each problem instance generate two lists l_1, l_2 of k random integers in the interval $[1, n]$. List l_1 corresponds to the tail list l_2 to the head of potential arcs $(i, j) \in R$. Starting from the first entry of the list add an arc $(l_1[i], l_2[i])$ to R, if $l_1[i] \neq l_2[i]$, $(l_1[i], l_2[i]) \notin R$, and $R \cup (l_1[i] \neq l_2[i])$ remains acyclic; otherwise skip this arc. We construct problem instances for $k = \frac{n}{4}, \frac{n}{2}, n, 2 \cdot n$. As some infeasible arcs have been generated, the actual number of added precedences is less than k. If *name* is the name of the TSPLIB-instance, then

name.0: pure ATSP–instance,
name.1: additional $k = \frac{n}{4}$ potential precedence relationships,
name.2: additional $k = \frac{n}{2}$ potential precedence relationships,
name.3: additional $k = n$ potential precedence relationships,
name.4: additional $k = 2 \cdot n$ potential precedence relationships.

Problem	n	$\|R\|$	Opt.	Heur.	BC–root bounds	GAP	BC–tree #N, Level	# cuts	#LPs	CPU
br17.0	18	0	39	39	–	–	0, 0	9	8	0:00.57
br17.1	18	4	41	41	[40, 41]	2.50	6, 3	29	33	0:03.65
br17.2	18	8	47	58	[44, 47]	6.82	8, 4	181	127	0:20.18
br17.3	18	14	49	54	[49, 52]	0.00	2, 1	96	44	0:06.72
br17.4	18	17	76	81	–	–	0, 0	31	11	0:00.55
p43.0	44	0	27950	28000	[27945,27950]	0.00	452	60	551	5:35.48
p43.1	44	9	27990	55260	[27989,28010]	0.00	18, 4	162	109	1:18.40
p43.2	44	20	[28175,28330]	55790	[28110,28410]	?	2134,52		7526	–*
p43.3	44	37	[28483,28670]	82370	[28242,28780]	?	1140,31	49246	13065	–*
p43.4	44	50	[69569,82960]	110620	[55972,82990]	?	1730,39	116681	24107	–*
ry48p.0	49	0	14422	15091	[14324,14444]	0.68	28,7		95	0:35.98
ry48p.1	49	11	[15220,15935]	16813	[14678,16181]	?	3406,24		14473	–*
ry48p.2	49	23	[15940,17022]	20736	[15287,17831]	?	1268,20	36673	16242	–*
ry48p.3	49	42	[18252,20334]	26874	[17740,21568]	?	742,22	47110	14821	–*
ry48p.4	49	58	[29967,31446]	35479	[28694,32246]	?	670,20	73704	18274	–*
ft53.0	54	0	690	7481	–	–	0,0		23	0:09.40
ft53.1	54	12	[7438, 7570]	10623	[7160, 8406]	?	4738,26		13031	–*
ft53.2	54	25	[7739, 8335]	11732	[7411,10314]	?	1928,23	34254	14028	–*
ft53.3	54	48	[9473,10682]	12587	[9138,11585]	?	504,18	44444	12883	–*
ft53.4	54	63	14425	15644	[13780,14628]	4.68	156,19	10814	2850	50:03.97
ft70.0	71	0	38673	39342	[38653,38673]	0.05	6, 3		25	0:23.30
ft70.1	71	17	39313	43441	[39153,40947]	0.41	682,16	4218	2408	88:00.63
ft70.2	71	35	[39862,41778]	44193	[39541,43776]	?	1684,17	16520	8352	–*
ft70.3	71	68	[41482,44732]	48767	[40935,46714]	?	500,14	22280	7567	–*
ft70.4	71	86	[52269,53882]	57794	[50642,55846]	?	556,18	43915	8802	–*
kro124p.0	101	0	36230	38776	[36106,36998]	0.34	38,6		169	5:39.13
kro124p.1	101	25	[37916,42845]	54463	[37500,43244]	?	158,14	5349	2098	–*
kro124p.2	101	49	[38621,45848]	54173	[38210,47520]	?	114,10	6508	1934	–*
kro124p.3	101	97	[41478,55649]	73177	[40428,59006]	?	82,10	7326	1995	–*
kro124p.4	101	131	[64858,80753]	93501	[58631,85035]	?	134,12	18674	3332	–*

–* : time limit of 5 hours CPU–time exceeded

Table 5.10. TSPLIB instances with additional precedences

It was surprising to see that although the pure ATSP–instances could be solved to optimality in a reasonable amount of time this was not possible for most of these randomly generated problem instances. All runs were stopped after 5 hours of CPU–time and the optimality GAP (see column *Opt.*) was still that large that for most instances there is no hope that these would have been solved to optimality in another 5 hours of computing time.

Therefore, we recognized two reasons. First, note that the results of the initial heuristic are getting worse the more precedences are involved (e.g., p43.1–4). That is an indicator that the heuristics we are working with at the moment, do not handle the precedence structure properly. The results of the LP–exploitation heuristic are better but in most cases still far away from the value of the optimal solution.

Second, recall that the pure ATSP instances derived from the stacker crane application (*rbgxxx.0*) are easy. All of them can be solved just using the subtour elimination constraints (SEC). This conjectures that the precedence structure of the considered real–life instances is well captured by the strengthened versions of the SEC, namely by the π–, σ–, (π, σ)–, and pcb–inequalities.

In contrast, the TSPLIB instances are hard, i.e., more complex inequalities than the SEC are needed to solve them to optimality. These inequalities and their strengthened versions are not yet used in the implementation. For example, we used a 2–matching separation routine that was originally designed to be used for the unconstrained TSP. Whenever a violated inequality is found we strengthen according to the precedences. But very seldomly such a violated 2–matching constraint is detected, although a strengthened versions is violated. We believe that a direct separation routine for the strengthened inequalities will improve the computational results.

In a second experiment we have chosen the TSPLIB problem instance *br17.atsp* and iteratively added randomly generated precedence relationships. The relationship $i \prec j$ is added, if it has not been generated before, if it is not implied by a transitivity relation, and if the instance remains feasible. Instance *br17.x.i* is derived from *br17.x.(i-1)* by adding one additional precedence relationship. The computational results for these instances are summarized in Table 5.11.

All but three instances are easy to solve; two of the instances can be considered to be hard. Note that for the problem instances of this class the upper bound on the root always equals the optimal solution value, whereas there is a gap of up to 20% on the lower bound for instances *br17.x.10*, *br17.x.12*, and *br17.x.13*.

Problem	n	$\|R\|$	Opt.	Heur.	BC–root bounds	BC–root GAP	BC–tree #N, Level	# cuts	#LPs	CPU
br17.x.1	18	1	39	39	–	–	0, 0	16	10	0:00.48
br17.x.2	18	2	39	39	–	–	0, 0	22	12	0:00.68
br17.x.3	18	3	39	39	–	–	0, 0	18	10	0:00.50
br17.x.4	18	4	39	39	–	–	0, 0	18	10	0:00.50
br17.x.5	18	5	42	42	[40, 42]	5.00	20, 6	121	74	0:06.90
br17.x.6	18	6	42	65	–	–	0, 0	68	28	0:01.75
br17.x.7	18	7	47	65	–	–	0, 0	150	55	0:04.15
br17.x.8	18	8	47	65	[44, 47]	6.82	4, 2	132	57	0:04.70
br17.x.9	18	9	47	65	[44, 47]	6.82	30,11	190	125	0:13.95
br17.x.10	18	10	55	65	[45, 55]	22.22	1030,26	9298	5344	13:47.20
br17.x.11	18	11	55	65	[53, 55]	3.77	8, 4	434	158	0:17.77
br17.x.12	18	12	55	81	[44, 55]	25.00	874,27	6245	3965	10:49.67
br17.x.13	18	13	55	81	[44, 55]	25.00	188,18	1768	1022	2:21.72
br17.x.14	18	13	68	81	[67, 68]	1.49	4, 2	164	59	0:04.25
br17.x.15	18	14	68	81	[67, 68]	1.49	4, 2	198	72	0:06.18
br17.x.16	18	14	68	68	–	–	0, 0	86	27	0:01.88
br17.x.17	18	13	84	89	[74, 84]	13.51	24, 6	389	149	0:15.15
br17.x.18	18	14	84	89	[83, 84]	1.20	2, 1	97	32	0:02.02
br17.x.19	18	14	139	143	–	–	0, 0	42	15	0:00.88
br17.x.20	18	15	143	160	–	–	0, 0	25	11	0:00.47
br17.x.21	18	15	233	271	–	–	0, 0	27	12	0:00.52
br17.x.22	18	16	252	252	–	–	0, 0	54	11	0:00.55
br17.x.23	18	14	294	294	–	–	0, 0	21	10	0:00.37
br17.x.24	18	14	294	331	–	–	0, 0	3	3	0:00.13
br17.x.25	18	14	294	294	–	–	0, 0	1	2	0:00.15
br17.x.26	18	14	297	297	–	–	0, 0	0	1	0:00.12
br17.x.27	18	15	297	297	–	–	0, 0	0	1	0:00.08
br17.x.28	18	14	297	297	–	–	0, 0	0	1	0:00.05

Table 5.11. TSPLIB instances *br17.atsp* with random precedences

5.9 Summary and conclusion

In this chapter an algorithm for the exact solution of instances of the Sequential Ordering Problem has been presented, that is based on an investigation of the Sequential Ordering polytope $SOP(n, P)$. We have seen that it was difficult to study the facial structure of $SOP(n, P)$ for general classes of instances. This is mainly due to the reason that the corresponding polytope may change dramatically with the precedence relationships even for fixed n. Several classes of valid inequalities have been derived that are based on facet–defining ATSP–inequalities. It will be challenging to derive a general "strengthening–procedure" for valid and facet–defining inequalties for the asymmetric travelling salesman polytope in case that precedences are present.

Despite of theoretical difficulties in describing $SOP(n, P)$ properly, an algorithm has been described that is capable of solving to optimality within a reasonable amount of time most of the small and medium sized problem instances that arise in practical applications. If the instance could not be solved to optimality, good lower bounds have been derived. In contrast to other approaches this algorithm can deal with a varying number of precedences.

For the instances considered the SEC and their strengthened version as π–, σ-, (π, σ)–, and pcb–inequalities are the most important classes of inequalities. 2–matching constraints do not really help in solving the instances, mainly due to the reason that no separation routine for their strengthened version exist.

We have seen that in the practical applications we considered, the corresponding ATSP–instances are "easily solvable". The described algorithm has problems in solving problem instances that were artificially derived out of hard ATSP instances. To overcome these problems more separation routines for (asymmetric) inequalities are needed, e.g., derived from ATSP–facets. Recently Fischetti and Toth [FT94] developed separation routines for D_k– and odd CAT–inequalities. They made their implementations available to the author. Their embedding into the branch&cut algorithm will certainly improve its performance.

Furthermore, it seems necessary to develop more exact and heuristic separation routines that explicitly take the precedences into account instead of following the strategy of running ATSP–separators and then strengthen the cuts due to the precedence structure.

The heuristics used so far need to be improved. A possible step would be to allow non-feasible intermediate sequences within the exchange heuristics. We conjecture that for most of the unsolved instances the calculated lower bounds are "relatively good". Therefore, we believe that some of the up to now unsolved instances could have been solved to optimality, if a better upper bound could be supplied to the branch&cut algorithm, as the size of the branch&cut trees would reduce dramatically.

As a by–product we obtained a branch&cut algorithm for the ATSP. The only separation routines used for pure ATSP–instances are the ones for subtour elimination constraints, 2–matching constraints, and the simple heuristic for T_k–inequalities. To the best of our knowledge the only published results of a branch&cut approach for the ATSP is by Fischetti and Toth [FT94]. Although it seems difficult to compare implementations running on different platforms, we note that the computing times presented in this chapter are comparable to those by Fischetti and Toth. From our point of view this is due to the advanced branch&cut framework ABACUS and to the use of CPLEX instead of XMP. Fischetti and Toth use additional separation routines for D_k–inequalities and odd CATs. Therefore, the number of iterations of their algorithm is less than for ours.

Chapter 6

Hamiltonian path problems with time windows

Time constrained sequencing and routing problems arise in many practical applications. In Chapter 4 we have seen an application of the asymmetric Hamiltonian path problem with time windows (AHPPTW) originating in the sequencing of jobs to be performed by the stacker crane in an automatic storage system. The AHPPTW is a generalization of the asymmetric Hamiltonian path problem (AHPP) where each node has to be visited within one specified time interval.

So far, not much attention has been paid to the design of exact algorithms to solve the problem. As far as we know, all of them are based on implicit enumeration techniques (branch&bound, dynamic programming). A polyhedral approach to solve problem instances to optimality is known to work well for the TSP, ATSP (see [JRR94], [FT94]), and for the precedence constrained ATSP (cmp. Chapter 5). A question that has not been answered is whether a polyhedral approach can be applied successfully in order to solve the time constrained ATSP and AHPP.

In this chapter an exact branch&cut algorithm to solve problem instances of the AHPPTW to optimality is described. New classes of valid inequalities and a computational comparison of two different models are presented. Preliminary computational results solving problems of up to 50 nodes are reported. All results of this chapter were obtained in joint work with Matteo Fischetti (University of Padova).

6.1 Introduction and notation

The **asymmetric Hamiltonian path problem with time windows (AHPPTW)** can be described in graph theoretical terms in the following way.

Consider a directed, complete graph $D = (V, A_n)$ on $n := |V|$ nodes and nonnegative arc weights c_{ij} associated with each arc $(i, j) \in A_n$. Furthermore, we assume that we are given a processing time $p_i \geq 0$, a release date $r_i \geq 0$ and a deadline $d_i \geq r_i$ for every node $i \in V$. The release date r_i denotes the earliest possible, the deadline d_i the latest possible starting time for processing node $i \in V$. Throughout this chapter we assume that arc weights, processing times, release dates and deadlines are integer values. For the deadline we also allow $d_i = \infty$. The time delay for processing node j immediately after node i is given by

$$\vartheta_{ij} := p_i + c_{ij}.$$

If not stated differently, we assume throughout this chapter that the triangle inequality on ϑ is satisfied, i.e.,

$$\vartheta_{ij} \leq \vartheta_{ik} + \vartheta_{kj}, \text{ for all } i, j, k \in V.$$

The interval $[r_i, d_i]$ is called the **time window** of node i, the **width** of the time window is given by $d_i - r_i$. The time window for node $i \in V$ is called **active** if $r_i > 0$ or $d_i < \infty$. A time window $[0, \infty)$ is called **relaxed**. In the sequel we denote with t_i the **start time** for processing node $i \in V$.

The problem is to find a sequence of the nodes with minimal cost such that for every node $i \in V$ the start time t_i for processing (visiting) node $i \in V$ lies within the given time window $[r_i, d_i]$. We will deal with the case that waiting times are allowed, i.e., one may arrive at a node $i \in V$ earlier than r_i and wait until the node is released. In the objective function we consider the travel distance only, waiting times for processing certain nodes are not taken into consideration. In the literature other objective functions are also studied. One example would be to minimize the route duration, i.e., the difference between the starting time of the first node in the path and the completion time of the last node.

By introducing an additional node (depot) d whose time window is relaxed (i.e., $[0, \infty)$) the AHPPTW can be transformed into an asymmetric travelling salesman problem with time windows (ATSPTW). In the ATSPTW we are interested in finding a cost–minimal tour through all nodes starting and ending at the depot node d and satisfying the time window constraints for all nodes.

Note that the AHPPTW reduces to the standard asymmetric Hamiltonian path problem (AHPP) for $p_i = 0, r_i = 0$, and $d_i = +\infty$ for every $i \in V$. As the AHPP is an $\mathcal{NP}$–hard problem, we cannot expect that, except for some trivial cases, the problem will be easier, if the additional time window constraints are involved. In fact, Savelsbergh [Sav85] showed that it is already strongly $\mathcal{NP}$–complete to find a feasible solution for the TSPTW. We want to mention that this already follows from an older result of Garey and Johnson [GJ77], who showed that it is strongly $\mathcal{NP}$–complete to find a feasible schedule for nonpreemptive single machine scheduling with release–times and deadlines. Ober [Obe92] adapted Savelsbergh's proof and showed that this also holds for the AHPPTW. Furthermore, Tsitsiklis [Tsi92] showed that the TSPTW with general time windows is strongly $\mathcal{NP}$–complete, even if all points are on a line and all processing times equal 0.

Although problems of this type arise in many practical applications, not much attention has been paid to that problem so far. In most publications **exact algorithms** play a minor role and the authors concentrate on the design of heuristics, often based on local search, see [Sav91] among others. To our knowledge there are only a few approaches that were aimed at solving to optimality problem instances of the (symmetric) TSP with time windows.

Christofides et al. [CMT81] describe a branch&bound algorithm in which the lower bound calculation is performed via a state space relaxation in a dynamic programming scheme. They report the solution of problem instances of up to 50 nodes with "moderately tight" time windows.

Baker [Bak83] also describes a branch&bound algorithm where the lower bound calculations are reduced to the solution of the dual problem of a relaxation. The problem that has to be solved is a longest path problem on an acyclic network. The algorithm performs well on problems of up to 50 nodes when only a small percentage of the time windows overlap.

Dumas et al. [DDGS93] present a dynamic programming algorithm for the TSPTW that is able to solve problems of up to 200 nodes and "fairly wide" time windows. Here reductions

of the state space and the state transitions are performed that are based on the time window structure.

It seems difficult to compare the different approaches as there exists no standard test set of problem instances for the TSPTW. The size of the problems that can be solved, typically expressed by the number of nodes, is extremely dependent on the structure of the time windows. In all cases, the authors only use randomly generated data. The case where the time windows are active for only a subset $W \subset V$ and all others are relaxed to $[0, \infty)$ is considered to be more difficult to solve by means of implicit enumeration (see, e.g., Baker [Bak83], who relaxed up to 50% of all time windows).

As far as we know, there are no publications where the attempt to address the problem by means of a branch&cut algorithm is reported. Applegate and Cook [AC91] report on a cutting plane algorithm for the **job–shop scheduling problem (JSSP)**. In the JSSP we are given a set of machines, a set of jobs to be scheduled on these machines, for each job a specified processing order through the machines, processing times for each job on each machine, and constraints that each machine can handle at most one job at a time. By considering only one machine we can derive the earliest arrival time of a job on that machine. Thus, we obtain release dates of the jobs. Therefore, many of the inequalities used for the JSSP can easily be adapted to be valid for the AHPPTW. We will point out to these inequalities later. The results Applegate and Cook report on the JSSP were not very promising, as several 10x10–problem instances could not be solved.

Some **generalizations** of the ATSPTW are discussed in the literature. These problem classes mainly occur in vehicle routing problems, where the ATSPTW is, e.g., a subproblem in a *"cluster–first, route–second approach"*. Here the nodes that have to be visited are clustered heuristically and in the following routing step one tries to solve the arising ATSPTW within each cluster to optimality. These related problems are known as m–ATSPTW, pick–up–and–delivery problems and dial–a–ride problems with time windows. Solving these problems to optimality is typically done by means of implicit enumeration (dynamic programming, branch&bound). But here, too, these approaches are restricted to the case of small size with tight time windows, i.e, time windows with small width. In other applications soft time windows are considered instead of the hard time windows that can not be violated. In these cases, a violation of the time windows is allowed, but results in an additional penalty cost.

For a survey on time constrained routing and scheduling problems see [DLSS88, DDSS93], among others. In Desrosiers et al. [DDSS93] the interested reader can find a comprehensive survey on the TSPTW.

Recall, that t_i was the starting time for processing node $i \in V$, and that for notational convenience a path P consisting of the arc set $\{(v_i, v_{i+1}) \mid i = 1, ..., k-1\}$ is denoted by its node set, i.e., $P = (v_1, v_2, ..., v_{k-1}, v_k)$. To simplify the notation within this chapter we need the following definitions.

(6.1.1) Definition.
Suppose we are given a digraph $D = (V, A)$ with cost coefficients c_{ij} for all $(i, j) \in A$, processing times p_i and time windows $[r_i, d_i]$ for all nodes $i \in V$.

(i) A Hamiltonian path is **feasible**, if each node is visited within its time window, i.e., $r_i \leq t_i \leq d_i$.

(ii) A path $P = (v_1, ..., v_k), 2 \leq k \leq n$, is said to be **infeasible**, if it does not occur as a subpath in any feasible Hamiltonian path.

(iii) For a given path $P = (v_1, ..., v_k), 2 \leq k \leq n$, let $[P]$ denote its **transitive closure**, i.e., $[P] := \{(v_i, v_j) \mid v_i, v_j \in P, 1 \leq i < j \leq k\}$.

(iv) For a given path $P = (v_1, ..., v_k)$ the earliest possible arrival time at v_k when going through path P is denoted by $\theta(P)$.

(v) A path $P = (v_1, ..., v_k)$ is called **minimal**, if $t_i > r_{v_i}$ for all $i = 2, ..., k$, i.e., the starting time for processing each but the first node in this path is greater than its release time.

◇

Given a path $P = (v_1, ..., v_k)$, the arrival time $\theta(P)$ at node v_k (with respect to path P) can be calculated by

$$\theta(P) = \max\{... \max\{\max\{r_{v_1} + \vartheta_{v_1 v_2}, r_{v_2}\} + \vartheta_{v_2 v_3}, r_{v_3}\}....\}.$$

If the path P is minimal, it is not necessary to take waiting times at nodes into consideration. Therefore, the above expression reduces to

$$\theta(P) = r_{v_1} + \sum_{i=1}^{k-1} p_{v_i} + \sum_{i=1}^{k-1} c_{v_i v_{i+1}} = r_{v_1} + \sum_{i=1}^{k-1} \vartheta_{v_i v_{i+1}}.$$

Note that if $P = (v_1, ..., v_k)$ violates the time window for v_k but is not minimal, there exists an $l, 1 < l < k$, such that $P' = (v_l, ..., v_k)$ is infeasible (and minimal).

In order to distinguish between feasible and infeasible paths we need a more specific characterization of the infeasibility of a path.

(6.1.2) Lemma.
Suppose we are given a digraph $D = (V, A)$ with cost coefficients c_{ij} for all $(i, j) \in A$, processing times p_i and time windows $[r_i, d_i]$ for all nodes $i \in V$ and that the triangle inequality on ϑ is satisfied, i.e., $\vartheta_{ij} \leq \vartheta_{ik} + \vartheta_{kj} \forall, i, j, k \in V$.
A path $P = (v_1, ..., v_k)$ is **infeasible**, if one of the following conditions is satisfied:

(i) P violates the time window for v_k, i.e., $\theta(P) > d_{v_k}$.

(ii) There exists a $v_i \in V, v_i \notin P$, such that both paths $P_1 = (v_i, v_1, ..., v_k)$ and $P_2 = (v_1, ..., v_k, v_i)$ violate the given time windows.

Proof.

(i) Obvious, since path P cannot be a subpath of a feasible Hamiltonian path (time window for v_k is violated).

(ii) Since in any feasible Hamiltonian path node v_i cannot be sequenced either in front or behind P.

□

Note that the conditions of Lemma (6.1.2) do not yield a complete description of all infeasible paths. In case that condition (6.1.2)(ii) is satisfied, we say that node v_i cannot be **covered by path** P. Note, that if the triangle inequality on ϑ is not satisfied, the path $(v_i, v_1, ..., v_k)$ might be infeasible but the path $(v_i, v_j, v_1, ..., v_k)$ is feasible. Instead of considering $\vartheta_{v_i v_1}$ for sequencing v_i in front of v_1 the shortest path on ϑ from v_i to v_1 has to be taken into account.

6.2 Preprocessing

As for many other combinatorial optimization problems preprocessing is an important part of an efficient implementation. Its main aim is to construct a "tighter" equivalent formulation of the problem, such that no optimal solution to the original problem is lost and each optimal solution to the tighter problem corresponds to an optimal solution to the original problem.

For the AHPPTW this includes three main steps: tightening the time windows, constructing precedences among the nodes, and fixing variables permanently. We detect paths $P = (v_1, v_2)$ infeasible due to the criteria given by Lemma (6.1.2). If such a path is detected, the corresponding arc (v_1, v_2) cannot be used in any feasible solution and is therefore deleted from the feasible arc set. If it is possible to derive precedence relationships between the nodes in V, we can apply the methodology that has been developed for the precedence constrained ATSP (see Chapter 5).

In the following we always assume that we are given a (not necessarily complete) digraph $D = (V, A)$, cost coefficients c_{ij} for all arcs $(i, j) \in A$, time windows $[r_i, d_i]$ and processing times p_i for all nodes $i \in V$. Recall that the minimum time delay for processing j immediately after i was defined by $\vartheta_{ij} = p_i + c_{ij}$.

6.2.1 Tightening of the time windows

In this section we present criteria that allow us to increase the release date (resp. to decrease the deadline) of certain nodes. These reductions have already been presented in [DDS92, DDSS93].

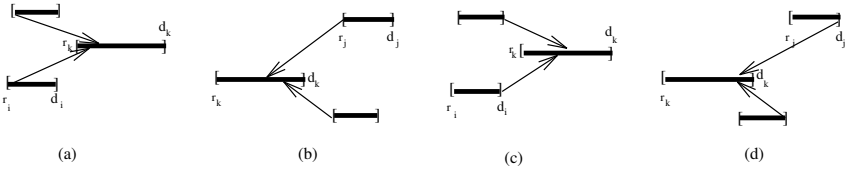


Figure 6.2.1

Release date adjustment

If the earliest arrival time at node $k \in V$ from any of its possible predecessors is bigger than its release date r_k (see Figure 6.2.1(a)), the release date of k might be increased, i.e.,

$$r_k = \max\{r_k, \min_{(i,k)\in A}\{r_i + \vartheta_{ik}\}\} \quad \forall k \in V \text{ s.t. } \delta^-(k) \neq \emptyset. \tag{6.2.3}$$

In order to avoid waiting times at the possible successor nodes of $k \in V$, the earliest possible starting time of k, and therefore its release date might be shifted (see Figure 6.2.1(b)), i.e.,

$$r_k = \max\{r_k, \min\{d_k, \min_{(k,j)\in A}\{r_j - \vartheta_{kj}\}\}\} \quad \forall k \in V \text{ s.t. } \delta^+(k) \neq \emptyset. \tag{6.2.4}$$

Due date adjustment

If the deadline d_k of node $k \in V$ is bigger than the latest possible arrival time at node k from any of its predecessors, the deadline might be decreased (see Figure 6.2.1(c)), i.e.,

$$d_k = \min\{d_k, \max\{r_k, \max_{(i,k)\in A}\{d_i + \vartheta_{ik}\}\}\} \quad \forall k \in V \text{ s.t. } \delta^-(k) \neq \emptyset. \tag{6.2.5}$$

If the latest possible departure time from node $k \in V$ in order to fulfill all time window constraints for its successors is less than its deadline, then it might be decreased (see Figure 6.2.1(d)), i.e.,

$$d_k = \min\{d_k, \max_{(k,j)\in A}\{d_j - \vartheta_{kj}\}\} \quad \forall k \in V \text{ s.t. } \delta^+(k) \neq \emptyset. \tag{6.2.6}$$

6.2.2 Construction of precedences

In case that the time windows for two nodes i and j, $i, j \in V$, are "nonoverlapping" (i.e., $r_j + \vartheta_{ji} > d_i$) we know that node i has to precede node j in any feasible solution to the AHPPTW. Therefore, precedences among the nodes in V might be derived and all the methodology developed for the SOP can also be applied to the AHPPTW.

