Railway scheduling problems and their decomposition

Dissertation
im Fachbereich Mathematik/Informatik der
Universität Osnabrück

Christian Strotmann

Osnabrück, Juli 2007

Abstract

Railway scheduling problems are quite popular scheduling and optimization problems which are treated in a large variety of papers and projects. Many special and even quite general situations have been investigated theoretically and also a variety of applied approaches tested on real-world instances has been developed.

This thesis mainly deals with the problem of scheduling trains in railway networks with respect to given routings, fixed minimal travelling times, and other constraints like time-windows. It combines the theory of some well-known scheduling models with its applications in railway scheduling. The railway scheduling problems considered in this work are closely related to job-shop scheduling problems with blocking and some additional constraints. Therefore part of this research is related to these shop scheduling problems. Theoretical scheduling models are extended, complexity results are derived and solution methods are proposed. Most results are applied to the considered railway scheduling problems. In addition to approaches which treat railway problems as a whole also decomposition methods for these problems and corresponding solution methods are presented. These solution methods are tested and compared with simple greedy procedures.

Contents

1 Introduction

Railway scheduling problems are the topic of a large variety of publications in the fields of applied mathematics, computer science and technical engineering. Quite abstract approaches treating special cases and also more general problems can be found as well as applied approaches solving real-world problems.

This thesis traces the idea to build abstract models of railway scheduling problems using graph models known from the classical scheduling literature and solve them with different approaches. Both railway scheduling problems as well as the underlying classical scheduling problems, namely job shop scheduling problems with blocking, are treated. This thesis is mainly based on two publications from Mascis and Pacciarelli [45],[44] and the EU-project COMBINE II [1] [26]. It supplements and continues the work done there.

The railway scheduling problems considered in this thesis consist of the problem to build schedules for a given set of trains moving in a railway network. The network is divided into block sections, which are small parts of the network, e.g. a certain segment of a track may define a block section. More precise descriptions and examples will be given later. Mainly two different types of block sections based on different safety systems are used by rail companies and thus are modelled here. Fixed block sections can contain only one train at a time, whereas in moving block sections trains may follow each other within the same section when keeping a certain safety distance. The rail network may contain block sections of both types. For each train moving through the network a route, i.e. a physically feasible sequence of block sections, where the train has to move through, is given. The difficulty now is to solve conflicts between trains, which use the same block sections, i.e. to choose feasible sequences for such trains. This problem is modelled in terms of a special graph model, namely the alternative graph model. Different additional constraints and objective functions are integrated.

In this thesis different approaches are presented in order to solve the considered problems. On one hand methods which treat the problems as a whole are proposed. These methods are formulated quite general, such that the methods themselves or their main basic ideas may be used for a variety of similar scheduling problems.

On the other hand also decomposition methods are proposed. These decomposition approaches are implemented specifically for railway scheduling problems and are based on a physical decomposition of the railway network, i.e. a division of the large railway network into smaller local networks. Such physical decompositions of railway network are practiced in real-world systems for example by the German railways.

[1] Christian Strotmann took part in this project as scientist as well as his supervisor Prof. Dr. Peter Brucker.

Some of the ideas and modelling details may be used to tackle other problems like for example supply chain scheduling or any kind of job-shop-like problems whose structure allows some physical decomposition.

As stated above for this thesis it is assumed that fixed routes are given for all trains moving in a railway network. Of course, also problems where routing decisions have to be made are worth considering. But strategies to integrate routing decisions into the proposed models and solution methods are postponed to further research. In this thesis it also is assumed that all data (minimal travelling times, etc.) are fixed. This is a good first approximation. By integrating a quite small amount of extra time into these travelling times, trains should be able to abide these times even if they have to brake or accelerate in between. This first approximation should be good enough to build train schedules. If more precise travelling times are needed which may also depend on the sequencing of trains at meeting points a travelling time calculation (simulation) could be integrated. Even this topic must be postponed to further research as the emphasis of this thesis lies on the scheduling aspect of railway problems.

This thesis is organized as follows. Problems and notations are described in Section 2. Both, shop scheduling as well as railway scheduling problems are introduced. Section 3 gives a survey on important existing literature concerning shop scheduling problems with blocking (and other constraints) and railway scheduling problems. In Section 4 graph models for shop scheduling and railway scheduling problems are described. Complexity results for both, shop scheduling problems as well as railway scheduling problems are given in Section 5. After that different solutions approaches like greedy heuristics, enumerative methods and local search heuristics are presented in Section 6. Existing approaches are described and ideas for new approaches are developed. A decomposition approach and corresponding solution methods are treated in Section 7. In Section 8 implementation details and computational results are given and discussed. Finally Section 9 contains some concluding remarks.

2 Problem description

In this section a description of the problems treated in this thesis is given. In the first part (2.1) different (job-)shop scheduling problems are formulated. In the second part (2.2) a detailed description of the considered railway scheduling problems is given.

2.1 Shop scheduling problems

Part of this thesis is dedicated to shop scheduling problems including blocking restrictions which are closely related to railway scheduling problems. In this section the classical job-shop scheduling problem and a variety of generalizations are introduced.

2.1.1 The classical job-shop problem

The classical job-shop problem may be formulated as follows. There are m machines $M_1, \ldots, M_m$ and n jobs $J_1, \ldots, J_n$. A job J_j consists of n_j operations $O_{ij} (i = 1, \ldots, n_j)$ which have to be processed in the order $O_{1j} \rightarrow O_{2j} \rightarrow \ldots \rightarrow O_{n_j j}$. Operation O_{ij} has to be processed on a dedicated machine $\mu_{ij} \in \{M_1, \ldots, M_m\}$ without preemption for $p_{ij} > 0$ time units. Each machine can process only one job at a time. Furthermore it may be assumed $\mu_{ij} \neq \mu_{i+1,j}$ for all $j = 1, \ldots, n$ and $i = 1, \ldots, n_j - 1$, i.e. machine repetition is forbidden. If not stated differently all data are assumed to be integer in this thesis.

In case of the classical job-shop problem sufficient buffer space is assumed to be available between the machines, i.e. a job can always wait in a buffer between the processing on two different machines.

To simplify the notation the operations are identified by numbers $1, \ldots, N$, where $N = \sum_{j=1}^{n} n_j$. The processing time of operation v is denoted by p_v, the machine on which it must be processed by $\mu(v)$, and the job it belongs to by $j(v)$. It is convenient to introduce two artificial operations 0 and $* = N + 1$ with processing time 0. These operations are called source and sink and model the start and the end of a schedule. For an operation $v = O_{ij}$ its successor is defined by $\sigma(v) = O_{i+1,j}$. If v is the last operation of a job its successor $\sigma(v)$ is defined to be $* = N+1$. Symmetrically the predecessor for an operation $v = O_{ij}$ is defined by $\psi(v) = O_{i-1,j}$. If v is the first operation of a job its predecessor $\psi(v)$ is defined to be the source 0.

A schedule for the problem is denoted by $\mathbb{S} = (s_v)$ where s_v is the starting time associated to operation v. The objective is now to determine a feasible schedule with minimal

makespan $C_{max} = \max_{j=1}^{n} C_j$, where C_j is the completion time of $O_{n_j j}$, i.e. the completion time of job J_j.

A special case of the JSP is the flow-shop problem (FSP). In a FSP each job consists of exactly m operations, and for any operation O_{ij} of a job J_j the associated machine is defined to be $\mu_{ij} = M_i$.

The classical JSP is known to be $\mathcal{NP}$-hard as it generalizes the classical flow-shop problem (see Brucker [9]). Many papers have been written on job-shop scheduling and a variety of algorithms in order to compute 'good' or even optimal solutions has been proposed (e.g. see Dell'Amico and Trubian [21], Aarts et al. [3], Brucker et al. [11], Nowicki and Smutnicki [53]).

To model additional constraints, such as release-dates, due-dates, deadlines, transportation delays, perishability constraints, etc., arbitrary time-lags may be introduced in connection with the JSP.

In a JSP with arbitrary time-lags additional restrictions of the form $s_u + l_{uv} \leq s_v$ with arbitrary (integer) l_{uv} are added. This problem covers the following special cases:

- a release-date r_v for the start of operation v,
- a deadline d_v for the start (or end) of operation v,
- no-wait constraints for operations. (A no-wait constraint means that an operation has to start immediately when its job predecessor has finished.)
- problems with objective function L_{max}.

For problems with no-wait constraints the concept of **no-wait** operations is introduced. A no-wait operation has to start immediately after the completion of its job predecessor. Let u be an arbitrary operation but the first of a job and $\psi(u)$ its job predecessor. If u is a no-wait operation then $s_u = s_{\psi(u)} + p_{\psi(u)}$ must hold, i.e. in addition to the usual constraint $s_u \geq s_{\psi(u)} + p_{\psi(u)}$ also $s_u \leq s_{\psi(u)} + p_{\psi(u)} \Leftrightarrow s_{\psi(u)} \geq s_u - p_{\psi(u)}$ must hold. A job-shop problem where all operations (but the first of a job) are no-wait operations is abbreviated by NWJSP (no-wait job-shop problem).

A variety of other restrictions are also special cases of arbitrary time lags. In the next part problems with another type of constraint, namely blocking restrictions, are described.

2.1.2 Job-shop problems with blocking

In this section job-shop problems with blocking constraints are introduced. A blocking constraint means that a job which has been completed on a machine cannot leave this machine until the next machine is free and thus blocks it. Such a situation occurs for example if no buffer space between the machines is available.

For problems with blocking constraints it is distinguished between two types of operations. A **blocking** operation blocks its machine even after its completion until its jobsuccesor starts on the associated machine. An **ideal** operation leaves its machine immediately after completion. This situation is known from the classical job-shop problem.

Using the notation introduced above for example the following situations occurs if two blocking operations u and v, which are not the last of their jobs, have to use the same machine M_k. If u precedes v on M_k, $s_v \geq s_{\sigma(u)}$ must hold as v has to wait until u leaves M_k, i.e. until $\sigma(u)$ starts. If otherwise v precedes u, $s_u \geq s_{\sigma(v)}$ must hold as u has to wait until v leaves the machine.

A job-shop problem where all operations are blocking except the last operation of each job is called blocking job-shop problem and abbreviated by BJSP. Of course an equivalent problem where all operations are blocking can be formulated by introducing at the end of each job an artificial operation, which has length zero and has to be processed on an always available machine.

Dealing with problems with blocking restrictions the question if so-called **swapping** of blocking operations is allowed or not arises. Swapping of operations may occur if a set of blocking operations exists where each one is waiting for a machine occupied by another operation in the set. Thus, the sole solution to this situation - note the blocking restriction - is that all operations of the set switch (swap) to their next machine simultaneously, i.e. the corresponding successor operations start simultaneously. This so-called swapping may be allowed or not.

If for an operation a swap is allowed this operation is called **swap operation** otherwise it is called **no-swap operation**. A scheduling problem where all blocking operations are swap (no-swap) operations is called swap (no-swap) problem. Note that in a BJSP the last operations of all jobs are ideal. Moreover swapping makes no sense for the last operations of jobs as they leave the system after their completion.

In the further sections the abbreviations IJSP for the classical (ideal) JSP, BWSJSP for the blocking JSP with swap allowed, and BNSJSP for the blocking job-shop problem with no-swap allowed are used.

2.2 Railway scheduling problems

In this section the railway scheduling problems treated in this thesis are described. Firstly a basic problem is introduced. After that this basic problem is extended by a variety of additional constraints.

2.2.1 The basic railway scheduling problem

There is a close similarity between railway scheduling problems and job-shop scheduling problems with blocking and no-swap allowed. Usually, a railway network is divided into block sections. Block sections correspond with machines in the job-shop problem. A train going on a fixed route from some origin to some destination corresponds with a job. A route is a sequence of block sections and passing a block section is an operation of the train. The minimal travel time needed to pass a block section is the processing time of the operation. Depending on the applied safety system additional constraints, such as blocking constraints, must be satisfied.

Railway companies apply different safety concepts. Some important concepts which will be discussed in this work, are:

(1) fixed block safety systems (with fast and slow trains),

(2) moving block safety systems,

(3) a combination of these two safety systems.

In a fixed block safety system a train cannot enter a block section if this block section is still occupied by another train. Note, that in this thesis it is assumed, that any train fits in every fixed block section, where it has to move through. This is a reasonable assumption as it holds in most real-world systems, too. (Of course, a generalization where trains occupy two or more block sections could be modelled by adjusting slightly the models and techniques presented later. But such considerations are not subject of this thesis.)

When a train leaves a fixed block section this takes a small amount $\varepsilon > 0$ of time, i.e. $\varepsilon > 0$ is the time period in which the train is present in two block sections. Thus, a subsequent train can enter the block section only $\varepsilon > 0$ after the entrance of the previous train in its next block section. This constraint covers the no-swap constraint, as then trains cannot swap because of these temporal constraints. The no-swap constraint is particularly important for two trains going in opposite direction on the same line, as they cannot swap for physical reasons.

In order to create a more detailed model different weights $\epsilon_i > 0$ for different trains or even $\epsilon_{ij} > 0$ for different train operations may be introduced. This does not modify the argumentation considerably. (As in this thesis it is assumed, that any train fits in every fixed block section, where it has to move through, it also can be assumed that the exit time of a train from a certain fixed block section is smaller than the minimum travelling time in its next fixed block section.)

Besides the described (slow) trains also **fast** trains can move through a rail network with fixed block safety system. In this thesis a fast train is a train which is only allowed to enter the next block section if the next two block sections are not occupied by other trains. The definition of fast trains in this thesis is based on a railway signalling system with signals which may be green, yellow or red. A red signal indicates the section behind the signal to be occupied. A yellow signal means that the next block section is free, but the second section behind the signal is occupied. A green signal indicates the next two block sections to be free.

Of course, other types of fast trains, which need more than two free block sections could be considered. Such trains could be integrated by modifying the model presented in Section 4.2 slightly.

Another safety system is the moving block safety system. In real-world rail networks this safety system is based on satellite control or on digital radio transmission. In such a system trains can follow each other within the same (moving) block section if they keep a sufficient safety distance.

Like for fixed block sections it can be assumed that any train fits in every moving block section, i.e. any block section is long enough for each train. Additionally in this thesis it is assumed that all moving block sections are one-way single-track sections, i.e. no switches are present in such sections and trains are only allowed to move through the sections in one predefined direction. More general models can be implemented by dividing moving block sections into different parts, i.e. sections for single lines, switches, etc., and building a route for each train through these sections.

The problem is now to determine sequences for trains which use the same block sections such that a corresponding schedule is feasible. This problem is called feasibility-problem. If additionally an objective function is given, the problem of determining a feasible solution with minimal objective value is an optimization-problem.

2.2.2 Additional constraints and objective functions

Railway problems may include a large variety of additional constraints, such as:

(1) (lower and) upper bounds for travelling times of trains,

(2) release-dates and deadlines for trains,

(3) starting and ending constraints for trains,

(4) connection constraints between trains, and

(5) out-of-service intervals for certain block sections.

Lower bounds for travelling times of trains are the minimal travelling times already described above. An upper bound for the travelling time of a train through a certain block section may result from a corresponding minimal speed restriction.

Release-dates and deadlines may be given for the entrances of trains in certain block sections. These may for example result from restrictions on arriving or departure times of trains at stations.

A starting constraint for a train in the first block section on its route means that this train must be the first one in this block section. Such a constraint may result from a situation where the train is physically already present in this block section at the start of a schedule. Symmetrically an ending constraint for a train in the last block section on its route means that this train must be the last one in this block section. Such a constraint may result from a situation where the train must physically stay in this block section until the end of a schedule. Of course, even sequences for sets of trains starting or ending in certain block sections may be predefined.

Connection constraints between trains may for example occur if passengers of one train should be able to catch another at a station. Then one train has to wait for the arrival of another before leaving the station.

An out-of-service interval for a block section may result from restoration activities and means that this block section is out of service, i.e. is not usable, for a certain time period.

Of course, other constraints could be included in railway problems but are not considered in this thesis.

Again both the feasibility- and the optimization-problem may be treated. In case of the optimization-problem additionally different objective functions like for example $C_{\max}$ and $L_{\max}$ may be considered.

3 Literature review

In the literature a large variety of papers dealing either with any kind of job-shop scheduling problems or with railway scheduling problems can be found. But the combination of job-shop scheduling theory and its application in railway scheduling, like investigated in this present thesis, is considered only in a few papers. The content of this section is twofold. The first part of this section summarizes literature dealing with shop scheduling problems and especially those with blocking constraints. In the second part some important papers and results concerning railway scheduling are reported.

3.1 Shop scheduling (with blocking or no-wait constraints)

The classical job shop scheduling problem is subject of a large number of scientific papers. Many extensions, modifications, and also special cases of this problem are treated in literature. Both, complexity results as well as solution procedures are derived. In this thesis only an overview on some important milestones in the research of job-shop scheduling and especially on problems with blocking constraints is given.

In the last decades the model which was most frequently applied when considering the classical job-shop and related problems was the *disjunctive graph model*, which was proposed by Roy and Sussmann in 1964 [62]. Based on this and related graph models some famous solution procedures have been developed like the branch and bound procedures by Carlier and Pinson [17] and Brucker et al. [11] and the tabu search approaches by Dell'Amico and Trubian [21] and Nowicki and Smutnicki [52, 53].

Generalizations of the classical job-shop problem like problems with transport robots have been studied for example in Knust [36] and Strotmann [65]. Local search heuristics for problems with multi-purpose machines can be found for example in Mastrolilli and Gambardella [46] and Hurink et al. [30]. A survey on a large variety of shop scheduling problems including extensions and special cases of job-shop problems and also complexity results can for example be found in Brucker and Knust [12]. The web-pages of the OR-group from the University of Osnabrueck [57] summarize the latest complexity results for flow-shop and job-shop problems with or without preemption and other constraints.

Another class of extensions of shop problems are problems with limited buffer space. Note, that these problems may contain blocking problems as special cases, as in most considerations even problems with zero buffer capacity are included. Papadimitriou and Kannelakis [55] investigate flow-shop problems with limited intermediate buffers, which are of first-in-first-out type and cannot be bypassed. They derive complexity results and

develop a heuristic for the two machine problem with a single-capacity-buffer between the machines. Brucker et al. [10] provide solution procedures for both, flow shop and job shop problems and even for different buffer models. Nieberg [50] develops local search heuristics for flow-shop and job-shop problems with pairwise buffers, i.e. a buffer for each ordered pair of machines. In all these publications the swapping of jobs (operations) is allowed. Swapping occurs if a set of jobs $J_{j_1}, \ldots, J_{j_k}$ is processed on a set of machines $M_{m_1}, \ldots, M_{m_k}$ and the next machines where the jobs have to be processed are $M_{m_{i+1}}$ for jobs $J_{j_i}, \;\; i = 1, \ldots, k-1$ and M_{m_1} for job J_{j_k}. Thus, the sole solution to this situation - note the blocking restriction - is that all jobs of the set switch to their next machine simultaneously. This so-called swapping may be allowed or not. Note, that for problems with limited buffer space the buffers may also be involved in such swapping situations.

In case of no-wait problems a variety of complexity results can be found in the literature. Röck [58, 59, 60] derived complexity results for no-wait flow-shop problems with different objective functions, like e.g. C_{max}, L_{max}, and other constraints like unit processing times and different resource constraint environments. The complexity of 2-machine and 3-machine no-wait job-shops was for example investigated by Sahni and Cho [63] and Sriskandarajah and Ladet [64]. A comprehensive summary of complexity results in this field can be found in Hall and Sriskandarajah [28].

Important for this thesis are publications on job-shop and (for complexity issues) flow-shop problems with blocking constraints. For such problems only a few publications can be found. Kamoun and Sriskandarajah [34] show NP-completeness for a two-stage flow-shop problem with two machines at the second stage, blocking, and minimum cycle time objective function. Complexity results for 2-machine flow-shops with blocking and setup times are presented by Logendran and Sriskandarajah [42]. In their publication a machine requires a setup time before processing a job. The setup times can be performed in anticipation of an arriving job, called anticipatory setup. Martinez et al. [43] study the complexity of flow-shop problems including another type of blocking constraint. There an operation blocks its machine until the successor operation leaves its machine. Of course, the complexity of some job-shop problems with blocking can be derived from the results above by simple special case reduction.

Looking for solution procedures only a few papers can be found for shop-scheduling problems with no-wait or blocking restrictions. Gilmore and Gomory [27] present a polynomial algorithm for the problem $F2 \mid no-wait \mid C_{max}$. Kravchenko [38] develops a polynomial algorithm for a specific two-machine no-wait job-shop scheduling problem. A flow-shop problem with blocking arising in an industrial context is studied by Mc Cormick et al. [49]. Mascis and Pacciarelli [45] develop greedy heuristics and branch&bound methods for job-shop problems with blocking or no-wait constraints. As a basis of their considerations they introduce the *alternative graph model*. This model is

a generalization of the disjunctive graph model (see above). The alternative graph model also builds the basis for the following considerations in this present thesis. In [44] Mascis and Pacciarelli model a variety of problems by means of alternative graphs. They model for example problems arising in steel works and even railway scheduling problems can be found.

Mati et al. [47] present a special tabu search procedure for the blocking job-shop problem. A main part of their paper is based on the geometric approach for the job-shop problem with two jobs which was developed by Brucker [8]. This geometric approach itself is based on a publication from Akers and Friedman [5]. Mati et al. use a version of the geometric approach to repair an unfeasible solution resulting by a modification of a feasible solution.

Summarizing the state of the research done in the field of shop problems with blocking or no-wait constraints, the following can be stated. A lot of work was done in the field of no-wait problems, most of it dedicated to complexity issues for flow-shops. Only a few publications deal with blocking job-shops and corresponding solution procedures.

3.2 Railway scheduling

Many papers and also research projects deal with questions how to schedule trains in rail networks. The variety of problems considered in the literature contains quite special applications as well as very complex problems dealing with large networks and many different constraints. In this thesis only a small survey with respect to some important papers and projects can be given.

Most of the publications dealing with railway problems treat special problems arising in the context of large railway networks. Especially variations of the problem of scheduling trains on a single-line track are investigated in many papers. For example Higgins et al. [29] develop branch&bound methods for railway problems with a single track and some sidings, where trains can pass each other. Cai and Goh [13] consider single track problems including sidings (here called passing loops), too. They develop a fast heuristic based on an integer-programming formulation. They also provide a proof for NP-completeness of the considered problem. In Brännlund et al. [7] single track problems with sidings are modelled as integer programming problems and based on this Lagrangian relaxation solution approaches are developed. Nou [51] considers such single track problems, too, and provides heuristics based on Lagrangian relaxation. For one-way single track problems Caprara et al. [14] suggest heuristic algorithms which are based on multigraph-formulation and Lagrangian relaxation. Carey and Lockwood [16] provide

solutions methods for single track lines including stations. Based on a mixed integer programming formulation they apply heuristic decomposition, i.e. trains are dispatched one by one and then are redispatched in order to improve solutions.

Another subproblem of complex railway problems is treated in Zwaneveld et al. [67]. In their publication models and algorithms for routing trains through railway stations are given. Other publications dealing with railway problems for stations are from Carey and Carville [15]. Here, the problem of choosing platforms and routes for trains in stations is treated.

In the scheduling literature the problem of scheduling trains in a large railway network with respect to some additional constraints and especially the problem of finding feasible solutions for some of these problems is treated only in a few publications. Of course a variety of quite applied publications deals with problems like how to operate a complex railway network, how to re-schedule trains, etc., but most of them propose models designed to support human dispachters (see e.g. Jovanović and Harker [32, 33], Kraay and Harker [37]). Models and approaches for real-time scheduling of trains can be found in Rodriguez [61]. Fully automated systems and also the basic models are treated less frequently. A branch&bound method for the problem of dispatching trains in a large network according to a given timetable is presented in Dessouky et al. [22]. Another approach for scheduling trains in rail networks is developed in Dorfman and Medanic [23]. Some publications treat the problem of re-scheduling trains when perturbations occur. An heuristic approach for such problems can be found in Törnquist [66]. A recent publication based on the alternative graph model is the paper from D'Ariano et al. [20]. There a branch&bound procedure for scheduling trains is presented.

A survey of a large variety of publications treating railway problems and developing corresponding optimization methods is given in Cordeau et al. [19]. Some latest results concerning a large variety of problem types arising in the context of railways can be found in[1] and [2], where for example topics as planning problems in general, decision support systems, safety aspects, passenger interface systems, timetabling problems, power supply, etc. are treated.

The author of this thesis took part in the EU-project COMBINE II, which is a follow-up project of the EU-project COMBINE. COMBINE II deals with railway problems and also decomposition approaches for railway scheduling problems. It provides an approach in order to support human dispatchers operating a complex railway network. Additionally basic models and concepts also suitable for designing automated systems are developed. The considerations of the project are based on the alternative graph model from Mascis and Pacciarelli [44, 45] (see above). These authors took part in the project, too. Details of the EU-project COMBINE II can be found in [26]. More detailed information on

the COMBINE II TMS (Traffic Management System) can be found in Mazzarello and Ottaviani [48].

As stated above the models in this thesis are based on the alternative graph model from Mascis and Pacciarelli [44, 45]. Some of the basic ideas of the decomposition approaches presented here were also used in the COMBINE II project, but Section 7 of the present thesis goes one further step. Here basic models and solution methods for a completely automated system are developed.

4 Graph models

In this section both job-shop as well as railway scheduling problems are modelled in terms of graphs. In the first part the disjunctive graph model for the classical job-shop problem is introduced and extended to more general problems including blocking restrictions. The second part describes a graph model for the railway scheduling problems introduced in Section 2.2.

4.1 Modelling job-shop scheduling problems

In this section an introduction to graph models for shop scheduling is given. These models are the basis for the following considerations about both, solution methods for some shop scheduling problems and models and solution methods for railway scheduling problems. Firstly the classical job-shop problem (IJSP) and the well-known disjunctive graph model are introduced. After that the basic model is generalized by introducing arbitrary time-lags. Based on this the alternative graph model is described and job-shop problems with blocking constraints are modelled in terms of the alternative graph model.

4.1.1 The disjunctive graph model

The classical job-shop problem can be formulated in terms of the disjunctive graph model, which was developed by Roy & Sussman [62] and later successfully applied to the job-shop problem and its extensions (see also Brucker [9]).

A disjunctive graph G consists of a set V of nodes, a set C of directed arcs (conjunctions), and a set D of undirected arcs (disjunctions). Considering the JSP the corresponding disjunctive graph $G = (V, C, D)$ is defined as follows:

- The set V of nodes represents the set of all operations. For the two artificial operations 0 and $*$ two artificial nodes, the source node 0 representing the start of a schedule and the sink node $* = N + 1$ representing its end, are introduced. Thus, $V = \{0, 1, \ldots, N, N + 1\}$.
- The set C of conjunctions represents the set of precedence constraints between consecutive operations of the same job. For each operation v a conjunction $v \rightarrow \sigma(v)$ is introduced. This conjunction is weighted by p_v. (Note that conjunctions of the form $v \rightarrow *$ are included.) Additionally conjunctions $0 \rightarrow v$ are introduced for each operation v which is the first of its job $j(v)$. These conjunctions are weighted

by 0. A conjunction $u \rightarrow v$ with weight p_u means that in a feasible schedule $\mathbb{S} = (s_\nu)$ the condition $s_u + p_u \leq s_v$ must be satisfied.

- The set D of disjunctions represents the different orders in which jobs on the same machine may be scheduled. It consists of undirected arcs between all pairs of operations which have to be processed on the same machine, i.e. for each pair u, v of operations with $\mu(u) = \mu(v)$ (and $j(u) \neq j(v)$) D contains an undirected arc $u - v$ weighted by the pair (p_u, p_v) indicating that either $s_u + p_u \leq s_v$ or $s_v + p_v \leq s_u$ must be satisfied.

With this graph model the problem of finding a feasible schedule for the job-shop problem is equivalent to the problem of fixing a direction for each disjunction such that the corresponding graph contains no cycles of positive length. In this work a positive cycle means a directed cycle with the sum of all its arc lengths being positive. When fixing the direction of arc $u - v$ to $u \rightarrow v$ ($v \rightarrow u$) the weight p_u (p_v) becomes relevant. A set S of fixed disjunctions is called **selection**. The corresponding selection is called **complete**, iff for each disjunction a direction has been fixed. It is **consistent**, iff the graph $G(S) = (V, C \cup S)$ corresponding to a selection S contains no positive cycle. If S_e is a complete consistent selection with $S \subset S_e$, S_e is called an **extension** of S. An **optimal extension** of S is an extension with minimal objective function value. Given a selection S the length of a longest (directed) path between nodes i and j in $G(S)$ (the sum of all arc weights on such a path) is denoted by $l^S(i, j)$.

On one hand each complete consistent selection S represents a feasible schedule. A feasible starting time s_i for an operation i then can be given by the length $l^S(0, i)$ of a longest $0 - i$-path in $G(S)$. (W.l.o.g. the starting time s_0 of the source operation 0 may assumed to be 0.) A path from 0 to $*$ in $G(S)$ with length s_*, i.e. a longest $0 - *$-path, is called **critical path**, . It determines the makespan C_{max} of the solution.

The starting times for a corresponding earliest-start-schedule (ESS) can be calculated by longest-path calculation in the graph $G(S)$, e.g. by a longest-path-version of the Floyd-Warshall algorithm (see Ahuja et al. [4]). Note that for regular objective functions, like C_{max} and L_{max} an ESS is an optimal one among all schedules respecting the chosen constraints.

On the other hand for each feasible schedule $\mathbb{S}$, a complete selection can be constructed by choosing for each disjunction a direction, which is respected in the schedule (as one direction of each disjunction must be respected). Of course, this complete selection must be consistent. Otherwise the resulting graph would contain a positive cycle, and thus, a feasible schedule could not fulfill all chosen constraints, which is a contradiction. The ESS corresponding to this complete consistent selection is not worse than $\mathbb{S}$ and therefore

a complete consistent selection representing an optimal schedule (namely the corresponding ESS) always exists.

Thus, the problem of finding an optimal solution for the problem is equivalent to the problem of finding a complete consistent selection S for the corresponding graph which minimizes the starting time s_* of the sink.

Note that for the classical job-shop problem a consistent selection S means that $G(S)$ is acyclic, i.e. contains no directed cycle, as any cycle would be a positive one.

Additional constraints, such as release-dates, duedates, deadlines, transportation delays, perishability constraints, no-wait constraints etc. can be modelled by introducing arbitrary time-lags of the form $s_u + l_{uv} \leq s_v$ with arbitrary (integer) l_{uv}. They are represented by arcs $u \rightarrow v$ with weights l_{uv} (precendence constraints including the start or end operations 0 and $* = N + 1$ are also possible).

As already mentioned above (Section 2.1.1) this problem covers a variety of special cases, e.g.:

- A release-date r_v can be modelled by a precedence constraint $0 \rightarrow v$ with $l_{0v} = r_v$.
- A deadline d_v for the start (end) of operation v can be modelled by a precedence constraint $v \rightarrow 0$ with $l_{v0} = -d_v$ ($l_{v0} = -d_v + p_v$).
- A no-wait constraint for an operation i (with job predecessor $\psi(i)$) can be modelled by introducing a precedence constraint $i \rightarrow \psi(i)$ with weight $-p_{\psi(i)}$. This models $s_{\psi(i)} \geq s_i - p_{\psi(i)}$ and thus together with $s_i \geq s_{\psi(i)} + p_{\psi(i)}$ it models the no-wait constraint $s_i = s_{\psi(i)} + p_{\psi(i)}$.
- Problems with objective function L_{max} can be formulated similar to C_{max}-Problems by modifying the weights of arcs from the last operations of all jobs to the sink. If v is the last operation of a job the weight of arc $v \rightarrow *$ is set to or $p_v - q_v$, where q_v is a due-date for the completion time of operation v. Then, minimizing the length of a critical path (or the starting time of $*$) in the corresponding graph is equivalent to minimizing L_{max}.

The introduction of arbitrary time-lags and some other requirements leads to a more general graph model, namely the alternative graph model.

4.1.2 The alternative graph model - Modelling job-shop problems with blocking

As mentioned above many applications of the JSP require a more general and variable modelling. For example blocking constraints (see Section 2.1.2) cannot be modelled in terms of the disjunctive graph model. A graph model of a situation where two blocking operations u and v have to be processed on the same machine M_k is depicted in Figure 1.

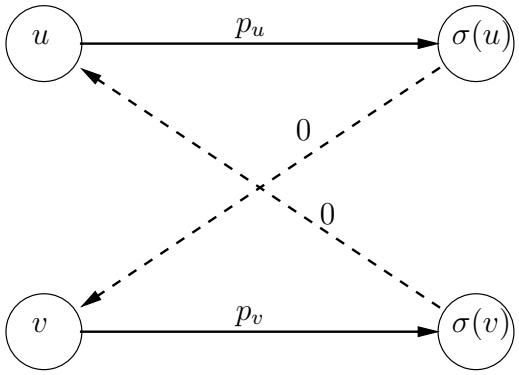

Figure 1: Two blocking operations to be processed on the same machine

If u precedes v on M_k, $s_v \geq s_{\sigma(u)}$ must hold as v has to wait until u leaves M_k, i.e. until $\sigma(u)$ starts. If otherwise v precedes u, $s_u \geq s_{\sigma(v)}$ must hold. Thus, these constraints can be modelled as the pair of dashed arcs depicted in Figure 1, where one arc has to be chosen in order to fix a certain processing sequence.

In order to formulate a more general graph model which covers these kinds of constraints the so-called **alternative graph model** is introduced. The alternative graph model is a generalization of the disjunctive graph model and was developed by Mascis and Pacciarelli [44]. An alternative graph $G = (V, C, A)$ consists of a set V of nodes, a set C of fixed arcs (conjunctions) $u \rightarrow v$ with arbitrary weights l_{uv}, and a set A of pairs of alternative arcs (alternative pairs) $\{u \rightarrow v, h \rightarrow k\}$ with arbitrary weights a_{uv} and a_{hk}.

Similar to the case of a disjunctive graph a set S which contains at most one arc of each alternative pair is called a **selection**. A selection is called **complete** iff it contains exactly one arc out of each alternative pair. Given a selection S let $G(S) = (V, C \cup S)$. A selection S is called **consistent** iff the corresponding graph $G(S)$ contains no positive cycle. The definitions of extension and optimal extension are applied to the alternative graph model simultaneously.

Obviously the alternative graph model covers the disjunctive graph model as a disjunction $u - v$ can be modelled as a pair $\{u \rightarrow v, v \rightarrow u\}$ of alternative arcs. Especially arbitrary time-lags are included in the alternative graph model as arbitrary arc weights are allowed.

The alternative graph model is quite general. It can be used to model a large variety of scheduling problems including blocking restrictions, arbitrary time-lags and other constraints (see also Mascis & Pacciarelli [44]). A generalization of the alternative graph model described above is introduced by Kampmeyer [35]. There pairs of alternative sets of arcs are considered instead of pairs of arcs. Then situations can be modelled where a machine is blocked by an operation until a set of successors has been started. Such constraints arise for example in problems where statements of computer programs have to be scheduled on different units of a computer processor. There certain data has to be stored in a register unit (and thus blocks it) until all statements using this data have been started.

The alternative graph model can especially be used to model job-shop problems with blocking constraints (see also Mascis & Pacciarelli [45]). As for the classical job-shop problem and the disjunctive graph model for each operation a node is introduced in the alternative graph and nodes 0 and $* = N + 1$ are added. The set of fixed arcs for the alternative graph equals the set of conjunctions of the disjunctive graph for the corresponding classical JSP. In contrast to disjunctive arcs now processing sequences on machines are modelled by alternative pairs. Different types of operations (blocking or ideal) require different pairs of alternative arcs. The alternative pair for two blocking operations to be processed on the same machine was already introduced in Figure 1. The corresponding situation where one of the operations is ideal is shown in Figure 2. The situation where both operations are ideal is known from the classical job-shop.

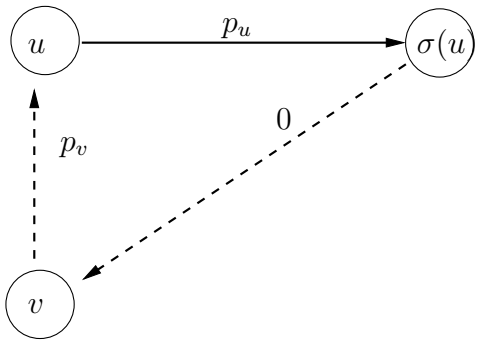

Figure 2: Pair of alternative arcs for a blocking operation u and an ideal operation v.

In general the alternative graph $G = (V, C, A)$ for a JSP with blocking is as follows:

- The set V of nodes represents the set of all operations (i.e. the starting times of operations). For the two artificial operations 0 and $*$ two artificial nodes, the source node 0 representing the start of a schedule and the sink node $* = N+1$ representing its end, are introduced. Thus $V = \{0, 1, \ldots, N, N + 1\}$.

- The set C of conjunctions represents the set of precedence constraints between consecutive operations of the same job. For each operation v a conjunction $v \rightarrow \sigma(v)$ is introduced. This conjunction is weighted by p_v. (Note that conjunctions of the form $v \rightarrow N+1$ are included.) Additionally conjunctions $0 \rightarrow v$ are introduced for each operation v which is the first of its job $j(v)$. These conjunctions are weighted by 0.

- The set A of pairs of alternative arcs represents the different orders in which jobs on the same machine may be scheduled. It consists of pairs of alternative arcs for all pairs u and v of operations (with $j(u) \neq j(v)$) which have to be processed on the same machine. If both operations u and v are blocking the pair of arcs from Figure 1 is introduced. If only u is blocking the pair of arcs from Figure 2 is introduced. The pair $(u \rightarrow v, v \rightarrow u)$ with weights p_u and p_v belongs to A if u and v are ideal.

As for the classical JSP each complete consistent selection for the proposed alternative graph is associated with a feasible schedule for the corresponding job shop problem with blocking and vice versa.

Thus the problem of finding a feasible solution is equivalent to the problem of finding a complete consistent selection. Again finding an optimal solution for the C_{max} and the L_{max}-problem corresponds to the problem of finding a complete consistent selection which minimizes s_*.

The data for an example of a blocking job-shop problem (JSPB) is given in Table 1. The corresponding alternative graph modelling this example is depicted in Figure 3. Note that due to more clarity not all alternative arcs are shown.

	J_1			J_2		J_3		
Operation	1	2	3	4	5	6	7	8
Machine	1	2	3	1	2	3	1	2
Processing time	2	2	2	1	2	1	1	2

Table 1: Example of a job-shop problem with blocking.

In Figure 4 this graph together with a corresponding complete consistent selection is shown. A critical path determining the makespan of an associated schedule is drawn in bold arrows.

Figure 5 depicts the ESS associated with the selection in Figure 4. Here a hatched area represents the time period where a blocking operations v stays on machine $\mu(v)$ after its

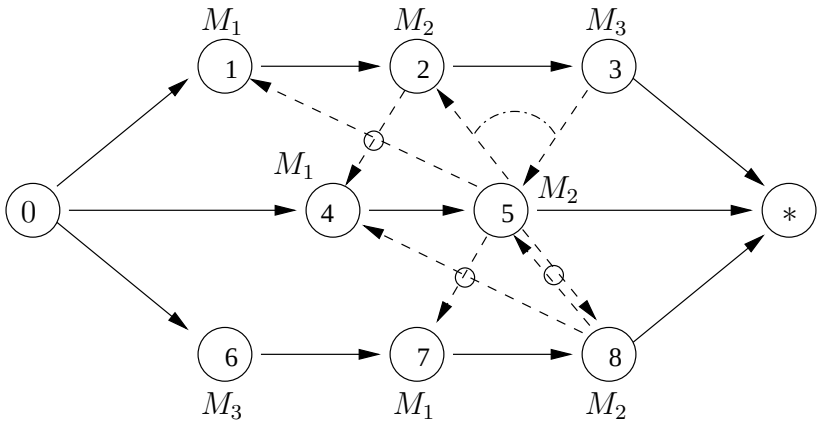

Figure 3: Alternative graph for the example from Table 1.