Let $i \prec j$ denote the fact that node i has to precede node j in every feasible solution. For notational convenience we introduce the **precedence digraph** $P = (V, R)$ that is defined on the same node set as D_n and where an arc $(i, j) \in R$ represents a precedence relationship $i \prec j$. Note that P must be acyclic and can be assumed to be transitively closed.

For a given arc $(i, j) \in R$ we know that arc (j, i) cannot be used in any feasible Hamiltonian path, as this would violate the precedence relationship among these nodes. Furthermore, for any nodes $i, j, k \in V$ with $(i, j) \in R$ and $(j, k) \in R$ we can conclude that arc (i, k) cannot be used in any feasible Hamiltonian path as node j has to be sequenced in between. These arcs are then eliminated from the feasible arc set A.

6.2.3 Elimination of arcs

For all arcs $(i, j) \in A$ it is checked, if they can be used in a feasible Hamiltonian path. Therefore, we start with the feasible path $P = (i, j)$. We now try to enlarge this path by sequencing nodes $Q = \{v_1, ..., v_k\}, v_l \in V \setminus \{i, j\}, l = 1, ..., k$, around P. If this cannot be done without constructing an infeasible path P', we know that arc (i, j) cannot be used in any feasible solution and it is eliminated from the feasible arc set A. For sets Q of small cardinality ($|Q| = 1, 2$) this can be checked by enumerating all possible paths.

For $Q = \{k\}$ we have to check, if both paths $P_1 = (i, j, k)$ and $P_2 = (k, i, j)$ are infeasible, i.e., if

$$\begin{cases} \max\{r_i + \vartheta_{ij}, r_j\} + \omega_{jk} > d_k & \text{and} \\ r_k + \omega_{ki} > d_i \quad \text{or} \quad r_k + \omega_{ki} + \omega_{ij} > d_j \end{cases}$$

where ω_{ij} denotes the shortest path on ϑ from i to j. Note that if k is in any precedence relationship with i or j, only one of the conditions has to be checked. If the triangle inequality on ϑ is satisfied, then ω_{ij} might be substituted by ϑ_{ij}.

For $Q = \{k, l\}$ the six paths $P_1 = (i, j, k, l), P_2 = (i, j, l, k), P_3 = (k, l, i, j), P_4 = (l, k, i, j), P_5 = (k, i, j, l), P_6 = (l, i, j, k)$ have to be checked. If there exist precedences among the nodes, certain paths do not have to be considered.

For $|Q| > 2$ it seems computationally too expensive to apply this procedure.

6.3 Modelling

In this section we provide two different ways of modelling the AHPPTW as integer linear programs. Based on these models we implemented branch&cut algorithms to solve instances of the AHPPTW to optimality. In Section 6.7 (preliminary) computational results on a set of random and real–life data will be given.

Model 1 is the standard model that can be found in many publications. This model involves arc variables as well as node variables. The time window restrictions are modelled with the help of a "big M". As in contrast to other publications the objective function is defined on the arc variables only it is possible to get rid of both the node variables and the "big M". Therefore, in model 2 the time window constraints are modelled by a new class of inequalities, the so–called **infeasible path constraints**.

Suppose we are given a digraph $D = (V, A)$ with node set V and feasible arc set A. For each arc $(i, j) \in A$ we introduce a binary variable $x_{ij} \in \{0, 1\}$ with the interpretation:

$$x_{ij} = \begin{cases} 1, & (i,j) \in A \text{ is in the Hamiltonian path,} \\ 0, & \text{otherwise.} \end{cases}$$

6.3.1 Model 1

Miller, Tucker, and Zemlin [MTZ60] proposed to substitute the subtour elimination constraints for the TSP

$$x(A(W)) \leq |W| - 1 \quad \forall\, W \subset V, |W| \geq 2 \tag{6.3.7}$$

by a smaller class of inequalities, but at the expense of extra free variables $t_i, i = 1, ..., n$. The MTZ–subtour elimination constraints can be written as

$$\begin{array}{rcll} t_i - t_j + (n-1)x_{ij} & \leq & n-2 & i,j = 1,...,n, i \neq j \\ 1 \quad \leq \quad t_i & \leq & n-1 & i = 1,...,n. \end{array}$$

The MTZ–formulation of the subtour elimination constraints results in a weak LP–formulation (see, e.g., [PS88]), as the MTZ–inequalities are not facet defining for the corresponding travelling salesman polytope. But they offer the advantage that they can easily be modified to take side constraints into account (see [DL91]). For the ATSP with time windows we interpret the t–variables to be time variables giving the starting times for processing the nodes. In the following let M be a large value.

The MTZ–inequalities for the ATSPTW can be written as

$$\begin{array}{rcll} t_i + \vartheta_{ij} - (1 - x_{ij}) \cdot M & \leq & t_j & i,j = 1,...,n, i \neq j \\ r_i \quad \leq \quad t_i & \leq & d_i & i = 1,...,n. \end{array} \tag{6.3.8}$$

With the help of the binary arc variables x_{ij} and the integer node variables t_i the AHPPTW can be formulated as an integer linear program as follows:

$$
\begin{array}{llll}
 & \min c^T x & & \\
\text{s.t. } (1) & x(A) & = n-1 & \\
(2) & x(\delta^-(i)) & \leq 1 & \forall\ i \in\ V \\
(3) & x(\delta^+(i)) & \leq 1 & \forall\ i \in\ V \\
(4) & t_i + \vartheta_{ij} - (1 - x_{ij}) \cdot M & \leq t_j & \forall\ (i,j) \in A \\
(5) & t_i & \leq d_i & \forall\ i = 1, ..., n \\
(6) & t_i & \geq r_i & \forall\ i = 1, ..., n \\
(7) & x_{ij} & \in \{0,1\} & \forall\ (i,j) \in A \\
(8) & t_i & \in \mathbb{N} & \forall\ i = 1, ..., n
\end{array}
\tag{6.3.9}
$$

Due to (1) the solution consists of $n-1$ arcs, (2) and (3) guarantee that at most 1 arc is entering and leaving each node. Inequalities (4) avoid subtours *(MTZ–subtour elimination constraints)* and assure that whenever j is sequenced immediately after i ($x_{ij} = 1$), the starting time t_j for processing node j cannot be less than the starting time of node i plus its processing time and travel time from i to j. If j is not sequenced immediately after i ($x_{ij} = 0$), the inequality is always valid due to "big M". Inequalities (5) and (6) guarantee that node $i \in V$ is processed within its time window. Note that an individual "big M_{ij}" might be defined for each inequality in (4), satisfying

$$M_{ij} \geq d_i + p_i + c_{ij} - r_j.$$

(1)–(3),(7) together with the subtour elimination constraints (6.3.7) forms the standard LP–formulation for the asymmetric Hamiltonian path problem.

It is easy to see that every feasible solution x of (1)–(8) (if existent) is the incidence vector of a feasible Hamiltonian path satisfying all given time windows. Due to the time windows the above system might have no solution. In the sequel we always assume that a feasible solution exists.

This model has some **disadvantages**. First the MTZ–inequalities (6.3.8) are not very strong and they can be lifted in several ways (see Section 6.5.4). Furthermore, it is known from practical experience that a "big M"–modelling will cause computational problems. Our computations confirmed that observation. The fractional solutions we obtained with the use of this model in a cutting plane algorithm do not have such a "nice" structure as for the pure ATSP, where (at least during the first iterations) the fractional values tend to be $\frac{1}{2}$, $\frac{1}{3}$, or $\frac{2}{3}$. But even more important is that the x– and t–variables are only weakly linked via the MTZ–inequalities, i.e., the structure of the time windows will only have small influence on the Hamiltonian path described by the x–variables.

As we are interested in the polyhedral structure defined by the convex hull of all feasible solutions of (6.3.9), we denote with

$$P_1^{TW} := conv\{(x,t) \in \mathbb{R}^{A \times V} \mid (x,t) \text{ satisfies (6.3.9)(1)–(8)}\}$$

the AHPTW–polytope based on model 1.

6.3.2 Model 2

Due to the fact that the objective function is defined only on the arc variables it is possible to model the time window restrictions by an additional class of inequalities, the so–called **infeasible path constraints**. The corresponding model is the following:

$$
\begin{array}{llll}
 & \min c^T x & & \\
\text{s.t. } (1) & x(A) & = n-1 & \\
(2) & x(\delta^-(i)) & \leq 1 & \forall\, i \in V \\
(3) & x(\delta^+(i)) & \leq 1 & \forall\, i \in V \\
(4) & x(A(W)) & \leq |W|-1 & \forall\, W \subset V,\ 2 \leq |W| \\
(5) & x(P) & \leq k-2 & \forall \text{ infeasible paths} \\
 & & & P=(v_1, v_2, ..., v_k) \\
(6) & x_{ij} & \in \{0,1\} & \forall\, (i,j) \in A
\end{array}
\tag{6.3.10}
$$

As far as we know, this model for the AHPPTW has not been defined before in the literature. A similar model can easily be defined for the ATSPTW.
In (6.3.10)(5) $x(P)$ denotes the sum of the variables corresponding to the path $P = (v_1, ..., v_k)$, i.e., $x(P) = \sum_{i=1}^{k-1} x_{v_i v_{i+1}}$. Inequalities (6.3.10)(5) forbid infeasible paths, i.e., paths violating the given time windows. Therefore, each solution x of (6.3.10)(1)–(6) is the incidence vector of a feasible Hamiltonian path, and vice versa. The formulation of the infeasible path constraints as stated in (6.3.10)(5) is weak. In Section 6.5.1 we present several stronger inequalities of this type.
Note that no "big M" modelling is required in this model. Moreover, only arc variables are present, which implies that no linking constraints between arc and node variables are necessary. This suggests that this model is superior to the later one. The computational results presented in Section 6.7 will confirm this expectation. But observe, that the use of this model is restricted to the case of the standard TSP-objective, namely to minimize the sum of travel times between the nodes.
Similar to model 1 we denote with

$$P_2^{TW} := conv\{x \in \mathbb{R}^A \mid x \text{ satisfies (6.3.10)(1)–(6)}\}.$$

the convex hull of all feasible solutions of (6.3.10).

6.4 Dimension of the polytope

In this section we show that it is an $\mathcal{NP}$–hard problem to determine the dimension of the AHPPTW polytope. First recall from the first section that it is an $\mathcal{NP}$–complete problem to find a feasible solution. Therefore, it is $\mathcal{NP}$–hard to decide (for the general case) whether the polytope is empty or not.

In the following we restrict ourselves to study two "simple" special instances of the AHPPTW. In the first, only one time window is active. We show that the solution of a min–cost Hamiltonian path problem is required to calculate the dimension of the corresponding polytope. Next, we consider the case where two time windows are active. To prove results on the dimension of the corresponding polytope we need the additional assumption that the release dates for the active time windows are greater than a certain big value M. For both cases we restrict our analysis to the case where only the x–variables are present (model 2).

6.4.1 Relaxation I : Only one active time window

Consider the following instance of the AHPPTW, where a complete, loop–free digraph $D = (V, A)$ with $n := |V|$ nodes, $n \geq 4$, $|A| = n \cdot (n-1)$ arcs, and only one active time window $[r_1, d_1]$ associated with node 1 is given. All other time windows are relaxed to $[0, +\infty)$. Assume that $\vartheta_{ik} + \vartheta_{kj} \geq \vartheta_{ij}\ \forall i, j, k \in V$.

Partition the node set as follows :

$$V = \{1\} \cup Q \cup W$$

where

$$Q := \{j \in V \setminus \{1\} \mid p_j + c_{j1} > d_1\}$$

and

$$W := V \setminus (\{1\} \cup Q).$$

That is Q contains all nodes that cannot be sequenced before node 1 without violating the time window for node 1.

Construct an undirected **auxiliary graph** $G_a = (V \setminus \{1\}, E)$, where

$$E := \{ij \mid i, j \in V \setminus \{1\}, i \neq j, p_i + p_j + min\{c_{ij} + c_{j1}, c_{ji} + c_{i1}\} \leq d_1\}.$$

That is $ij \in E$, if and only if either $(i, j, 1)$ or $(j, i, 1)$ or both are feasible paths. Note that the nodes of Q are isolated in G_a (some nodes in W can also be isolated, i.e., be incompatible with all $j \in V$).

Consider the following equations:

$$\begin{array}{lllll} (1) & x_{j1} & = & 0, & \forall\ j \in Q \\ (2) & x(\delta^+(1)) & = & 1, & \text{iff node 1 cannot be the final node} \\ & & & & \text{of a feasible Hamiltonian path} \\ (3) & \sum_{j \in S_h} x(\delta^-(j)) & = & |S_h| - x(S_h : 1), & h = 1, ..., m \end{array} \tag{6.4.11}$$

where $S_h \subseteq V \setminus \{1\}, h = 1, ..., m$ are the m connected components of G_a. Note that adding up the m equations (3) leads to

$$\sum_{h=1}^{m} (\sum_{j \in S_h} x(\delta^-(j))) = \sum_{h=1}^{m} |S_h| - \sum_{h=1}^{m} x(S_h : 1),$$

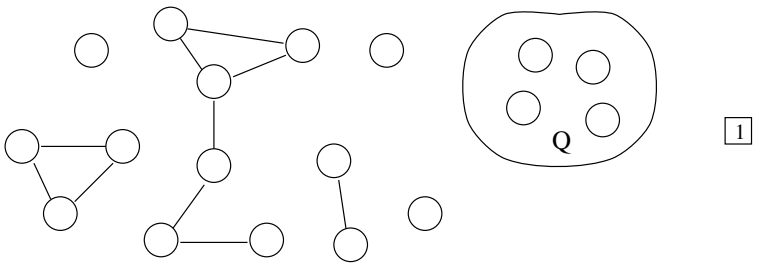


Figure 6.4.2 The auxiliary graph G_a

which can be rewritten as

$$x(A) = n - 1. \tag{6.4.12}$$

Therefore, 6.4.12 is a combination of 6.4.11(3) and hence dependent. Furthermore, note that it requires the solution of a min–cost Hamiltonian path problem to check the condition of equation (2).

(6.4.13) Lemma.
If there is only one time window active in the AHPPTW, then the equations 6.4.11(1)–(3) are valid for P_2^{TW}.

Proof. Obvious for equations (1) and (2).
W.l.o.g. assume that the time window for node 1 is active. Let now $S_h \subseteq V \setminus \{1\}$ be any connected component of G_a. Note that no edge can cross the cut $[S_h, V \setminus (\{1\} \cup S_h)]$.
Consider any feasible path x, and let k be the starting node. Hence $x(\delta^-(k)) = 0$. The left hand side of (3) is then

$$\sum_{j \in S_h} x(\delta^-(j)) = \begin{cases} |S_h| - 1, & \text{if } k \in S_h, \\ |S_h|, & \text{otherwise.} \end{cases}$$

Assume first that $k \in S_h$ and $x(S_h : 1) = 0$, and let $j \notin S_h$ be such that $x_{j1} = 1$. Then we have that edge $kj \in E$, a contradiction (see Figure 6.4.3(a)).

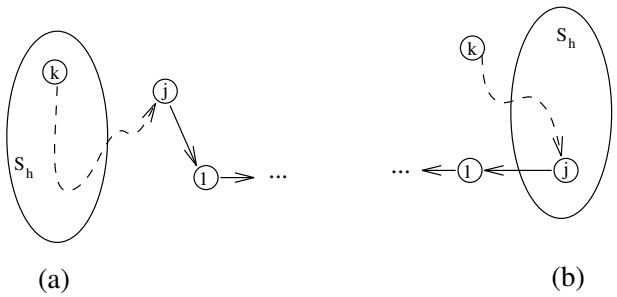


Figure 6.4.3

Now assume that $k \notin S_h$ and $x(S_h : 1) = 1$. Let $j \in S_h$, s.t. $x_{j1} = 1$, and note that edge $kj \in E$, again a contradiction (see Figure 6.4.3(b)). □

(6.4.14) Lemma.
For any instance of the AHPPTW where only the time window for one node, say node 1, is active, the equations 6.4.11(1)–(3) are linearly independent.

Proof. The $m + |Q|$ equations (1) and (3) (and (2) if active) are linearly independent. To see this, it is sufficient to provide for each equation of the family, say $ax = a_0$, a point $x \in \mathbb{R}^A$ such that all the equations but $ax = a_0$ are satisfied.
For equation $x_{j1} = 0 (\forall\, j \in Q)$ set x to be the (characteristic vector of the) path $(S_{h_j}, 1, S_1, S_2,$ $..., S_m)$ with $S_{h_j} = \{j\}$ consisting of a singleton node $j \in Q$ (see Figure 6.4.4(a)).
For equation 6.4.11(3) set x to be the two subpaths $(1, S_1, ..., S_m)(S_h)$ (see Figure 6.4.4(b)).
And if equation 6.4.11(2) is active, set x to be the cycle on $n-1$ nodes $(S_1, ..., S_m)$, $1 \notin S_i$, $i = 1, ...m$ (see Figure 6.4.4(c)).

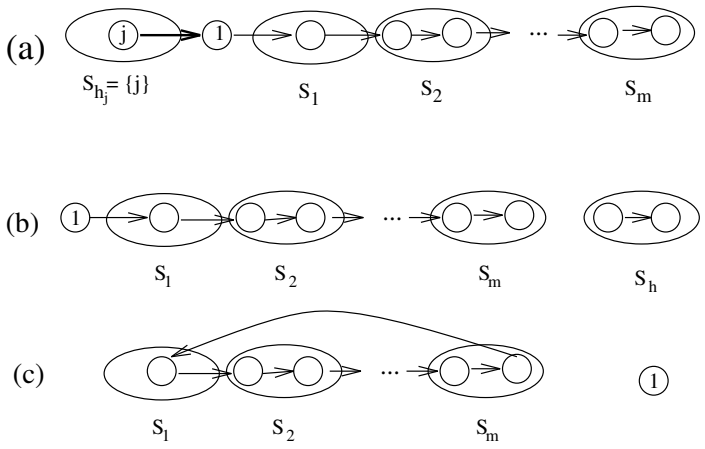


Figure 6.4.4

□

We now show that (1) and (3), plus (2) if correct, form a (minimal) equation system for the (relaxed) AHPPTW polytope P_2^{TW}, i.e., that no other independent equation exists. Let

$$\mu = \begin{cases} 1, & \text{if the condition in (2) is satisfied,} \\ 0, & \text{otherwise.} \end{cases}$$

(6.4.15) Theorem.
Given an instance of the AHPPTW on $n \geq 4$ nodes where only the time window for one node is active, then

$$dim(P_2^{TW}) = |A| - (|Q| + m + \mu).$$

Proof. We give a direct proof, consisting of exhibiting $|A| - (|Q| + m + \mu) + 1$ affinely independent vertices of the polytope P_2^{TW}.
Consider first the face F of P_2^{TW} induced by $x(\delta^-(1)) \geq 0$, containing all feasible paths starting with node 1.

Since we assume $d_j = +\infty, \forall\ j \neq 1$, every Hamiltonian path starting with node 1 is feasible, hence a minimal equation system for this face is known to be

$$\begin{array}{lllll} (i) & x_{j1} & = 0 & \forall\ j \in V \setminus \{1\}, \\ (ii) & x(\delta^-(j)) & = 1 - x_{j1} & \forall\ j \in V \setminus \{1\}, \\ (iii) & x(\delta^+(1)) & = 1. & \end{array}$$

Note that the right hand side of (ii) is 1. Therefore, $dim(F) = |A| - 2n + 1$, thus there exist $dim(F) + 1 = |A| - 2n + 2$ affinely independent vertices of this face. We need $(n - 1 - |Q|) + (n - 1 - m) + (1 - \mu)$ additional points, which we construct in the following way:

A. For each $j \in (V \setminus \{1\}) \setminus Q$, construct the feasible path $(j, 1, ...)$. They are affinely independent due to equations (i) (since $x_{j1} = 0$ for all the remaining points so far considered). Note that equations (ii)–(iii) still hold for these points.

B. For each $h = 1, 2, ..., m$, consider the component S_h. Let T_h be any tree spanning S_h. Choose any seed node $r_h \in S_h$ and give an orientation to the edges in T_h so as to obtain a directed tree (arborescence) rooted at r_h. Then consider the nodes $v \in S_h \setminus \{r_h\}$ in any sequence visiting each node after its father node in the arborescence. Let i be the father node of v in the arborescence.
Since $iv \in E$, one of the two paths $(v, i, 1, ...)$ or $(i, v, 1, ...)$ is feasible and satisfies all the equations (ii) except those with $j = v$ or $j = i$.
Because of the particular sequence in visiting the nodes, all the points constructed so far satisfy the equations (ii) written for $j = v$, hence the new point is affinely independent.
The above construction produces $|S_h| - 1$ new points for each component S_h, i.e., $n - 1 - m$ new points in total.

C. If $\mu = 1$, we are done; otherwise, the last point to be constructed is $(..., 1)$. It is affinely independent, since equation (iii) is satisfied by all the points so far constructed (since $n \geq 4$).

□

Call the instance of the AHPPTW **1–regular**, if only one time window is active and the associated auxiliary graph G_a is connected. This implies $Q = \emptyset$.

(6.4.16) Corollary.
Suppose we are given a 1–regular instance of the AHPPTW on $n \geq 4$ nodes, then

$$dim(P_2^{TW}) = |A| - 1 - \mu.$$

□

6.4.2 Relaxation II : Two active time windows

Now consider the case defined on a complete loop–free digraph $D = (V, A)$, $n := |V|$ and $n \geq 4$, where only two time windows are active. W.l.o.g. assume that nodes 1 and 2 correspond to the nodes with the active time windows. The time windows for all nodes in $V \setminus \{1, 2\}$ are relaxed, i.e., $[0, \infty)$. Furthermore, assume that the release dates for the two active time windows are so that all nodes in $V \setminus \{1, 2\}$ can be sequenced before $\{1, 2\}$, i.e., $r_1 \geq M$ and $r_2 \geq M$.

Case 1: Both arcs feasible

We study first the case in which both arcs (1,2) and (2,1) are feasible, i.e.,

$$\begin{array}{rcl} r_1 + p_1 + c_{12} & \leq & d_2 \\ r_2 + p_2 + c_{21} & \leq & d_1 \end{array} \tag{6.4.17}$$

hold.

Note that due to the construction of the time windows, every feasible path can be "shifted" so as to visit either node 1 or node 2 at its release time.

Indeed, let x be the (characteristic vector of) any Hamiltonian path of the form $(..., k_1, k_2, ..., k_r, ...)$, where $k_1 \in \{1, 2\}$ and $k_r = 3 - k_1$. Then x is feasible, if and only if

$$r_{k_1} + (p_{k_1} + c_{k_1,k_2}) + ... + (p_{k_{r-1}} + c_{k_{r-1},k_r}) \leq d_{k_r}.$$

Let

$$\sigma = \begin{cases} 1, & \text{if no path of the form } (..., 1, k, 2, ...) \\ & \text{or } (..., 2, k, 1, ...) \text{ is feasible,} \\ 0, & \text{otherwise.} \end{cases}$$

The conditions above can easily be checked by verifying

$$r_1 + p_1 + c_{1k} + p_k + c_{k2} \leq d_2$$

or

$$r_2 + p_2 + c_{2k} + p_k + c_{k1} \leq d_1$$

for all $k \in V \setminus \{1, 2\}$.

Now consider the following equations

$$\begin{array}{lll} (1) & x(A) & = n - 1 \\ (2) & x_{12} + x_{21} & = 1 \qquad (\text{ if } \sigma = 1). \end{array} \tag{6.4.18}$$

The two equations above are clearly valid and linearly independent.

(6.4.19) Theorem.
Assume we are given an instance of the AHPPTW on $n \geq 4$ nodes, where only the time windows for two nodes, say nodes 1 and 2, are active, $r_1 \geq M$ and $r_2 \geq M$, where M is big enough such that all nodes in $V \setminus \{1, 2\}$ can be sequenced before $\{1, 2\}$, and both arcs between nodes 1 and 2 are feasible. Then

$$dim(P_2^{TW}) = |A| - 1 - \sigma.$$

Proof. Consider the face F of P_2^{TW} induced by the valid inequality $x_{12} + x_{21} \leq 1$. Because of our assumptions, every Hamiltonian path x, s.t. $x_{12} + x_{21} = 1$ is feasible w.r.t. the time window, and hence belongs to F. Since $x_{12} + x_{21} \leq 1$ defines a facet of the (unconstrained) Hamiltonian path polytope P_H, we have $dim(P_H) = |A| - 1$ affinely independent points in F. If $\sigma = 1$, we are done. Otherwise every feasible path x such that $x_{21} = x_{12} = 0$ produces an additional independent point.
Therefore, we have $|A| - 1 + (1 - \sigma)$ affinely independent points in P_2^{TW}, from which the result follows. □

Case 2: Only one arc feasible

We now address the case in which conditions 6.4.17 do not hold. If both arcs (1,2) and (2,1) are not feasible (i.e., $r_1 + p_1 + c_{12} > d_2$ and $r_2 + p_2 + c_{21} > d_1$), then the instance itself is not feasible and $P_2^{TW} = \emptyset$.
Therefore, assume w.l.o.g. that (1,2) is feasible, but (2,1) is not, i.e.,

$$\begin{aligned} r_1 + p_1 + c_{12} &\leq d_2 \\ r_2 + p_2 + c_{21} &> d_1. \end{aligned}$$

This implies that every feasible path must visit node 1 before node 2.