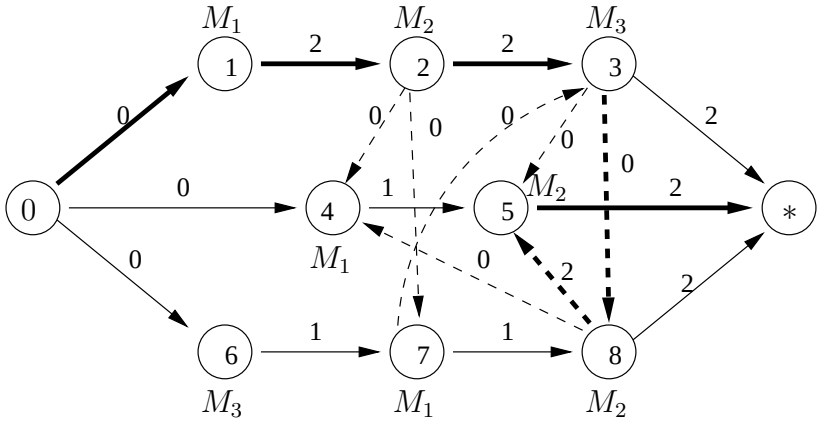

Figure 4: A feasible solution (complete consistent selection) corresponding to Figure 3.

completion. Operation v blocks this machine while waiting for the next machine of the corresponding job $j(v)$ to become available for its successor operation $\sigma(v)$.

As mentioned above dealing with problems with blocking restrictions the question if the swapping of blocking operations is allowed or not arises.

A swapping situation results in a cycle of alternative arcs in the corresponding solution graph. These alternative arcs represent blocking and therefore have weight 0. Thus, the situation results in a zero length cycle, which is allowed. If swapping of certain blocking operations is forbidden (no-swap operations) an (arbitrary) small weight $\epsilon > 0$ is put on the corresponding alternative arcs (instead of zero weights) in order to make the above mentioned cycle positive and thus the corresponding solution infeasible. Note, that still all data may be assumed to be integer as all data could be multiplied by a suitably large

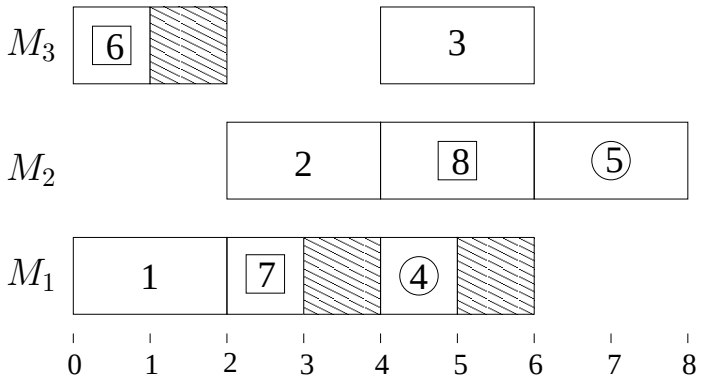

Figure 5: Gantt-Chart for the solution depicted in Figure 4.

positive integer N and the small $\epsilon > 0$ could be replaced by 1.

In a no-swap problem all alternative arcs of blocking operations get an (arbitrary) small weight $\epsilon > 0$, whereas in swap problems these weights are zero. Note that in a JSPB the last operations of all jobs are ideal and thus no weights of corresponding alternative arcs have to be modified.

A situation where two operations swap is depicted in Figure 6. Of course, such a swap is prevented by introducing $\epsilon > 0$ as the zero-length-cycle then becomes positive and thus, a selection including such a situation is inconsistent.

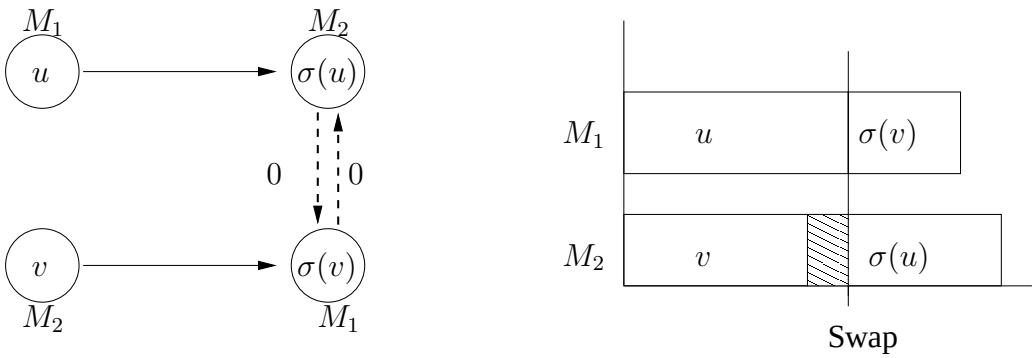

Figure 6: Situation where two operations swap.

For the problem instance introduced above (Table 1, etc.) a schedule where a swap of operations arises at time 4 is depicted in Figure 7. Here operations 2,4, and 6 swap at time 4 and thus operations 3, 5 and 7 have to start simultaneously at this time. The corresponding graph is shown in Figure 8. The zero length cycle indicating the swap is drawn in bold arcs.

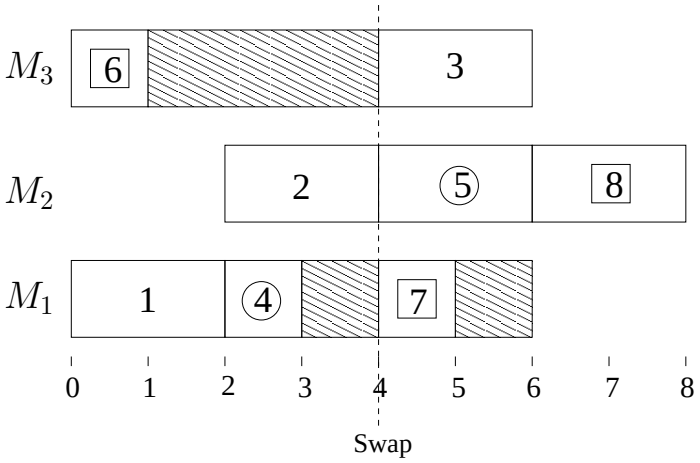

Figure 7: Schedule for example from Table 1. Swap at time 4.

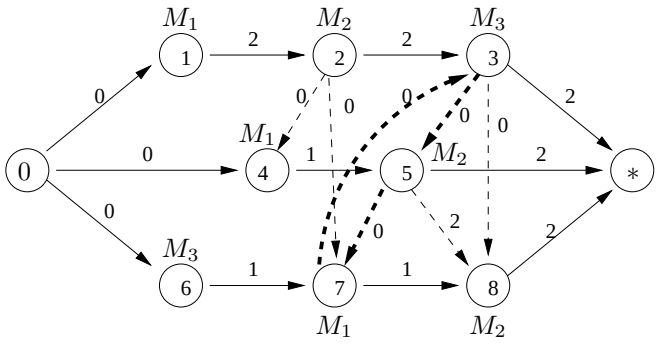

Figure 8: Graph corresponding to schedule in Figure 7.

A main difference between the two models - alternative and disjunctive graphs - is the following. Considering the disjunctive graph for a classical job-shop problem for any consistent selection an extension always exists whereas this is not the case for a general alternative graph and even not for the blocking job-shop problems BNSP and BWSP (see also Mascis and Pacciarelli [44]). An example of a consistent selection for a blocking job-shop problem having no extension is depicted in Figure 9. Two jobs have to be processed on three machines in the same order. The dashed arcs are the alternative arcs chosen in the given consistent selection. As any choice of the remaining alternative pair (dotted arcs) leads to a positive cycle, no extension exists.

A more detailed discussion on the complexity of a variety of problems which can be modeled by alternative graphs will follow later in Section 5.

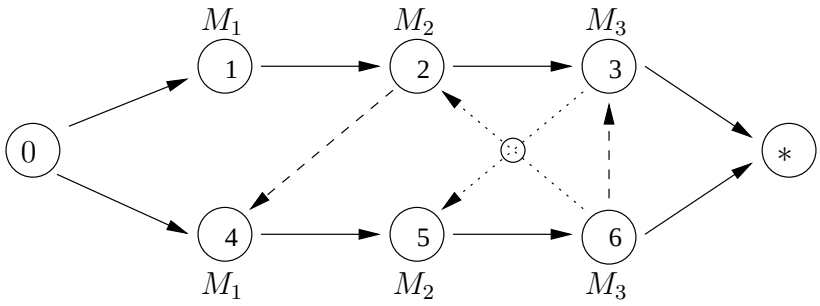

Figure 9: Blocking JSP where consistent selection (dashed arcs) has no extension.

4.2 Modelling railway scheduling problems

In this section the alternative graph model for railway scheduling problems is described. Firstly the basic problem is modelled in Section 4.2.1. In Section 4.2.2 the model for different additional constraints and different objective functions is discussed. The basic model and also some of the additional constraints were already introduced by Mascis and Pacciarelli [44] and applied to the EU-projects COMBINE and COMBINE II.

4.2.1 The basic model

In an alternative graph model for a railway scheduling problem the nodes correspond to train operations (entry time of a train in a specific block section). As in the job-shop case two dummy nodes 0 and $*$ are introduced. For some of the additional constraints explained later artificial nodes must be introduced. As for the shop problems proposed above the set of all these nodes is denoted by V.

Fixed arcs (conjunctions) are defined as follows. Each node $i \notin \{0, *\}$ has a unique successor $\sigma(i)$ corresponding to the operation which follows operation i on the route of the associated train. If operation i is the last operation of the train $\sigma(i) = *$ is set. Besides the arcs $i \rightarrow \sigma(i)$ which are labeled with the minimal time p_i for performing operation i (i.e. for passing the corresponding block sections) arcs $0 \rightarrow i$ for all operations i which are the first operations of trains (jobs) are introduced. The default weights for these arcs are 0. Later the possibility of modeling earliest starting times (release dates) by introducing arcs $0 \rightarrow j$ with non-negative labels l_{0j} is discussed and can especially be applied to the first operations of trains.

For completing the basic model the introduction of suitable pairs of alternative arcs in order to model sequencings of trains (at block sections where conflicts could occur) is

needed. As already described in Section 2.2.1 railway companies apply different safety concepts which can be modelled by alternative arcs.

In a fixed block safety system a (regular slow) train cannot enter a block section if this block section is still occupied by another train. If i and j are train operations using the same fixed block section, then either $s_{\sigma(i)} + \varepsilon \leq s_j$ or $s_{\sigma(j)} + \varepsilon \leq s_i$ must hold. As mentioned above $\varepsilon > 0$ is a small number which can be interpreted as time needed for a train to leave the corresponding fixed block section (exit time), i.e. the time period in which the train is present in two block sections. The situation of two trains using the same fixed block section then is represented by the pair $\{\sigma(i) \to j, \sigma(j) \to i\}$ of alternative arcs with $l_{\sigma(i)j} = l_{\sigma(j)i} = \varepsilon$. A corresponding situation is shown in Figure 10. Note that this is a blocking job-shop situation with no swap allowed.

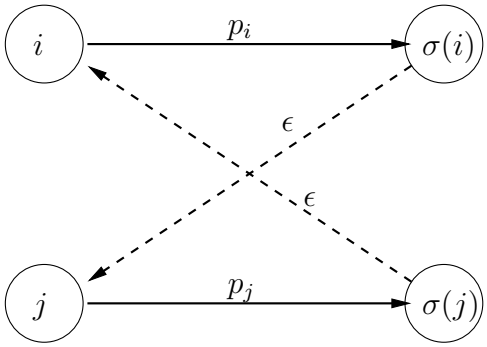

Figure 10: Two train operations i and j using the same block section.

Remember that the number $\epsilon > 0$ can be interpreted as the time needed for a train to leave a certain block section. Introducing different weights $\epsilon_j > 0$ for different trains or even $\epsilon_{ij} > 0$ for different train operations does not modify the argumentation considerably. As in this thesis it is assumed, that any train fits in every fixed block section, where it has to move through, it also can be assumed that the exit time of a train from a certain fixed block section is smaller than the minimum travelling time in its next fixed block section. In this thesis the case of an unique number $\epsilon > 0$ is treated if not stated different.

As described before, assigning a small weight $\epsilon > 0$ ($\epsilon_j > 0$, $\epsilon_{ij} > 0$) to the alternative arcs also models the no-swap constraint which is very important in the case of two trains going in opposite direction on the same track. Of course such two trains cannot swap for physical reasons and by weights $\epsilon > 0$ ($\epsilon_j > 0$, $\epsilon_{ij} > 0$) this is prevented in feasible solutions (complete consistent selections). As for practical reasons all data could be given in seconds or even smaller units, also in this case all data can be assumed to be integer. Of course, also situations where a train is at the end of its route can be modelled. Then a

pair of the type already depicted in Figure 2 must be introduced. If both trains are at the end of their route a pair of the type $\{j \to i, i \to j\}$ is introduced (see also Section 4.1.2).

So far the situation of slow trains moving in a railway network is modelled. For fast trains the model is slightly different. Remember that in a fixed block safety system a fast train is only allowed to enter a block section if the next two block sections are not occupied by other trains. This case is modelled by pairs of alternative arcs which are different from the situation above. In the case of a fast train A and a slow train B moving in the same direction through a sequence of block sections the corresponding alternative arcs are shown in Figure 11.

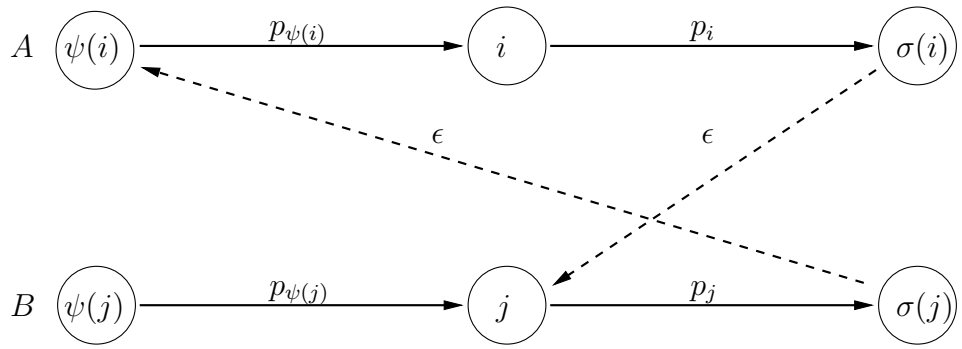

Figure 11: Fast train operation i (belonging to train A) and slow train operation j (belonging to train B) using the same block section.

For the situation in Figure 11 it is assumed, that i and j are neither the first nor the last operations of the associated trains. For situations involving the first or last operations of trains (trains at the beginning or end of their routes) analogue alternative pairs can be introduced. Situations with two fast trains, situations involving track switches, trains going in opposite direction, etc. are modeled analogously. An example with a slow and a fast train going in opposite direction is depicted in Figure 12. Both trains travel through block section M_i. This leads for example to the corresponding alternative pair $\{\sigma(u) \to \psi(v), \sigma(v) \to u\}$.

The moving block safety system described in Section 2.2.1 can be implemented as shown in Figure 13, where two trains have the same route. In this figure $\{i \to j, \sigma(j) \to \sigma(i)\}$ and $\{j \to i, \sigma(i) \to \sigma(j)\}$ are pairs of alternative arcs which are labeled by positive safety distances (here c, d, e and f), i.e. minimal amounts of time between trains entering or leaving the corresponding moving block section. If $i \to j$ is chosen from the first pair then $j \to i$ cannot be chosen from the second pair (otherwise there is the positive cycle $i \to j \to i$) and $\sigma(i) \to \sigma(j)$ has to be selected, which is compatible with $i \to j$. Thus, in

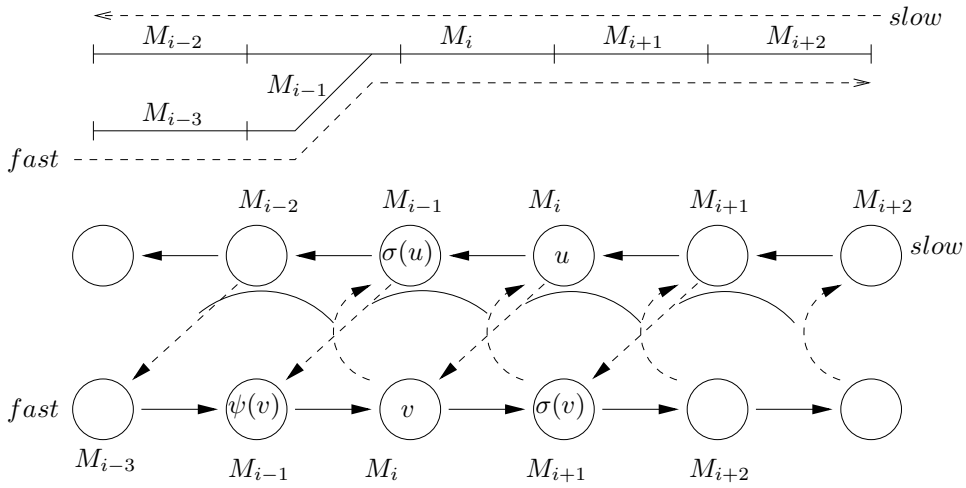

Figure 12: Two trains in a situation where a track switch is involved.

a complete consistent selection either $i \to j$ and $\sigma(i) \to \sigma(j)$ or $j \to i$ and $\sigma(j) \to \sigma(i)$ have to be selected.

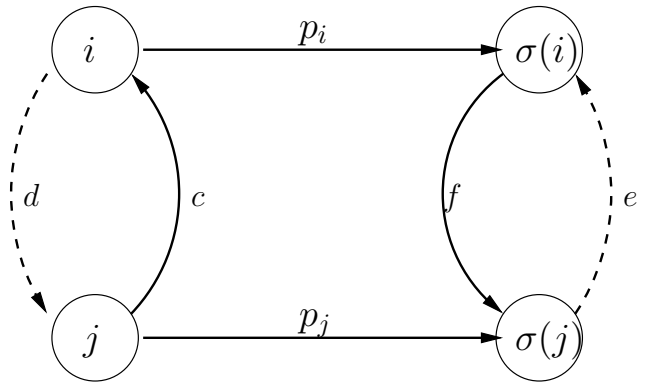

Figure 13: Model of two trains using the same moving block section.

In this thesis it is assumed that all moving block sections are one-way single-track sections. More general models can be implemented by dividing moving block sections into different parts, i.e. sections for single lines, switches, etc., and building a route for each train through these sections. Then suitable pairs of alternative arcs modelling the different sequences of trains can be introduced.

The moving block system may be combined with the fixed block system then modelling complex railway systems with different safety systems by a unique graph formulation.

Note that in both cases, fixed block and moving block sections, the corresponding alternative pairs are not incident with the nodes 0 and $*$.

4.2.2 Modelling additional constraints and objective functions

In this section the additional constraints introduced in Section 2.2.2 are modelled in terms of the alternative graph model. It will be introduced how to model constraints (1) to (5) and some objective functions. Most of these additional constraints only require to introduce suitable additional fixed arcs.

A lower bound for the travelling time of a train through a block section was already introduced for the basic model. It corresponds to the processing time of the associated operation i and thus is modelled by the weight p_i for the arc $i \rightarrow \sigma(i)$. An upper bound $k \geq 0$ for this travelling time may be set by introducing the fixed arc $\sigma(i) \rightarrow i$ with weight $-k$. This models the constraint $s_i \geq s_{\sigma(i)} - k \Leftrightarrow s_{\sigma(i)} - s_i \leq k$ and thus the travelling time for the corresponding train and block section to be less or equal than k. Obviously $k \geq p$ must hold since otherwise a positive cycle occurs. Constraints of Type (1) are depicted in Figure 14 (a).

A release-date r_i and/or a deadline d_i for the entrance of a train in a certain block section can be modelled by introducing arc $0 \rightarrow i$ with weight r_i and/or arc $i \rightarrow 0$ with weight $-d_i$, where i is the corresponding operation representing the entrance of this train in this block section. Release-date and deadline constraints are depicted in Figure 14 (b) and (c).

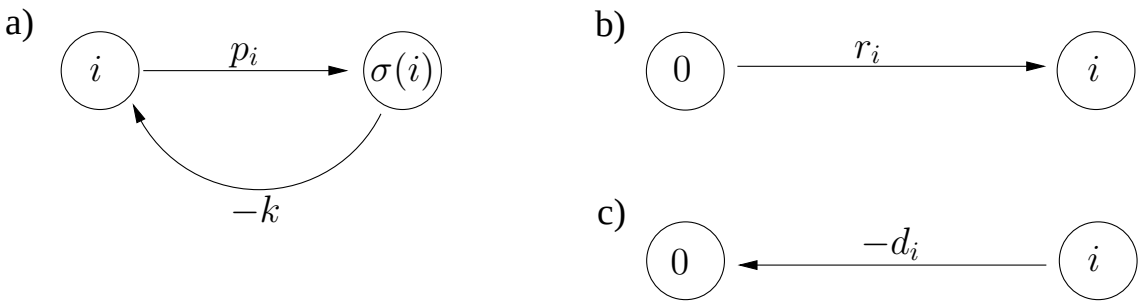

Figure 14: Different additional constraints modelled in terms of the alternative graph.

Starting and ending constraints for trains in fixed block sections are modelled by fixing alternative pairs. If for example a train starts on a certain fixed block section and is present there already at the start of a plan, it must be the first train on this block section and the corresponding alternative pairs are fixed accordingly. These alternative pairs are deleted

from the set of all alternative pairs and the chosen arcs are added to the set of fixed arcs. On the other hand if a train ends on a certain fixed block section and remains there until the end of a plan, it has to be the last train on that block section and again alternative pairs are fixed accordingly. Again these alternative pairs are deleted from the set of all alternative pairs and the chosen arcs are added to the set of fixed arcs. An example of a starting constraint is depicted in Figure 15.

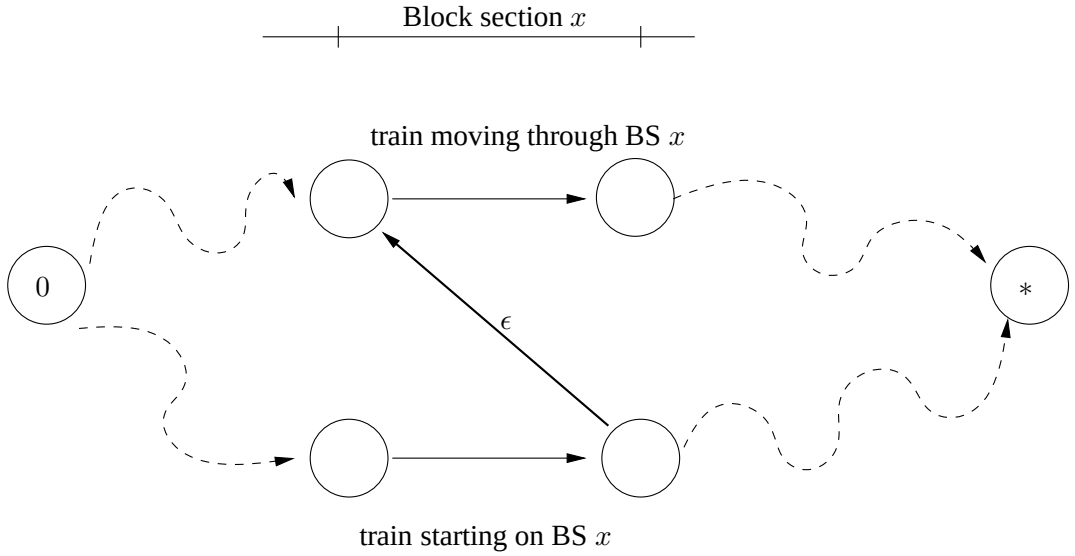

Figure 15: Starting and ending constraints modelled in terms of the alternative graph.

Starting and ending constraints for sets of trains already present or ending in moving block sections can be modelled analogously by fixing alternative pairs.

Also several connection constraints may be modelled by introducing fixed arcs. If for example a train B has to wait at a station platform for another train A a fixed arc $i \rightarrow j$ is introduced where i represents the entrance of train A in the station block section and j represents the exit of train B from its station block section (more precisely the entrance of train B in the following block section). The weight of the arc is set to w modeling the minimal time B has to wait before leaving its block section after the arrival of A in its block section, i.e. w is a suitable number modelling a minimal time which has to be between the arrival of A at the platform and the exit of B from the platform. An example for a constraint of Type (4) is depicted in Figure 16.

Modelling out-of-service intervals for certain block sections is different from the situations above, as not only additional fixed arcs but also additional nodes and alternative pairs have to be introduced. Details of the model for an out-of-service interval situation

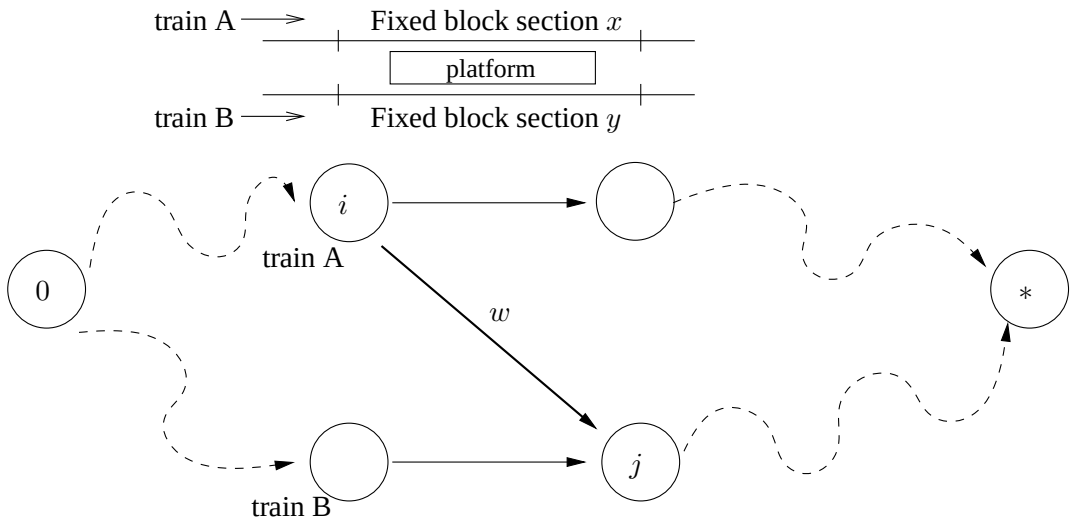

Figure 16: Connection constraints modelled in terms of the alternative graph.

are depicted in Figure 17. There the situation is modelled where a slow train has to travel through a block section which is out of service for a certain time period. Situations for fast trains or moving block sections are modelled analogously. In the case depicted in Figure 17 for the out-of-service interval $s = [b, b + l]$ of a certain block section x two artificial nodes b_{sx} and e_{sx} are introduced. Additionally there are fixed arcs $0 \rightarrow b_{sx}$, $b_{sx} \rightarrow e_{sx}$, $e_{sx} \rightarrow 0$ and $e_{sx} \rightarrow *$ with weights $b, l, -b - l$ and 0, where b is the beginning of the out-of-service interval and l its length. All these fixed arcs together model the constraint that block section x is out-of-service exactly from b to $b+l$. The alternative pair drawn in dashed lines then models the two possibilities either to schedule the train before or after the out-of-service interval in the corresponding block section. It can easily be seen that introducing an out-of-service interval for a certain block section is similar to the introduction of an additional train moving through this section exactly within the time period $s = [b, b + l]$.

The railway scheduling problem (with or without additional constraints) now can be formulated as follows. Find a complete consistent selection S of alternative arcs for the corresponding alternative graph $G = (V, C, A)$. Such a complete consistent selection S again defines an earliest start schedule $(s_i)_{i \in V}$ where the starting time of operation i is equal to the length of a longest path from 0 to i in $G(S) = (V, C \cup S)$.!!!

Besides the task to compute feasible solutions one can go further and try to find good or optimal solutions. Given a certain objective function the goal is to compute solutions which minimize this function.

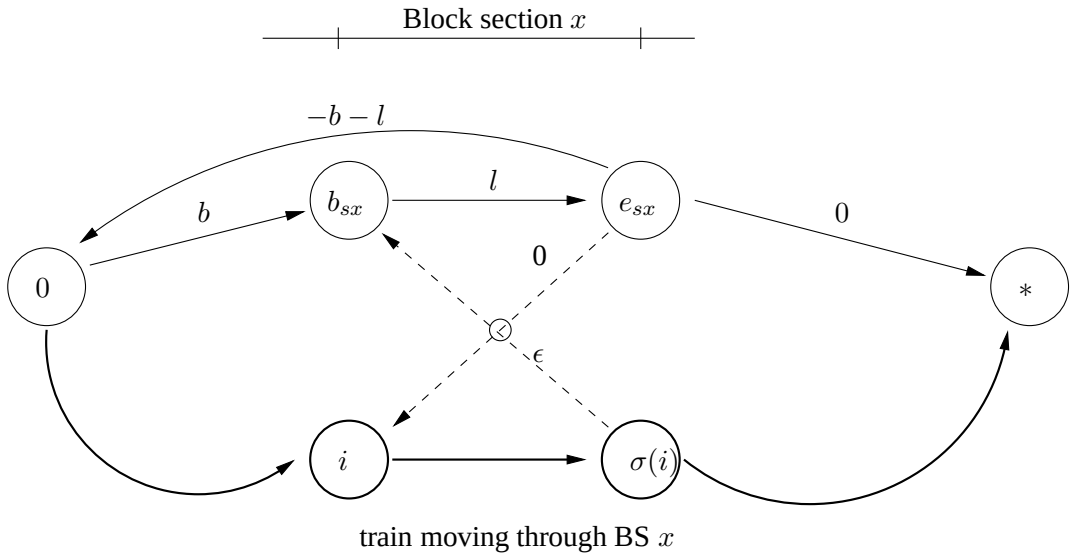

Figure 17: An out-of-service interval modelled in terms of the alternative graph.

In addition to the large variety of different constraints also different objective functions can be suitably integrated in the alternative graph model. Obviously the makespan objective C_{max} as well as the maximum lateness objective L_{max} can be modelled as shown above in the case of blocking job-shop problems. Also generalizations such as maximum lateness of starting and ending times of arbitrary operations (e.g. entrances/exits of trains in block sections) can be implemented in an easy way. This can be done by setting due-dates for these operations in terms of fixed arcs. For example $i \rightarrow *$ with weight $-q_i$ ($p_i - q_i$) would model q_i as the due-date for the start (completion) of i. Note that modelling problems with due-dates and a corresponding objective function, all arcs of type $v \rightarrow *$ must represent such due-date constraints. Such arcs with weight 0 at the end of jobs must be deleted or modified in order to represent due-dates.

For the objective functions proposed above the goal is to find a complete consistent selection S for a given alternative graph $G = (V, C, A)$ which minimizes the starting time s_* of the sink.

Based on the starting times in a schedule $\mathbb{S} = (s_i)_{i \in V}$ optimal speeds for all train operations can be calculated. For example consider a train which has to wait in a block section before entering the next because that one is occupied by another train. Then its speed in the section can be adjusted to the total time it has to stay in the block section.

In the next section complexity results for different shop scheduling and railway problems are presented.

5 Complexity Results

In this section complexity results for shop scheduling problems with blocking and especially for railway scheduling problems are presented.

Two different types of problems are considered concerning their complexity, namely decision problems on one hand and optimization problems on the other hand. For an introduction to the theory of complexity and a summary of important results the reader is referred to Garey & Johnson [24]. In this thesis a selection of decision problems is shown to be NP-complete and some optimization problems are shown to be NP-hard. The different types of complexity are closely related. An optimization problem is called NP-hard if the corresponding decision problem is NP-complete. If a problem is shown to be NP-complete or NP-hard this means that is is very unlikely to solve it in polynomial time (unless $P = NP$).

Before starting with presenting complexity results, a notation is introduced which is based on the well-known $\alpha|\beta|\gamma$-scheme (see Brucker [9]) but adjusted in order to describe the problems discussed in this thesis. New values for the machine environment, and objective functions are described in Table 2.

Field	Value	Meaning
α	$Railway$	railway scheduling problem from Section 2.2.1
	$Railway, FB, slow$	railway scheduling problems with only fixed block sections and slow trains
γ	$f \leq UB$	problem of deciding whether a feasible solution with objective value at most UB exists or not
	$feas$	problem of deciding whether a feasible solution which respects all constraints exists or not

Table 2: New values in the α- and γ-field.

Table 3 summarizes new values for the β-field. The new field β_b describes blocking restrictions and the field β_s contains new types of precedence constraints which are important in the context of railway scheduling problems. The structure of a railway network and corresponding routes of trains are characterized by the field β_n.

With the notations above the elementary reductions depicted in Figure 18 and 19 can be made. An arc $P \rightarrow Q$ in the reduction graph means that problem P reduces to problem Q.

Field	Value	Meaning
β_b	$\bullet$	all operations are ideal
	$blocking$	all operations are blocking except the last of a job
	$blocking - op$	each operation is defined to be ideal or blocking by the problem instance
	$blocking(\epsilon)$	each machine is available again only $\epsilon > 0$ after a job starts on the next machine (has left the machine)
	$blocking - op(\epsilon)$	describes the combination of the $blocking - op$ case and the ϵ-constraint
β_s	$\bullet$	no additional precedence constraints are given
	$start$	each job/train may be determined to be the first one on its first machine (starting constraints)
	end	each job/train may be determined to be the last one on its last machine (ending constraints)
	$prec$	arbitrary precedence constraints may be given
β_n	$single - line$	trains move on a single line in both directions
	$grid$	the railway network has gridlike structure
	$\bullet$	the railway network has arbitrary structure

Table 3: New values in the β-field.

Obviously a flow-shop problem reduces to a corresponding job-shop problem, as it is a special case (Figure 18 (a)). For problems with blocking and no-swap constraints the Railway problem with fixed block sections and slow trains lies between flow-shop and job-shop problems (see Figure 18 (b)).

The elementary reductions for different blocking constraints depicted in Figure 19 can be derived by special case reductions.

Graphs describing elementary reductions for other problem characteristics can be found in Brucker [9]. The corresponding reduction graph for objective functions even holds when considering feasibility instead of optimization problems.

In the following complexity results for both machine scheduling and railway scheduling problems are discussed. In the first part the complexity of some feasibility problems for flow-shops and job-shops is analyzed. The complexity of associated optimization problems is deduced afterwards. The second part deals with the complexity of railway scheduling problems. Finally in the third part all complexity results are summarized. In particular borders between polynomially solvable and NP-complete (NP-hard) problems are identified.

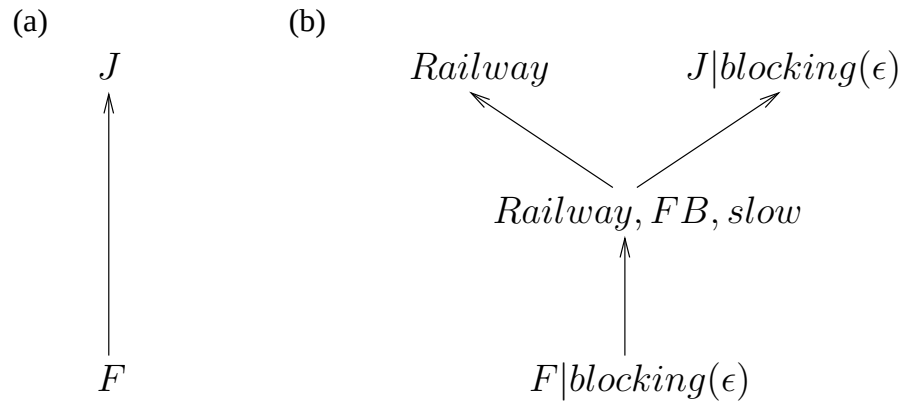

Figure 18: Elementary reductions for shop scheduling and railway problems.

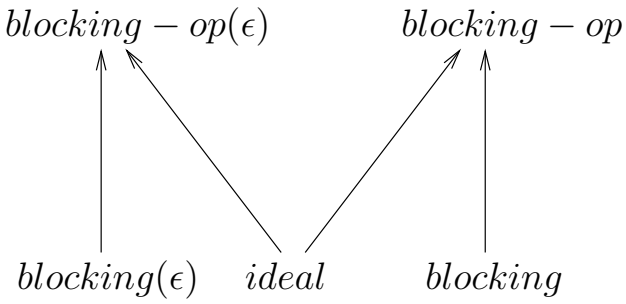

Figure 19: Elementary reductions for different blocking characteristics.

5.1 Shop scheduling problems with blocking

This subsection deals with complexity results for shop scheduling problems. On one hand these results will build a basis for considerations about railway scheduling and related problems. On the other hand they will be used to identify borders between polynomially solvable and NP-complete (NP-hard) problems.

Some decision problems in the field of shop scheduling problems with blocking are shown to be NP-complete. That means it is unlikely to decide with a polynomial effort of computation time whether a feasible solution exists or not (unless $P = NP$).

Papadimitriou & Kanellakis [55] consider flow-shop problems with FIFO-buffers between consecutive machines. A FIFO-buffer of capacity b can contain at most b jobs at a time. The jobs must leave the buffer in the same order as they entered it, this is the FIFO property. Moreover no job is allowed to bypass a buffer. Thus, only permutation plans are feasible solutions to these problems. A basic result published by Papadimitriou & Kanellakis [55] is

Proposition 5.1 *Problem $F2|$FIFO-buffer, b=1$|C_{max} \leq UB$ is NP-complete (in the strong sense).*

Proof:

The following proof is a version of the original proof from Papadimitiriou and Kanellakis but partially adjusted to the notations in this work.

For showing NP-completeness a reduction from problem 3MI (3-dimensional matching of integers) is made. For problem 3MI a set of n (positive) integers $A = \{a_1, \ldots, a_n\}$ and a set of $2n$ (positive) integers $B = \{b_1, \ldots, b_{2n}\}$ are given. The question is: Is there a partition of B into n pairs $(p_i, q_i), i = 1, \ldots, n$ such that for all i; $a_i + p_i + q_i = c$ with $c = (\frac{1}{n})(\sum_{i=1}^{n} a_i + \sum_{i=1}^{2n} b_i)$? This problem is known to be NP-complete (in the strong sense) (see Garey and Johnson [24]).

Now starting from an instance of 3MI an instance for problem $F2|$FIFO-buffer, b=1$|C_{max} \leq UB$ is constructed. It can be assumed that $\frac{c}{4} < a_i, b_i < \frac{c}{2}$ and that all a_i, b_i are multiples of $4n$. This can be achieved by adding a sufficiently large integer to all a_i and b_i and then multiplying all integers by $4n$. If an integer $Z > 0$ is added to all a_i and b_i then $\frac{c}{4}$ increases by $\frac{3}{4}Z$ and $\frac{c}{2}$ increases by $\frac{3}{2}Z$. Thus, the values a_i and b_i increase faster than $\frac{c}{4}$ and slower than $\frac{c}{2}$ when increasing Z. Eventually for some large Z the relations above are fulfilled.

Note, that the problem stays the same when modifying the data as described above. The transformation does not affect the existence of a solution for the 3MI problem and thus leads to an equivalent problem. Note, that $c > 8$ holds, as $\frac{c}{2} > a_i \geq 4$.

Constructing an instance for the flow-shop problem out of the 3MI instance $4n + 1$ jobs with execution times (p_{1i}, p_{2i}) are introduced as follows:

- *(i) $n-1$ jobs $K_1, \ldots, K_{n-1}$ with $(p_{1K_i}, p_{2K_i}) = (\frac{3c}{2}, 2)$ and a job K_0 with $(p_{1K_0}, p_{2K_0}) = (0, 2)$ and a job K_n with $(p_{1K_n}, p_{2K_n}) = (\frac{3c}{2}, 0)$,*
- *(ii) a job B_i with processing times $(p_{1B_i}, p_{2B_i}) = (1, b_i)$ for each $1 \leq i \leq 2n$,*
- *(iii) a job A_i with processing times $(p_{1A_i}, p_{2A_i}) = (\frac{c}{2}, a_i + c)$ for each $1 \leq i \leq n$.*

The upper bound UB is defined to be $n(2c + 2)$. This construction of an instance for problem $F2|$FIFO-buffer, b=1$|C_{max} \leq UB$ is obviously of polynomial effort.