Construct a bipartite graph $B = (N^+ \cup N^-, E)$, where each node $k^+ \in N^+$ corresponds to the arc $(1, k) \in \delta^+(1), k \neq 2$, each node $h^- \in N^-$ to the arc $(h, 2) \in \delta^-(2), h \neq 1$. We have that an edge $k^+h^- \in E$ whenever there exists a feasible path x, such that $x_{1k} = x_{h2} = 1$.

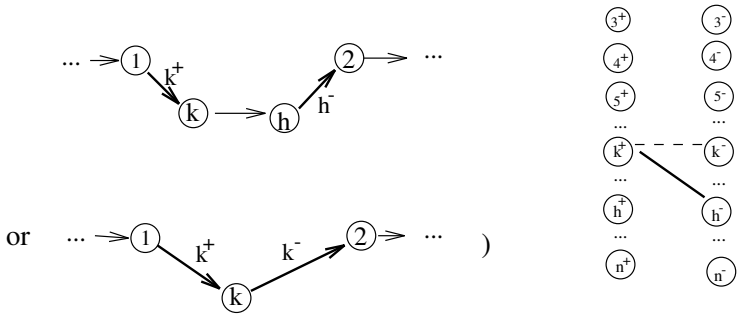


Figure 6.4.5

In other words k^+h^- belongs to E, if

$$\begin{aligned} r_1 + p_1 + c_{1k} + p_k + c_{kh} + p_h + c_{h2} &\leq d_2, \quad \text{if } h \neq k, \\ r_1 + p_1 + c_{1k} + p_k + c_{k2} &\leq d_2, \quad \text{if } h = k. \end{aligned}$$

Let $S_1, S_2, ..., S_m$ be the m connected components of B (where $S_i \subseteq N^+ \cup N^-, i = 1, ..., m$). Consider the following equations

$$\begin{array}{lll} (1a) & x(A) & = n - 1 \\ (1b) & x_{21} & = 0 \\ (2) & x(\delta^+(1)) & = 1 \\ (3) & x(\delta^-(2)) & = 1 \\ (4) & \sum_{k^+ \in S_i} x_{1k} - \sum_{h^- \in S_i} x_{h2} & = 0 \qquad \text{for } i = 1, 2, ..., m \end{array} \tag{6.4.20}$$

Note that in case $S_i = \{k^+\}$ (resp. $S_i = \{h^-\}$), the equation 6.4.20(4) becomes $x_{1k} = 0$ (resp. $x_{h2} = 0$). Moreover, in case $E = \emptyset$ they imply $x_{12} = 1$ because of 6.4.20(2).

(6.4.21) Lemma.
Assume we are given an instance of the AHPPTW on $n \geq 4$ nodes, where

- only the time windows for two nodes, say nodes 1 and 2, are active,
- $r_1 \geq M$ and $r_2 \geq M$, where M is big enough such that all nodes in $V \setminus \{1, 2\}$ can be sequenced before $\{1, 2\}$,
- and only one arc between the active nodes is feasible,

then equations 6.4.20(1)–(4) are valid for P_2^{TW}.

Proof. The validity of equations (1),(2),(3) is obvious.
We now prove the validity of equation (4). Take any feasible path $\bar{x}$ and let $\bar{k}, \bar{h} \in V$ be s.t. $\bar{x}_{1\bar{k}} = \bar{x}_{\bar{h}2} = 1$ (possibly $\bar{k} = \bar{h}$). By definition, if $(..., 1, \bar{k}, ..., \bar{h}, 2, ...), \bar{k} \neq 2$ (and hence $\bar{h} \neq 1$) then $(\bar{k}^+, \bar{h}^-) \in E$, hence $(\bar{k}^+, \bar{h}^-)$ belong to the same component, say $S_{\bar{i}}$. Let $\bar{i} = 0$, if $\bar{x}_{12} = 1$; i.e., $\bar{k} = 2$ and $\bar{h} = 1$. Therefore, all equations (4) are satisfied by $\bar{x}$, since both sums are either 0 (when $i \neq \bar{i}$) or 1 (when $i = \bar{i}$). □

Note that adding up all equations 6.4.20(4) leads to

$$x(\delta^+(1)) - x_{12} = x(\delta^-(2)) - x_{12}.$$

Hence one of the equations 6.4.20(2) and (3) is redundant.

(6.4.22) Lemma.
Assume we are given an instance of the AHPPTW on $n \geq 4$ nodes, where

- only the time windows for two nodes, say nodes 1 and 2, are active,
- $r_1 \geq M$ and $r_2 \geq M$, where M is big enough such that all nodes in $V \setminus \{1, 2\}$ can be sequenced before $\{1, 2\}$,
- and only one arc between the active nodes is feasible,

then the equations 6.4.20(1a),(1b),(2),(4) are linearly independent.

Proof. We prove the linear independence by exhibiting for each such equation, say $ax = a_0$, a point $x \in \mathbb{R}^A$ satisfying all the equations except $ax = a_0$.

– For equation (1a) consider the Hamiltonian cycle $(1, 2, ...)$. Note that all equations (4) are of the form $0 - 0 = 0$.(See Figure 6.4.6(a)).

– For equation (1b) consider the cycle $(1, 2)$ and the Hamiltonian path through the remaining nodes (see Figure 6.4.6(d)). (All equations (4) are of the form $0 - 0 = 0$.)

– For equation (2) consider the Hamiltonian path $(2, , ..., 1)$. Note that again all equations (4) are of the form $0 - 0 = 0$.

– For equation (4) consider the path $(1, a)$ and the Hamiltonian cycle $(2, ...)$ through the nodes in $V \setminus \{1, a\}$ where $a \in S_i$ (see Figure 6.4.6(b)).
If $S_i \cap N^+ = \emptyset$, i.e., $S_i = \{h^-\}$, then take the point shown in Figure 6.4.6(c). Note that these points do necessarily violate the redundant equation (3).

□

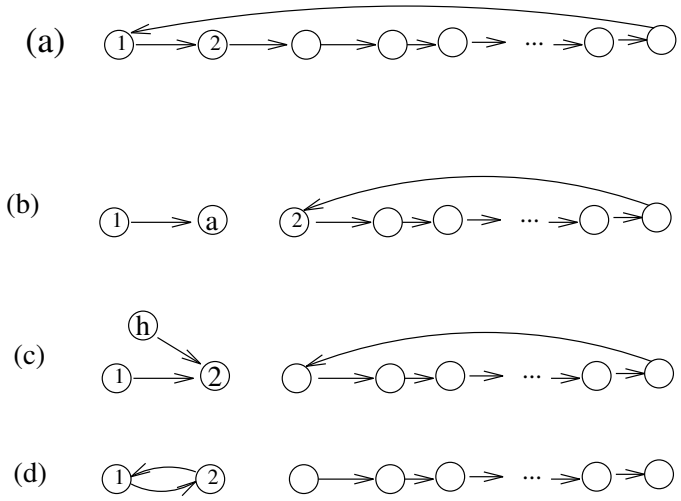


Figure 6.4.6

We next show that the inequalities 6.4.20(1),(2),(4) form a (minimal) equation system for P_2^{TW}, having rank $m+3$.

(6.4.23) Theorem.
Assume we are given an instance of the AHPPTW on $n \geq 4$ nodes, where

- only the time windows for two nodes, say nodes 1 and 2, are active,
- $r_1 \geq M$ and $r_2 \geq M$, where M is big enough such that all nodes in $V \setminus \{1,2\}$ can be sequenced before $\{1,2\}$,
- and only one arc between the active nodes is feasible,

then

$$dim(P_2^{TW}) = |A| - (m+3).$$

Proof. We construct $n \cdot (n-1) - (m+3) + 1$ affinely independent $x \in P_2^{TW}$.

Consider first the face F of P_2^{TW} induced by $x_{12} \leq 1$. The dimension of F is easily established, since there is a one–to–one correspondence between F and the (unconstrained) Hamiltonian path polytope on $n-1$ nodes, P_{n-1}^H. Therefore, we have $dim(P_{n-1}^H) + 1 = ((n-1)(n-2)-1)+1$ affinely independent feasible Hamiltonian paths x having $x_{12} = 1$, and hence satisfying

$$\begin{array}{lll} (i) & x_{1k} & = 0 \quad \forall\ k = 3,\ldots,n \\ (ii) & x_{h2} & = 0 \quad \forall\ h = 3,\ldots,n \end{array}$$

Note that we have an equation (i) and (ii) for each node in $N^+ \cup N^-$. We need $2 \cdot (n-2) - m$ additional points $x \in P_2^{TW}$ which we obtain as follows.

For $i = 1, 2, .., m$, consider the component S_i and let $T_i \subseteq E$ be a tree spanning S_i. Choose any root node $r_i \in S_i$, and give an orientation to the $|S_i| - 1$ edges of T_i so as to obtain a directed tree (arborescence) rooted at r_i.

Then consider, in turn, the nodes $v^\alpha \in S_i \setminus \{r_i\}, \alpha \in \{+,-\}$, in a sequence visiting each node v^α after its father node in the arborescence. For each such node v^α, let $i^{-\alpha}$ be its father

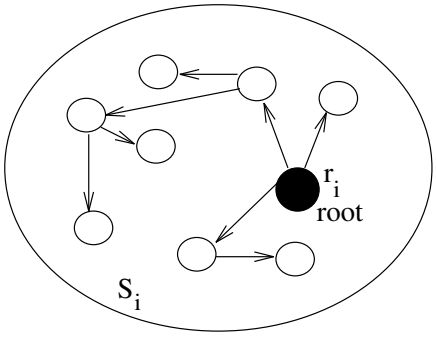


Figure 6.4.7

in the arborescence, and construct the feasible point

$$\begin{array}{rll} (1, v, i, 2, ...) & \text{if} & v^\alpha = v^+ \\ (1, i, v, 2, ...) & \text{if} & v^\alpha = v^- \\ (1, v, 2, ...) & \text{in case} & v = i. \end{array}$$

Note that this point satisfies all equations (i)–(ii), except the two equations associated with the arcs in $\delta^+(1) \cup \delta^-(2)$ corresponding to the nodes v^α and $i^{-\alpha}$ of the bipartite graph B. Because of the particular sequence in visiting the nodes of S_i, all previously constructed points satisfy the equation (i) and (ii) associated with v^α, hence the new point is affinely independent from the previous ones.

The above construction produces $|S_i| - 1$ new points for each component S_i, hence $2 \cdot (n - 2) - m$ new points in total. The thesis follows. □

We call the problem instance of the AHPPTW **2–regular**, if only the time windows for two nodes, say nodes 1 and 2, are active, $r_1 \geq M$ and $r_2 \geq M$, where M is big enough such that all nodes in $V \setminus \{1, 2\}$ can be sequenced before $\{1, 2\}$, and when its associated bipartite graph B is connected (thus implies that all the arcs $(1, k)$ and $(h, 2)$ can be part of a feasible path). In this case 6.4.20(1),(2),(3) form a minimal equation system for P_2^{TW}, i.e., we have the following corollary.

(6.4.24) Corollary.
Suppose we are given a 2–regular instance of the AHPPTW on $n \geq 4$ nodes, then

$$dim(P_2^{TW}) = |A| - 4.$$

□

6.5 Valid inequalities

As every feasible Hamiltonian path is a solution to the AHPP, we know that every valid and in particular every facet defining inequality for the asymmetric Hamiltonian path polytope is valid for the time constrained polytopes P_1^{TW} and P_2^{TW}. But in most cases these inequalities can be lifted in several ways. Not many classes of valid inequalities for the time constrained ATSP can be found in the literature. In Desrochers, Laporte [DL91] lifted versions of the Miller–Tucker–Zemlin subtour elimination constraints and lifted lower and upper bound constraints on the t–variables can be found.

If we ignore the setup costs c_{ij}, the AHPPTW is related to *single machine sequencing* or *scheduling problems*, sequencing a set of jobs V with processing time p_i for all nodes $i \in V$ on a single machine such that given time window constraints are satisfied. Balas [Bal85] introduced inequalities for this class of problems, that can be used for the AHPPTW, if node variables (as in model 1) are used. Applegate and Cook [AC91] performed computational experiments with several of these classes for the job shop scheduling problem and claimed that they might be helpful for the solution of these problems within a polyhedral approach. Note that in the job shop scheduling problem another objective, namely the minimization of the makespan, is considered.

Recall that the AHPTW–polytopes are defined as follows

$$P_1^{TW} := conv\{(x,t) \in \mathbb{R}^{A \times V} \mid (x,t) \text{ satisfies } (6.3.9)(1)\text{–}(8)\}$$

and

$$P_2^{TW} := conv\{x \in \mathbb{R}^A \mid x \text{ satisfies } (6.3.10)(1)\text{–}(6)\}.$$

In this section we summarize the inequalities valid for P_1^{TW} and P_2^{TW} known from the literature and state new classes of valid inequalities. This section is organized as follows. First we give inequalities involving x–variables only, next we list inequalities with x– and t–variables, and inequalities using only t–variables. Finally general procedures for obtaining valid cuts are described. The preliminary computational experiments showed that model 2 is superior to model 1. Therefore, many of the inequalities presented here (in particular those involving the node variables t_i) are just listed for the sake of completeness but are not yet used, and probably will not be used, in the branch&cut code.

6.5.1 Infeasible path constraints

These inequalities are derived from the fact that certain paths are infeasible, i.e., they violate a time window constraint. For a given infeasible path $P = (v_1, ..., v_k)$ the **basic version** of these inequalities is given by

$$x(P) \leq k - 2 \quad \forall \text{ infeasible paths } P = (v_1, v_2, ..., v_k).$$

There exist several possibilities to lift this inequality. In the following we discuss in detail one of these lifted inequalities, the so–called **tournament constraints**. This is followed by a list of other inequalities derived from of a lifting of the basic version.

Tournament constraints

(6.5.25) Lemma.
For all infeasible paths $P = (v_1, ..., v_k)$ the inequality

$$\sum_{i=1}^{k-1} \sum_{j=i+1}^{k} x_{v_i v_j} \leq k - 2$$

is valid with respect to P_1^{TW} and P_2^{TW}.

Proof. Note that $\sum_{i=1}^{k-1} \sum_{j=i+1}^{k} x_{v_i v_j} = k-1$ implies $x_{v_i v_{i+1}} = 1$ for $i = 1, ..., k-1$. Therefore, the inequality is valid iff $x(P) \leq k - 2$ is valid, i.e., the path P is infeasible. □

Figure 6.5.8 gives an example of a tournament inequality derived from an infeasible path on 4 nodes.

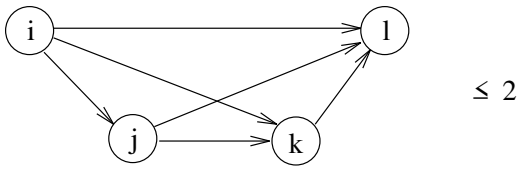


Figure 6.5.8 Tournament constraint

The **separation problem** for the tournament constraints can be solved with the help of a simple enumeration procedure. Suppose we are given a (fractional) point x^*. According to Savelsbergh [Sav94] there are only polynomially many paths P_k for which $\sum_{i=1}^{k-1} \sum_{j=i+1}^{k} x^*_{v_i v_j} - k + 2$ is greater than 0. Note that for this conclusion the assumption that $\sum_{i \in V} x^*_{ij} = \sum_{j \in V} x^*_{ij} = 1$ is necessary. These paths can easily be detected by enumeration (backtrack as soon as $\sum_{i=1}^{k-1} \sum_{j=i+1}^{k} x^*_{v_i v_j} - k + 2 \leq 0$).

Lifted tournament constraints

The tournament constraints are not very strong, as they can be further lifted in several ways, in the case when other infeasible paths through the nodes $v_1, ..., v_k$ exist. For the tournament constraint depicted in Figure 6.5.8, e.g., the variable x_{ki} might be added to the left hand side, if the path (j, k, i, l) is infeasible.

(6.5.26) Definition.
Given a node set $W \subset V$. With $\Phi[W]$ we denote a **permutation** of the nodes in W. ◇

(6.5.27) Theorem.
Let $P = (v_1, ..., v_k)$ be a path and $S := (v_2, ..., v_{k-1})$, $Q := (v_1, ..., v_{k-1})$ be two subpaths.
(a) If all paths $P' := (\Phi[Q], v_k)$ are infeasible, then the inequality

$$x(A(Q)) + x(Q : v_k) \leq k - 2$$

is valid with respect to P_1^{TW} and P_2^{TW}.

(b) If all paths $P' := (v_1, \Phi[S], v_k)$ are infeasible, then the inequality

$$x(v_1 : S) + x(A(S)) + x(S : v_k) + x_{v_1 v_k} \leq k - 2$$

is valid with respect to P_1^{TW} and P_2^{TW}.

Proof.

(a) If the condition is satisfied, all paths (Q, v_k) are infeasible. Thus $x(A(Q)) + x(Q : v_k) = k - 1$ only holds for an infeasible path.

(b) Note that $x(A(S)) \leq k - 3$ and that $x_{v_1 v_k} = 1$ implies $x(v_1 : S) = x(S : v_k) = 0$. If $x_{v_1 v_k} = 0$ holds, we know that $x(v_1 : S) + x(A(S)) + x(S : v_k) = k - 1$ only holds for infeasible paths.

□

Note that the inequality in (6.5.27)(a) is a strengthening of the subtour elimination constraint $x(A(Q)) \leq k-2$. The inequality in (6.5.27)(b) has a similar structure as the precedence forcing constraint (PFC)

$$x(j : W) + x(W : j) + x(A(W)) + x(W : i) \leq |W|$$

for a given precedence relationship $i \prec j$ (see Section 5.4.1). Note that for sets $U \subset W$ a similar PFC can be stated. This is not true for the inequality of type (6.5.27)(b) because for a subset $U \subset S$, the infeasibility of the paths $(v_1, \Phi[U], v_k)$ can no longer be guaranteed. Furthermore, it is not possible to strengthen the inequality by lifting the variables corresponding to the "reverse arcs" $(v_k : S)$ and $(S : v_1)$, as the infeasibility of all paths using these arcs cannot be guaranteed.

It is not easy to decide whether all the paths $(\Phi[Q], v_k)$ and $(v_1, \Phi[S], v_k)$ are infeasible. For this the solution of an AHPPTW on Q, resp. S, is necessary in general. But there exist conditions that imply the infeasibility of the paths and that can easily be checked. Some of these conditions are given in the following lemma.

(6.5.28) Lemma.
Let $P = (v_1, ..., v_k)$ be an infeasible path and $S := (v_2, ..., v_{k-1})$, $Q := (v_1, ..., v_{k-1})$ be two subpaths.

(a) If

$$\min_{v_i \in Q}\{r_{v_i}\} + \sum_{i=1}^{k-1} p_{v_i} + \sum_{i=1}^{k-1} \min\{c_{v_i v_j} \mid (v_i, v_j) \in A, v_j \in Q \cup \{v_k\}\} > d_k,$$

then all paths $P' := (\Phi[Q], v_k)$ are infeasible.

(b) If

$$r_{v_1} + \min\{c_{v_1 v_j} \mid (v_1, v_j) \in A, v_j \in S\} + \sum_{i=1}^{k-1} p_{v_i} + \sum_{i=1}^{k-1} \min\{c_{v_i v_j} \mid (v_i, v_j) \in A, v_j \in S \cup \{v_k\}\} > d_k$$

then all paths $P' := (v_1, \Phi[S], v_k)$ are infeasible.

Proof. By definition. □

In case that none of the conditions of Lemma (6.5.28) is satisfied, no efficient way of strengthening the tournament constraints is known. The lifted arcs can be found by enumerating all paths through P, which is computationally exhaustive for longer paths.

Generalized tournament constraints

In case that the triangle inequality on ϑ is satisfied the tournament constraint

$$\sum_{i=1}^{k-1} \sum_{j=i+1}^{k} x_{v_i v_j} \leq k-2$$

can easily be generalized using a clique lifting technique as described in Balas and Fischetti [BF93b]. We obtain the following theorem.

(6.5.29) Theorem.
Let $P = (v_1, ..., v_k)$ be an infeasible path and let $S_i \subset V, i = 1, ..., k$ be disjoint node sets containing a node of the path P, i.e., $v_i \in S_i, \forall i = 1, ..., k, S_i \cap S_j = \emptyset$ for all $i \neq j$. Assume that $\vartheta_{ij} \leq \vartheta_{ik} + \vartheta_{kj}$ holds for all $i, j, k \in V$.
Then the inequality

$$\sum_{i=1}^{k} x(A(S_i)) + \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} x(S_i : S_j) \leq \sum_{i=1}^{k} |S_i| - 2$$

is valid with respect to P_1^{TW} and P_2^{TW}, if

(i) the infeasibility of the path P is based on the fact that the time window for v_k is violated, or

(ii) the infeasibility of the path P is based on the fact that a node $v_l \notin P$ can not be covered by path P, and $v_l \notin \cup_{i=1}^{k} S_i$.

Proof. Note that the inequality can only be violated by an feasible path P', if

$$\begin{aligned} x(A(S_i)) &= |S_i| - 1 \quad \forall i = 1, ..., k, \text{ and} \\ \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} x(S_i : S_j) &= k-1 \end{aligned}$$

hold. But this implies that the infeasible path (possibly including some detours through the S_i) has to be used. □

See Figure (6.5.9)(a) for an example of a generalized tournament constraint based on an infeasible path of four nodes.

Note, that the assumption of the valid triangle inequality on ϑ is essential. Otherwise, a detour through the nodes of S_i could result in a feasible path. In the case that the triangle inequality does not hold we obtain the inequality

$$\sum_{i=1}^{k} x(A(S_i)) + \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} x_{v_i v_j} + \sum_{\substack{i=2 \\ |S_i|>1}}^{k} \sum_{\substack{j=1 \\ |S_j|>1}}^{i-1} x(v_i : S_j) \leq \sum_{i=1}^{k} |S_i| - 2.$$

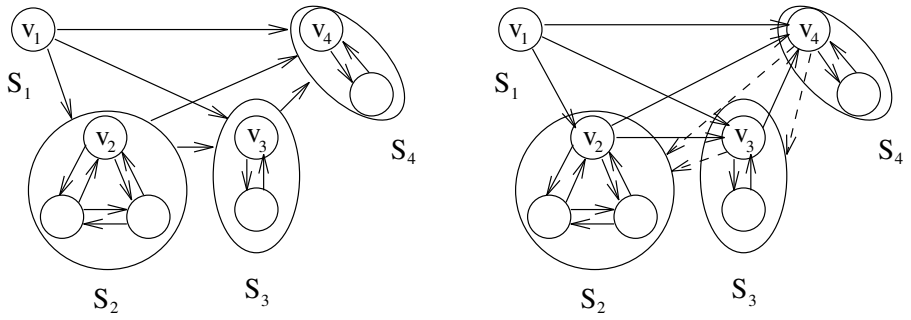


Figure 6.5.9 Generalized tournament constraints

(See Figure (6.5.9)(b) for an example of an inequality of this type.) Note, that for the case that there exist a S_i with $|S_i| \geq 2$ the validity of this inequality is not based on the infeasibility of path P. For this case, the inequality is valid for the unconstrained ATSP as well and is a generalization of the C3–inequality (see Section 1.3). To see the validity of the inequality for the ATSP, assume that there exists a feasible path H violating the inequality. It is easy to see that H has to satisfy the two trivially valid inequalities

$$\begin{aligned} \sum_{i=1}^{k} x(A(S_i)) &\leq |S_i| - 1 \qquad \text{and} \\ \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} x_{v_i v_j} + \sum_{\substack{i=2 \\ |S_i|>1}}^{k} \sum_{\substack{j=1 \\ |S_j|>1}}^{i-1} x(v_i : S_j) &\leq k - 1 \end{aligned}$$

with equality; a contradiction.

Note that for the path $P = (v_1, v_2, v_3)$ and $|S_1| = |S_3| = 1$ and $|S_2| = k$ for some $k \geq 2$, we obtain a T_k–inequality.

Concatenation of paths

It can happen that paths are feasible in themselves but that the combination of these feasible paths leads to an infeasibility.

(6.5.30) Definition.
Suppose we are given a family $\mathcal{P} = \{P_1, P_2, ..., P_k\}$ of node disjoint simple paths, $P_i \cap P_j = \emptyset, \forall i, j = 1, ..., k, i \neq j$. Let π be a permutation of the indices of $\mathcal{P}$.
The path $P = (P_{\pi(1)}, P_{\pi(2)}, ..., P_{\pi(k)})$ is called a **concatenation** of the paths in $\mathcal{P}$. $\diamond$

(6.5.31) Theorem.
Let $\mathcal{P} = \{P_1, P_2, ..., P_k\}$ be a family of node disjoint simple paths and let $[P_i]$ denote the transitive closure of path P_i. Furthermore, assume that the triangle inequality $\vartheta_{ij} \leq \vartheta_{ik} + \vartheta_{kj}$ holds for all $i, j, k \in V$. If any concatenation of the paths in $\mathcal{P}$ is infeasible, the inequality

$$\sum_{i=1}^{k} x([P_i]) \leq \sum_{i=1}^{k} |P_i| - k - 1$$

is valid with respect to P_1^{TW} and P_2^{TW}.

Proof. Let $P_i = (x^i_{v_1}, ..., x^i_{v_l})$ denote the i–th path in $\mathcal{P}$.
Note that $\sum_{i=1}^{k} x([P_i]) = \sum_{i=1}^{k} |P_i| - k$ implies $x^i_{v_j, v_{j+1}} = 1$ for all $j = 1, ..., |P_i| - 1$ in each path $P_i \in \mathcal{P}$. But at least one arc has to be forbidden as any concatenation of the paths is infeasible. $\square$

For the case $\mathcal{P} = \{P_1, P_2\}$ we show one way of a further strengthening of such an inequality.

(6.5.32) Theorem.
Let $P_1 = (v_1, v_2, ..., v_k)$ and $P_2 = (u_1, u_2, ..., u_l)$ be two node disjoint simple paths and assume that the triangle inequality $\vartheta_{ij} \leq \vartheta_{ik} + \vartheta_{kj}$ holds for all $i, j, k \in V$. If any concatenation of the paths P_1 and P_2 is infeasible, then the inequality

$$x([P_1]) + x([P_2]) + \sum_{i=1}^{k-1} \sum_{j=2}^{l} x_{v_i u_j} \leq k + l - 3$$

is valid with respect to P_1^{TW} and P_2^{TW}.