It has to be shown that for the flow-shop problem a schedule with makespan at most UB exists if and only if the 3MI problem has a solution. As UB is the sum of all p_{1j} and also

the sum of all p_{2j}, the makespan C of a plan is less or equal to UB iff $C = UB$. Thus, there cannot be any idle time in such a plan for one of the machines. As a consequence K_0 must be scheduled first and K_n last. Otherwise there would be idle time on the second machine at the beginning of a schedule or on the first machine at the end, respectively.

First it is shown that any feasible schedule for the flow-shop problem defines a partition of B into n pairs $\{b_{i_1}, b_{i_2}\}$ such that $a_i + b_{i_1} + b_{i_2} = c$. Every feasible schedule for the flow-shop instance has to consist of n segments (see Figure 20). This can be shown by induction on the number of segments. It will now be shown that the first part of the schedule in $[0, 2c + 2]$ must look like in Figure 20. From an identical argument then the induction step for $[i(2c + 2), (i + 1)(2c + 2)]$ follows.

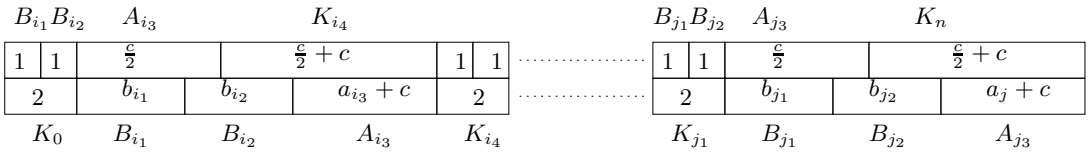

Figure 20: Job patterns for a feasible schedule

In any feasible schedule for the flow-shop problem two jobs B_{i_1}, B_{i_2} have to follow the first job K_0. Otherwise there would be an idle time on the second machine (see Figure 21). (If an A_j follows K_0 this is obvious, as $c > 8$. If a job B_i and an A_j follow K_0 then $b_i < \frac{c}{2} - 1$ holds since c and b_i are multiples of $4n$ and $b_i < \frac{c}{2}$. Thus, idle time on the second machine occurs. Scheduling a job K_j instead of A_j would also lead to idle time on the second machine, as its processing time on the first machine is even longer.)

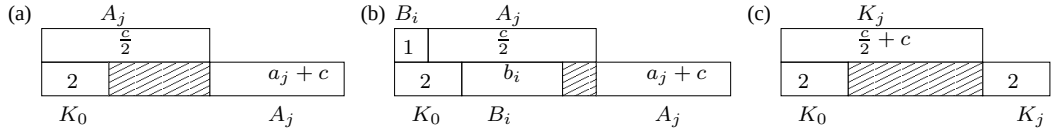

Figure 21: Job patterns, which do not provide a feasible schedule.

The next job then must be an A_{i_3} (see Figure 22). Otherwise, choosing a job B_{i_3}, a buffer overflow (idle time on the first machine, respectively) would occur. Choosing a job K_{i_3} an idle time on the second machine (as $b_{i_1} + b_{i_2} < c < \frac{3c}{2}$) would arise.

Similar arguments as used above provide a job K_{i_4} to be the next one. K_{i_4} must be chosen since a no job B_{i_4} can be used. A job A_{i_4} followed by a job B_{i_5} or A_{i_5} cannot be used, too. All these choices would cause a buffer overflow (idle time on the first machine, respectively) as $b_{i_1} + b_{i_2} + a_{i_3} + c > \frac{3c}{2} > c + 1$ (see Figure 23 (a), (b)). Using a job A_{i_4} followed by a job K_{i_5} would also lead to contradiction when looking at the next

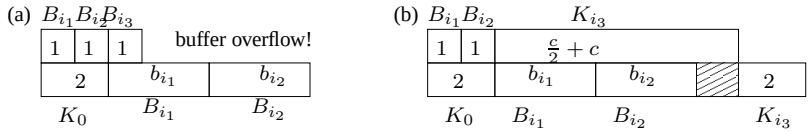

Figure 22: Job patterns, which do not provide a feasible schedule.

possible job. Then $b_{i_1} + b_{i_2} + a_{i_3} + c + a_{i_4} + c < 4c - 2$ *holds, so that scheduling a job* K_{i_6} *would cause an idle time on the second machine (see Figure 23 (c)). Furthermore* $b_{i_1} + b_{i_2} + a_{i_3} + c + a_{i_4} + c > 3c$ *holds, which leads to buffer overflow (idle time on the first machine, respectively) when scheduling a job* A_{i_6} *or* B_{i_6} *next. Thus, job* K_{i_4} *completes the segment and the next segment starts with* K_{i_4} *on the second machine.*

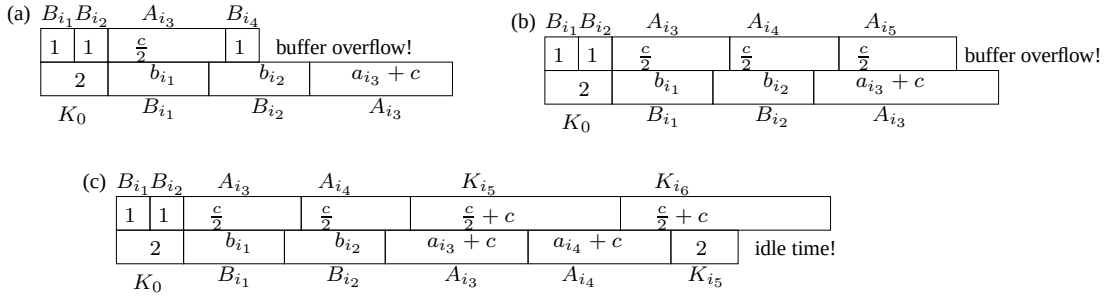

Figure 23: Job patterns, which do not provide a feasible schedule.

Additionally it is shown that K_{i_4} *finishes on the first machine exactly when* A_{i_3} *finishes on the second. Therefore it has to be shown that* $b_{i_1} + b_{i_2} + a_{i_3} + c = 2c$*. In the case* $b_{i_1} + b_{i_2} + a_{i_3} + c < 2c$ *there must be idle time on the second machine. In case* $b_{i_1} + b_{i_2} + a_{i_3} + c > 2c$ *it is clear that* $b_{i_1} + b_{i_2} + a_{i_3} + c - 2c = b_{i_1} + b_{i_2} + a_{i_3} - c$ *is a multiple of* $4n$ *and no job can follow* K_{i_4} *(a job* B_i *because of buffer overflow (idle time on the first machine, respectively) and jobs* A_i *and* K_i *because* $b_{i_1} + b_{i_2} + a_{i_3} + c + 2 < \frac{3c}{2} + c = \frac{c}{2} + 2c$*, which means idle time on the second machine). Thus,* $b_{i_1} + b_{i_2} + a_{i_3} + c = 2c$ *holds and a partition for the 3MI problem can be constructed from the feasible flow-shop schedule.*

Conversely, given a partition for the 3MI problem a feasible schedule without idle times for the flow-shop problem can be constructed using the pattern from Figure 20. This completes the proof and problem $F2|$*FIFO-buffer, b=1*$|C_{max} \leq UB$ *is shown to be* NP*-complete (in the strong sense).* □

Note, that in the proof above zero processing times are allowed and thus, NP-completeness for problems including those is shown. But zero processing times can be disallowed by multiplying all processing times by a large positive integer and replacing zero times by 1. This does not modify the argumentation.

From the complexity result above other complexity results for a variety of shop scheduling problems with blocking can be derived step by step. In a first step the following result for blocking flow-shop problems can be stated.

Proposition 5.2 *Problem $F3|$blocking$|C_{max} \leq UB$ is NP-complete (in the strong sense).*

Proof: *Problem $F3|$blocking$, p_{2j} = 0|C_{max} \leq UB$ is equivalent to the 2-machine flow-shop problem with a FIFO-buffer of capacity $b = 1$ between the machines (see Hall & Sriskandarajah [28]), as the buffer can be interpreted as an additional machine with zero processing times. Thus, the problem of deciding whether a feasible solution with at most makespan UB exists is NP-complete (in the strong sense). Therefore the more general problem with arbitrary processing times on the second machine (i.e. $F3|$blocking$|C_{max} \leq UB$) is NP-complete (in the strong sense), too. (If zero processing times are wanted to be disallowed this can obviously be done by multiplying all processing times by a sufficiently large number N and setting zero processing times to 1. This does not modify the argumentation.)* □

Setting time-windows $[0, UB]$ for the completion time of all operations for the problem $F3|$blocking$|\ldots$ and asking for a solution respecting these time-windows leads to an equivalent formulation of the decision problem $F3|$blocking$|C_{max} \leq UB$. Thus, the more general problem with arbitrary time-windows is NP-complete (in the strong sense), too. This leads to

Proposition 5.3 *Problem $F3|$blocking$, r_{ij}, d_{ij}|$feas is NP-complete (in the strong sense).*

From the results above some results for job-shop problems with blocking can be derived by simple special case reduction.

Proposition 5.4 *(a) Problem $J|$blocking$|C_{max} \leq UB$ is NP-complete (in the strong sense).*

(b) Problem $J|$blocking$, r_{ij}, d_{ij}|$feas is NP-complete (in the strong sense).

Note that all considerations so far do only hold in the case of standard blocking problems and thus with swap allowed in the case of job-shop problems.

For no-swap problems a more complex argumentation is needed. Consider problem $F3|$blocking$(\epsilon)|C_{max} \leq UB$, namely the 3-machine flow-shop problem with blocking,

where each machine is available again only $\epsilon > 0$ after a job starts on the next machine (has left the machine). This means, that ϵ is added to the weight of every alternative arc. In Figure 24 the Problem $F3|\text{blocking}|..$ is modeled in terms of an alternative graph. The blocking restrictions allow only permutation solutions to be feasible, as they disallow the overtaking of jobs. Consider the situation depicted in Figure 25 in terms of an alternative graph. There the sequence of two jobs i and j is chosen different on the machines M_i and M_{i+1}, as the corresponding alternative arcs are chosen in different directions. This choice is infeasible as a positive cycle occurs. By induction the permutation property is proved. For a complete consistent selection this means that one has parallel alternative arcs between each pair of jobs (see Figure 26).

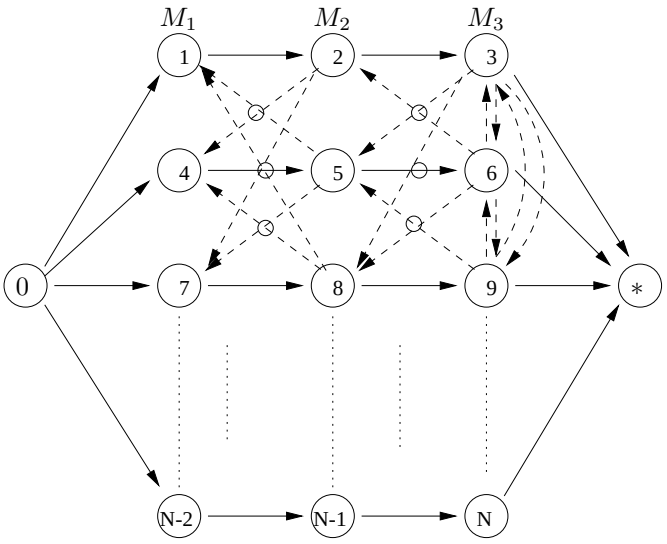

Figure 24: Alternative graph for problem $F3|\text{blocking}|..$.

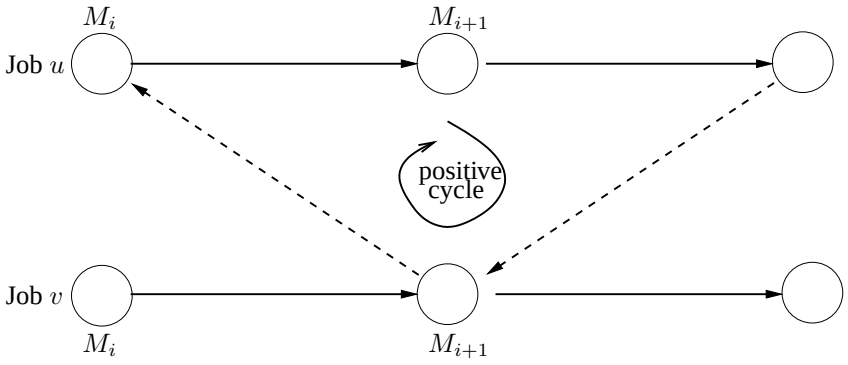

Figure 25: Jobs which are sequenced differently on consecutive machines.

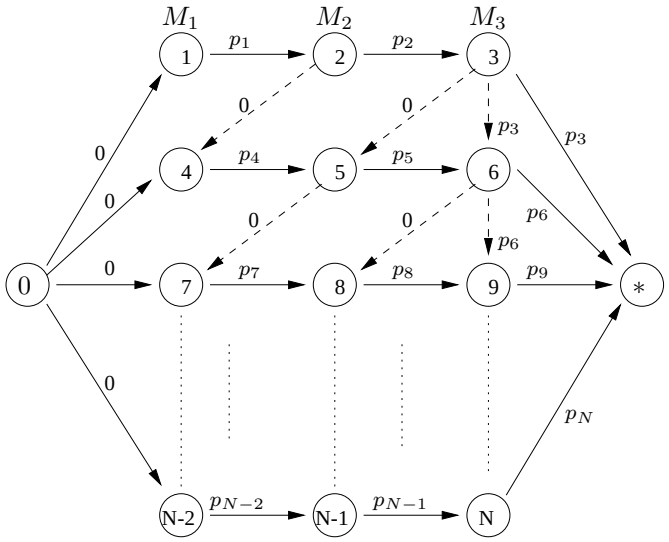

Figure 26: Solution graph for problem $F3|\text{blocking}|..$.

Obviously replacing the zero weights of alternative arcs by a small number $\epsilon > 0$ does not modify the argumentation considerably. This leads to

Proposition 5.5 *Problem $F3|$blocking$(\epsilon)|C_{max} \leq UB$ is NP-complete (in the strong sense).*

With the same argumentation as for the case without ϵ - setting time-windows $[0, UB]$ for the completion time of all operations for the problem $F3|\text{blocking}(\epsilon)|$ and asking for a solution respecting these time-windows - an equivalent formulation of the decision problem $F3|\text{blocking}(\epsilon)|C_{max} \leq UB$ is given. Thus, the more general problem with arbitrary time-windows is NP-complete, too.

Proposition 5.6 *Problem $F3|$blocking$(\epsilon), r_{ij}, d_{ij}|$feas is NP-complete (in the strong sense).*

For job-shop problems with blocking and the ϵ-constraint one can prove NP-completeness by simple special case reduction. The ϵ-constraint covers the case of $\epsilon > 0$ being arbitrary small and thus the no-swap problem (see above). As the problem with the ϵ-constraint is NP-complete (in the strong sense) for any $\epsilon > 0$ especially the problem with swap not allowed ($\epsilon > 0$ arbitrary small) is NP-complete (in the strong sense). In the following it is not distinguished between problems with the ϵ-constraint and no-swap problems. The results for job-shop problems with blocking are summarized in

Proposition 5.7 *(a) Problem $J|$blocking$(\epsilon)|C_{max} \leq UB$ is NP-complete (in the strong sense).*

(b) Problem $J|$blocking$(\epsilon), r_{ij}, d_{ij}|$feas is NP-complete (in the strong sense).

For job-shop problems with blocking and no swap allowed Mascis & Pacciarelli [44] derived a different type of complexity result, which is based on the representation of the problem in terms of alternative graphs. Using the notations introduced above they proved the following

Proposition 5.8 *Consider a blocking job-shop problem with no swap allowed which is given in terms of an alternative graph. Furthermore let S be a partial consistent selection for this graph. Then the problem of deciding whether an extension of S exists or not is NP-complete (in the strong sense).*

Note that this proposition cannot be proved for a general situation in the case swap is allowed. A more detailed description of those results can be found in [44].

Leading over from feasibility to optimization problems the following two results which can be derived directly from the above results are important.

Conclusion 5.1 *(i) Problem $F3|$blocking$|C_{max}$ is NP-hard.*

(ii) Problem $F3|$blocking$(\epsilon)|C_{max}$ is NP-hard.

Even for problems with no-wait constraints such as $F3|$no-wait$|C_{max}$ NP-hardness is proved (see Röck [60]). For the objective function L_{max} the no-wait flow-shop problem is already NP-hard for two machines (see Röck [59]). Of course then the corresponding job-shop problems are also NP-hard. But even in more special cases like unit processing times, these job-shop problems remain NP-hard for three machines and pseudopolynomially solvable for two machines, i.e. $J3|$no-wait$, p_{ij} = 1|C_{max}$ is NP-hard and $J2|$no-wait$, p_{ij} = 1|C_{max}$ is NP-hard in the weak sense. It is pseudopolynomially solvable (see Sriskandarajah & Ladet [64] and Kubiak [39]).

5.2 Railway scheduling problems

The complexity results derived above imply some railway scheduling problems introduced in Section 2.2 to be NP-complete.

Problem $F3|\text{blocking}(\epsilon)|C_{max} \leq UB$ can be viewed as a special railway problem or as a specific part of a railway problem with fixed block signalling system. In particular this shop scheduling problem corresponds to the problem of scheduling a set of slow trains all travelling in the same direction through three fixed block sections with respect to a given global time-window. Thus, this leads to a more general result.

Proposition 5.9 *Let be given a rail network with fixed block signalling, a set of trains with given routes and the corresponding travelling times for all pairs* { *train, block section* }*. Then the problem of finding a feasible solution with respect to a given global time-window (upper bound for the makespan) is NP-complete (in the strong sense).*

Of course, the case of mixed signalling systems is NP-complete, too. Another complexity result, which is relevant for railway scheduling problems is Proposition 5.8. Let be given a rail network with fixed block signalling, a set of trains with fixed routes and the travelling times for all pairs (train, block section). Then the problem of deciding whether a feasible solution with respect to some starting and ending constraints of trains corresponds to a special job-shop problem where a special partial selection is given and an extension is asked for. Then Proposition 5.8 already indicates that even railway problems without time-windows may be very hard to solve.

A formal reduction - strongly oriented towards a similar reduction from Arbib et al. [6] - can be made from problem 3-SAT. This leads to the following proposition.

Proposition 5.10 *Given a railway network with fixed block safety system and slow trains with given fixed routes and initial positions (starting constraints) the problem of deciding whether a feasible solution (complete consistent selection for the corresponding alternative graph) exists or not is NP-complete (in the strong sense).*

Proof: *It is clear, that the given problem is in NP. One can guess a complete selection and prove its consistency in polynomial time by looking for a positive cycle in the corresponding graph.*

In order to show NP-completeness 3-SAT will be reduced to a special case of the railway problem. This proof is strongly oriented towards a similar proof from Arbib et al. [6]. Arbib et al. show NP-completeness for a problem where packets have to move through a so-called packet switching network. Here trains moving in a railway network are considered instead of packets. An instance of 3-SAT is given by clauses $F_0 \wedge F_1 \wedge \ldots F_m$ with variables $x_0, x_1, \ldots, x_n$, where $F_j = (l_{j1} \vee l_{j2} \vee l_{j3})$.

Now a railway network and trains moving on given routes in this network are associated to this instance of 3-SAT. The railway network is composed out of special rudimentary elements (fixed block sections) which are shown in Figure 27. In the railway network shown in Figure 28 these fixed block sections appear in different orientations. For example the fixed block section of type (b) appears at all four borders of the rectangle in the corresponding four different orientations.

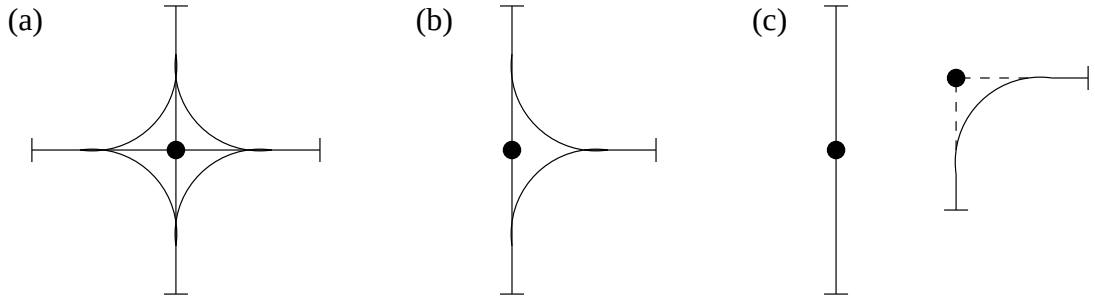

Figure 27: Different layouts for fixed block sections.