Proof. Let $m := \sum_{i=1}^{k-1} \sum_{j=2}^{l} x_{v_i u_j}$.
If $m = 0$, the inequality is valid due to Theorem (6.5.31).
Note that if $m \geq 1$, the inequalities $x([P_1]) \leq k - 1 - m$ and $x([P_2]) \leq l - 1 - m$ hold. The result follows. $\square$

In Theorems (6.5.31) and (6.5.32) the triangle inequality is required in the concatenation of the paths. If this property does not hold, it is necessary to use the shortest path between the paths in $\mathcal{P}$ in order to connect them.

In the inequalities in (6.5.31) and (6.5.32) we used the tournament constraint in each path in $\mathcal{P}$. Note that any of the infeasible path inequalities presented in the following could have been used instead.

Other infeasible path inequalities

Given an infeasible path $P = (v_1, ..., v_k)$, one possible way of strengthening the basic infeasible path constraint $x(P) \leq k-2$ to a tournament constraint $x([P]) \leq k-2$ was already presented. But there exist several other ways to lift the basic infeasible path constraint. In this section, we give some of these inequalities. Note that the validity of these inequalities is only based on the assumption of the infeasibility of path $P = (v_1, ..., v_k)$.

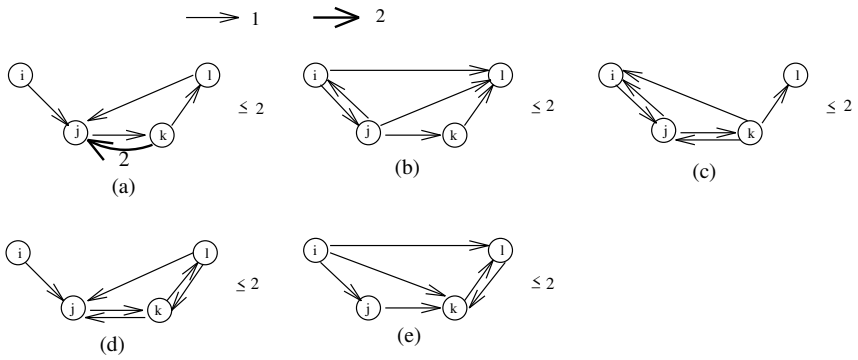


Figure 6.5.10 Lifted infeasible path constraints (1)

(6.5.33) Theorem. (Lifted path inequalities)
Let $P = (v_1, v_2, ..., v_k)$ be an infeasible path and $x(P) := \sum_{j=1}^{k-1} x_{v_j v_{j+1}}$. Then the following inequalities are valid for P_1^{TW} and P_2^{TW}:

$$(a) \quad x(P) + \; 2\sum_{j=2}^{k-2} x_{v_{k-1}v_j} + \sum_{j=2}^{k-3}\sum_{l=j+1}^{k-2} x_{v_l v_j} + \sum_{j=2}^{k-2} x_{v_k v_j} \leq k-2$$

$$(b) \quad x(P) + \; \sum_{j=1}^{k-2} x_{v_j v_k} + \sum_{j=2}^{k-2}\sum_{l=1}^{j-1} x_{v_j v_l} \leq k-2$$

$$(c) \quad x(P) + \; \sum_{j=2}^{k-1}\sum_{l=1}^{j-1} x_{v_j v_l} \leq k-2$$

$$(d) \quad x(P) + \; \sum_{j=2}^{k-1}\sum_{l=j+1}^{k} x_{v_l v_j} \leq k-2$$

$$(e) \quad x(P) + \; \sum_{j=3}^{k} x_{v_1 v_j} + \sum_{j=3}^{k-1}\sum_{l=j+1}^{k} x_{v_l v_j} \leq k-2$$

Proof.
Due to the infeasibility of the path P the inequality $x(P) \leq k-2$ holds.

(a) Assume that $m := 2\sum_{j=2}^{k-2} x_{v_{k-1}v_j} + \sum_{j=2}^{k-3}\sum_{l=j+1}^{k-2} x_{v_l v_j} + \sum_{j=2}^{k-2} x_{v_k v_j}$.
As all arcs with coefficient 2 are leaving node v_{k-1} and as there are only $k-3$ nodes entered by lifted arcs, we know that $m \leq k-2$. Due to $x(\delta^-(i)) \leq 1$, $x(\delta^+(i)) \leq 1$ and the fact that cycles are not feasible, it follows that $x(P) \leq k-2-m$. This proves the result.

(b) Assume that $m := \sum_{j=1}^{k-2} x_{v_j v_k} + \sum_{j=2}^{k-2} \sum_{l=1}^{j-1} x_{v_j v_l}$.
It is easy to see that $m \leq k-2$ and $x(P) \leq k-2-m$ hold. The result follows.

(c) Assume that $m := \sum_{j=2}^{k-1} \sum_{l=1}^{j-1} x_{v_j v_l}$.
Due to $x(\delta^-(i)) \leq 1$ and $x(\delta^+(i)) \leq 1$, we know that $m \leq k-2$ and $x(P) \leq k-2-m$ hold. The result follows.

(d) Analogous.

(e) Analogous. □

Figure 6.5.10 gives examples of these inequalities for the infeasible path $P = (i, j, k, l)$. These inequalities can also be further lifted in the case where other infeasible paths through the nodes of P exist.

For the sake of completeness and without stating the inequalities explicitly or giving an explicit proof of their validity further lifted inequalities for infeasible paths of length 4 are given in Figures 6.5.11–6.5.13. To avoid fractional coefficients all values of the variables in Figure 6.5.13 have been multiplied by 2 or 3.

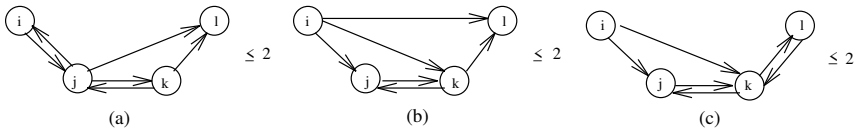


Figure 6.5.11 Lifted infeasible path constraints (2)

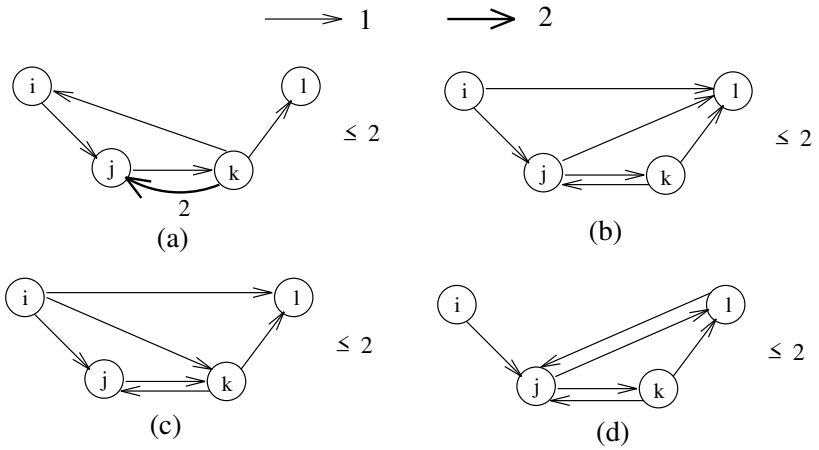


Figure 6.5.12 Lifted infeasible path constraints (3)

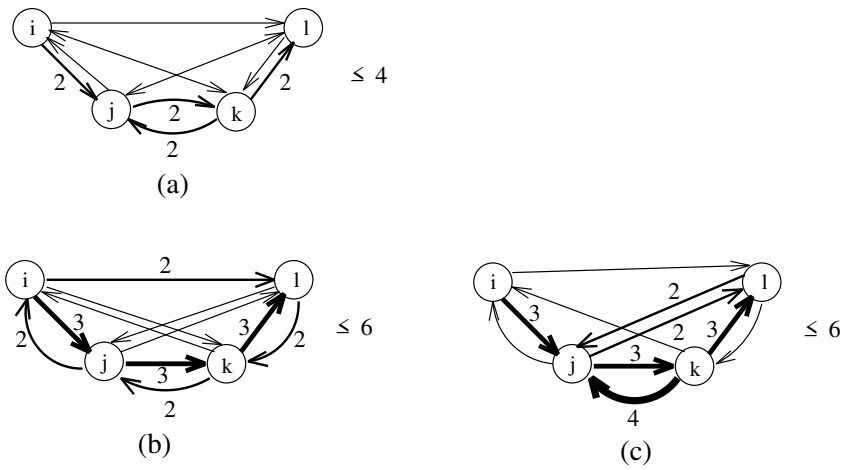


Figure 6.5.13 Lifted infeasible path constraints (4)

6.5.2 A lifting procedure

In this section, we describe a lifting procedure, called **V–lifting**, that can be used to construct new families of valid infeasible path inequalities for the AHPPTW. Assume that the triangle inequality on ϑ is satisfied, i.e., $\vartheta_{ij} \leq \vartheta_{ik} + \vartheta_{kj}$ holds for all $i, j, k \in V$.
Suppose we are given the valid infeasible path constraint $\alpha x \leq \alpha_0$ where the infeasibility of the underlying infeasible path $P = (v_1, ..., v_m)$ is based on the fact that P violates the time window for v_m. Let $\beta x \leq \alpha_0 + 1$ be an inequality such that the following conditions are satisfied:

$$\begin{array}{ll} (1) & \exists k \in V, \text{s.t. } \alpha_{ij} = \beta_{ij} \quad \forall\ (i,j) \notin \delta(k) \\ (2) & \alpha_{ij} = 0 \quad \forall (i,j) \in \delta(k) \\ (3) & \exists u, v \neq k \text{ s.t. } \beta_{uv} = \beta_{uk} = \beta_{kv} = 1 \\ & \text{and } \beta_{ij} = 0 \quad \forall\ (i,j) \in \delta(k) \setminus \{(u,k),(k,v)\} \end{array}$$

As the triangle inequality on ϑ is satisfied, the inequality $\beta x \leq \alpha_0 + 1$ is valid as well. Now, adding up the following three valid inequalities:

$$\begin{array}{lllll} \frac{1}{2} & [& \alpha x & \leq \alpha_0 &] \\ \frac{1}{2} & [& \beta x & \leq \alpha_0 + 1 &] \\ \frac{1}{2} & [& x_{uk} + x_{kv} + 2x_{uv} & \leq 2 &] \end{array}$$

we obtain

$$\frac{1}{2}\alpha x + \frac{1}{2}\beta x + \frac{1}{2}(x_{uk} + x_{kv}) + x_{uv} \leq \frac{1}{2}\alpha_0 + \frac{1}{2}\alpha_0 + \frac{1}{2} + 1$$

that is

$$\alpha x + x_{uk} + x_{kv} + x_{uv} \leq \alpha_0 + 1 + \lfloor \frac{1}{2} \rfloor$$

i.e., we add the term $x_{uk} + x_{kv} + x_{uv}$ to the left hand side and increase the right hand side by 1. Note that the coefficient of variable x_{uv} will increase by 1 when the V–lifting operation is applied. Figure 6.5.14 gives an example of a lifted inequality derived from a tournament constraint.

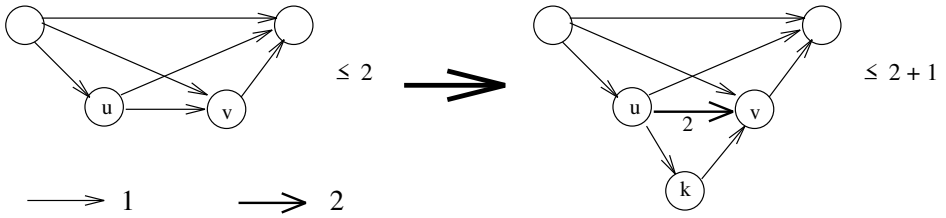


Figure 6.5.14 Lifted tournament constraint

(6.5.34) Remarks.

1. In Lemma (6.1.2) we have seen another type of infeasibility, i.e., a path $P = (v_1, ..., v_m)$ is infeasible, if a node $v_l \notin \{v_1, ..., v_m\}$ cannot be covered by P. If in the above lifting procedure $k \neq v_l$ holds, the V–lifting procedure can also be applied to this type of inequality.
2. The V–lifting procedure is in some analogy to the T–lifting for the ATSP described by Fischetti [Fis92]. Here to a given ATSP–inequality $\alpha x \leq \alpha_0$ "triangles" $x_{uk} + x_{kv} + x_{uv}$ are attached while the right hand side is increased by 1. But in contrast the inequality and the triangles "intersect" not in an arc (u, v) but only in one single node k. In the T_k–inequalities we have seen an example of a T–lifted subtour elimination constraint (SEC). Note that the T–lifting procedure cannot be applied to the infeasible path constraints, as the obtained inequality is not necessarily valid. Here the underlying inequality $\alpha x \leq \alpha_0$ must be valid for the ATSP.
3. It is easy to see that the lifting procedure as stated above is not valid for the ATSP. To see this consider $\alpha x \leq \alpha_0$ to be subtour elimination constraint $x(A(W)) \leq n - 2$ on $n - 1$ nodes. The V–lifted inequality $x(A(W)) + x_{uk} + x_{kv} + x_{uv} \leq n - 1$ is obviously not valid, since a tour is cut off.
   We conjecture that the V–lifting procedure is valid for the ATSP as well under the stronger condition

$$
\begin{aligned}
&(1') \quad \exists k \in V, \text{s.t. } \alpha_{ij} = \beta_{ij} \quad \forall\, i, j \notin \delta(k), \\
&(2') \quad \alpha_{ij} = 0 \quad \forall (i, j) \in \delta(\{k, l\}), k \neq l \\
&(3') \quad \exists u, v \neq k \text{ s.t. } \beta_{uv} = \beta_{uk} = \beta_{kv} = 1 \\
&\qquad \text{and } \beta_{ij} = 0 \quad \forall\, (i, j) \in \delta(k) \setminus \{(u, k), (k, v)\}
\end{aligned}
$$

   i.e., one node remains isolated after the lifting. A proof remains open.

6.5.3 Generalized predecessor/successor–inequality

In Chapter 5 the predecessor, successor, and predecessor–successor inequalities have been presented. These inequalities introduced by Balas et al. [BFP92] take the precedence structure among the nodes into consideration and can be strengthened for the time constrained AHPP.

Generalized (π, σ)–inequality

The (π, σ)–inequalities can be described as follows. We are given two node sets $X, Y \subset V$ such that X has to precede Y, i.e., $i \prec j$ for all pairs $i \in X, j \in Y$. Furthermore, we are given a node set $S \subset V$ such that X is contained in S and Y in its complement $\bar{S}$. In order not to violate the precedence relationships among the nodes in X and Y, a feasible path cannot cross the cut $(S : \bar{S})$ through an arc incident with $W := \pi(X) \cup \sigma(Y)$. Therefore, the (π, σ)–inequality

$$x(S \setminus W : \bar{S} \setminus W) \geq 1$$

is valid. Due to the time window restrictions, other paths from $S \setminus W$ to $\bar{S} \setminus W$ might be infeasible (cmp. Figure 6.5.15). They can also be eliminated from the left hand side.

(6.5.35) Lemma.
Given $D = (V, A)$ and a precedence digraph $P = (V, R)$. Let $X, Y \subset V$, s.t. $i \prec j \ \forall$ pairs $i \in X, j \in Y$, $W := \pi(X) \cup \sigma(Y)$. Assume that the triangle inequality on ϑ is satisfied.
Set Q to be the infeasible arc set

$$\begin{aligned} Q := \quad & \{(u, v) \in A \mid \exists i \in X, j \in Y, \text{ s.t. } (i, u, v, j) \text{ is infeasible}\} \ \cup \\ & \{(i, k) \text{ and } (k, j) \in A \mid \exists i \in X, j \in Y, \text{ s.t. } (i, k, j) \text{ is infeasible}\}. \end{aligned}$$

Then for all $S \subset V$, s.t. $X \subset S, Y \subset \bar{S}$ the inequality

$$x((S \setminus W : \bar{S} \setminus W) \setminus Q) \geq 1$$

is valid with respect to P_1^{TW} and P_2^{TW}.

Proof. For any feasible Hamiltonian path the subpath from the node of X visited last to the node of Y visited first cannot traverse any node of W without violating a precedence relationship. Furthermore, it cannot use an arc of Q without being infeasible. Thus, an arc in $(S \setminus W : \bar{S} \setminus W) \setminus Q$ has to be used. □

If $X = \{i\}$ and $Y = \{j\}$, the generalized (π, σ)–inequality is called simple. The **separation problem** for the simple generalized (π, σ)–inequalities can heuristically be solved via a separation procedure similar to the one used for the weak (π, σ)–inequalities (see Chapter 5.4.20).

Suppose we are given a fractional point x^*. Set up a capacitated LP–solution digraph $D^* = (V, A^*)$ with $(i, j) \in A^*$, if $x^*_{ij} > 0$. To each $(i, j) \in A^*$ associate a capacity $c^*_{ij} = x^*_{ij}$. For all $i \prec j$ apply the following procedure. Construct a digraph $\tilde{D} = (\tilde{V}, \tilde{A})$ from D^* by deleting

- all nodes in $\pi(i) \cup \sigma(j)$,
- all nodes k such that (i, k, j) is infeasible, i.e., $r_i + \vartheta_{ik} + \vartheta_{kj} > d_j$,
- all arcs (u, v) such that (i, u, v, j) is infeasible, i.e., $r_i + \vartheta_{iu} + \vartheta_{uv} + \vartheta_{vj} > d_j$.

If we do not succeed to send one unit of flow from i to j in $\tilde{D}$, the minimum capacity cut in $\tilde{D}$ separating i from j defines a violated simple generalized (π, σ)–inequality.

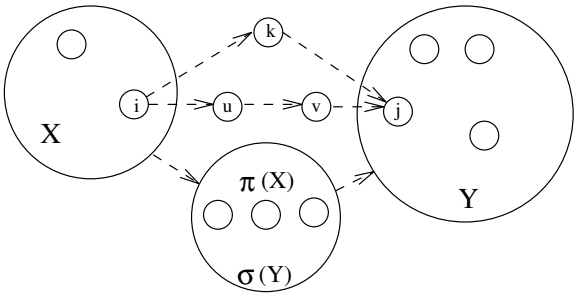


Figure 6.5.15

Generalized π–inequality

In a similar way the π–inequality (see Section 5.4.9) can be strengthened.

(6.5.36) Lemma.
Given an instance of the AHPPTW defined on the digraph $D = (V, A)$ with a node $d \in V$ such that $|\delta^+(d)| = 0$. Assume that the triangle inequality on ϑ is satisfied.
Let $S \subset V \setminus \{d\}, \bar{S} := V \setminus S$, and set Q to be the infeasible arc set

$$\begin{aligned} Q := \quad & \{(u,v) \in A \mid \exists i \in S, \text{ s.t. } (i,u,v,d) \text{ is infeasible}\} \ \cup \\ & \{(i,k) \text{ and } (k,d) \in A \mid \exists i \in S, \text{ s.t. } (i,k,d) \text{ is infeasible}\}. \end{aligned}$$

Then the inequality

$$x((S \setminus \pi(S) : \bar{S} \setminus \pi(S)) \setminus Q) \geq 1$$

is valid for P_1^{TW} and P_2^{TW}.

Proof. Let H be a feasible Hamiltonian path. Let h be the node of S visited last. Note that neither h nor the successor of h can be in $\pi(S)$. Furthermore, H cannot leave S via an arc contained in Q without being infeasible. □

In analogy to the generalized (π, σ)–inequalities, an heuristic separation routine can be designed for the generalized π–inequalities. For all $i \in V$ perform the following operations. First delete all nodes in $\pi(i)$ from D^*; next delete all nodes k such that (i,k,d) is infeasible, and then delete all arcs (u,v) from D^* such that (i,u,v,d) is infeasible. If it is not possible to send a flow of value one from i to d, the minimum capacity cut in D^* that separates i from d defines a violated generalized π–inequality.

Generalized σ–inequality

(6.5.37) Lemma.
Given an instance of the AHPPTW defined on the digraph $D = (V, A)$ with a node $s \in V$ such that $|\delta^-(s)| = 0$. Assume that the triangle inequality on ϑ is satisfied.
Let $S \subset V \setminus \{s\}, \bar{S} := V \setminus S$, and set Q to be the infeasible arc set

$$\begin{aligned} Q := \quad & \{(u,v) \in A \mid \exists j \in S, \text{ s.t. } (s,u,v,j) \text{ is infeasible}\} \ \cup \\ & \{(s,k) \text{ and } (k,j) \in A \mid \exists j \in S, \text{ s.t. } (s,k,j) \text{ is infeasible}\}. \end{aligned}$$

Then the inequality

$$x((\bar{S} \setminus \sigma(S) : S \setminus \sigma(S)) \setminus Q) \geq 1$$

is valid for P_1^{TW} and P_2^{TW}.

Proof. Let H be a feasible Hamiltonian path. Let h be the node of S visited first. Note that neither h nor the predecessor of h can be in $\sigma(S)$. Furthermore, H cannot enter S via an arc contained in Q without being infeasible. □

The separation routine for the generalized σ–inequalities is similar to the one for the generalized π–inequalities. In contrast to the latter routine the nodes in $\sigma(i)$, the nodes k corresponding to infeasible paths (s, k, i) and the arcs corresponding to infeasible paths (s, u, v, i) are deleted. The flow is sent from s to i and the minimum capacity cut that separates s from i defines a violated simple generalized σ–inequality.

6.5.4 Strengthening of the MTZ–inequalities

In Section 6.3.1 the MTZ–subtour elimination constraints for the AHPPTW have been introduced,i.e.,

$$t_i + \vartheta_{ij} - (1 - x_{ij}) \cdot M_{ij} \leq t_j \tag{6.5.38}$$

Taking the reverse arcs $(j, i) \in A$ and infeasible arc combinations into account Desrochers and Laporte [DL91] observed that these inequalities can be lifted. In case that precedences get involved the MTZ–inequalities can even be further strengthened.

Reverse arcs

(6.5.39) Theorem. (Lifted MTZ–inequalities)(Desrochers, Laporte, 91)
Let $a_{ji} := \max\{\vartheta_{ji}, r_i - d_j\}$, $M_{ij} \geq d_i + \vartheta_{ij} - r_j$. Then for all $i, j = 1, ..., n, i \neq j$ the inequality

$$t_i + \vartheta_{ij} - (1 - x_{ij}) \cdot M_{ij} + (M_{ij} - \vartheta_{ij} - a_{ji}) \cdot x_{ji} \leq t_j$$

is valid for P_1^{TW}.

Proof. See [DL91]. □

Infeasible paths

These inequalities can still be further strengthened by taking infeasible arc combinations into account. For the sake of simplicity we give an example for the case where the path (k, i, j) is infeasible.

(6.5.40) Lemma. (Desrochers, Laporte, 91)
Let $i, j, k \in V$ such that $r_k + \vartheta_{ki} + \vartheta_{ij} > d_j$. Let $a_{ji} := \max\{\vartheta_{ji}, r_i - d_j\}$, $b_{ki} \leq M - c_{ij} - \min\{d_k + c_{ki}, d_i\}$ and $M \geq \max_{ij}\{c_{ij} + c_{ji}\}$. Then the inequality

$$t_i + \vartheta_{ij} - (1 - x_{ij}) \cdot M + (M - \vartheta_{ij} - a_{ji}) \cdot x_{ji} + b_{ki} x_{ki} \leq t_j$$

is valid for P_1^{TW}.

Proof. See [DL91]. □

Precedences

The standard MTZ–inequality (6.5.38) can be strengthened in case that a precedence relationship $i \prec j$ is involved. Therefore, assume $i \prec j$. As i must be scheduled before j, we have that $t_i \leq t_j$ and, even more, that the inequality

$$t_i + \vartheta_{ij} x_{ij} \leq t_j \tag{6.5.41}$$

is valid. But this inequality is dominated when $d_i + \vartheta_{ij} \leq r_j$, since

$$t_i + \vartheta_{ij} x_{ij} \leq d_i + \vartheta_{ij} x_{ij} \leq d_i + \vartheta_{ij} \leq r_j (\leq t_j)$$

is a valid inequality.

So far, only the case where j is sequenced directly after i has been considered in the MTZ–inequalities. It could happen that one or more nodes are sequenced between i and j. We now give a strengthening of the MTZ–inequality taking possible nodes k into account that are sequenced in between.
As $\sum_{j\in V} x_{ij} = \sum_{i\in V} x_{ij} = 1$, the inequality (6.5.41) can be further strengthened, since

$$t_i + \vartheta_{ij}x_{ij} + \sum_{k\neq i,j} (\omega_{ik} + \vartheta_{kj})x_{kj} \leq t_j \tag{6.5.42}$$

is a valid inequality.

6.5.5 Strengthening of the bounds on the t-variables

Desrochers and Laporte [DL91] observed that the bounds on the t–variables $r_i \leq t_i \leq d_i$ can also be lifted, taking other arc combinations into account.

(6.5.43) Theorem. (Lifted t–bounds)(Desrochers, Laporte, 91)
Let $a_{ji} := \max\{0, r_j - r_i + \vartheta_{ji}\}$ and $b_{ij} := \max\{0, d_i - d_j + \vartheta_{ij}\}$. Then the inequalities

$$\begin{array}{lll} (i) & r_i + \sum_{\substack{j=1\\ i\neq j}}^{n} a_{ji}x_{ji} & \leq t_i \quad \forall\ i \in V \\ (ii) & d_i - \sum_{\substack{j=1\\ i\neq j}}^{n} b_{ij}x_{ij} & \geq t_i \quad \forall\ i \in V \end{array}$$

are valid for P_1^{TW}.

Proof. See [DL91]. □

6.5.6 Inequalities based on t–variables

There is a relation between the AHPPTW and one–machine scheduling problems with time windows. This problem is to sequence n jobs on a single machine subject to a given set of time windows while minimize a certain objective function. Balas [Bal85] and Dyer and Wolsey [DW90] considered the case where only release dates are present. The inequalities they derived can be used for the AHPPTW too. In the following we give two examples. So far, these inequalities are not used in the branch&cut code.

In scheduling problems, very often only processing times p_i but no set–up costs c_{ij} are present. Therefore, it is helpful to construct an equivalent problem instance where the processing times are increased and the cost coefficients are decreased as much as possible. To calculate the amount by which the processing times, resp. the cost coefficients, might be changed, the following assignment problem is solved:

$$\begin{array}{lll} & \min c^T x & \\ \text{s.t. } (1) & \sum_{i\in V} x_{ij} & = 1 \quad \forall j \in V \\ (2) & \sum_{j\in V} x_{ij} & = 1 \quad \forall i \in V \\ (3) & x_{ij} & \geq 0 \quad \forall (i,j) \in A \end{array}$$

assuming that $c_{ii} = \infty \; \forall \; i \in V$. On the other hand, consider the dual problem

$$\begin{array}{lll} & \max \sum_{i\in V}(u_i + v_i) & \\ \text{s.t.} & c_{ij} - u_i - v_j & \geq 0 \quad \forall i,j \in V. \end{array}$$

Update now processing time and cost coefficients by

$$\begin{aligned} \tilde{p}_i &:= p_i + u_i + v_i, \\ \tilde{c}_{ij} &:= c_{ij} - u_i - v_j. \end{aligned}$$

Alternatively, one can proceed heuristically: Subtract the biggest value a_i, resp. b_i, from each row, resp. column, in the cost matrix $C = (c_{ij})_{\substack{i=1,\ldots,n \\ j=1,\ldots,n}}$ such that all c_{ij} remain nonnegative and set

$$\begin{aligned} \tilde{p}_i &:= p_i + a_i + b_i, \\ \tilde{c}_{ij} &:= c_{ij} - a_i - b_j. \end{aligned}$$

Basic cuts

This class of inequalities has been introduced by Dyer and Wolsey [DW90].
Let $S \subset V$ be a subset of the jobs to be processed. By considering the order in which the jobs in S are scheduled, it is easy to check that the inequality

$$\sum_{j\in S} \tilde{p}_j \cdot t_j \geq \min_{j\in S}\{r_j\} \cdot \sum_{j\in S} \tilde{p}_j + \sum_{i\neq j\in S} \tilde{p}_i \cdot \tilde{p}_j$$

is satisfied by all possible sequences of the jobs.

Two–job cuts

This class of inequalities has been introduced by Balas [Bal85]. It is a sharpening of the basic cut for a pair $\{i, j\}$.

Suppose $r_j < r_i + \tilde{p}_i$ and $r_i < r_j + \tilde{p}_j$. The two–job cut involving only processing times can be written as

$$(\tilde{p}_i + r_i - r_j) \cdot t_i + (\tilde{p}_j + r_j - r_i) \cdot t_j \geq \tilde{p}_i \cdot \tilde{p}_j + r_i \cdot \tilde{p}_j + r_j \cdot \tilde{p}_i$$

(see [AC91]). By verifying that the inequality holds for the earliest possible starting times of the two possible orderings of i and j, it is easy to see that the inequality is valid.

Balas [Bal85] already stated a sharpened version of the two–job cut involving order dependent processing times. This inequality can be written as

$$(\omega_{ij} + r_i - r_j) \cdot t_i + (\omega_{ji} + r_j - r_i) \geq \omega_{ij} \cdot \omega_{ji} + r_i \cdot \omega_{ji} + r_j \cdot \omega_{ij}.$$

Violated inequalities of this class can be found by enumeration of all i and j satisfying $r_j < r_i + \tilde{p}_i$ and $r_i < r_j + \tilde{p}_j$.

6.5.7 General cutting planes

Clique cuts

We now present a general scheme for obtaining valid cuts that is due to Balas [Bal85]. We present it for the more general case where node and arc variables are present. This concept is based on the enumeration of all feasible solutions on a small node set. Therefore, suppose, we are given a set $S \subset V$ of small cardinality, and let (x^*, t^*) be a fractional (LP) solution. The problem is to find a (most violated) inequality $\alpha x + \beta t \leq \gamma$ with support on S, i.e.,

$$\begin{array}{lll} \alpha_{ij} & \geq 0 & \forall\, i,j \in S, \\ \alpha_{ij} & = 0 & \forall\, i \notin S \text{ or } j \notin S, \\ \beta_j & \geq 0 & \forall\, j \in S, \\ \beta_j & = 0 & \forall\, j \notin S, \end{array}$$

and which is violated by (x^*, t^*).

Let $\sigma := \alpha x + \beta t - \gamma$ denote the degree of violation. Now, the following linear program determines a valid cut:

$$\begin{array}{llll} & & \max \sigma & \\ (6.5.44) & \text{s. t. } (1) & \sum \alpha_j + \sum \beta_j & = 1 \\ & (2) & \alpha x^{(k)} + \beta t^{(k)} - \gamma & \leq 0 \quad \forall \;\; k = 1, ..., \mathcal{K} \end{array}$$

where (1) is a normalization condition and $(t^{(1)}, x^{(1)}), ..., (t^{(\mathcal{K})}, x^{(\mathcal{K})})$ correspond to all the feasible solutions on S to our problem. With the help of this LP we determine an inequality that maximizes the degree of violation and is satisfied by all feasible solutions.

If the number of feasible solutions on the set S is too big to be added explicitly to the LP (6.5.44), this problem can be solved by **row generation**: Start with a triple (α, β, γ) and see whether there exists a k, s.t. $\alpha x^{(k)} + \beta t^{(k)} - \gamma > 0$; if yes, add the corresponding inequality to the LP, solve the LP, and proceed further.

General clique cuts

Let $D^* = (V, A^*)$ be the LP–solution support digraph, such that $(i,j) \in A^*$, if $x^*_{ij} > 0$. The cuts presented in this section were motivated by the fact that (at least for the problem instances we consider) D^* very often does not contain a feasible solution. If this is the case, we have that

$$x(A^*) \leq n - 2 \tag{6.5.45}$$

is a valid cut. As D^* is usually very sparse, an enumeration procedure similar to one for the separation of the tournament constraints (see Section 6.5.1) might be used to check if D^* contains a feasible solution (backtrack as soon as the path is infeasible).

If D^* does contain feasible solutions, but only those that are not better than the best one known so far, the inequality (6.5.45) can also be used. But note that this inequality might cut off feasible points but not the optimal solution.

A similar argumentation can be applied for smaller node sets $S \subset V$. If in D^* no feasible solution exists on S, the inequality

$$x(A^*(S)) \leq |S| - 2$$

is a valid cut.

6.6 The Branch&Cut Algorithm

Based on the general concept presented in Chapter 1.2 two branch&cut algorithms for the AHPPTW were implemented. One implementation is based on the model with the extra node variables and the big M (model 1, see Section 6.3.1), the other on arc variables only (model 2, see Section 6.3.2). One of the main aims of these implementations is to compare the computational performance of these two models (see also Section 6.7). In this section we briefly sketch some of the implementation details.

Problem enlargement

Note that if $\pi(i) \neq \emptyset$ (resp. $\sigma(i) \neq \emptyset$), the degree constraint $x(\delta^-(i) \leq 1$ (resp. $x(\delta^+(i) \leq 1)$ turns to be an equation. Furthermore, several separation routines require the existence of a fixed starting and ending node of a feasible Hamiltonian path. As it is more convenient for the implementation, the problem instances are enlarged by two nodes, a source node s and destination node d, and arcs $\{(s,j) \mid j \in V \setminus \{s,d\}, \pi(j) = \emptyset\} \cup \{(i,d) \mid i \in V \setminus \{s,d\}, \sigma(i) = \emptyset\}$ with cost coefficients zero. Furthermore, precedences are added such that s is always the starting node and d is always the final node of any feasible Hamiltonian path.

Preprocessing

The preprocessing reductions described in Section 6.2 are repeated iteratively until they fail. The order in which the tests are applied is the following:

(1) Construction of precedences (see Section 6.2.2).

(2) Tightening of the time windows; the conditions are checked in the order (6.2.3), (6.2.4), (6.2.5), (6.2.6).

(3) Elimination of arcs $(i,j) \in A$ with $|Q| = 1, 2$ (see Section 6.2.3). The elimination procedure for $|Q| = 2$ is called only if all other tests failed.

Heuristics

In the current version of the branch&cut code only very simple primal heuristics are used to obtain feasible solutions. Before entering the cutting plane phase **initial heuristics** are called in order to obtain a feasible starting sequence. The following procedures are applied:

(1) Check if the trivial sequence $(1, 2, 3, ..., n-1, n)$ is feasible.

(2) Sort the nodes due to increasing release dates and check if this sequence is feasible.

(3) Sort the nodes due to increasing deadlines and check if this sequence is feasible.

(4) Sort the nodes due to increasing midpoints of the time windows $m_i := r_i + \frac{d_i - r_i}{2}$ and check if this sequence is feasible.

(5) Apply a nearest feasible neighbor heuristic starting with each node.

The best solution found by any of the procedures is the initial feasible solution.

In order to make use of the information contained in the current LP–solution x^* an **LP–exploitation heuristic** is run at each iteration of the branch&cut algorithm. Therefore, the LP–solution digraph $D^* = (V, A^*)$ is constructed where $(i,j) \in A^*$, if and only if $x^*_{ij} > 0$. In D^* all feasible solutions are enumerated. It is possible to backtrack, as soon as the path

becomes infeasible or the cost of the path is higher than the cost of the best feasible solution so far. The computational results showed that D^* very seldomly contains a feasible solution.

We are aware of the fact, that, especially in the initial step, more sophisticated heuristics are required to improve the efficiency of the implementations.

Initial LP

The LP is initialized with the variables corresponding to the 5–nearest–neighbours–digraph and to the best feasible solution determined by the initial heuristic. For model 1 all node variables t_i are added, too. We start with the equations

$$\begin{aligned} x(\delta^-(i)) &= 1 \quad \forall i \in V \cup \{d\} \\ x(\delta^+(i)) &= 1 \quad \forall i \in V \cup \{s\}. \end{aligned}$$

In the implementation of model 1 for all variables x_{ij} the MTZ–inequalities (6.5.38), resp. (6.5.42) in case that $i \prec j$, are added, too.

Separation strategy

The separation routines are called in the following order:

(1) Subtour elimination constraints.
(2) Pool Separation.
(3) π–inequalities.
(4) σ–inequalities.
(5) (π, σ)–inequalities.
(6) Infeasible path constraints.
(7) Lifted 3–cycles.
(8) General cuts $x(A^*) \leq n - 2$ (if used).

Whenever one of the procedures generates a cutting plane all subsequent routines (except (8)) are skipped.

If the number of inequalities in the pool is large, it is computational exhaustive to check if all these inequalities are satisfied by the current LP–solution x^*. Therefore, inequalities for which exact and fast separation routines exist, such as subtour elimination constraints, are never separated through the pool. At the moment being it is only checked if the infeasible path constraints (6.5.27)(a), (6.5.27)(b), and (6.5.32) are satisfied.

The separation of the infeasible path constraints consists of several steps. Our computational experience showed that once an infeasible path constraint was generated the cutting plane algorithm often tries to react to the generated cut by keeping the structure of this forbidden low–cost path by taking a short "detour" or a "short cut". Note, that a "short cut" might result in a feasible path. In the heuristic separation procedure it is checked first if a minor modification, in fact only an enlargement, of an already generated infeasible path constraint is violated.

If no violated inequality is found, the enumerative separation procedure for tournament constraints (see Section 6.5.1) is run. If a violated tournament constraint is found, it is checked, if it can be lifted to an inequality of type 6.5.27(a)(b). If yes, the lifted inequality is added, otherwise the tournament constraint.

If no cut is found, it is verified if a concatenation of paths creates an infeasibility. So far,

only a simple procedure is used in which paths corresponding to variables with value 1 are concatenated.

Nodes i, j, k violating the lifted 3–cycle inequality $x_{ij}+x_{jk}+x_{ki}+2\cdot x_{ji} \leq 2$ are determined by enumeration.

As the LP–exploitation heuristic is based on an enumeration of all feasible path in $D^* = (V, A^*)$, the information if the inequality $x(A^*) \leq n-2$ is valid is got almost "for free". But these cuts are not very strong and very dense. Therefore, the decision whether to branch or not is not taken on the detection of such cuts. There exist a flag indicating if these cuts should be used or not and the cut is added only if another cutting plane has been generated. The computational results in Section 6.7 (see Table 6.7) show that these cuts are not useful in solving the given problem instances to optimality.

Further implementation details

Branching: Branching is performed on x–variables only. The branching variable x_{ij} is chosen to be the one closest to 0.5. If there exist several such variables, the one with highest cost coefficient c_{ij} is chosen. Other branching strategies like

- branch on variables that occur in the best solution so far,
- branch on variables (arcs) that are incident with nodes with small in– or outdegree in the input digraph,
- branch on variables that have a small in– and outdegree in the LP–solution digraph,
- branch on the fractional variables with an highest cost coefficient,

etc., should be tested.

Enumeration strategy: As enumeration strategy Depth–First–Search is applied. For the TSP it is known that this strategy does not always give the best results (see [JRT92]). Therefore, the implementation of other strategies like Best–First–Search and Breadth–First–Search is necessary.

Pricing frequency: Nonactive variables are priced out after each iteration.

Tailing off: If within one node of the branch&cut tree cuts are added, but the increase in the objective function is not sufficiently large enough, a "tailing off" of the cutting plane algorithm is detected and a branching step is performed. This is done if the improvement over the last k_i LPs is less than $p_i\%$ a branching step is performed. In the current implementation the values $k_1 = 20, p_1 = 1\%$ and $k_2 = 10, p_2 = 0\%$ are used. This means that we branch, if over the last 10 LPs no improvement or over the last 20 LPs only an improvement of 1% has been achieved.

6.7 Computational results

The branch&cut codes were implemented in C on a SUN SPARC Station 10. The implementations are based on a branch&cut framework developed and implemented by Michael Jünger. (Note that the advanced branch&cut framework ABACUS, that was used for the SOP, only supports 0/1–variables and could therefore not be used for model 1!) The subtour elimination and 2–matching separation routines were implemented by Michael Jünger, too. As LP–solver we used CPLEX 2.2 .

Two sets of data were tested. The problem instances of the first set (*rbg...*) were derived from the stacker crane application described in Section 4. The second class consists of randomly generated problem instances based on the TSPLIB instance *br17.atsp* (see [Rei91]) and on problem instance *rbg27a*.

We wanted to get an idea which influence the number of nodes with active time windows and their width will have on the computational difficulty of the problem instance. Therefore, we constructed several **random instances**.

First, we generated time windows and processing times to the cost matrix *br17.atsp*. These values were taken randomly from fixed intervals (e.g., for the instances in *br17.a.m*, $1 \leq m \leq n$ the time window width is taken from the interval $[70, 140]$ and processing times from the interval $[30, 80]$, see Table 6.1). As a reference path we have chosen the second–nearest–neighbour Hamiltonian path. The times for visiting each node in this path were chosen to be the midpoints of the time windows. We generated a random ordering O of the nodes. For problem instance *br17.i.m* of class i only the time windows of the first $m, 0 \leq m \leq n$, nodes of O are active, the time windows for all other nodes are relaxed to $[0, \infty)$. Thus, *br17.i.0* corresponds to the pure ATSP instance (enlarged by a source and destination node) and the difference between instances *br17.i.(m-1)* and *br17.i.m* is that the time window for node $O(m)$ is activated.

Second, we kept the cost matrix of *rbg27a* and the interval for the processing times fixed and varied the interval for the time window width. Beside that the same procedure as described above is applied to obtain the different problem instances *rbg27.i.m*.

In Table 6.1 the CPU–times required to solve these random instances to optimality are given. These results are based on model 2. More detailed tables can be found in the appendix.

We observe that the problem instances are easily solvable if only a few time windows are active ($m < \frac{n}{6}$). The most difficult instances are in the range $\frac{n}{4} \leq m \leq \frac{n}{2}$, whereas the instances for $m > \frac{2}{3} \cdot n$ are easily solvable again. For the polyhedral approach both the instances with small and large time window width (*rbg27.i.m*, $i \in \{a, d, e, f\}$) can be considered to be easier than the instances with "medium sized" time windows (*rbg27.i.m*, $i \in \{b, c\}$).

This collection of problem instances is definitely not large enough to make final conclusions. But we think that the problem instances for $m = 1, 2$ are easy because they are "more or less" AHPP instances for which it is known that a polyhedral approach can successfully be applied to solve these instances to optimality. We think that the instances for $m > \frac{2}{3} \cdot n$ are easier than those for smaller m because the preprocessing already reduces the number of variables and the precedence constraints and the infeasible paths give a certain structure to the problem instance. Thus, there are not that many "degrees of freedom" for the solutions. But further tests with will be necessary to obtain a better understanding of this behaviour.

In the following results are given only for the real–life instances *rbg...* and for one class of

randomly generated data, namely *br17.a.m.* The tables for the other randomly generated instances can be found in the appendix.

In Table 6.2 the effect of the preprocessing procedure is summarized. We observe that the number of arcs was reduced by approx. 50 %. Note that the sum of columns $|R1|, |R2|$, Fix1, and Fix2 can be larger than the difference between $|A_n| - |A|$, as after the fixing of variable x_{ij} to 0 in a following iteration it can be detected that $j \prec i$. Only for the real life instances a huge number of time window tightenings could be performed. The most effective tightening is the one due to (6.2.3).

Tables 6.3 and 6.4 give a short summary of the computational results achieved with the two implementations. First observe that half of the small instances (< 30 nodes) could be solved to optimality at the root and that all other small instances were solved in a reasonable amount of time. The branch&cut trees are moderately sized. If the problem instance was not solved to optimality in the root, the GAP at the root ($\frac{opt-lb}{lb}$) is in most cases less than approx. 6% for instances *rbg..* and less than 10-20% for instances *br17.a.m.* The use of further separation routines will close this gap.

If we compare the computational performance of the two models, we observe that for most instances the use of model 1 results in less LPs that had to be solved (see column # LPs) and in smaller branch&cut trees. But despite of this fact the computing times are higher (see column CPU). This confirms that model 2 is superior to model 1 from a computational point of view.

In Tables 6.5 and 6.6 it is summarized which cuts were generated. The pure ATSP constraints (subtour elimination, 2–matching, lifted 3–cycles) only play a minor role. Moreover, only a few cuts were generated from the pool. Most generated cuts are based on the precedence relationships or on infeasible paths.

Influence of the number of active time windows

	x = (a)	x = (b)	x = (c)	x = (d)	x = (e)	x =(f)
tw-width	[70, 140]	[35, 70]	[70, 140]	[35, 70]	[10 150]	[100,150]
$\mathbf{p}_i$	[30, 80]	[15, 40]	[15, 40]	[30, 80]	[15, 40]	[15, 40]
br17.x.01	0:01.03	0:00.62	0:00.65	0:00.67	0:00.70	0:00.63
br17.x.02	0:00.85	0:00.78	0:01.52	0:00.67	0:06.13	0:54.12
br17.x.03	0:02.45	25:34.73	6:19.08	0:46.67	2:50.68	0:01.48
br17.x.04	13:42.65	8:25.32	0:01.98	0:46.67	0:01.52	0:02.42
br17.x.05	84:00.42	5:52.40	0:01.63	0:08.50	1:39.80	0:04.53
br17.x.06	89:43.17	6:09.52	0:00.87	0:02.87	0:01.62	0:05.40
br17.x.07	77:18.20	1:15.62	1:43.23	3:08.37	8:43.35	0:03.30
br17.x.08	27:18.40	1:21.75	0:00.70	0:59.18	4:22.30	0:04.52
br17.x.09	0:38.62	0:05.68	0:02.65	0:02.93	0:01.83	0:20.65
br17.x.10	1:44.63	0:03.25	0:01.55	0:00.97	0:13.08	0:03.85
br17.x.11	0:54.47	0:01.08	0:00.82	0:00.75	0:42.38	0:03.27
br17.x.12	1:23.72	0:00.87	0:00.82	0:04.37	0:28.67	0:01.78
br17.x.13	0:25.03	0:09.83	0:01.07	0:01.53	0:01.55	0:01.15
br17.x.14	0:01.78	0:00.72	0:00.63	0:00.62	0:01.18	0:01.87
br17.x.15	0:02.67	0:00.47	0:00.55	0:00.63	0:00.73	0:00.62
br17.x.16	0:00.53	0:00.43	0:00.48	0:00.37	0:00.43	0:00.57
br17.x.17	0:00.43	0:00.28	0:00.53	0:00.32	0:00.43	0:00.52

	x = (a)	x = (b)	x = (c)	x = (d)	x = (e)	x = (f)
tw-width	[10, 30]	[30, 50]	[50, 70]	[70, 90]	[90 110]	[110, 130]
$\mathbf{p}_i$	[15, 40]	[15, 40]	[15, 40]	[15, 40]	[15, 40]	[15, 40]
rbg27.x.01	0:01.87	0:01.47	0:01.37	0:01.68	0:01.43	0:01.33
rbg27.x.02	0:02.08	0:04.63	0:01.58	0:02.43	0:09.88	0:02.37
rbg27.x.03	0:02.42	14:43.22	0:02.35	0:23.90	4:17.08	0:01.43
rbg27.x.04	0:02.00	30:35.48	0:14.78	14:06.63	29:03.77	3:47.27
rbg27.x.05	0:07.27	145:16.18	126:47.22	78:38.15	31:30.08	0:05.52
rbg27.x.06	0:11.90	145:21.18	132:56.93	105:52.32	296:59.02	0:05.33
rbg27.x.07	15:31.18	192:59.80	125:15.77	16:18.07	26:12.70	0:08.63
rbg27.x.08	0:40.65	37:46.88	143:04.85	18:21.47	27:32.85	14:21.73
rbg27.x.09	1:55.68	86:31.70	282:06.47	60:13.47	29:38.80	1:39.48
rbg27.x.10	1:46.63	18:51.37	47:07.07	5:38.75	35:47.53	72:44.03
rbg27.x.11	2:07.73	13:36.68	52:00.05	2:00.78	16:12.08	74:47.43
rbg27.x.12	0:43.33	0:53.67	31:35.33	2:14.38	4:24.17	48:35.70
rbg27.x.13	1:05.65	0:42.10	34:56.53	3:27.47	2:58.13	18:52.82
rbg27.x.14	0:41.97	0:11.82	12:54.52	0:09.60	0:04.37	16:06.28
rbg27.x.15	9:07.85	0:48.65	2:55.93	0:10.47	0:03.93	12:16.13
rbg27.x.16	1:00.88	0:08.15	3:12.05	0:09.30	0:02.70	4:26.95
rbg27.x.17	0:37.33	0:05.13	2:47.28	0:06.72	1:15.88	0:18.05
rbg27.x.18	0:03.27	0:04.18	0:10.60	0:04.82	0:45.83	0:08.27
rbg27.x.19	0:03.32	0:13.18	0:06.82	0:04.90	0:18.35	0:08.85
rbg27.x.20	0:01.75	0:02.90	0:16.52	0:04.42	0:20.87	0:05.58
rbg27.x.21	0:01.37	0:02.60	0:11.37	0:10.48	0:07.37	0:04.73
rbg27.x.22	0:01.10	0:01.70	0:03.08	0:04.13	0:03.08	0:06.07
rbg27.x.23	0:01.07	0:01.62	0:01.92	0:02.87	0:04.13	0:06.00
rbg27.x.24	0:01.05	0:01.32	0:02.90	0:02.55	0:02.20	0:02.83
rbg27.x.25	0:01.13	0:01.08	0:02.72	0:01.53	0:01.72	0:02.52
rbg27.x.26	0:01.35	0:01.62	0:02.13	0:02.75	0:01.47	0:02.03
rbg27.x.27	0:01.02	0:01.03	0:01.30	0:01.17	0:01.22	0:01.75

Table 6.1. Computing time for random instances

Preprocessing

	A_n	Iter.	$\|R1\|$	$\|R2\|$	TW1	TW2	TW3	TW4	Fix1	Fix2	$\|A\|$
rbg10a	110	2	27	29	0	3	0	0	0	0	54
rbg10b	110	10	32	19	25	0	5	0	30	0	41
rbg16a	272	3	94	98	0	3	4	0	0	1	79
rbg16b	272	3	54	49	0	1	1	0	0	2	167
rbg17a	306	39	62	26	235	0	4	0	74	5	174
rbg19a	380	3	154	153	0	1	3	0	0	2	71
rbg19b	380	2	90	79	0	2	1	0	0	0	211
rbg19c	380	39	74	36	245	0	4	0	72	0	229
rbg20a	420	32	158	47	141	0	10	0	198	10	95
rbg27a	756	46	142	61	438	0	3	0	92	15	487
rbg48a	2352	105	577	360	1639	0	6	0	247	0	1288
rbg49a	2450	114	734	580	1842	1	5	0	102	1	1083
rbg50a	2550	3	485	429	0	3	2	0	0	7	1629
rbg50b	2550	115	732	591	1964	1	4	0	102	0	1175
rbg50c	2550	108	611	450	1604	0	4	0	172	0	1396
br17.a.00	306	1	0	0	0	0	0	0	0	0	306
br17.a.01	306	1	0	0	0	0	0	0	0	0	306
br17.a.02	306	2	1	2	0	0	0	0	4	0	299
br17.a.03	306	2	3	5	0	0	0	0	4	0	294
br17.a.04	306	2	6	9	0	0	0	0	8	0	283
br17.a.05	306	8	31	12	14	0	0	0	36	5	244
br17.a.06	306	7	34	18	11	0	0	0	39	9	226
br17.a.07	306	6	38	25	9	0	0	0	36	9	216
br17.a.08	306	6	43	33	8	0	0	0	34	6	206
br17.a.09	306	6	49	42	7	0	0	0	37	8	184
br17.a.10	306	6	55	49	6	0	0	0	27	7	180
br17.a.11	306	6	63	60	5	0	0	0	25	4	164
br17.a.12	306	6	72	72	4	0	0	0	29	7	134
br17.a.13	306	13	92	84	13	0	0	0	31	20	96
br17.a.14	306	13	101	98	11	0	0	0	16	18	86
br17.a.15	306	13	111	113	9	0	0	0	7	14	70
br17.a.16	306	9	121	129	3	1	1	0	3	6	51
br17.a.17	306	9	133	146	6	30	8	17	0	1	26

Table 6.2: Preprocessing

$\|A_n\|$	:	Number of arcs in input digraph
Iter.	:	Number of preprocessing loops
$\|R1\|$	:	Number of precedence relationships among the nodes (including transitively derived relationships)
$\|R2\|$	:	Number of eliminated transitive relationships arcs
TW1	:	Number of release date adjustments due to (6.2.3).
TW2	:	Number of release date adjustments due to (6.2.4).
TW3	:	Number of deadline adjustments due to (6.2.5).
TW4	:	Number of deadline adjustments due to (6.2.6).
Fix1	:	Number of variables fixed to 0 due to the criterion described in Section 6.2.3 ($\|Q\| = 1$).
Fix2	:	Number of variables fixed to 0 due to the criterion described in Section 6.2.3 ($\|Q\| = 2$).
$\|A\|$	:	Number of remaining arcs / variables

We summarize the keys to the tables on the following pages.