With each clause F_i a subnet of the whole railway network is associated. More precisely with F_i the railway subnet N_{F_i} is associated with rudimentary elements

$$u_i, a_{ij} \text{ and } b_{ij}, \text{ where } j \in \{1, 2, 3\}.$$

These rudimentary elements have connections

$$\{(a_{ij}, a_{i,j+1}), (b_{ij}, b_{i,j+1}) \mid j = 1, 2\} \cup \{(a_{ij}, b_{ij}) \mid j = 1, 2, 3\} \cup \{(b_{i2}, u_i)\}.$$

Additionally with each pair $X_k = (x_k, \bar{x}_k)$ is associated a subnet N_{x_k} with elements

$$\{x_k, v_k, \bar{x}_k\}$$

and connections

$$\{(x_k, v_k), (v_k, \bar{x}_k)\}.$$

These subnetworks are embedded in a rectangular gridlike network of sufficient size. Of course the minimal size is bounded by a polynomial in the size of the instance of 3-SAT. This ensures the correctness of the transformation. How to embed the subnetworks is defined by specifying coordinates in the grid.

Embedding the sections of the subnetworks in the grid the following coordinates are chosen for the N_{F_i}:

- *a_{ij} gets the coordinates $(3i + j, 5),\ 0 \leq i \leq m,\ 1 \leq j \leq 3$,*

- b_{ij} *gets the coordinates* $(3i + j, 4),\ 0 \leq i \leq m,\ 1 \leq j \leq 3$*,*
- u_i *gets the coordinates* $(3i + 2, 3),\ 0 \leq i \leq m$*.*

For the sections of N_{x_k} *the following coordinates are chosen:*

- x_k *gets the coordinates* $(3k + 1, 1),\ 0 \leq k \leq n$*,*
- v_k *gets the coordinates* $(3k + 2, 1),\ 0 \leq k \leq n$*,*
- $\bar{x}_k$ *gets the coordinates* $(3k + 3, 1),\ 0 \leq k \leq n$*,*

In order to have a more comfortable description of the routes of trains additional names for special sections in the network are introduced.

- *The sections* $(3i + 2, 2),\ 0 \leq i \leq m$ *are denoted* bu_i *('below* u_i*'),*
- *the sections* $(3k + 2, 0),\ 0 \leq k \leq n$ *are denoted* bv_k *('below* v_k*'),*
- *the sections* $(3k + 1, 2),\ 0 \leq k \leq n$ *are denoted* ax_k *('above* x_k*'), and*
- *the sections* $(3k + 3, 2),\ 0 \leq k \leq n$ *are denoted* $a\bar{x}_k$ *('above* $\bar{x}_k$*').*

With $p = \max\{3m + 1, 3n + 1\} + 1$ *all sections are contained in the rectangular network with corners* $(0, 0), (p, 0), (p, 6)$*, and* $(0, 6)$*. This is a polynomially large railway network. Skipping some details, a sketch of the railway network is drawn in Figure 28.*

Now (slow) trains are introduced. Firstly L *trains* $t_1, \ldots, t_L$ *are defined. The idea is to have enough trains* t_i *to occupy all block sections on a common part of their routes. Therefore the routes of the trains* t_i *have to be defined first and* L *will be defined later.*

Train t_i *starts in a section located at coordinates* $(0, -i), 1 \leq i \leq L$ *and has to be the first train there, i.e. it blocks this section from the beginning of any plan. The route of* t_i *is composed out of three parts. Firstly* t_i *passes all sections along the following route, which is specified by turning points only:*

$$(0, -i), (0, 0), (p, 0), (p, 2), (1, 2), (1, 4).$$

After that t_i *travels along the following path:*

$$(1, 4), (1, 5), (3, 5), (3, 4), \ldots,$$

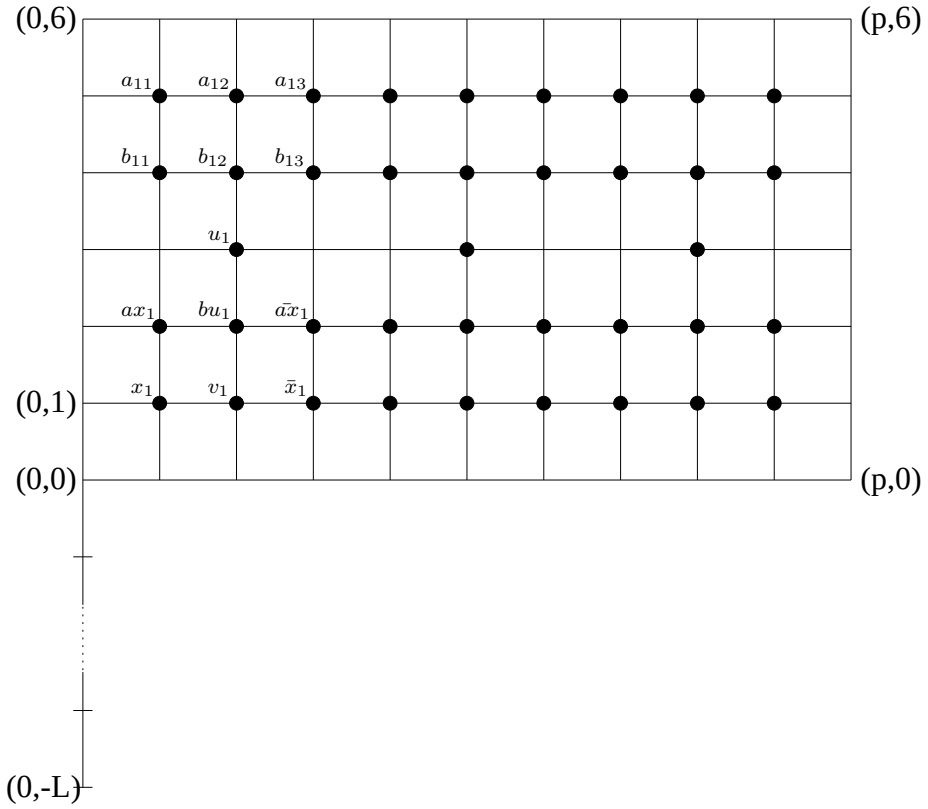

Figure 28: Sketch of the gridlike railway network.

$$(3j+1,4),(3j+1,5),(3j+3,5),(3j+3,4),\ldots,$$
$$(3m+1,4),(3m+1,5),(3m+3,5),(3m+3,4).$$

The last part of t_i's route is:

$$(3m+3,4),(p,4),(p,6),(0,6),(0,1).$$

Thus, $(0,1)$ is the final destination of all these L trains. Now L is defined to be the number of sections along any route of these trains between $(1,0)$ and $(0,2)$, extremes included. Note, that this number is well-defined as any of the trains passes a finite number of sections between $(1,0)$ and $(0,2)$. This number is bounded by $7(p+1)$ as any section is visited by any train at most once. The route of a train t_i is depicted in Figure 29.

Next $3(m+1)$ trains p_{ij}, where $0 \leq i \leq m, 1 \leq j \leq 3$, are placed at their initial block sections at a_{ij}. Their routes depend on the instance of 3-SAT in the following way:

- *If $l_{i1} = x_k$, then p_{i1} travels the route $a_{i1}, b_{i1}, b_{i2}, bu_i, ax_k, x_k$ otherwise if $l_{i1} = \bar{x}_k$ it travels route $a_{i1}, b_{i1}, b_{i2}, bu_i, a\bar{x}_k, \bar{x}_k$,*

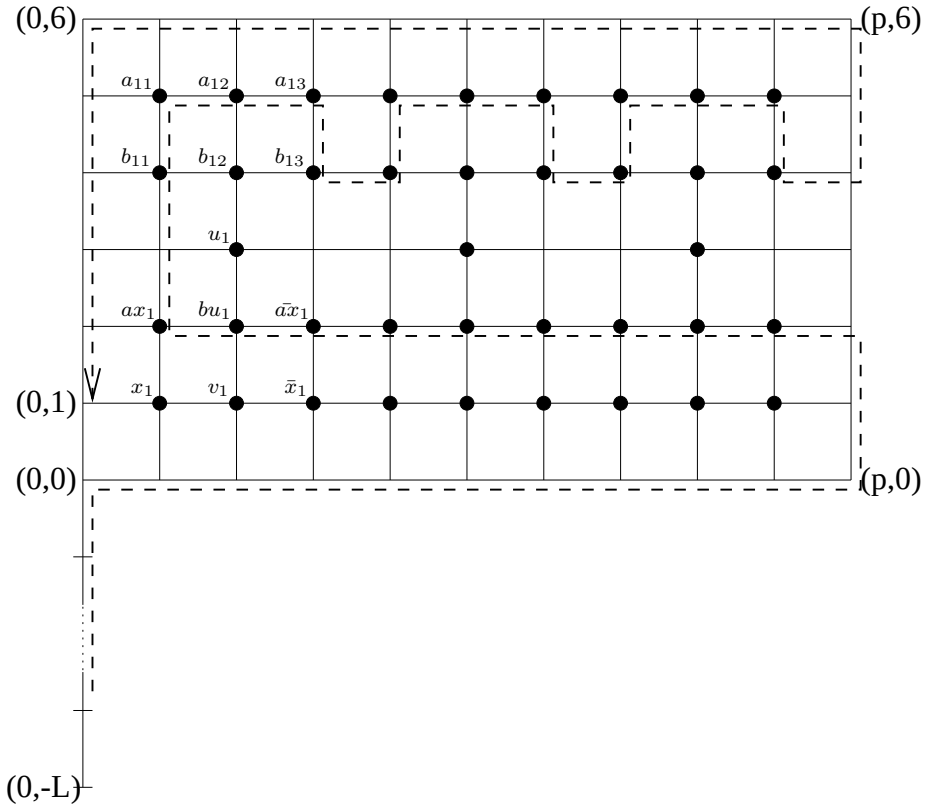

Figure 29: Routes of trains t_i.

- *if* $l_{i2} = x_k$*, then* p_{i2} *travels the route* a_{i2}, bu_i, ax_k, x_k *otherwise if* $l_{i2} = \bar{x}_k$ *it travels route* $a_{i2}, bu_i, a\bar{x}_k, \bar{x}_k$*,*
- *if* $l_{i3} = x_k$*, then* p_{i3} *travels the route* $a_{i3}, b_{i3}, b_{i2}, bu_i, ax_k, x_k$ *otherwise if* $l_{i3} = \bar{x}_k$ *it travels route* $a_{i3}, b_{i3}, b_{i2}, bu_i, a\bar{x}_k, \bar{x}_k$*.*

Additionally $2(n+1)$ *trains* q_k *and* $\bar{q}_k$ *where* $0 \leq k \leq n$ *are placed at their initial block sections at* x_k *and* $\bar{x}_k$ *respectively. The route of a train* q_k *is*

$$x_k, v_k, bv_k, (0,0), (0,-L),$$

whereas the route of $\bar{q}_k$ *is*

$$\bar{x}_k, v_k, bv_k, (0,0), (0,-L).$$

The routes of trains p_{ij}*,* q_k*, and* $\bar{q}_k$ *are depicted in Figure 30.*

The last train introduced is a special train l *which starts in the section located at co-ordinates* $(0,1)$*. It travels straight on to its destination in* $(0,-L)$*. This train plays an*

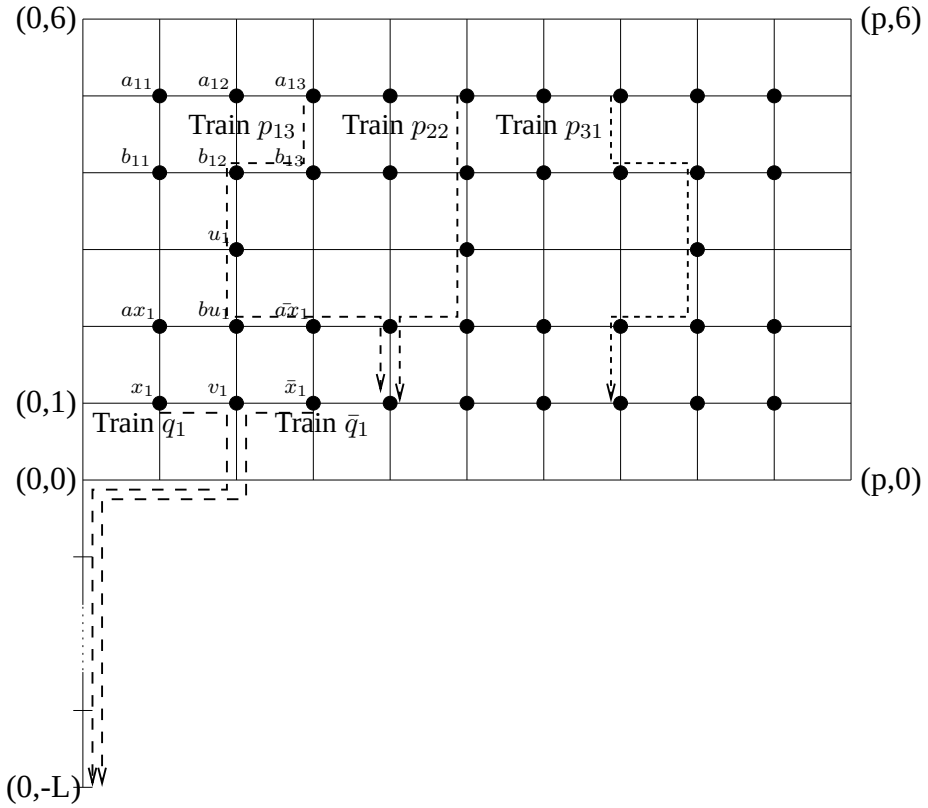

Figure 30: Routes of trains p_{ij}, q_k, and $\bar{q}_k$.

important role as it obviously has to wait for completing its journey until all trains t_i have left the sections on the y-axis. Note that each block section, where trains end, is supposed to have some sidings, were only these ending trains can be parked.

The travelling time of any of the trains is set to 10 for any block section and the the exit-time ϵ is set to 1. The resulting instance of a railway scheduling problem is obviously polynomial in the size of the 3-SAT instance.

In order to complete this proof it has to be shown, that the instance of the 3-SAT problem is satisfiable if and only if a feasible solution for the railway problem exists.

Let μ be a satisfying assignment for the 3-SAT problem. According to μ trains have to be driven in a way that they can reach their destinations. Firstly trains q_k travels into section v_k, if $\mu(x_k) = TRUE$. Otherwise ($\mu(x_k) = FALSE$) $\bar{q}_k$ travels into section v_k. By construction of the railway network and since μ is a satisfying assignment, for each subnet N_{F_i} at least one train p_{ij} can reach its destination. All other trains p_{ij} are parked in the two 'parking sections' b_{i2} and u_i. Next all trains t_i can leave the y-axis and occupy

block sections on their route between coordinates $(1, 0)$ *and* $(0, 2)$*. Now the special train* l *can reach its destination and consequently all other trains, especially the* t_i *can complete their journey. This completes one direction of the proof.*

Starting with a feasible solution for the railway scheduling problem, now a satisfying assignment for the 3-SAT problem is constructed. The design of the railway network and the routes of trains allows special train l *to reach its destination only if all trains* t_i *have left the* y*-axis. By the choice of* L *and since* l *occupies the destination of all trains* t_i*, all trains* p_{ij} *have either to travel to their destinations (*x_k *or* $\bar{x}_k$*) or to be parked in* b_{i2} *or* u_i*. Each subnet* N_{F_i} *provides two parking sections only. Thus, at least one train* p_{ij} *has to be driven to its final destination for each* N_{F_i}*. This implies, that for each pair* $(x_k, \bar{x}_k)$ *one of the trains has to move into section* v_k*. This provides a satisfying assignment for the 3-SAT problem, which completes the proof.*

□

A railway problem which is obviously polynomially solvable is the problem $Railway, FB, slow|single-line, start|feas$. If there are starting constraints for trains at both ends of the single line the problem is infeasible. Otherwise a feasible schedule can be computed by planning trains one by one respecting the starting constraints.

5.3 Classification of complexity results

In this section the complexity results derived above are summarized and supplemented by additional results from the literature. Especially the hardest problems which are known to be polynomially solvable and the easiest problems which are shown to be NP-complete or NP-hard are identified. Scheduling problems with different blocking restrictions are considered first. After that, railway problems are analyzed where starting restrictions are given.

For shop scheduling problems the classical flow-shop problem with two machines ($F2||C_{max}$) is known to be polynomially solvable (see Johnson [31]). But already the problem with three machines can be proved to be NP-hard (see Lenstra et al. [41]), i.e. the corresponding decision problem is NP-complete. Thus, the reduction graph in Figure 19 provides such problems with $\beta_b \in \{ideal, blocking - op, blocking - op(\epsilon)\}$ to be NP-complete (NP-hard).

For blocking problems the 2-machine flow-shop with makespan objective is polynomially solvable (see Gilmore and Gomory [27]). The corresponding 3-machine flow-shop problem with $\beta_b \in \{blocking, blocking(\epsilon)\}$ is NP-complete (NP-hard) (see Proposition

5.2 and 5.5). Even with the no-swap restriction, which becomes important for job-shop problems, the problem is NP-complete.

Thus, for blocking problems with makespan objective the border between polynomially solvable and NP-complete (NP-hard) problems lies between 2-machine and 3-machine flow-shop problems. Minimal and maximal open problems can be derived using the CLASS-program from Plaggenborg [56]. In addition to the reduction graphs from Figure 18 (a) and 19 also the reduction graph for the number of machines and for objective functions (see Brucker [9]) is used. The results for machine scheduling problems can be summarized as follows:

- **maximal polynomially solvable:**

$F2\|blocking(\epsilon)\|C_{max}$	equiv. problem in Gilmore & Gomory [27]
$F2\|\|C_{max}$	Johnson [31]
$F2\|blocking\|C_{max}$	equiv. problem in Gilmore & Gomory [27]

- **minimal NP-hard:**

$F2\|\|\sum C_i$	Garey et al. [25]
$J2\|\|C_{max}$	Lenstra & Rinnooy Kan [40]
$F3\|blocking\|C_{max}$	Proposition 5.2
$F3\|\|C_{max}$	Garey et al. [25]
$F3\|blocking(\epsilon)\|C_{max}$	Proposition 5.2
$F2\|\|L_{max}$	Lenstra et al. [41]

- **minimal open:**

$J2|blocking|C_{max}$
$F2|blocking - op|C_{max}$
$F2|blocking|L_{max}$
$F2|blocking(\epsilon)|\sum C_i$
$F2|blocking|\sum C_i$
$F2|blocking(\epsilon)|L_{max}$
$F2|blocking - op(\epsilon)|C_{max}$
$J2|blocking(\epsilon)|C_{max}$

- **maximal open:**

  $F2|blocking - op|C_{max}$
  $F2|blocking - op(\epsilon)|C_{max}$
  $J|blocking(\epsilon)|\sum w_iC_i$
  $J|blocking|\sum w_iC_i$
  $J2|blocking(\epsilon)|\sum w_iT_i$
  $J2|blocking(\epsilon)|\sum w_iU_i$
  $J2|blocking|\sum w_iT_i$
  $J2|blocking|\sum w_iU_i$

The feasibility problem for a flow-shop with blocking, the ϵ-constraint and starting constraints ($F|blocking(\epsilon), start|feas$) is obviously polynomially solvable. It can be interpreted as a railway problem where slow trains move on a single line in the same direction. As stated above the railway problem $Railway, FB, slow|single - line, start|feas$ is polynomially solvable. However, the more general railway problem with fixed block safety system and slow trains ($Railway, FB, slow|start|feas$) is NP-complete, as shown in Proposition 5.10. A closer look on the associated proof shows already problem $Railway, FB, slow|grid, start|feas$ to be NP-complete. Thus, for Railway scheduling problems the border between polynomially solvable and NP-complete problems lies between problems with only a single line and problems with a gridlike railway network. However, there is a large gap between railway problems with starting constraints which are known to be polynomially solvable and NP-complete, respectively.

In the next section different solution procedures, exact methods and heuristic approaches, are presented.