Key to Tables 6.3, 6.4 and 6.7:

n	:	Number of nodes
Opt	:	Optimal solution value.
BC-root	:	Quality of the solution at the root LP, namely
... bounds		upper and lower bounds on the root node of the branch&cut tree, stated in the form $[lb, ub]$. If the problem instance is solved to optimality in the root, this is stated by "–".
... GAP	:	optimality gap at the root node ($\frac{opt-lb}{lb} \cdot 100$).
BC–tree	:	Size of the branch&cut tree characterized by
... # N	:	the number of nodes,
... level	:	the highest level (root is at level 0).
# cuts	:	Number of generated cutting planes.
# LPs	:	number of linear programs that were solved.
CPU	:	CPU–time to solve instances to optimality (on SUN SPARC 10)

Key to Tables 6.5 and 6.6:

Pool	:	Number of cuts generated from the pool.
SEC	:	Number of generated subtour elimination constraints.
TMC	:	Number of generated 2–matching constraints.
π	:	Number of generated π–inequalities (5.4.9).
σ	:	Number of generated σ–inequalities (5.4.13).
(pi, σ)	:	Number of generated (π, σ)–inequalities (5.4.17).
Inf.Path	:	Number of generated infeasible path constraints.
SH	:	Number of generated inequalities generated by shrinking procedure (5.6.36).
3C	:	Number of generated lifted 3–cycle constraints.
GC	:	Number of generated cuts $x(A^*) \leq n - 1$ (cmp. 6.5.45).

Computational comparison of the two models

Model 1									
			BC-root		BC–tree				
Problem	n	Opt.	bounds	GAP	#N	Level	# cuts	#LPs	CPU
rbg10a	12	149	–	–	1	0	19	8	0:00.48
rbg10b	12	108	–	–	1	0	0	1	0:00.25
rbg16a	18	179	–	–	1	0	10	6	0:00.58
rbg16b	18	142	[133, 194]	6.77	35	8	306	183	0:41.02
rbg17a	19	146	–	–	1	0	2	3	0:02.10
rbg19a	21	217	–	–	1	0	2	2	0:00.52
rbg19b	21	182	[175, 290]	4.00	161	12	2713	1221	4:22.43
rbg19c	21	190	[182, 201]	4.40	151	12	1472	571	2:19.08
rbg20a	22	210	–	–	1	0	0	1	0:01.80
rbg27a	29	268	[266, 268]	0.75	21	5	526	149	1:09.28
rbg48a	50	[455,856]	[455,856]	?	?	?	?	?	—*
rbg49a	51	[410,920]	[410,920]	?	?	?	?	?	—*
rbg50a	52	414	–	–	1	0	499	88	22:02.87
rbg50b	52	[447,940]	[447,940]	?	?	?	?	?	—*
rbg50c	52	[508,888]	[508,888]	?	?	?	?	?	—*
Model 2									
			BC-root		BC–tree				
Problem	n	Opt.	bounds	GAP	#N	Level	# cuts	#LPs	CPU
rbg10a	12	149	–	–	1	0	23	11	0:00.37
rbg10b	12	108	–	–	1	0	0	1	0:00.20
rbg16a	18	179	–	–	1	0	13	7	0:00.48
rbg16b	18	142	[133, 194]	6.77	43	8	248	156	0:12.20
rbg17a	19	146	–	–	1	0	2	3	0:01.73
rbg19a	21	217	–	–	1	0	4	3	0:00.45
rbg19b	21	182	[175, 290]	4.00	113	16	1647	797	1:37.30
rbg19c	21	190	[182, 201]	4.40	175	12	1644	633	0:59.05
rbg20a	22	210	–	–	1	0	0	1	0:01.92
rbg27a	29	268	[266, 268]	0.75	27	6	610	178	0:47.78
rbg48a	50	[455,856]	[455,856]	?	?	?	?	?	—*
rbg49a	51	[410,920]	[410,920]	?	?	?	?	?	—*
rbg50a	52	414	–	–	1	0	264	57	1:56.90
rbg50b	52	[448,940]	[448,940]	?	?	?	?	?	—*
rbg50c	52	[508,888]	[508,888]	?	?	?	?	?	—*

Table 6.3: Computational results for data 1

Model 1									
			BC-root		BC–tree				
Problem	n	Opt.	bounds	GAP	#N	Level	# cuts	#LPs	CPU
br17.a.00	19	25	–	–	1	0	16	11	0:02.40
br17.a.01	19	25	–	–	1	0	17	11	0:01.75
br17.a.02	19	25	–	–	1	0	25	10	0:02.12
br17.a.03	19	27	–	–	1	0	44	22	0:04.28
br17.a.04	19	34	[28, 37]	21.43	?	?	?	?	—*
br17.a.05	19	37	[30, 42]	23.33	?	?	?	?	—*
br17.a.06	19	37	[30, 42]	23.33	?	?	?	?	—*
br17.a.07	19	37	[30, 44]	23.33	?	?	?	?	—*
br17.a.08	19	37	[30, 44]	23.33	?	?	?	?	—*
br17.a.09	19	54	[49, 68]	10.20	151	14	522	366	1:37.67
br17.a.10	19	55	[49, 111]	12.24	275	17	1936	944	3:44.63
br17.a.11	19	55	[50, 60]	10.00	141	14	1007	482	2:02.60
br17.a.12	19	57	[52, 74]	9.62	105	13	703	350	1:03.37
br17.a.13	19	62	[52, 71]	19.23	95	9	357	209	0:30.97
br17.a.14	19	62	[57, 109]	8.77	3	1	19	9	0:01.60
br17.a.15	19	69	[57, 69]	21.05	17	5	71	40	0:03.45
br17.a.16	19	81	–	–	1	0	0	1	0:00.67
br17.a.17	19	81	–	–	1	0	0	1	0:00.48

Model 2									
			BC-root		BC–tree				
Problem	n	Opt.	bounds	GAP	#N	Level	# cuts	#LPs	CPU
br17.a.00	19	25	–	–	1	0	14	12	0:00.65
br17.a.01	19	25	[25, 29]	0.00	3	1	20	17	0:01.03
br17.a.02	19	25	–	–	1	0	18	12	0:00.85
br17.a.03	19	27	–	–	1	0	57	28	0:02.45
br17.a.04	19	34	[27, 37]	25.93	3199	35	12225	8781	13:42.65
br17.a.05	19	37	[30, 42]	23.33	16959	44	87563	43055	84:00.42
br17.a.06	19	37	[30, 44]	23.33	18883	38	100667	48251	89:43.17
br17.a.07	19	37	[30, 44]	23.33	16201	37	88099	41922	77:18.20
br17.a.08	19	37	[30, 44]	23.33	6709	31	36650	18621	27:18.40
br17.a.09	19	54	[49, 68]	10.20	161	15	677	433	0:38.62
br17.a.10	19	55	[49,111]	12.24	363	14	2204	1103	1:44.63
br17.a.11	19	55	[50, 72]	10.00	171	12	1311	605	0:54.47
br17.a.12	19	57	[52, 74]	9.62	355	23	3017	1234	1:23.72
br17.a.13	19	62	[52, 71]	19.23	153	15	1020	431	0:25.03
br17.a.14	19	62	[57,109]	8.77	11	3	35	27	0:01.78
br17.a.15	19	69	[57, 69]	21.05	19	5	72	51	0:02.67
br17.a.16	19	81	–	–	1	0	0	1	0:00.53
br17.a.17	19	81	–	–	1	0	0	1	0:00.43

Table 6.4: Computational results for data 2

	Pool	SEC	TMC	π	σ	(π, σ)	Inf.Path	SH	3C	GC	$\sum$
Model 1											
rbg10a	0	8	0	3	0	0	8	0	0	0	19
rbg10b	0	0	0	0	0	0	0	0	0	0	0
rbg16a	0	4	0	0	0	0	6	0	0	0	10
rbg16b	36	53	0	88	1	95	68	0	1	0	306
rbg17a	0	2	0	0	0	0	0	0	0	0	2
rbg19a	0	2	0	0	0	0	0	0	0	0	2
rbg19b	111	143	15	907	351	807	268	222	0	0	2713
rbg19c	52	177	4	184	0	120	985	0	2	0	1472
rbg20a	0	0	0	0	0	0	0	0	0	0	0
rbg27a	12	44	0	76	0	305	94	3	4	0	526
rbg50a	0	88	5	107	0	293	6	0	0	0	499
Model 2											
rbg10a	0	8	0	7	0	0	8	0	0	0	23
rbg10b	0	0	0	0	0	0	0	0	0	0	0
rbg16a	0	3	0	0	0	0	10	0	0	0	13
rbg16b	10	44	0	58	1	73	70	0	2	0	248
rbg17a	0	2	0	0	0	0	0	0	0	0	2
rbg19a	0	2	0	0	0	0	2	0	0	0	4
rbg19b	30	121	7	502	160	413	211	232	1	0	1647
rbg19c	76	180	4	191	0	149	1119	0	1	0	1644
rbg20a	0	0	0	0	0	0	0	0	0	0	0
rbg27a	14	56	0	71	0	290	188	4	1	0	610
rbg50a	0	24	3	76	0	161	0	0	0	0	264

Table 6.5: Generated cuts for data 1 (solved instances only)

	Pool	SEC	TMC	π	σ	(π, σ)	Inf.Path	SH	3C	GC	$\sum$
Model 1											
br17.a.01	0	16	0	0	0	0	0	0	1	0	17
br17.a.02	0	25	0	0	0	0	0	0	0	0	25
br17.a.03	0	28	0	1	0	3	10	0	2	0	44
br17.a.09	10	136	5	65	0	48	250	0	18	0	522
br17.a.10	79	326	17	291	5	183	1044	0	70	0	1936
br17.a.11	10	143	16	81	3	213	542	1	8	0	1007
br17.a.12	12	89	3	100	3	115	379	0	14	0	703
br17.a.13	0	37	2	8	0	20	287	0	3	0	357
br17.a.14	0	3	0	0	0	0	16	0	0	0	19
br17.a.15	0	8	5	10	0	4	44	0	0	0	71
br17.a.16	0	0	0	0	0	0	0	0	0	0	0
br17.a.17	0	0	0	0	0	0	0	0	0	0	0
Model 2											
br17.a.00	0	14	0	0	0	0	0	0	0	0	14
br17.a.01	0	18	0	0	0	0	0	0	2	0	20
br17.a.02	0	18	0	0	0	0	0	0	0	0	18
br17.a.03	0	25	0	8	0	2	20	0	2	0	57
br17.a.04	607	2022	497	68	11	90	9394	16	127	0	12225
br17.a.05	42	9651	1408	1219	24	116	74354	0	791	0	87563
br17.a.06	8	10865	1558	1434	1	48	85866	7	888	0	100667
br17.a.07	0	8900	1409	1014	6	28	76090	4	648	0	88099
br17.a.08	261	5965	565	831	5	175	28719	5	385	0	36650
br17.a.09	12	108	14	85	2	46	392	5	25	0	677
br17.a.10	96	360	22	247	2	123	1408	0	42	0	2204
br17.a.11	13	121	21	135	1	234	783	4	12	0	1311
br17.a.12	45	391	77	325	0	211	1990	2	21	0	3017
br17.a.13	2	100	27	93	0	43	756	0	1	0	1020
br17.a.14	0	5	4	0	0	2	24	0	0	0	35
br17.a.15	0	17	4	0	0	7	44	0	0	0	72
br17.a.16	0	0	0	0	0	0	0	0	0	0	0
br17.a.17	0	0	0	0	0	0	0	0	0	0	0

Table 6.6: Generated cuts for data 2 (solved instances only)

Influence of general clique cuts

We summarize the computational results obtained with the general cuts $x(A^*) \leq n-2$, that are valid in case that the LP–solution digraph $D^* = (V, A^*)$ does not contain a feasible solution (see also Section 6.5.7). We describe the results just for the implementation of model 2.

With general clique cuts									
			BC-root		BC–tree				
Problem	n	Opt.	bounds	GAP	#N	Level	# cuts	#LPs	CPU
rbg10a	12	149	–	–	1	0	22	10	0:00.37
rbg10b	12	108	–	–	1	0	0	1	0:00.22
rbg16a	18	179	–	–	1	0	13	7	0:00.53
rbg16b	18	142	[135,143]	5.19	17	6	406	185	0:36.05
rbg17a	19	146	–	–	1	0	2	3	0:01.97
rbg19a	21	217	–	–	1	0	4	3	0:00.52
rbg19b	21	182	[177,182]	2.82	53	11	3962	1299	11:05.57
rbg19c	21	190	[185,190]	2.70	59	11	1863	557	1:28.83
rbg20a	22	210	–	–	1	0	0	1	0:01.82
rbg27a	29	268	[266,268]	0.75	13	4	1388	359	2:29.97
rbg50a	52	414	–	–	1	0	465	78	3:22.32
br17a.00	19	25	–	–	1	0	18	12	0:00.73
br17a.01	19	25	[25, 29]	0.00	3	1	31	22	0:01.72
br17a.02	19	25	[25, 29]	0.00	3	1	23	18	0:01.45
br17a.03	19	27	[27, 37]	0.00	3	1	70	34	0:03.98
br17a.04	19	34	[27, 37]	25.93	30021	47	200432	98479	151:03.23
br17a.05	19	37	[30, 42]	23.33	21021	45	120738	55911	104:58.98
br17a.06	19	37	[30, 44	23.33	?		?	?	?
br17a.07	19	37	[30, 44]	23.33	21159	47	135532	60160	98:05.23
br17a.08	19	37	[30, 44]	23.33	7423	36	57141	23311	35:01.48
br17a.09	19	54	[49, 68]	10.20	159	12	1194	493	0:44.92
br17a.10	19	55	[49,109]	12.24	405	20	4253	1585	3:27.55
br17a.11	19	55	[50, 74]	10.00	889	26	18131	5347	13:07.78
br17a.12	19	57	[52, 74]	9.62	115	13	2016	676	1:14.08
br17a.13	19	62	[52, 71]	19.23	167	18	1733	581	0:38.65
br17a.14	19	62	[57,109]	8.77	17	5	40	34	0:02.25
br17a.15	19	69	[57, 69]	21.05	19	5	80	52	0:02.77
br17a.16	19	81	–	–	1	0	0	1	0:00.58
br17a.17	19	81	–	–	1	0	0	1	0:00.47

Table 6.7

It is interesting to observe that for most instances the use of these cuts results in smaller branch&cut trees and in better bounds on the root. But despite this fact, the overall computing times are higher (compare also Tables 6.3 and 6.4). The reason is that more cuts are generated, the general clique cuts are very dense and typically not very strong, as they can be strengthened taking arcs into account, that correspond to variables with LP value zero. But on the other hand, they force the LP–solution to contain feasible solutions or at least structures that can easily be extended to feasible solutions. Therefore, the LP–exploitation heuristic tends to give better results. Further computational experiments with these cuts will be necessary. E.g., it might be promising to use these cuts only in the root of the branch&cut tree.

Finally we mention, that this computational experience with these cuts extend in an analogous way to the instances not mentioned in this table.

6.8 Conclusions and outlook

In this chapter a polyhedral approach to solve problem instances of the AHPPTW to optimality has been described. In the preliminary computational results we have seen that this approach is capable of solving problem instances up to 30–50 nodes with varying number and width of time window to optimality.
We emphasize that the implementational expense for the branch&cut code is higher than for an implementation of a dynamic programming or branch&bound algorithm. But from our point of view already now this approach is comparable to the exact algorithms based on implicit enumeration, if not for all nodes the time windows are active or the time window width is relatively large. The computational performance of the branch&cut code can further be improved by

- the implementation of more sophisticated heuristics,
- the implementation of further heuristic separation routines for the infeasible path constraints,
- the implementation of further separation routines for the ATSP (see [FT94]),
- the use of a more sophisticated branch&cut framework (e.g., ABACUS 0.1).

This will be part of future research.

In the following we state two points that we consider interesting to be analysed.

An alternative model

The precedence constrained AHPP can be modelled by means of ATSP–variables and Linear Ordering variables (see [AEGS93]). A similar model can be stated for the time constrained AHPP. In addition to model 2 additional arc variables y_{ij} are introduced for all $(i, j) \in A_n$. The Linear Ordering constraints defined on the y–variables can be used to model the time window restrictions.

Beforehand, it is not clear if this model is competitive to the models presented in this section. The y–variables offer the advantage that they capture the "transitivity" of the infeasible paths but the disadvantage that a linking of the x– and y–variables is required. For the SOP it has been shown in a computational comparison that the combined model does not give better results than the one based on the x–variables only (see [AEGS93]). We presume that this holds for the AHPPTW as well. But computational tests will be necessary to confirm that fact.

Knapsack constraints

For two given nodes i and j the time windows define a knapsack with $W_{ij} = d_j - r_i$. It will be necessary to determine inequalities that take care of the fact that only a certain amount of other nodes can be packed into this time interval (knapsack). Up to now it is not clear how to decide for a given fractional solution $\bar{x}$ how many items are packed into a time interval between two nodes.

Conclusions and outlook

On–line optimization

In this thesis we have described a practical problem that we encountered in the on–line optimization of the movements of a stacker crane in an automatic storage system. We have seen a problem, the so–called on–line Hamiltonian path problem, that has not been studied in that form in the literature so far.

In 1989 Lovász et al. wrote:

> *"We believe that many other problems can and should be analyzed from the perspective of on–line algorithms."[LST89].*

As many practical applications are of an on–line character as described in this thesis, we support this point of view. But despite the practical importance of on–line problems and the fact that on–line optimization problems are mathematically interesting on their own, the scientific literature on handling complex on–line situations is almost nonexisting. Even case studies on practical problems can be found very seldomly in the literature.

Although an optimization package has been described that is running with success in everyday production in a complex FMS, we are aware of the fact that the approach presented in this thesis may not be the best one to handle on–line decisions of the type described here. We believe that deeper theoretical investigations are necessary to get a better understanding of on–line problems.

The concept of competitiveness, which has been used so far to compare the theoretical performance of different on–line algorithms, leads to very pessimistic results. Furthermore, this concept does not properly capture some of the restrictions that typically occur in complex on–line situations (time windows, priorities, etc.). Therefore, we consider it important to develop other approaches to evaluate on–line algorithms.

Fundamental research in modelling complex on–line situations, such as the stacker crane application, seems to be important. The main point will be to develop models that capture all restrictions that occur in the practical on–line application and that can then be attacked by algorithmic tools. One should keep it mind that it is very important to meet the time restrictions that typically occur in practical applications.

In order to derive efficient algorithms for complex on–line situations, we believe that the on–line problem cannot be addressed only from the combinatorial point of view. For instance for the stacker crane application, which has been discussed in this thesis, it seems to us promising to combine the stochastic and combinatorial structure of the on–line problem in one model, that might then contain stochastic and/or nonlinear components.

AHPP with side constraints

We derived lower bounds on the value obtained by an optimal on–line strategy by analyzing two off–line Combinatorial Optimization problems that have not yet been studied intensively in the literature, namely to the asymmetric Hamiltonian path problem with precedence constraints, also called sequential ordering problem (SOP), and the asymmetric Hamiltonian path problem with time windows (AHPPTW). No attempts to solve these problems to optimality by means of polyhedral methods have been published so far.

We performed polyhedral investigations on these problems and derived several classes of new valid inequalities. Branch&cut algorithms have been implemented, that are able to solve reasonable sized real–life problem instances to optimality. For the SOP the branch&cut algorithm outperforms the codes based on implicit enumeration, since it is able to solve instances with a varying number of precedences. For the AHPPTW the code in its current preliminary version is at least comparable to other approaches. We expect that the implementation of further separation routines and the use of further classes of inequalities will improve the code.

But despite the impressive computational results we believe that further theoretical and computational investigations on both problems are necessary. This is on the one hand due to the fact that routing problems with additional side constraints (e.g., precedences or time windows) occur in many practical applications. On the other hand there are several open questions that are interesting from a theoretical point of view and that have been mentioned in the corresponding sections. For the SOP for instance, it seems promising to derive a general lifting procedure to derive valid inequalities from ATSP–inequalities. Furthermore, we suggest to develop more exact and/or heuristic separation routines that explicitly take care of the precedences among the nodes.

For the AHPPTW the time windows have an "unpleasant" influence on the structure of the underlying polytopes, which makes it difficult to derive "nice" facial results. We believe that there is no hope to develop as comprehensive results about their facial structure as it has been done for the ATSP or many other Combinatorial Optimization problems. For instance, it is already a hard combinatorial problem to determine the dimension of these polytopes. One intuitive reason for this fact is, that there exists no complete characterization for the infeasible paths of length 1, corresponding to variables that can a priori be fixed to 0. This results in the feature that the minimal equation system cannot be determined, as not all implicit equations of the form $x_{ij} = 0$ are known. If it is possible to derive conditions to characterize all these variables and if that can be efficiently be computed (as it is the case for the SOP), this will improve both, the theoretical understanding of the AHPPTW and the computational performance of the branch&cut code.

Based on the computational experience we gathered so far, we presume that a combination of polyhedral and enumerative methods will be a good way to attack the AHPPTW. For instance, it might be worth to enumerate all infeasible paths on the nodes of a generated tournament constraint, and then to add the strengthened tournament constraint to the LP. Furthermore, we consider interesting to analyze if the clique cuts, that have been described in Section 6.5.7, can help in solving the problem instances to optimality.

As a by–product we obtained a branch&cut code for the asymmetric travelling salesman problem, whose computational performance on hard problem instances from TSPLIB is comparable to the codes published in the literature. A fine–tuning of the parameters of the branch&cut code, originally developed for the SOP, the implementation of the exact separation of the T_k–inequalities and the embedding of the separation routine, recently developed

by Fischetti and Toth, will improve the performance of the code.

Modelling of FMSs

The modern FMSs are complex systems characterized by many interdependencies that cannot be modelled properly. Typically, the optimization of the FMS is addressed by a hierarchical approaches reducing the overall problem to the optimization of certain subproblems – with unpredictable influences on the rest of the system. This results in an enormous loss of the potential of this technology. It would be better to optimize the whole process instead of suboptimizing on a tool–by tool or machine–by machine basis. But still the research is far away from giving satisfactory answers to this problem. Further research activities, especially on the modelling of such systems, are necessary.

Appendix A

List of symbols and abbreviations

We summarize the most important abbreviations and mathematical symbols that have been used in different parts of the thesis.

c_{ij}	:	cost coefficient for arc (i, j).
$P = (V, R)$	:	precedence digraph; if $(i, j) \in R$, then i has to precede j in every feasible solution.
$i \prec j$	:	precedence relationship, i has to precede j in every feasible solution. This implies $(i, j) \in R$.
$\pi(i)$	:	Set of predecessors of node i. If $k \in \pi(i)$, then $k \prec i$.
$\pi(S)$	:	$\cup_{i \in S} \pi(i)$.
$\sigma(i)$	:	Set of successors of node i. If $k \in \sigma(i)$, then $i \prec k$.
$\sigma(S)$	:	$\cup_{i \in S} \sigma(i)$.
p_i	:	processing time for node i.
r_i	:	release time of node i.
d_i	:	due date for node i.
$[r_i, d_i]$	:	time window for node i.
ϑ_{ij}	:	time delay for processing j after i, defined as $\vartheta_{ij} = p_i + c_{ij}$.
ω_{ij}	:	shortest path on ϑ from i to j.
$\Phi[W]$	:	Permutation of the nodes in W (p. 150).
P_T^n	:	Asymmetric travelling salesman polytope.
$\tilde{P}_T^n$	:	Monotone asymmetric travelling salesman polytope.
$P_{PC}(n, P)$	:	Precedence constrained asymmetric travelling salesman polytope.
$SOP(n, P)$	:	Sequential ordering polytope.
P_1^{TW}	:	Time constrained asymmetric Hamiltonian path polytope involving x– and t–variables.
P_2^{TW}	:	Time constrained asymmetric Hamiltonian path polytope involving only x–variables.

AGV	:	automatically guided vehicle
AHPP	:	asymmetric Hamiltonian path problem
AHPPTW	:	asymmetric Hamiltonian path problem with time windows (see page 131)
ATSP	:	asymmetric travelling salesman problem
AGV	:	Flexible Manufacturing system
JSSP	:	job–shop scheduling problem (see page 133)
ATSPTW	:	asymmetric travelling salesman problem with time windows (see page 132)
SEC	:	subtour elimination constraint
SOP	:	sequential ordering problem (see Chapter 5)
TSP	:	travelling salesman problem
TSPTW	:	travelling salesman problem with time windows

Appendix B

Statistics on SOP

Generated cuts for pure ATSP instances

In the following table the number of generated cuts for the pure ATSP instances is given. The upper part contains the runs without the heuristic T_k–inequality separation, the lower the runs with this separation routine.

Problem	SEC	TMC	T_k	$\sum$
br17	9	0	–	9
p43	136	36	–	172
p43x2	214	53	–	267
ry48p	100	50	–	150
ft53	30	0	–	30
ft70	22	0	–	22
kro124p	133	45	–	178
br17	9	0	0	9
p43	125	37	40	202
p43x2	137	25	40	202
ry48p	94	42	32	168
ft53	26	0	2	28
ft70	22	0	0	22
kro124p	121	47	17	185

Table B.1: Generated cuts for ATSP–instances

Problem	Name of the problem instance.
SEC	Number of generated subtour elimination constraints.
TMC	Number of generated 2–matching constraints.
T_k	Number of generated T_k-inequalities.
$\sum$	Total number of generated cuts.

Distribution of computing time for ATSP instances

In this table we summarize the proportion of time spent in the different parts of the implemented branch&cut code. The entries have to be read as percent of total time spent in a certain part, e.g., for instance *br17* 8.7% of the total computing time is spent in the initialization phase.

As in the previous table the upper, resp. lower, part contains the values obtained without, resp. with, T_k–separation. Note, that the major time is spent on the solution of the LPs and in the LP–exploitation routine. The results do not differ significantly with or without the T_k–inequality separation routine.