6 Solution methods

In this section a summary of different approaches to get feasible and/or good solutions for railway scheduling and related problems is presented. Existing techniques appearing in the literature as well as new ideas are described. When considering optimization problems the approaches are mainly developed for the objective function C_{max} (and L_{max}) but can be easily adapted to other objectives like maximum lateness of an arbitrary set of operations (see above). All methods are based on the representation of these problems in terms of the alternative graph model. Thus, most of the proposed algorithms can also be used to solve other problems if formulated by means of alternative graphs.

6.1 Greedy heuristics

In this section some simple heuristics in order to compute feasible solutions (complete consistent selections) for problems formulated by means of alternative graphs are presented. Most of these heuristics were developed by Mascis and Pacciarelli [44, 45] and are based on a generic greedy strategy combined with different priority rules. In addition to the priority rules presented in [44, 45] a new rule is described.

The main idea of the generic greedy approach is to fix successively arcs of alternative pairs according to some priority rule. In between alternative arcs which are induced by others are fixed. The algorithm ends with a complete consistent selection or stops if for an alternative pair no choice is possible, i.e. if both arcs of the pair would create a positive length cycle in the graph.

The generic algorithm is as follows:

```
1  begin
2  S := ∅;
3  Preprocess graph, i.e. choose all alternative arcs, which
     are implied by the problem instance itself;
4  while A ≠ ∅ do
5     begin
6     Select an alternative pair ((h,k),(i,j)) ∈ A;
7     Select arc (i,j), i.e. S := S ∪ {(i,j)}; A := A − {((h,k),(i,j))};
8     while ∃((u,v),(p,q)) ∈ A : l(v,u) + a_uv > 0 do
9        begin
10       if l(q,p) + a_pq > 0 then
```

```
11          STOP, the procedure failed in finding a feasible
             solution;
12       else
13          Select arc (p, q), i.e.
              S := S ∪ {(p, q)}; A := A − {((u, v), (p, q))};
14       end
15    end
16 end
```

Listing 1: Generic greedy algorithm

The preprocessing algorithm in Step 3 fixes arcs which are implied by the problem instance itself. It mainly applies the inner loop (Step 8) of the heuristic to the alternative graph. In Step 6 and 7 different priority rules are used to choose alternative arcs which are fixed next. Mascis and Pacciarelli ([44]) proposed four different rules.

- AMCC (Avoid Maximum Current $C_{\max}$) selects the pair $((h,k),(i,j)) \in A$; such that
$$l(0,h) + a_{hk} + l(k,*) = \max_{(u,v)\in A}\{l(0,u) + a_{uv} + l(v,*)\}.$$
The alternative arc (h,k) would increase the makespan of $G(S)$ most, if selected. Hence AMCC chooses (i,j).

- SMCP (Select Most Critical Pair) selects the pair $((h,k),(i,j)) \in A$; such that
$$\min\{l(0,h) + a_{hk} + l(k,*), l(0,i) + a_{ij} + l(j,*)\}$$
is maximized. Then the arc (i,j) with $l(0,h) + a_{hk} + l(k,*) \geq l(0,i) + a_{ij} + l(j,*)$ is chosen.

- SMBP (Select Most Balanced Pair) selects the pair $((h,k),(i,j)) \in A$; such that
$$\mid \{l(0,h) + a_{hk} + l(k,*) - l(0,i) - a_{ij} - l(j,*)\} \mid$$
is minimized. Then the arc (i,j) with $l(0,h) + a_{hk} + l(k,*) \geq l(0,i) + a_{ij} + l(j,*)$ is chosen.

- SMSP (Select Max Sum Pair) selects the pair $((h,k),(i,j)) \in A$; such that
$$\mid \{l(0,h) + a_{hk} + l(k,*) + l(0,i) + a_{ij} + l(j,*)\} \mid$$
is maximized. Then the arc (i,j) with $l(0,h) + a_{hk} + l(k,*) \geq l(0,i) + a_{ij} + l(j,*)$ is chosen.

In this thesis an additional rule is used, namely:

- FCFS (First Come First Serve) selects the pair $((h,k),(i,j)) \in A$; such that

  $$\min\{l(0,h), l(0,i)\}$$

  is minimized. Then the arc (i,j) with $l(0,h) \geq l(0,i)$ is chosen.

In the case of the absence of a path from i to j its length is set to $l(i,j) = -\infty$. In the implementations for this thesis it is set to a large negative number, respectively. It will be seen later, that all five rules can also be used as branching rules for enumerative algorithms.

As can be seen from the given algorithm a very simple constraint propagation technique is included. Alternative arcs which are implied by others can be found by simple longest-path-considerations. That means, if for an alternative arc $i \rightarrow j$ with weight a a path from j to i with weight $b > -a$ already exists the choice of $i \rightarrow j$ would lead to a positive cycle and thus, its alternative arc is implied. For some quite simple examples like flow-shop problems with blocking the algorithm always finds a complete consistent selection. Examples, where during the algorithm a consistent selection is reached, which has no extension, but where all alternative arcs seem to be selectable, are more complex. In this context arcs are said to seem selectable if they do not cause an immediate contradiction if chosen.

In the next section more sophisticated constraint propagation techniques are developed.

6.2 Constraint propagation techniques

In this Section constraint propagation techniques for the alternative graph model and especially for job-shop problems with blocking and for the railway scheduling problems described above are presented.

The constraints in job-shop problems with blocking are more restrictive than in a corresponding classical job-shop. Feasible solutions for the classical job-shop in general do not stay feasible for the blocking problem. Moreover no-swap restrictions reduce the number of feasible solutions additionally. On one hand this complicates the problem of finding feasible solutions. On the other hand it may be a chance to apply effectively constraint propagation methods.

Especially in railway problems situations occur where the choice of an alternative arc or a set of alternative arcs implies many other alternative arcs to be chosen in a feasible

solution. Consider the situation depicted in Figure 31 where two trains travel on a single track in the same direction without any possibilities to overtake. Then the choice of one alternative $\sigma(u) \to v$ arc representing the train sequence in a block section M_i implies all other arcs to be chosen in the same direction.

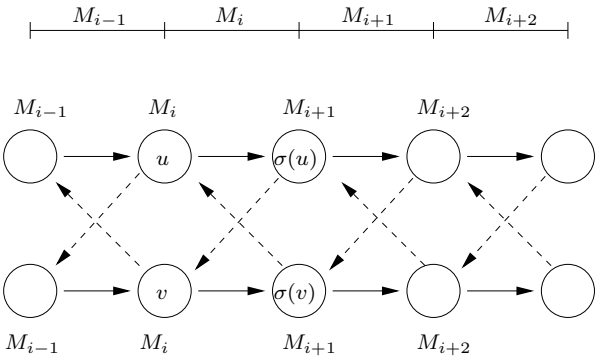

Figure 31: Two trains travelling on a single track.

These ideas are now generalized and described in a formal way. Consider two pairs of alternative arcs $(i \to j, h \to k)$ and $(s \to t, p \to l)$. Moreover let $M_{i\to j}$ and $M_{s\to t}$ be the sets of alternative arcs which are implied by $i \to j$ and $s \to t$. By definition let be $i \to j \in M_{i\to j}$ and $s \to t \in M_{s\to t}$.

As a first step obviously the following can be stated:

Remark 6.1 *Let S be a partial consistent selection for the alternative graph $G = (V, C, A)$ and $(i \to j, h \to k)$ be an additional alternative pair. If $G(S \cup \{i \to j\})$ contains a positive cycle then either $h \to k$ is implied by S or S has no extension (if even $G(S \cup \{h \to k\}$ contains a positive cycle).*

Remark 6.1 leads to a very simple methods to enlarge sets $M_{s\to t}$. All alternative arcs have to be checked whether they lead to a positive cycle with the current set $M_{s\to t}$. If an arc produces a positive cycle, its alternative is added to $M_{s\to t}$ and so on. If at a point $M_{s\to t}$ has no extension, the initial arc $s \to t$ cannot be chosen in a complete consistent selection. Otherwise it is known that whenever arc $s \to t$ is chosen in a selection then all arcs of $M_{s\to t}$ have to be chosen in order to stay feasible. This technique was already integrated in the greedy algorithms from Section 6.1 and can obviously be implemented in polynomial time.

By the strategy described above sets of implied arcs $M_{i\to j}$ can be initialized. In order to enlarge these sets or to fill them more effectively the following remarks can be stated.

Remark 6.2 *If $s \to t \in M_{i\to j}$, then $M_{s\to t} \subset M_{i\to j}$.*

A direct consequence is:

Remark 6.3 *If $s \to t \in M_{i\to j}$ and $i \to j \in M_{s\to t}$, then $M_{s\to t} = M_{i\to j}$.*

Notice that $s \to t \in M_{i\to j}$ means, that the choice of $s \to t$ is implied by the choice of $i \to j$. Reversing this argument leads to

Remark 6.4 *Let $(i \to j, h \to k)$ and $(s \to t, p \to l)$ be two alternative pairs. If $s \to t \in M_{i\to j}$ then $h \to k \in M_{p\to l}$.*

Proof: *Let $p \to l$ be chosen in a complete consistent selection. Suppose that even $i \to j$ is chosen. Then $s \to t \in M_{i\to j}$ implies $s \to t$ to be chosen, which is a contradiction, as only one arc of the pair $(s \to t, p \to l)$ can be fixed. Thus, $h \to k$ must be fixed.* □

Furthermore it can be proven

Conclusion 6.1 *Let $(i \to j, h \to k)$ and $(s \to t, p \to l)$ and $(a \to b, c \to d)$ be alternative pairs with $a \to b \in M_{i\to j}$ and $c \to d \in M_{s\to t}$.*
Then also $p \to l \in M_{i\to j}$ and $h \to k \in M_{s\to t}$

Proof: *From Remark 6.2 it follows that $M_{a\to b} \subset M_{i\to j}$ ($M_{c\to d} \subset M_{s\to t}$) and Remark 6.4 provides $h \to k \in M_{c\to d}$ ($p \to l \in M_{a\to b}$).*
Thus, $p \to l \in M_{i\to j}$ and $h \to k \in M_{s\to t}$. □

Summarizing the main considerations above three properties of the sets of implied arcs can be identified, which are:

(1) Reflexivity: $i \to j \in M_{i\to j}$ for all alternative arcs $i \to j$.

(2) Set-Transitivity: $s \to t \in M_{i\to j} \Rightarrow M_{s\to t} \subset M_{i\to j}$ for all all alternative arcs $i \to j$ and $s \to t$.

(3) Alternative-Symmetry: $s \to t \in M_{i\to j} \Rightarrow h \to k \in M_{p\to l}$ for all alternative pairs $(i \to j, h \to k)$ and $(s \to t, p \to l)$.

These properties may help to effectively enlarge the sets of implied arcs. They may even be used to propagate constraints, which were already set for example in a branch&bound procedure. For all algorithms which are based on the alternative graph model the choice of alternative arcs may be replaced by the choice of the corresponding sets. This leads to

Notation 6.1 *For an alternative pair* $(i \rightarrow j, h \rightarrow k)$ *the pair of sets* $(M_{i\rightarrow j}, M_{h\rightarrow k})$ *is called* **pair of alternative sets**.

For special cases of problems probably additional properties may be identified.

Next lower bounds and constraint propagation techniques, which will be helpful especially for enumerative algorithms, will be described. The techniques are mainly designed for exploiting the existence of a given upper bound UB for the objective function value in C_{max}-problems. They were developed for the classical job-shop problem and already transferred to blocking problems by Mascis and Pacciarelli [45].

A lower bound presented by Mascis and Pacciarelli [45] is based on the concept of **cliques**. A clique of operations is a set K of operations where no two of them can be processed simultaneously, i.e. for $i, j \in K$ one has either $s_j \geq s_i + p_i$ or $s_i \geq s_j + p_j$. This is for example the case for a set of operations to be processed on the same machine.

For blocking problems the time when a machine becomes available again after the processing of an operation i possibly does not only depend on p_i but also on the constraints on $s_{\sigma(i)}$. Thus, the processing time p_i may be dynamically replaced by larger values in optimization procedures for such problems.

For a clique K an operation $v \in K$ is called **input of** K, if in all feasible solutions v is processed before all other operations of K. It is called **output of** K if in all feasible solutions it is processed after all other operations of K.

Given a consistent selection S a release time $\bar{r}_i$ can be associated with each operation i. A lower bound for $\bar{r}_i$ is the length $l^S(0, i)$ of a longest $0 - i$-path in $G(S)$. Thus, a possible choice for $\bar{r}_i$ is $l^S(0, i)$.

Similarly a flow time $\bar{p}_i$ for i can be identified as a lower bound for the time period between the starting time of i and and the time when $j(i)$ leaves $\mu(i)$ in an optimal extension of S. A possible choice is $\bar{p}_i = p_i$, as $\bar{p}_i \geq p_i$ holds.

At last a delivery time $\bar{q}_i$ can be associated to i. It is defined as $\bar{q}_i = l^S(i, *) - \bar{p}_i$, i.e. it is a lower bound for the time period between the time when $j(i)$ leaves its machine and the makespan of an extension of S.

For blocking operations the values can be modified to $\bar{r}_i = \max\{l^S(0,i), l^S(0,\sigma(i)) - \bar{p}_i\}$, $\bar{p}_i = l^S(i,\sigma(i))$ and $\bar{q}_i = \max\{l^S(\sigma(i),*), l^S(i,*) - \bar{p}_i - \bar{r}_i + l^S(0,i)\}$.

If $\bar{r}_i = l^S(0,\sigma(i)) - \bar{p}_i$ this implies, that i is shifted right. But i may be the successor of another operation and thus, this right-shift could increase the makespan. Thus, $\bar{r}_i$ could be too large. The choice of $\bar{q}_i$ compensates too large values which may be chosen for $\bar{r}_i$. If $\bar{r}_i = l^S(0,i)$, then one has $\bar{q}_i = \max\{l^S(\sigma(i),*), l^S(i,*) - \bar{p}_i\}$, which is obviously okay. Otherwise one has $\bar{q}_i = \max\{l^S(\sigma(i),*), l^S(i,*) - \bar{p}_i - l^S(0,\sigma(i)) + \bar{p}_i + l^S(0,i)\} = \max\{l^S(\sigma(i),*), l^S(i,*) - (l^S(0,\sigma(i)) - l^S(0,i))\}$. Here the term $l^S(0,\sigma(i)) - l^S(0,i)$ compensates the fact, that $\bar{r}_i$ may be too large.

A lower bound for the makespan of an optimal extension of S can be computed by considering the corresponding single machine problem with release dates r_i and delivery times q_i for each clique K. A lower bound for such a problem is derived by building Jackson's preemptive schedule (see Carlier & Pinson [18]). Such a schedule can be computed with effort $O(|K|\log|K|)$ and has makespan

$$\max_{J\subset K}(\min_{j\in J} r_j + \sum_{j\in J} p_j + \min_{j\in J} q_j).$$

Thus, this is a suitable technique to compute lower bounds for example in branch&bound procedures.

The propagation technique presented by Mascis and Pacciarelli [45] is based on the investigation of **ascendant** and **descendant** sets.

Given a selection S let now be $r_i = l^S(0,i)$ and $q_i = l^S(i,*) - p_i$. Then given a bound UB for the makespan a set $J \subset N$ of operations is called ascendant set of c if $c \notin J$ and

$$\min_{j\in J\cup\{c\}} r_j + \sum_{j\in J\cup\{c\}} p_j + \min_{j\in J} q_j > UB.$$

In this case c is the output of the clique $K = J \cup \{c\}$. Then one can forbid all unselected alternative arcs $i \to j$ with weight a_{ij} which fulfill

$$l^S(c,i) + a_{ij} + \max_{k\in J}\{l^S(j,k)\} > 0.$$

This result can be derived by employing ideas from Carlier & Pinson [18] to alternative graphs. As c has to be processed after all operations in J the inequality above would indicate a positive cycle. For an ideal operation c all arcs $c \to j$ and for a blocking operation c all arcs $\sigma(c) \to j$ with $j \in J$ are forbidden.

The definition of a descendant set and the consequences for unselected alternative arcs can be derived symmetrically. $J \subset N$ is called descendant set of c if $c \notin J$ and

$$\min_{j \in J} r_j + \sum_{j \in J \cup \{c\}} p_j + \min_{j \in J \cup \{c\}} q_j > UB.$$

Then c is the input of the clique $K = J \cup \{c\}$ and one can forbid all unselected alternative arcs $i \rightarrow j$ with

$$\max_{k \in J} \{l^S(k,i)\} + a_{ij} + l^S(j,c) > 0.$$

The use of the constraint propagation techniques described above is twofold. On one hand they can be used to reduce the search space when looking for feasible solutions. Instead of selecting one arc out of an alternative pair, a set of implied arcs can be chosen for example in greedy heuristics. On the other hand the techniques can be used to compute lower bounds or to cut branches in branch&bound procedures. Some details of enumerative algorithms and branch&bound procedures are treated in the next section.

6.3 Enumeration techniques

In this Section enumeration techniques in order to find feasible solutions for job-shop scheduling problems with blocking and for railway scheduling problems or to prove that no such solution exists are presented. Based on this a Branch & Bound procedure to find good/optimal solutions is presented. Some versions of the described techniques can be found in the publication from Mascis and Pacciarelli [45].

A general sketch of an enumeration procedure for problems formulated by means of the alternative graph model is depicted in Listing 2. L is a list of incomplete selections. Starting from an empty selection in each step of the main loop an element of L (i.e. an incomplete selection) is chosen. For this selection two children are generated by selecting an unselected alternative pair and fixing the first or the second arc. For the choice of a parent selection from L different strategies are possible. The choice of an unselected alternative pair can be done due to the rules described in Section 6.1 (AMCC, SMCP, etc.).

```
1 begin
2 L := {∅};
3 while Stop criterion is not fulfilled do
4    begin
```

```
5    Choose parent S ∈ L;
6    L = L − {S};
7    Select an alternative pair ((h,k),(i,j)) ∈ A which is
       unselected in S;
8    Generate children S_1 = S ∪ {(h,k)} and S_2 = S ∪ {(i,j)};
9    for i=1,2 do
10      begin
11      if G(S_i) contains no positive cycle then
12         begin
13         Evaluate S_i and
14         possibly L = L ∪ {S_i};
15         end
16      end
17   end
18 end
```

Listing 2: Enumerative algorithm

Steps 5 and 7 define the branching scheme of the procedure. Depending on the choice of a parent solution $S \in L$ in Step 5 different strategies like depth-first or breadth-first-search can be realized. For example choosing in every step a parent with the maximal number of fixed alternative pairs leads to a depth-first-search. Which branches of the search tree are analyzed first depends on the priority rules used in Step 7.

The function *Evaluate* S_i investigates if a solution S_i should be added to L or not. For example in a situation where it can be seen that S_i has no extension it should not be added to L. By this function even a Branch&Bound procedure can be realized. In this case the function *Evaluate* S_i computes a lower bound LB for the objective value of any extension of S_i. This can be done using the techniques described in Section 6.2. Let UB be the objective value of the best feasible solution computed so far. Then S_i is added to L only if $LB < UB$.

The search can for example be stopped (Step 3) if a feasible solution is reached or if all feasible solutions have been investigated and an optimal solution has been found.

A specific Branch&Bound procedure for re-scheduling delayed trains in a railway network can be found in D'Ariano et al. [20].

6.4 Local search

In this Section local search heuristics like tabu search for both railway scheduling and job-shop problems are presented. Some similar ideas can be found in Pacciarelli and Pranzo [54]. The idea of local search is to explore the search space, i.e. the space of all solutions, by moving from one solution to another. These moves are done according to so-called neighbourhoods, which define a set of neighbour solutions for any solution in the search space. Note that the notion *solution* in this work does not necessarily mean a feasible solution but a complete selection, which may be consistent or inconsistent.

The intention of this Section is threefold. In the first part neighbourhoods for local search procedures and an underlying basic theory are described. Afterwards in the second part special repair procedures needed as subroutines for some of the neighbourhoods are discussed. Repair procedures are applied to infeasible solutions appearing during a search. They modify such infeasible solutions in order to restore feasibility. Finally in the third subsection details of the local search procedures such as tabu list strategies, etc. are presented.

6.4.1 Neighbourhood structures

The following considerations are related to problems with C_{max}-objective, but can easily be extended to L_{max} and other objectives. Before introducing different neighbourhood structures the following definition is made.