Problem	init.	LP	improve	separation	pricing	misc.	total
br17	8.7	56.5	13.0	0.0	4.3	17.4	0:00.38
p43	0.9	36.6	33.8	5.5	16.4	6.9	2:07.62
p43x2	0.5	34.4	34.9	5.3	17.3	7.5	3:50.53
ry48p	3.3	29.7	57.6	4.2	2.7	2.4	0:40.12
ft53	19.5	15.9	60.0	2.3	1.7	0.5	0:09.55
ft70	38.5	8.1	46.4	2.4	3.7	1.0	0:29.05
kro124p	4.9	7.7	84.2	0.8	1.5	0.9	6:27.63
br17	16.0	52.0	16.0	4.0	8.0	4.0	0:00.42
p43	0.7	25.5	23.0	5.8	41.1	4.0	2:50.77
p43x2	1.1	29.2	32.2	6.4	27.1	4.1	1:39.50
ry48p	3.2	26.2	55.9	6.2	6.5	1.9	0:40.80
ft53	17.2	13.7	61.8	1.7	4.2	1.4	0:11.82
ft70	38.7	7.3	46.8	3.0	3.7	0.5	0:29.17
kro124p	4.4	5.4	58.1	1.0	30.4	0.6	7:05.15

Table B.2: Distribution of computing time

Problem	Name of the problem instance.
LP	Time needed to solve the linear programs (in %)
improve	Time needed for the LP–exploitation heuristic (in %)
separation	Time needed for the separation of violated inequalities (in %)
pricing	Time needed to price out nonactive variables (in %)
misc	All other time (in %)
total	Total CPU time to solve the instance to optimality

Generated cuts for the instances of SOP

In this table we summarize the number of cuts generated by the different separation routines. The aim of this table is to give an idea which separation routines have been "important" for the solution of the instances.

Problem	SEC	PCB	SEC(π)	SEC(σ)	PI	PI-SIGMA	SIGMA	TMC	Shrink	T_k	$\sum$
ESC07	0	0	2	2	0	0	0	0	0	0	4
ESC11	1	0	3	3	1	0	2	1	0	1	12
ESC12	1	0	3	8	0	1	1	0	10	0	24
ESC14	1	0	9	9	0	0	0	0	0	0	19
ESC25	0	0	10	8	18	1	10	0	20	0	67
ESC47	63	0	164	120	916	390	858	2	107	0	2620
ESC63	0	0	33	88	313	1121	712	0	16	2	2285
ESC78	0	0	81	68	284	1176	379	0	202	19	2209
ESC98	1	0	16	18	0	0	0	0	0	0	35
prob.1	2	0	2	2	0	0	0	0	0	0	6
prob.2	0	0	1	1	0	0	0	0	0	0	2
prob.3	8	0	6	4	0	1	1	0	19	0	39
prob.5	456	0	556	641	1346	3783	1340	274	570	172	9138
prob.6	48	0	8	10	2	0	1	8	17	0	94
prob.7	1	0	2388	1670	596	0	734	463	1259	33	7144
prob.faw1	0	0	7	40	0	0	0	4	0	2	53
rbg016a	0	0	2	1	0	0	0	0	0	0	3
rbg016b	0	0	10	8	10	11	5	0	34	0	78
rbg017a	0	0	6	5	4	0	0	0	0	0	15
rbg019a	0	0	6	6	0	0	0	0	1	0	13
rbg019b	0	0	2	2	0	0	0	0	0	0	4
rbg021a	0	0	5	3	1	7	1	0	14	0	31
rbg023a	0	0	2	2	1	7	1	0	6	0	19
rbg048a	0	0	68	31	339	1734	275	2	229	15	2693
rbg049a	0	0	19	19	4	95	17	0	53	4	211
rbg050a	0	0	31	37	5	61	10	0	172	0	316
rbg050b	0	0	34	23	7	67	4	0	12	3	150
rbg050c	0	0	59	59	46	352	33	6	301	4	860
rbg068a	0	0	8	7	1	7	1	0	6	0	30

Table B.3. Generated cutting planes

Problem	Name of the problem instance.
SEC	Number of generated "pure" subtour elimination constraints.
PCB	Number of generated "strengthened" subtour elimination constraints as simple *pcb*–inequality.
SEC(π)	Number of generated "strengthened" subtour elimination constraints as π–inequality.
SEC(σ)	Number of generated "strengthened" subtour elimination constraints as σ–inequality.
PI	Number of generated weak π–inequalities.
PI-SIGMA	Number of generated weak σ–inequalities.
SIGMA	Number of generated weak (π, σ)–inequalities.
TMC	Number of generated 2–matching constraints.
Shrink	Number of cuts generated by shrinking procedure.
T_k	Number of generated T_k–inequalities.
$\sum$	Total number of generated cuts.

Distribution of computing time for SOP–instances

In the following table we summarize the distribution of the computing time spent in the different parts of the branch&cut code. The key is similar to the one for the ATSP instances.

Problem	init.	LP	improve	separation	pricing	misc.	total
ESC07	20.0	60.0	0.0	20.0	0.0	0.0	0:00.10
ESC11	11.8	52.9	0.0	23.5	0.0	11.8	0:00.23
ESC12	2.9	62.9	11.4	11.4	8.6	2.9	0:00.52
ESC14	6.2	60.4	10.4	10.4	10.4	2.1	0:00.38
ESC25	1.7	35.8	20.5	10.3	28.1	3.6	0:04.33
ESC47	0.1	12.9	56.5	10.2	19.2	1.0	11:48.05
ESC63	0.4	16.3	16.2	20.8	45.8	0.4	4:33.22
ESC78	0.1	22.7	15.2	19.8	41.6	0.7	19:10.20
ESC98	0.9	21.5	11.3	3.6	60.4	2.4	0:29.67
prob.1	42.9	42.9	0.0	14.3	0.0	0.0	0:00.08
prob.2	25.0	75.0	0.0	0.0	0.0	0.0	0:00.10
prob.3	2.5	40.4	24.8	11.2	16.8	4.3	0:01.67
prob.5	0.0	14.7	10.3	26.7	47.4	0.9	27:39.27
prob.6	8.4	8.8	57.2	1.7	23.5	0.5	0:37.95
prob.faw1	7.1	12.3	48.0	4.1	27.5	1.1	0:22.68
rbg016a	10.5	52.6	5.3	15.8	10.5	5.3	0:00.18
rbg016b	3.7	50.0	13.4	20.7	6.1	6.1	0:01.37
rbg017a	1.9	53.8	7.7	15.4	15.4	5.8	0:00.48
rbg019a	9.4	50.0	3.1	18.8	6.2	12.5	0:00.48
rbg019b	50.0	37.5	12.5	0.0	0.0	0.0	0:00.13
rbg021a	1.7	43.3	5.0	16.7	28.3	5.0	0:00.45
rbg023a	9.1	38.6	4.5	11.4	27.3	9.1	0:00.52
rbg048a	0.1	10.5	40.2	22.2	25.7	1.3	8:55.53
rbg049a	2.6	17.5	25.4	12.9	39.0	2.7	0:18.57
rbg050a	1.2	27.9	19.1	14.0	33.6	4.3	0:15.02
rbg050b	2.7	20.6	25.1	10.6	38.1	2.8	0:09.73
rbg050c	0.2	22.6	34.1	20.1	20.5	2.5	1:37.37
rbg068a	15.1	18.3	36.3	13.4	14.5	2.4	0:03.92

Table B.4. Timing statistic

Appendix C

Statistics on AHPPTW

In the following we give a more detailed statistic on the instances of the AHPPTW mentioned in Table 6.1. The key to this table is the same as given in Section 6.7 (see page 174)

			BC-root		BC–tree				
Problem	n	Opt.	bounds	GAP	#N	Level	# cuts	#LPs	CPU
br17b.01	19	25	–	–	1	0	17	10	0:00.62
br17b.02	19	25	–	–	1	0	21	9	0:00.78
br17b.03	19	32	[25, 88]	28.00	4859	54	19248	12195	25:34.73
br17b.04	19	33	[29, 89]	13.79	1537	49	7107	3860	8:25.32
br17b.05	19	33	[30, 89]	10.00	893	30	5925	2946	5:52.40
br17b.06	19	33	[30, 82]	10.00	1191	38	4855	2868	6:09.52
br17b.07	19	33	[31, 44]	6.45	277	29	1664	849	1:15.62
br17b.08	19	33	[29, 81]	13.79	285	24	1887	864	1:21.75
br17b.09	19	33	[31, 33]	6.45	15	5	194	69	0:05.68
br17b.10	19	45	[33, 45]	36.36	19	4	74	44	0:03.25
br17b.11	19	45	–	–	1	0	5	4	0:01.08
br17b.12	19	48	–	–	1	0	4	3	0:00.87
br17b.13	19	58	[48, 60]	20.83	81	10	252	191	0:09.83
br17b.14	19	72	–	–	1	0	12	8	0:00.72
br17b.15	19	84	–	–	1	0	0	1	0:00.47
br17b.16	19	90	–	–	1	0	0	1	0:00.43
br17b.17	19	121	–	–	1	0	0	1	0:00.28
br17c.01	19	25	–	–	1	0	14	12	0:00.65
br17c.02	19	27	–	–	1	0	48	21	0:01.52
br17c.03	19	27	[25, 72]	8.00	1475	51	6213	3038	6:19.08
br17c.04	19	27	–	–	1	0	35	22	0:01.98
br17c.05	19	30	[30, 119]	0.00	5	2	27	19	0:01.63
br17c.06	19	30	–	–	1	0	23	11	0:00.87
br17c.07	19	31	[30, 120]	3.33	369	17	1371	1032	1:43.23
br17c.08	19	42	–	–	1	0	16	11	0:00.70
br17c.09	19	42	[42, 43]	0.00	7	3	52	38	0:02.65

Table C.1 (a).

			BC-root		BC–tree				
Problem	n	Opt.	bounds	GAP	#N	Level	# cuts	#LPs	CPU
br17c.10	19	42	[42, 43]	0.00	3	1	31	19	0:01.55
br17c.11	19	42	–	–	1	0	23	11	0:00.82
br17c.12	19	42	–	–	1	0	23	12	0:00.82
br17c.13	19	42	–	–	1	0	27	13	0:01.07
br17c.14	19	42	–	–	1	0	15	6	0:00.63
br17c.15	19	42	–	–	1	0	17	6	0:00.55
br17c.16	19	42	–	–	1	0	7	4	0:00.48
br17c.17	19	42	–	–	1	0	10	4	0:00.53
br17d.01	19	25	–	–	1	0	14	12	0:00.67
br17d.02	19	28	[25, 94]	12.00	103	14	233	286	0:27.37
br17d.03	19	34	[33, 155]	3.03	121	13	440	394	0:46.67
br17d.04	19	33	[33, 126]	0.00	3	1	82	39	0:05.75
br17d.05	19	34	[34, 87]	0.00	7	3	100	61	0:08.50
br17d.06	19	34	–	–	1	0	65	31	0:02.87
br17d.07	19	34	[34, 58]	0.00	569	23	2627	1884	3:08.37
br17d.08	19	34	[34, 59]	0.00	209	22	1164	688	0:59.18
br17d.09	19	34	[34, 54]	0.00	3	1	71	36	0:02.93
br17d.10	19	40	–	–	1	0	16	12	0:00.97
br17d.11	19	40	–	–	1	0	8	7	0:00.75
br17d.12	19	69	[62, 140]	11.29	19	5	256	74	0:04.37
br17d.13	19	69	[62, 93]	11.29	9	3	62	23	0:01.53
br17d.14	19	78	–	–	1	0	1	2	0:00.62
br17d.15	19	78	–	–	1	0	6	4	0:00.63
br17d.16	19	90	–	–	1	0	0	1	0:00.37
br17d.17	19	121	–	–	1	0	0	1	0:00.32
br17e.01	19	25	–	–	1	0	14	12	0:00.70
br17e.02	19	28	[27, 40]	3.70	15	6	129	57	0:06.13
br17e.03	19	27	[27, 75]	0.00	605	24	3118	1256	2:50.68
br17e.04	19	27	–	–	1	0	31	18	0:01.52
br17e.05	19	28	[28, 84]	0.00	399	26	1539	953	1:39.80
br17e.06	19	28	[28, 84]	0.00	3	1	43	15	0:01.62
br17e.07	19	28	[28, 44]	0.00	2221	40	8387	5103	8:43.35
br17e.08	19	28	[28, 84]	0.00	885	33	4692	2512	4:22.30
br17e.09	19	33	–	–	1	0	42	24	0:01.83
br17e.10	19	51	[48, 51]	6.25	29	12	173	123	0:13.08
br17e.11	19	53	[49, 53]	8.16	117	14	618	427	0:42.38
br17e.12	19	53	[49, 56]	8.16	63	9	446	286	0:28.67
br17e.13	19	54	–	–	1	0	53	21	0:01.55
br17e.14	19	54	–	–	1	0	33	20	0:01.18
br17e.15	19	54	–	–	1	0	15	9	0:00.73
br17e.16	19	60	–	–	1	0	0	1	0:00.43
br17e.17	19	60	–	–	1	0	1	2	0:00.43

Table C.1 (b).

Problem	n	Opt.	BC-root bounds	BC-root GAP	BC–tree #N	BC–tree Level	# cuts	#LPs	CPU
br17f.01	19	25	–	–	1	0	12	10	0:00.63
br17f.02	19	28	[25, 75]	12.00	221	25	822	486	0:54.12
br17f.03	19	27	[27, 139]	0.00	3	1	26	17	0:01.48
br17f.04	19	33	–	–	1	0	49	24	0:02.42
br17f.05	19	33	[33, 42]	0.00	5	2	64	38	0:04.53
br17f.06	19	33	[33, 43]	0.00	3	1	91	41	0:05.40
br17f.07	19	33	–	–	1	0	69	32	0:03.30
br17f.08	19	33	[33, 44]	0.00	3	1	82	38	0:04.52
br17f.09	19	33	[33, 41]	0.00	55	13	173	144	0:20.65
br17f.10	19	33	[33, 108]	0.00	3	1	94	29	0:03.85
br17f.11	19	33	–	–	1	0	94	30	0:03.27
br17f.12	19	33	–	–	1	0	77	16	0:01.78
br17f.13	19	45	–	–	1	0	47	13	0:01.15
br17f.14	19	45	[45, 51]	0.00	5	2	54	21	0:01.87
br17f.15	19	45	–	–	1	0	15	6	0:00.62
br17f.16	19	48	–	–	1	0	7	5	0:00.57
br17f.17	19	48	–	–	1	0	8	4	0:00.52

Table C.1 (c).

Problem	n	Opt.	BC-root bounds	BC-root GAP	BC–tree #N	BC–tree Level	# cuts	#LPs	CPU
rbg27.a.01	29	161	–	–	1	0	23	14	0:01.87
rbg27.a.02	29	161	–	–	1	0	11	13	0:02.08
rbg27.a.03	29	161	–	–	1	0	15	15	0:02.42
rbg27.a.04	29	161	–	–	1	0	11	14	0:02.00
rbg27.a.05	29	161	–	–	1	0	53	35	0:07.27
rbg27.a.06	29	161	[161,278]	0.00	3	1	88	53	0:11.90
rbg27.a.07	29	184	[182,297]	1.10	449	24	4844	2709	15:31.18
rbg27.a.08	29	199	[193,294]	3.11	5	2	336	143	0:40.65
rbg27.a.09	29	204	[197,292]	3.43	29	10	1050	404	1:55.68
rbg27.a.10	29	218	[209,291]	4.31	47	8	1144	461	1:46.63
rbg27.a.11	29	230	[213,299]	7.98	49	8	1325	481	2:07.73
rbg27.a.12	29	227	[221,331]	2.71	9	3	475	173	0:43.33
rbg27.a.13	29	230	[226,362]	1.77	11	3	759	245	1:05.65
rbg27.a.14	29	232	[230,373]	0.87	9	4	685	183	0:41.97
rbg27.a.15	29	270	[252,281]	7.14	255	15	7283	2239	9:07.85
rbg27.a.16	29	265	[258,346]	2.71	73	10	932	357	1:00.88
rbg27.a.17	29	265	[261,341]	1.53	47	9	626	238	0:37.33
rbg27.a.18	29	272	[268,342]	1.49	3	1	72	24	0:03.27
rbg27.a.19	29	272	[268,340]	1.47	3	1	52	22	0:03.32
rbg27.a.20	29	284	–	–	1	0	22	7	0:01.75
rbg27.a.21	29	287	–	–	1	0	16	4	0:01.37
rbg27.a.22	29	287	–	–	1	0	0	1	0:01.10
rbg27.a.23	29	287	–	–	1	0	0	1	0:01.07
rbg27.a.24	29	287	–	–	1	0	0	1	0:01.05
rbg27.a.25	29	335	–	–	1	0	0	1	0:01.13
rbg27.a.26	29	335	–	–	1	0	0	1	0:01.35
rbg27.a.27	29	344	–	–	1	0	0	1	0:01.02
rbg27.b.01	29	161	–	–	1	0	8	9	0:01.47
rbg27.b.02	29	161	[161,201]	0.00	3	1	22	27	0:04.63
rbg27.b.03	29	167	[164,212]	1.83	443	15	4184	3164	14:43.22
rbg27.b.04	29	170	[164,245]	3.66	729	18	8802	5764	30:35.48
rbg27.b.05	29	181	[165,263]	9.70	2673	31	39757	22861	145:16.18
rbg27.b.06	29	185	[176,281]	5.11	1485	28	33210	16898	145:21.18
rbg27.b.07	29	206	[180,282]	14.77	2889	23	54342	24391	192:59.80
rbg27.b.08	29	207	[194,394]	6.70	1319	37	12737	7435	37:46.88
rbg27.b.09	29	214	[197,363]	8.63	3303	30	31396	18138	86:31.70
rbg27.b.10	29	210	[202,395]	3.96	803	21	7685	4211	18:51.37
rbg27.b.11	29	220	[210,231]	4.76	211	18	4464	1893	13:36.68
rbg27.b.12	29	222	[219,231]	1.37	13	3	540	224	0:53.67
rbg27.b.13	29	224	[222,229]	0.90	19	5	377	185	0:42.10
rbg27.b.14	29	224	–	–	1	0	131	60	0:11.82
rbg27.b.15	29	234	[231,241]	1.30	19	5	375	187	0:48.65
rbg27.b.16	29	238	–	–	1	0	120	44	0:08.15
rbg27.b.17	29	238	–	–	1	0	81	31	0:05.13
rbg27.b.18	29	238	–	–	1	0	65	25	0:04.18
rbg27.b.19	29	262	[261,263]	0.38	9	4	199	79	0:13.18
rbg27.b.20	29	263	–	–	1	0	50	13	0:02.90
rbg27.b.21	29	267	–	–	1	0	45	13	0:02.60

Table C.4(a).

Problem	n	Opt.	BC-root bounds	BC-root GAP	BC–tree #N	BC–tree Level	# cuts	#LPs	CPU
rbg27.b.22	29	267	–	–	1	0	0	2	0:01.70
rbg27.b.23	29	267	–	–	1	0	0	1	0:01.62
rbg27.b.24	29	310	–	–	1	0	0	2	0:01.32
rbg27.b.25	29	310	–	–	1	0	0	1	0:01.08
rbg27.b.26	29	326	–	–	1	0	0	1	0:01.62
rbg27.b.27	29	344	–	–	1	0	0	1	0:01.03
rbg27.c.01	29	161	–	–	1	0	10	10	0:01.37
rbg27.c.02	29	161	–	–	1	0	6	11	0:01.58
rbg27.c.03	29	161	–	–	1	0	30	14	0:02.35
rbg27.c.04	29	162	[161,236]	0.62	3	1	112	63	0:14.78
rbg27.c.05	29	180	[162,255]	11.11	4067	44	32070	21014	126:47.22
rbg27.c.06	29	175	[163,290]	7.36	3207	39	39648	21688	132:56.93
rbg27.c.07	29	180	[166,295]	8.43	3431	38	37900	20728	125:15.77
rbg27.c.08	29	190	[169,303]	12.43	2407	38	41256	19221	143:04.85
rbg27.c.09	29	209	[184,300]	13.59	5861	37	97468	43261	282:06.47
rbg27.c.10	29	206	[189,300]	14.29	839	22	16106	7323	47:07.07
rbg27.c.11	29	236	[196,301]	20.41	2203	32	18921	10291	52:00.05
rbg27.c.12	29	226	[213,305]	6.10	627	19	14060	5328	31:35.33
rbg27.c.13	29	250	[231,342]	8.23	725	23	17830	6243	34:56.53
rbg27.c.14	29	251	[235,342]	6.81	677	24	7952	3283	12:54.52
rbg27.c.15	29	248	[239,301]	3.77	147	13	2180	834	2:55.93
rbg27.c.16	29	250	[240,298]	4.17	179	13	2493	969	3:12.05
rbg27.c.17	29	250	[244,261]	2.46	141	11	1806	795	2:47.28
rbg27.c.18	29	250	[249,250]	0.40	5	2	153	64	0:10.60
rbg27.c.19	29	250	[249,250]	0.40	3	1	119	47	0:06.82
rbg27.c.20	29	256	[253,256]	1.19	15	6	332	116	0:16.52
rbg27.c.21	29	267	[265,268]	0.75	23	5	187	96	0:11.37
rbg27.c.22	29	267	–	–	1	0	71	19	0:03.08
rbg27.c.23	29	267	–	–	1	0	22	9	0:01.92
rbg27.c.24	29	267	–	–	1	0	0	1	0:02.90
rbg27.c.25	29	306	–	–	1	0	2	2	0:02.72
rbg27.c.26	29	306	–	–	1	0	0	1	0:02.13
rbg27.c.27	29	324	–	–	1	0	0	1	0:01.30
rbg27.d.01	29	161	–	–	1	0	13	12	0:01.68
rbg27.d.02	29	161	–	–	1	0	12	16	0:02.43
rbg27.d.03	29	164	[163,227]	0.61	19	5	216	103	0:23.90
rbg27.d.04	29	168	[165,272]	1.82	377	22	4569	2916	14:06.63
rbg27.d.05	29	170	[164,274]	3.66	2339	33	20332	12708	78:38.15
rbg27.d.06	29	170	[163,269]	4.29	4613	34	30092	17591	105:52.32
rbg27.d.07	29	179	[175,253]	2.29	575	28	5610	2933	16:18.07
rbg27.d.08	29	181	[175,295]	3.43	493	17	6784	3148	18:21.47
rbg27.d.09	29	190	[178,282]	6.74	1251	19	20856	8723	60:13.47
rbg27.d.10	29	191	[185,288]	3.24	111	14	2289	910	5:38.75
rbg27.d.11	29	194	[192,197]	1.04	33	6	928	345	2:00.78
rbg27.d.12	29	206	[202,313]	1.98	35	7	1074	377	2:14.38
rbg27.d.13	29	221	[214,311]	3.27	29	9	1434	474	3:27.47
rbg27.d.14	29	221	–	–	1	0	77	31	0:09.60
rbg27.d.15	29	221	–	–	1	0	85	39	0:10.47

Table C.4(b).

Problem	n	Opt.	BC-root bounds	BC-root GAP	BC–tree #N	BC–tree Level	# cuts	#LPs	CPU
rbg27.d.16	29	221	–	–	1	0	90	32	0:09.30
rbg27.d.17	29	223	–	–	1	0	41	21	0:06.72
rbg27.d.18	29	229	–	–	1	0	5	3	0:04.82
rbg27.d.19	29	239	–	–	1	0	17	7	0:04.90
rbg27.d.20	29	239	–	–	1	0	1	3	0:04.42
rbg27.d.21	29	265	[261,272]	1.53	3	1	115	56	0:10.48
rbg27.d.22	29	253	–	–	1	0	2	2	0:04.13
rbg27.d.23	29	271	–	–	1	0	10	5	0:02.87
rbg27.d.24	29	279	–	–	1	0	4	3	0:02.55
rbg27.d.25	29	297	–	–	1	0	0	1	0:01.53
rbg27.d.26	29	306	–	–	1	0	0	1	0:02.75
rbg27.d.27	29	322	–	–	1	0	0	1	0:01.17
rbg27.e.01	29	161	–	–	1	0	7	10	0:01.43
rbg27.e.02	29	163	[161,192]	1.24	9	4	44	54	0:09.88
rbg27.e.03	29	165	[164,240]	0.61	245	16	1287	1093	4:17.08
rbg27.e.04	29	168	[164,246]	2.44	1583	25	7684	6185	29:03.77
rbg27.e.05	29	179	[174,244]	2.87	1003	20	8900	5335	31:30.08
rbg27.e.06	29	187	[174,275]	7.47	11093	45	76716	46792	296:59.02
rbg27.e.07	29	187	[179,302]	4.47	781	22	9398	4586	26:12.70
rbg27.e.08	29	187	[179,298]	4.47	613	23	11068	4513	27:32.85
rbg27.e.09	29	188	[180,298]	4.44	673	22	11885	4402	29:38.80
rbg27.e.10	29	190	[182,313]	4.40	659	21	15488	5236	35:47.53
rbg27.e.11	29	196	[193,313]	1.55	493	31	7535	2866	16:12.08
rbg27.e.12	29	196	[194,309]	1.03	115	16	1643	660	4:24.17
rbg27.e.13	29	200	[197,310]	1.52	123	17	1736	718	2:58.13
rbg27.e.14	29	200	–	–	1	0	58	30	0:04.37
rbg27.e.15	29	200	–	–	1	0	49	21	0:03.93
rbg27.e.16	29	211	–	–	1	0	19	12	0:02.70
rbg27.e.17	29	235	[233,238]	0.86	43	9	816	271	1:15.88
rbg27.e.18	29	235	[234,242]	0.43	35	9	561	193	0:45.83
rbg27.e.19	29	235	[234,239]	0.43	13	6	243	79	0:18.35
rbg27.e.20	29	235	[234,246]	0.43	23	6	218	119	0:20.87
rbg27.e.21	29	235	[234,235]	0.43	11	5	95	47	0:07.37
rbg27.e.22	29	252	–	–	1	0	39	21	0:03.08
rbg27.e.23	29	260	–	–	1	0	99	26	0:04.13
rbg27.e.24	29	260	–	–	1	0	45	13	0:02.20
rbg27.e.25	29	263	–	–	1	0	22	8	0:01.72
rbg27.e.26	29	263	–	–	1	0	24	6	0:01.47
rbg27.e.27	29	270	–	–	1	0	2	2	0:01.22
rbg27.f.01	29	161	–	–	1	0	5	9	0:01.33
rbg27.f.02	29	161	–	–	1	0	12	15	0:02.37
rbg27.f.03	29	161	–	–	1	0	9	8	0:01.43
rbg27.f.04	29	171	[166,244]	3.01	189	16	1063	858	3:47.27
rbg27.f.05	29	170	–	–	1	0	44	28	0:05.52
rbg27.f.06	29	170	–	–	1	0	51	26	0:05.33
rbg27.f.07	29	170	–	–	1	0	80	39	0:08.63
rbg27.f.08	29	175	[172,283]	1.74	449	35	4795	2141	14:21.73
rbg27.f.09	29	185	[183,303]	1.09	13	4	575	193	1:39.48

Table C.4(c).

Problem	n	Opt.	BC-root bounds	BC-root GAP	BC–tree #N	BC–tree Level	# cuts	#LPs	CPU
rbg27.f.10	29	196	[188,205]	4.26	245	15	8658	2499	72:44.03
rbg27.f.11	29	199	[191,214]	4.19	441	25	17063	4305	74:47.43
rbg27.f.12	29	202	[193,333]	4.66	725	26	17245	5124	48:35.70
rbg27.f.13	29	199	[195,216]	2.05	405	19	9502	2753	18:52.82
rbg27.f.14	29	199	[197,326]	1.02	499	24	9523	2904	16:06.28
rbg27.f.15	29	207	[198,216]	4.57	375	24	6892	2228	12:16.13
rbg27.f.16	29	217	[212,336]	2.36	111	16	2780	808	4:26.95
rbg27.f.17	29	220	–	–	1	0	253	62	0:18.05
rbg27.f.18	29	228	–	–	1	0	145	38	0:08.27
rbg27.f.19	29	232	–	–	1	0	138	46	0:08.85
rbg27.f.20	29	235	–	–	1	0	98	27	0:05.58
rbg27.f.21	29	235	–	–	1	0	99	25	0:04.73
rbg27.f.22	29	247	[246,247]	0.41	3	1	133	36	0:06.07
rbg27.f.23	29	247	[246,247]	0.41	3	1	98	32	0:06.00
rbg27.f.24	29	249	–	–	1	0	35	14	0:02.83
rbg27.f.25	29	263	–	–	1	0	39	11	0:02.52
rbg27.f.26	29	263	–	–	1	0	27	8	0:02.03
rbg27.f.27	29	270	–	–	1	0	4	3	0:01.75

Table C.4(d).

Bibliography

[Abd94] A. Abdel–Aziz Abdel–Hamid. *Combinatorial Optimization Problems Arising in the Design and Management of an Automatic Storage System.* PhD thesis, Technische Universität Berlin, Germany, 1994.

[ABN+88] S. Atherton, K. Butterbaugh, K.S. Nickerson, W.B. Powell, and Y. Sheffi. Maximizing profits for North American Van Lines' truckload division: A new framework for pricing and operations. *Interfaces*, 18(1):21–41, January–February 1988.

[AC91] D. Applegate and W. Cook. A computational study of the job–shop scheduling problem. *ORSA Journal on Computing*, 3:149–156, 1991.

[AEGS90] N. Ascheuer, L. Escudero, M. Grötschel, and M. Stoer. On identifying in polynomial time violated subtour elimination and precedence forcing constraints for the sequential ordering problem. In Kannan, R. and Pulleyblank,W.R., editors, *Integer Programming and Combinatorial Optimization*, pages 19–28. University of Waterloo, Waterloo, Ontario, 1990.

[AEGS93] N. Ascheuer, L. Escudero, M. Grötschel, and M. Stoer. A cutting plane approach to the sequential ordering problem (with applications to job scheduling in manufacturing). *SIAM Journal on Optimization*, 3:25–42, 1993.

[AG] J. Ashayeri and L. Gelders. Practice of simulation in manufacturing and logistic systems. *Belgian Journal of Operations Research, Statistics and Computer Science*, 29(1), ???

[Alb93] S. Albers. *The influence of lookahead in competitive on–line algorithms.* PhD thesis, Universität des Saarlandes, Saarbrücken, 1993.

[Asc89] N. Ascheuer. Ein Schnittebenenverfahren für ein Reihenfolgeproblem in der flexiblen Fertigung. Master's thesis, Universität Augsburg, Germany, 1989.

[Asc92] N. Ascheuer. *Das Simulationspaket AMSEL.* ZIB, 1992.

[Bak83] E.K. Baker. An exact algorithm for the time–constrained traveling salesman problem. *Operations Research*, 31(5):938–945, 1983.

[Bal85] E. Balas. On the facial structure of scheduling polyhedra. *Mathematical Programming Study*, 24:179–218, 1985.

[Bal89] E. Balas. The asymmetric assignment problem and some new facets of the travelling salesman polytope on a directed graph. *SIAM Journal on Discrete Mathematics*, 3:425–451, 1989.

[Bau90] P. Bauer. Optimale Steuerung des FALKE–Automaten, 1990. Documentation of a joint project of the University of Augsburg and Siemens AG, Augsburg, Germany.

[BB89] H.G. Bartels and S.G. Bartels. The facets of the asymmetric 5–city traveling salesman polytope. *Zeitschrift für Operations Research*, 33:193–197, 1989.

[BCSW86] J. Błazewisz, W. Cellary, R. Słowiński, and J. Węglarz. *Scheduling under resource constraints*, volume 7 of *Annals of Operations Research*. Baltzer, Basel, 1986.

[BDBK+90] S. Ben-David, A. Borodin, R.M. Karp, G. Tardos, and A. Wigderson. On the power of randomization in online algorithms. In *STOC90*, pages 379–386. ACM, May 1990.

[Ber92] D.J. Bertsimas. A vehicle routing problem with stochastic demand. *Operations Research*, 40(3):574–585, May 1992.

[BESW93] J. Błazewisz, K. Ecker, G. Schmidt, and J. Węglarz. *Scheduling in Computer and Manufacturing Systems*. Springer–Verlag, Berlin, 1993.

[BF92] E. Balas and M. Fischetti. The fixed–outdegree 1–arborescence polytope. *Mathematics of Operations Research*, 17:1001–1018, 1992.

[BF93a] E. Balas and M. Fischetti. A lifting procedure for the asymmetric traveling salesman polytope and a large new class of facets. *Mathematical Programming*, 58:1001–1018, 1993.

[BF93b] E. Balas and M. Fischetti. A lifting procedure for the asymmetric traveling salesman polytope and a large new class of facets. *Mathematical Programming*, A 58:325–352, 1993.

[BFP92] E. Balas, M. Fischetti, and W. Pulleyblank. The precedence constrained asymmetric traveling salesman polytope. Technical Report 15213, Carnegie Mellon University, Pittsburgh, 1992.

[BFR91] R.E. Burkard, B. Fruhwirth, and G. Rote. Vehicle routing in an automated warehouse: Analysis and optimization. Technical report, Technische Universität Graz, Austria, October 1991.

[BH93] D.J. Bertsimas and L.H. Howell. Further results on the probabilistic traveling salesman problem. *European Journal of Operations Research*, 65(1):68–95, 1993.

[BM76] J.A. Bondy and U.S.R. Murty. *Graph Theory with Applications*. American Elsevier, New York and Macmillan, London, 1976.

[BMRS] L. Bianco, A. Mingozzi, S. Ricciardelli, and M. Spadoni. Exact and heuristic procedures for the traveling salesman problem with precedence constraints, based on dynamic programming. Technical Report. (Publication around 1990. No further information available).

[BR91] D.J. Bertsimas and G. van Ryzin. A stochastic and dynamic vehicle routing problem in the euclidian plane. *Operations Research*, 39(4):601–615, July 1991.

[BR93a] D.J. Bertsimas and G. van Ryzin. Stochastic and dynamic vehicle routing in the euclidian plane with multiple capacitated vehicles. *Operations Research*, 41(1):60–76, July 1993.

[BR93b] D.J. Bertsimas and G. van Ryzin. Stochastic and dynamic vehicle routing with general arrival and demand distributions. *Advances in Applied Probability*, 25(4):947–978, 1993.

[CGS89] F.R.K. Chung, R.L. Graham, and M.E. Saks. A dynamic location problem for graphs. *Combinatorica*, 9(2):111–131, 1989.

[CKOS90] Y. Crama, A.W.J. Kolen, A.G. Oerlemans, and F.C.R. Spieksma. Throughput rate optimization in the automated assembly of printed circuit boards. *Annals of Operations Research*, 26:455–480, 1990.

[CL86] G. Chartrand and L. Lesniak. *Graphs & digraphs (Second edition)*. Wadsworth & Brooks/Cole Advanced Books and Software, Pacific Grove, California, USA, 1986.

[CL92] M. Chrobak and L.L. Larmore. The server problem and on–line games. In L.A. McGeoch and D.D. Sleator, editors, *On–line algorithms*, pages 11–64. AMS, 1992.

[CMT81] N. Christofides, A. Mingozzi, and P. Toth. State–space relaxation procedures for the computation of bounds to routing problems. *Networks*, 11:145–164, 1981.

[CR90] S. Chopra and G. Rinaldi. The graphical asymmetric travelling salesman polyhedron. In Kannan, R. and Pulleyblank,W.R., editors, *Integer Programming and Combinatorial Optimization*, pages 129–146. University of Waterloo, Waterloo, Ontario, 1990.

[CV93] J. Csirik and A. van Vliet. An on–line algorithm for multidimensional bin packing. *Operations Research Letters*, 13(3):149–158, April 1993.

[DDGS93] Y. Dumas, J. Desrosiers, E. Gelinas, and M.M. Solomon. An optimal algorithm for the traveling salesman problem with time windows. Technical Report G–91–51, GERARD , Montreal, Canada, 1991, revised 1993.

[DDS92] M. Desrochers, J. Desrosiers, and M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Reserach*, 40(2):342–354, 1992.

[DDSS93] J. Desrosiers, Y. Dumas, M. Solomon, and F. Soumis. Time constrained routing and scheduling. Technical Report G–92–42, GERARD , Montreal, Canada, 1993. (To appear as a chapter in the volume "Networks" in the *Handbooks in Operations Research and Management Science*).

[DL91] M. Desrochers and G. Laporte. Improvements and extensions to the Miller–Tucker–Zemlin subtour elimination constraints. *Operations Research Letters*, 10(1):27–36, 1991.

[DLL93] M. Dror, G. Laporte, and F. Louveaux. Vehicle routing with stochastic demands and restricted failures. *ZOR-Methods and Models of Operations Research*, 37(3):273–284, 1993.

[DLSS88] M. Desrochers, J.K. Lenstra, M.W.P. Savelsbergh, and F. Soumis. Vehicle routing with time windows: Optimization and approximation. In B.L. Golden and A.A. Assad, editors, *Vehicle routing: Methods and studies*, pages 65–84. North–Holland, 1988.

[DLT89] M. Dror, G. Laporte, and P. Trudeau. Vehicle routing with stochastic demands: Properties and solution frameworks. *Transportation Science*, 23(3):166–176, 1989.

[Dro93] M. Dror. Vehicle routing with uncertain demands as a stochastic program: properties of the corresponding solution. *European Journal of Operational Research*, 64(3):432–441, 1993.

[DW90] M. Dyer and L.A. Wolsey. Formulating the single machine sequencing problem with release dates as a mixed integer program. *Discrete Applied Mathematics*, 26:255–270, 1990.

[EL92] R. Euler and H. Le Verge. A complete and irredundant linear description of the asymmetric traveling salesman polytope on 6 nodes. Technical report, Working Paper, 1992.

[Esc88a] L.F. Escudero. An inexact algorithm for the sequential ordering problem. *European Journal of Operational Research*, 37:236–253, 1988.

[Esc88b] L.F. Escudero. On the implementation of an algorithm for improving a solution to the sequential ordering problem. *Trabajos de Investigacion-Operativa*, 3:117–140, 1988.

[Eul89] R. Euler. Odd cats and a general class of facet defining inequalities for the asymmetric travelling salesman polytope. Technical report, Carnegie Mellon University, Pittsburgh, USA, 1989.

[FGJ92] G. Fandel, Th. Gulledge, and A. Jones, editors. *New directions for Operations Research in Manufacturing*. Springer–Verlag, Berlin, 1992. Proceedings of a Joint US/German Conference, Gaithersburg, USA, July 30-31, 1991.

[FGL92] A. Friedman, J. Glimm, and J. Lavery. *The mathematical and computational science in emerging manufacturing technologies and management practices*. Society for Industrial and Applied Mathematics, Philadelphia, USA, 1992.

[Fin93] C.H. Fine. Developments in manufacturing technology and economic evaluation models. In S.C. Graves, A.H.G. Rinnooy Kan, and P.H. Zipkin, editors, *Logistics of production and inventory*, volume 4 of *Handbooks in Operations Research and Management Science*, chapter 14. North–Holland, Amsterdam, 1993.

[Fis89] M. Fischetti. Clique tree inequalities define facets of the asymmetric traveling salesman polytope. Technical report, Technical report OR/89/7(R), DEIS, University of Bologna, 1989. (To appear in *Discrete Applied Mathematics*).

[Fis91] M. Fischetti. Facets of the asymmetric traveling salesman polytope. *Mathematics of Operations Research*, 16:42–56, 1991.

[Fis92] M. Fischetti. Three facet–lifting theorems for the asymmetric traveling salesman polytope. In E. Balas, G. Cornuéjols, and R. Kannan, editors, *Integer Programming and Combinatorial Optimization*, pages 260–273. Carnegie Mellon University, Pittsburgh, USA, 1992.

[FKL+91] A. Fiat, R. Karp, M. Luby, L.A. McGeoch, D.D. Sleator, and N.E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12:685–699, 1991.

[FKT89] U. Faigle, W. Kern, and G. Turán. On the performance of on–line algorithms for partition problems. *Acta Cybernetica*, 9:107–119, 1989.

[FP90] L.F. Frantzeskakis and W.B. Powell. A successive linear approximation procedure for stochastic, dynamic vehicle allocation problems. *Transportation Science*, 24(1):40–57, February 1990.

[FRR90] A. Fiat, Y. Rabani, and Y. Ravid. Competitive k–server algorithms. In *Proceedings of 22nd Symposium on Theory of Algorithms*, 1990.

[FT92] M. Fischetti and P. Toth. An additive bounding procedure for the asymmetric TSP. *Mathematical Programming*, 53:173–197, 1992.

[FT94] M. Fischetti and P. Toth. A polyhedral approach for the exact solution of hard ATSP instances. Technical report, University of Bologna, April 1994.

[GJ77] M.R. Garey and D.S. Johnson. Two–processor scheduling with start–times and deadlines. *SIAM Journal on Computing*, 6:416–426, 1977.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[GLS88] M. Grötschel, L. Lovasz, and A. Schrijver. *Geometric algorithms and combinatorial optimization*. Springer Verlag, Berlin, 1988.

[GP77] M. Grötschel and M. Padberg. Lineare Charakterisierungen von Travelling Salesman Problemen. *Zeitschrift für Operations Research*, 21:1–7, 1977.

[GP85a] M. Grötschel and M. Padberg. Polyhedral computations. In E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, editors, *The Traveling Salesman Problem*. John Wiley & Sons, 1985.

[GP85b] M. Grötschel and M. Padberg. Polyhedral theory. In E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, editors, *The Traveling Salesman Problem*. John Wiley & Sons, 1985.

[GP86] M. Grötschel and W. Pulleyblank. Clique tree inequalities and the symmetric traveling salesman problem. *Mathematics of Operations Research*, 11:537–569, 1986.

[Grö77] M. Grötschel. *Polyedrische Charakterisierungen kombinatorischer Optimierungsprobleme*. Hain, Meisenheim am Glan, 1977.

[Grö92a] M. Grötschel. Discrete mathematics in manufacturing. In R. E. O'Malley, editor, *ICIAM91: Proceedings of the Second International Conference on Industrial and Applied Mathematics*, pages 119–145. SIAM, 1992.

[Gro92b] E.F. Grove. The harmonic online k–server algorithm is competitive. In L.A. McGeoch and D.D. Sleator, editors, *On–line algorithms*, pages 11–64. AMS, 1992.

[GRZ93] S.C. Graves, A.H.G. Rinnooy Kan, and P.H. Zipkin, editors. *Logistics of production and inventory*, volume 4 of *Handbooks in Operations Research and Management Science*. North–Holland, Amsterdam, 1993.

[Hel53] I. Heller. On the problem of shortest path between points, i. and ii. (abstracts). *Bulletin of the American Mathematical Society*, 59:551–552, 1953.

[HS92] M. Halldórsson and M. Szegedy. Lower bounds for on–line graph coloring. In L.A. McGeoch and D.D. Sleator, editors, *On–line algorithms*, pages 169–179. AMS, 1992.

[IEE90] *Computer Integrated Manufacturing*, 1990. Proceedings of Rensselaer's Second International Conference on Computer Integrated Manufacturing, May 21-23, 1990, Rensselaer Polytechnic Institute, Troy, New York.

[IW91] M. Imase and B.M. Waxman. Dynamic Steiner tree problem. *SIAM J. Disc. Math.*, 4(3):369–384, August 1991.

[Jac57] J.R. Jackson. Networks of waiting lines. *Operations Research*, 5:518–521, 1957.

[Jac63] J.R. Jackson. Jobshop–like queueing systems. *Management Science*, 10:131–142, 1963.

[Jai85] P. Jaillet. *The probabilistic traveling salesman problem*. PhD thesis, Department of Civil Engineering, MIT, 1985.

[JO88] P. Jaillet and A.R. Odoni. The probabilistic vehicle routing problem. In B.L. Golden and A.A. Assad, editors, *Vehicle Routing : Methods and Studies*, pages 293–318. Elsevier Science Publisher B.V. (North–Holland), 1988.

[JRR94] M. Jünger, G. Reinelt, and G. Rinaldi. The traveling salesman problem. Technical report, Universität zu Köln, February 1994. (To appear as a chapter in the volume "Networks" in the *Handbooks in Operations Research and Management Science*).

[JRT92] M. Jünger, G. Reinelt, and S. Thienel. Provably good solutions for the traveling salesman problem. Angewandte Mathematik und Informatik Report No. 92.114, Universität zu Köln, 1992.

[Kar91] R.M. Karp. An introduction to randomized algorithms. *Discrete Applied Mathematics*, 34:165–201, 1991.

[Kle92] J.P.C. Kleijnen. Simulation and optimization in production planning. *Decision Support Systems*, 8, 1992. (Only prelimary version available to the author).

[KM93] T. Krippner and H. Matejka. On-line Optimierung eines Testregallagers. Modellierung und Vergleich verschiedener Heuristiken. Master's thesis, Universität Augsburg, Germany, 1993.

[KMRS88] A.R. Karlin, M.S. Manasse, L. Rudolph, and D.D. Sleator. Competitive snoopy caching. *Algorithmica* **3**, pages 79–119, 1988.

[KP93] B. Kalyanasundaram and K. Pruhs. Online weighted matching. *Journal of Algorithms*, 14(3):478–488, 1993.

[KT92] H.A. Kierstead and W.T. Trotter. On–line graph coloring. In L.A. McGeoch and D.D. Sleator, editors, *On–line algorithms*, pages 85–92. AMS, 1992.

[Kuh55] H. Kuhn. On certain convex polyhedra (abstract). *Bulletin of the American Mathematical Society*, 61:557–558, 1955.

[KVV90] R. Karp, U. Vazirani, and V. Vazirani. An optimal algorithm for on–line bipartite matching. In *Proceedings, 22nd ACM Symposium on Theory of Computing*, pages 352–358, 1990.

[LL90] G. Laporte and F. Louveaux. Formulations and bounds for stochastic vehicle routing problems with uncertain supplies. In J.J. Gabewicz, J.-F. Richard, and L.A. Wolsey, editors, *Economic decision–making: games, econometrics and optimization*, pages 443–455. North–Holland, Amsterdam, 1990.

[LLM89] G. Laporte, F. Louveaux, and H. Mercure. Models and exact solutions for a class of stochastic location–routing problems. *European Journal of Operational Research*, 39:71–78, 1989.

[LLRS93] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. Sequencing and scheduling: Algorithms and complexity. In S.C. Graves, A.H.G. Rinnooy Kan, and P.H. Zipkin, editors, *Logistics of production and inventory*, volume 4 of *Handbooks in Operations Research and Management Science*, chapter 8. North–Holland, Amsterdam, 1993.

[LST89] L. Lovász, M. Saks, and W.T. Trotter. An on–line graph coloring algorithm with sublinear performance ratio. *Discrete Mathematics*, 75:319–325, 1989.

[MMS88] M.S. Manasse, L.A. McGeoch, and D.D. Sleator. Competitive algorithms for server problems. In *Proceedings of the 20th Annual ACM Symposium on the Theory of Computing*, pages 322–333, 1988.

[MMS90] M.S. Manasse, L.A. McGeoch, and D.D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11:208–230, 1990.

[MS91] L.A. McGeoch and D.D. Sleator. A strongly competitive randomized algorithm. *Algorithmica* **6**, pages 816–825, 1991.

[MS92] L.A. McGeoch and D.D. Sleator, editors. *On–line algorithms*. AMS, 1992. Proceedings of a DIMACS workshop, February 11-13, 1991.

[MTZ60] C.E. Miller, A.W. Tucker, and R.A. Zemlin. Integer programming formulations and traveling salesman problems. *J. Assoc. Comput. Mach.*, 7:326–329, 1960.

[NKT89] G.L. Nemhauser, A.H.G. Rinnooy Kan, and M.J. Todd, editors. *Optimization*, volume 1 of *Handbooks in Operations Research and Management Science*. North-Holland, Amsterdam, 1989.

[NW88] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, Chichester, 1988.

[Obe92] M. Ober. Asymmetrische Hamilton–Wege–Probleme mit Zeitfenstern. Master's thesis, Universität Augsburg, 1992.

[Par90] D.J. Parrish. *Flexible Manufacturing*. Butterworth-Heinemann, London, 1990.

[Pow86] W.B. Powell. A stochastic model of the dynamic vehicle allocation problem. *Transportation Science*, 20(2):117–129, May 1986.

[Pow87] W.B. Powell. An operational planning model for the dynamic vehicle allocation problem with uncertain demands. *Transp. Res. B-Vol.*, 21B(3):217–232, 1987.

[PR90] M. Padberg and G. Rinaldi. Facet identification for the symmetric traveling salesman polytope. *Mathematical Programming*, 47:219–257, 1990.

[PR91] M. Padberg and G. Rinaldi. A branch and cut algorithm for tyhe resolution of large–scale symmetric traveling salesman problems. *SIAM Review*, 33:60–100, 1991.

[PS88] M. Padberg and T.Y. Sung. An analytical comparison of different formulations of the travelling salesman problem. Technical report, Working Paper, New York University, 1988.

[Psa80] H.N. Psaraftis. A dynamic programming solution to the single vehicle many–to–many immediate dial–a–ride problem. *Transportation Science*, 14(2):130–154, 1980.

[Psa88] H.N. Psaraftis. Dynamic vehicle routing problems. In B.L. Golden and A.A. Assad, editors, *Vehicle Routing : Methods and Studies*, pages 223–248. Elsevier Science Publisher B.V. (North–Holland), 1988.

[PT91] W. Pulleyblank and M. Timlin. Precedence constrained routing and helicopter scheduling : Heuristic design. Technical Report Research report RC17154 (#76032), IBM T. J. Watson Research Center, Yorktown Heights, NY, USA, 1991. (To appear in Interfaces.).

[QS94] M. Queyranne and A. Schulz. Polyhedral approaches to machine scheduling. Technical Report 408/1994, Technische Universität Berlin, 1994.

[QW94] M. Queyranne and Y. Wang. Symmetric inequalities and their composition for asymmetric travelling salesman polytopes. Technical report, Working paper 90-MSC-002, University of British Columbia, Vancouver, 1990, revised 1994. (To appear in *Mathematical Programming*.).

[Rei91] G. Reinelt. TSPLIB – a travelling salesman problem library. *ORSA Journal on Computing*, 3:376–384, 1991.

[RT93] W.T. Rhee and M. Talagrand. On line bin packing with items of random size. *Mathematics of Operations Research*, 18(2):438–445, May 1993.

[Sav85] M.W.P. Savelsbergh. Local search for routing problems with time windows. *Annals of Operations Research*, 4:285–305, 1985.

[Sav90] M. W. P. Savelsbergh. An efficient implementation of local search algorithms for constrained routing problems. *European Journal of Operations Research*, 47:75–85, 1990.

[Sav91] M.W.P. Savelsbergh. The vehicle routing problem with time windows: minimizing route duration. Technical Report 91–03, Department of Mathematics and Computer Science, Eindhoven University of Technology, 1991.

[Sav94] M. Savelsbergh, 1994. Personal communication (1994) . School of Industrial and System Engineering, Georgia Institute of Technology, Atlanta, USA.

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, Chichester, 1986.

[SG83] W.R. Stewart Jr. and B.L. Golden. Stochastic vehicle routing: A comprehensive approach. *European Journal of Operational Research*, 14:371–385, 1983.

[Sim91] E.R. Sims, Jr. *Planning and Managing Industrial Logistics Systems*, volume 12 of *Advances in Industrial Engineering*. Elsevier, Amsterdam, 1991.

[Spi93] F.C.R. Spieksma. *Assignment and scheduling algorithms in automated manufacturing*. PhD thesis, University of Limburg, Maastricht, Netherlands, 1993.

[SS89] K.E. Stecke and R. Suri, editors. *Flexible Manufacturing Systems*, volume 8 of *Manufacturing Research and Technology*. Elsevier, Amsterdam, 1989. Proceedings of the Third ORSA/TIMS Conference on Flexible Manufacturing Systems: Operations Research Models and Applications.

[SSK93] R. Suri, J.L. Sanders, and M. Kamath. Performance evaluation of production networks. In S.C. Graves, A.H.G. Rinnooy Kan, and P.H. Zipkin, editors, *Logistics of production and inventory*, volume 4 of *Handbooks in Operations Research and Management Science*, chapter 5. North–Holland, Amsterdam, 1993.

[ST85] D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of ACM*, 28:202–208, 1985.

[ST91] R. Suri and S. de Treville. Full speed ahead. *OR/MS Today*, pages 34–42, June 1991.

[SV93] P. Solot and M. van Vliet. Analytic models for FMS design optimization : a survey. Technical report, EPFL Lausanne, January 1993.

[Swa91] J. Swain. World of choices. *OR/MS Today*, pages 81–103, October 1991.

[Tet90] U. Tetzlaff. *Optimal Design of Flexible Manufacturing Systems*. Physica–Verlag, Heidelberg, 1990.

[Tim89] M. Timlin. Precedence constrained routing. Master's thesis, Department of Combinatorics and Optimization, University of Waterloo, 1989.

[TK92] H. Tempelmeier and H. Kuhn. OR–Modelle zur Planung flexibler Fertigungssysteme. *OR Spektrum*, 14(4):177–192, 1992.

[Tre92] S. de Treville. Time is money. *OR/MS Today*, pages 30–34, October 1992.

[Tsi92] J. Tsitsiklis. Special cases of traveling salesman and repairman problems with time windows. *Networks*, 22:263–282, 1992.

Index