Definition 6.1 *Let $\mathcal{S}$ be the search space of an optimization or decision/feasibility problem and let $N : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{S})$ be a neighbourhood structure.*

(i) *N is called* **connected** *if any solution $s \in \mathcal{S}$ can be reached from any other solution $s' \in \mathcal{S}$ by doing a finite number of steps according to N.*

(ii) *Dealing with an optimization problem N is called* **opt-connected** *if from any solution $s \in \mathcal{S}$ a feasible optimal solution $s^* \in \mathcal{S}$ (if existent) can be reached by doing a finite number of steps according to N.*

(iii) *Dealing with a decision/feasibility problem N is called* **feasibility-connected** *if from any solution $s \in \mathcal{S}$ a feasible solution $s^f \in \mathcal{S}$ (if existent) can be reached by doing a finite number of steps according to N.*

The proposed approaches and neighbourhoods are based on the following theorem which is a direct consequence of a similar theorem for the classical JSP (see Brucker [9]).

Theorem 6.1 *Let S be a complete selection for a given alternative graph G.*
(i): If S is consistent, let P be a critical path in $G(S)$. If $\tilde{S}$ is a complete consistent selection with makespan smaller than S, at least one alternative arc of P does not belong to $\tilde{S}$.
(ii): If S is not consistent, let C be a positive cycle in $G(S)$. If $\tilde{S}$ is a complete consistent selection at least one alternative arc of C does not belong to $\tilde{S}$.

Proof: *Both, (i) and (ii) are obviously true.*

□

Using Theorem 6.1 we may think of two different approaches, i.e. two different types of neighbourhoods to be used during local search.

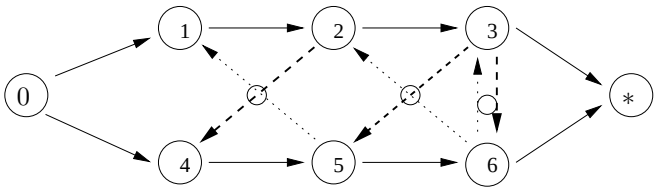

Figure 32: No replacement of a chosen alternative arc (bold dashed arcs) possible.

The first approach is based on the idea of moving from one feasible solution to another by applying a suitable neighbourhood. But for some complete consistent selections each replacement of an alternative arc leads to an infeasible situation (see Figure 32), i.e. to a positive cycle in the resulting graph. Therefore a neighbourhood must be able to replace more than one alternative arc at the same time, i.e. it must be able to repair such situations if possible. A generic formulation of this type of neighbourhood is the following.

Neighbourhood N_1: *Let S be a complete consistent selection for a given alternative graph G and P be a critical path in $G(S)$. Then the neighbourhood $N_1(S)$ is the set of all complete consistent selections which are obtained by the following two steps:*
(i) (At least) one alternative arc of an arbitrary critical path P is replaced by its alternative. (If P contains no alternative arcs, S is optimal and one can stop.)
(ii) If the corresponding new selection is inconsistent, this selection is repaired by replacing other alternative arcs suitably.

In general - for an arbitrary repair procedure - N_1 is neither opt-connected nor feasibility-connected.

Note that the basis for using N_1 is a given complete consistent selection and thus it is only eligible for optimization procedures. The problem of finding a complete consistent

selection is NP-complete for the problem with time-windows and even for the problem where a partial selection is given. Of course, for problems IJSP, BWSJSP and BNSJSP a feasible (but possibly very bad) starting solution can always be found, as all jobs can be scheduled one by one from the beginning to the end.

One special case of N_1 is a neighbourhood, where the first step allows only to replace exactly one alternative arc of a critical path. For problem BNSJSP Mati et al. [47] developed a tabu search procedure. They use a specific implementation of the described type of a N_1 neighbourhood. In their paper an extension of a geometric approach developed by Brucker [8] for the problem $J \mid n = 2 \mid C_{max}$ is used in order to repair infeasible solutions (inconsistent selections) in the second step. Unfortunately Mati et al. provide only very few computational results.

Before describing different approaches to repair inconsistent selections in the next section (second step of neighbourhood N_1) and discussing the relevant underlying theory another type of neighbourhood is proposed. This second approach is quite different. The second neighbourhood does not provide an approach to repair inconsistent selections after having replaced alternative arcs. Thus, it must be allowed to move to (and even start from) inconsistent complete selections. Case (ii) of the theorem above gives a hint how to come (back) to feasible solutions (complete consistent selections). A generic formulation of this type of neighbourhoods is the following.

Neighbourhood N_2: *Let S be a complete selection for a given alternative graph G.*
(i): If S is consistent, let P be an arbitrary critical path in $G(S)$. The neighbourhood $N_2(S)$ is the set of all complete selections derived by replacing one or more alternative arc of an arbitrary critical path P by its alternative. (If P contains no alternative arcs, S is optimal and the search process can be stopped.)
(ii): If S is not consistent, let C be a positive cycle in $G(S)$. Then the neighbourhood $N_2(S)$ is the set of all complete selections derived by replacing one or more alternative arc of an arbitrary positive cycle C by its alternative. (If $G(S)$ has a positive cycle C containing no alternative arcs, the problem is unfeasible and the search process can be stopped.)

Neighbourhood N_2 can also be used when not having an initial complete consistent selection and especially when looking only for a feasible solution. In this case only step (ii) of the neighbourhhod is used and the search process is stopped when having found a feasible solution, i.e. a complete consistent selection. As for N_1 one special case of N_2 is a neighbourhood where in both cases, (i) and (ii), only replacing exactly one alternative arc is allowed.

Obviously, N_2 has the following

Property 6.1 *The neighbourhood N_2 is both opt-connected and feasibility-connected.*

6.4.2 Repair procedures for inconsistent selections

In order to apply neighbourhoods of the N_1-type methods to repair infeasible solutions (inconsistent complete selection) resulting from step (i) of the neighbourhood are needed. A special approach for the BNSP can be found in Mati et al. [47]. As already mentioned above they use an extension of a geometric approach developed by Brucker et al. [8] for the problem $J \mid n = 2 \mid C_{max}$ in order to repair infeasible solutions (inconsistent complete selections) in the second step.

Now another repair procedure is described by defining a specific implementation of the neighbourhood N_1. This implementation of N_1 for the blocking job-shop problem with no swap allowed (BNSP) is defined as follows:

Neighbourhood N_{1A}: *Let G be the alternative graph corresponding to a blocking job-shop problem with no swap allowed (BNSP). Let S be a complete consistent selection and P be an arbitrary critical path in $G(S)$. Then the neighbourhood $N_{1A}(S)$ is the set of all complete consistent selections which are obtained by the following two steps:*
(i) One alternative arc $i \rightarrow j$ of an arbitrary critical path P is replaced by its alternative $h \rightarrow k$.
(ii) If the corresponding new selection is inconsistent, the entire job $j(k)$ of operation k is shifted to the end of a schedule by replacing all corresponding alternative arcs. Obviously this leads to a new complete consistent selection, as the partial schedule containing all jobs but $j(k)$ stays feasible. Scheduling $j(k)$ after all jobs cannot create a positive cycle, as then there are no arcs going back from $j(k)$ to any other job.

In Figure 33, 34, 35 and 36 an example is depicted which proves that N_{1A} is not opt-connected. For the complete consistent selection depicted in Figure 33 only one alternative arc (the dashed one) is located on a critical path. Replacing this alternative arc and repairing the new selection according to N_{1A} leads to the schedule depicted in Figure 34. The alternative graph with the corresponding complete consistent selection is depicted in Figure 35. Applying N_{1A} to this solution leads back to the initial solution. A better solution and thus, an optimal one as given in Figure 36 cannot be reached.

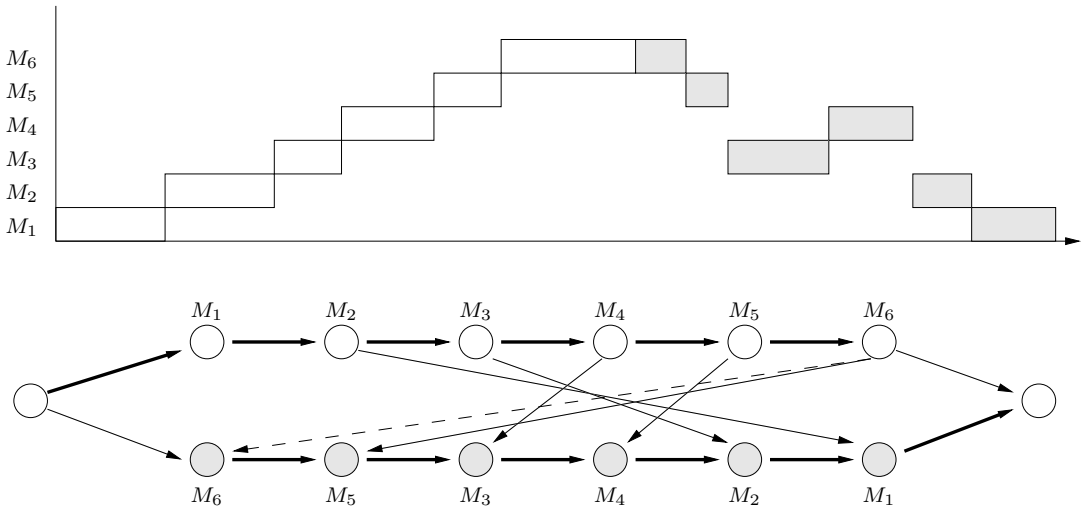

Figure 33: Complete consistent selection for a blocking JSP.

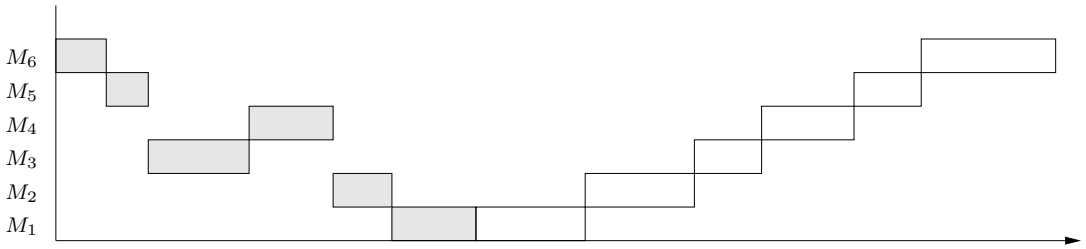

Figure 34: Schedule for Example from Figure 33 after one step of neighbourhood N_{1A}.

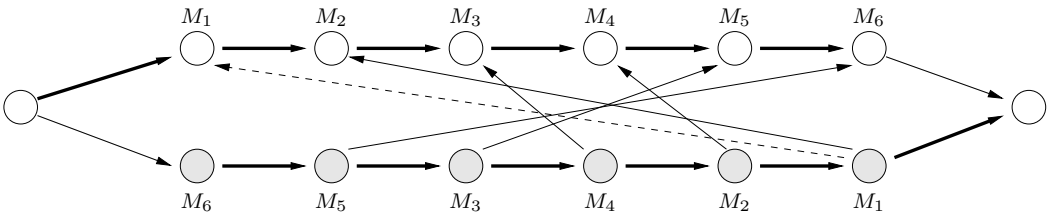

Figure 35: Complete consistent selection corresponding to Figure 34.

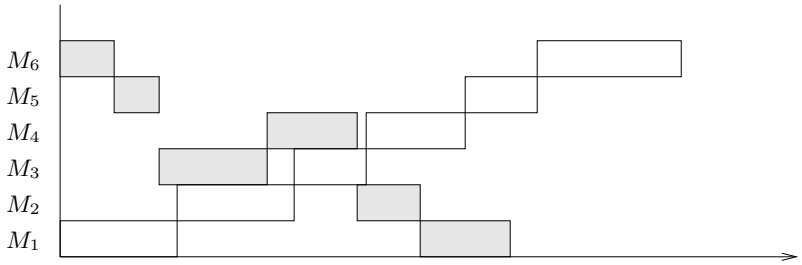

Figure 36: An optimal schedule for Example from Figure 33.

6.4.3 Local search strategies

Based on the proposed neighbourhoods different local search methods and especially different strategies for applying tabu search to the problem can be developed. In this section different ideas for approaches to get feasible or good solutions are described. These approaches include both, search methods which visit only feasible solutions (complete consistent selections) and those which allow to move to (and even start from) infeasible solutions (complete but inconsistent selections).

Firstly some basics of local search procedures are introduced. Especially the structure of so-called tabu search methods is presented. The general idea of local search is to move from one solution to another by following a neighbourhood structure. A generic local search procedure is depicted in Listing 3.

```
1  begin
2  Compute an initial solution s ∈ S;
3  best solution s* := s;
4  while Stop criterion is not fulfilled do
5     begin
6     determine a neighbour s' ∈ N(s);
7     s := s';
8     if c(s') < c(s*) then s* := s';
9     end
10 end
```

Listing 3: Generic local search

Tabu search is a special local search strategy which was applied very successfully to different job-shop problems in the past. Tabu search applies special strategies in order to avoid going back to solutions which have already been visited during the search process, i.e. to avoid following circuits during the search. This is done by storing attributes of already visited solutions in a list called tabu list. Going back to solutions with the same attributes as stored in the list is forbidden, those solutions are **tabu**.

A sketch of a general tabu search algorithm is drawn in Listing 4. Here $Cand(s)$ is the set of all neighbours of s which are not tabu. In Step 7 $s' \notin T$ means, that the attributes of s' are not contained in the tabu list T.

```
1  begin
2  Compute an initial solution s ∈ S;
3  best solution s* := s;
4  tabu list T := ∅;
5  while Stop criterion is not fulfilled do
6     begin
7     determine Cand(s) := {s' ∈ N(s)|s' ∉ T};
8     if Cand(s) ≠ ∅ then
9        determine s' ∈ Cand(s)
10       s := s';
11       T := T ∪ {s};
12       if c(s) < c(s*) then
13          s* := s;
14    end
15 end
```

Listing 4: Generic tabu search

When using neighbourhood N_1, and thus only visiting feasible solutions (complete consistent selections) tabu strategies similar to strategies known from the classical job-shop (see Nowicki and Smutnicki [52]) can be used. One possible strategy is the following. For any solution already visited a 4-tupel containing the following data is stored in the tabulist:

- the alternative arc which was reversed in order to get a neighbour of this solution,
- the alternative arcs preceding this arc on the considered critical path,
- the alternative arcs succeeding this arc on the considered critical path, and
- the corresponding value of the objective function.

A solution is **tabu**, if one of the 4-tupels in the tabulist is reconstructed. Obviously these criteria define more solutions to be tabu than the ones already visited. In order not to set solutions tabu which improve the currently best solution an **aspiration criterion** is introduced. It guarantees that solutions with objective value better than the currently best one are not tabu. A closer look shows that an aspiration criterion is already included. Solutions which are better than the best solution already visited cannot reconstruct one of these 4-tuples, as at least their objective value must be different (better).

Using neighbourhood N_2, different tabu criteria are needed in order to describe feasible and especially infeasible solutions which have already been visited. Which criteria are suitable could be analyzed by developing and testing different approaches.

In the next section a decomposition approach for railway scheduling problems is presented.

7 Problem decomposition

So far this thesis was dedicated to models and algorithms for railway and also job-shop scheduling problems as a whole. This section deals with a decomposition approach for specific railway scheduling problems. The approach is based on ideas which were already developed in cooperation with Dario Pacciarelli from the Universita 'Roma Tre' within the EU-project COMBINE II [2] [26]. There a basic model as well as some ideas for solution methods were introduced. In this thesis the model and corresponding approaches are elaborated in detail and supplemented. Practical methods as well as theoretical aspects are presented. The main accent is put on the goal to compute feasible solutions for railway networks which are decomposed into different local networks.

7.1 A decomposition model

In this subsection the decomposition model which is applied to the railway problem introduced above is described. Based on a geographical decomposition of the physical railway network in local networks the problem is divided into local problems corresponding to these local networks. These local problems are then solved and the whole process is coordinated at a higher level in order to produce a globally feasible solution.

Some additional assumptions on the structure of the given problems are made. Starting from the basic model also upper bounds for travelling times, time-window constraints (release-dates and deadlines), starting and ending constraints, and connection constraints between trains are allowed. The last two constraints are assumed to arise only inside local networks and not at borders between different local networks. General time constraints as well as out-of-service intervals are assumed to be absent, as they would complicate the decomposition model. Nevertheless even these constraints could be integrated by generalizing the model slightly. Some additional assumptions will be stated later when needed.

Describing the decomposition of the physical railway network the following notation is used. A **clearing point** is a point between two different block sections of a railway network, i.e. a point between different safety segments of the network. Each clearing point between two block sections corresponds to nodes in the alternative graph, which represent crossing times of trains at this physical point.

A geographical decomposition of a physical railway network can be defined by identifying borders and border sections (block sections) between adjacent local networks. By this

[2] Christian Strotmann took part in this project as scientist as well as his supervisor Prof. Dr. Peter Brucker.

the whole physical railway network is divided into local networks. Especially the set of all clearing points is divided into disjoint sets belonging to the different local networks. An example with two local railway networks is shown in Figure 37. This network is divided into two local networks. Additionally the routes of two (slow) trains A and B are depicted.

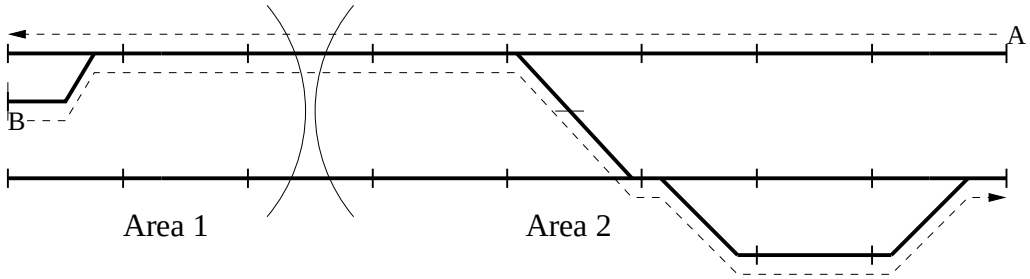

Figure 37: A railway network divided in two local networks.

In order to decompose the problem of scheduling traffic in the whole railway network into smaller problems corresponding to the local networks the alternative graph for the whole problem is divided into different local graphs. Note that each node of the alternative graph but the source and the sink is associated with exactly one certain clearing point in the physical network as it represents a crossing time of a train at this clearing point.

Based on the division of all clearing points the set of all nodes of the large alternative graph $G = (V, C, A)$ but the source and the sink is divided into disjoint sets corresponding to the local networks. So-called **basic local graphs** corresponding to the physical local networks are defined. Summarizing the considerations above now each node belongs to a basic local graph iff its clearing point belongs to the corresponding local network. By construction the node sets belonging to the basic local graphs together with nodes/sets 0 and $*$ build a disjoint decomposition of V.

If all corresponding arcs and alternative pairs are added to such a basic local graph, this graph represents the problem of scheduling traffic within the corresponding local network. But by considering only basic local graphs the scheduling of traffic at borders is missing. Therefore the local areas have to be enlarged by so-called **border elements** which are defined next. An arc is called **border crossing** if it connects nodes belonging to different basic local graphs, i.e. if it represents a constraint between events occurring within different local networks.

Each border element is a subgraph of the alternative graph for the whole problem. It corresponds to a single border, i.e. a block section lying on a border between two adjacent

local networks. It optionally includes parts of the graph corresponding to some neighboured block sections. More precisely the border element for a border is an alternative subgraph consisting of:

- All fixed arcs crossing this border.
- All pairs of alternative arcs where two of the four end nodes of the pair belong to different basic local graphs at this border.
- The set N of all nodes which are incident with these arcs and alternative arcs.
- All fixed arcs connecting nodes in N.
- All pairs of alternative arcs where all nodes which are incident to this pair belong to N.

Border elements represent traffic at the corresponding borders between different local networks. The idea of this definition is to include all fixed arcs and alternative arcs/pairs which are incident with nodes belonging to different basic local graphs. Note, that 0 or $*$ never belong to a border element.

In Figure 38 to 44 different situations arising at borders and the corresponding types of border elements are depicted. For the sake of clarity each node (representing the entrance of a train in a block section M_i) is placed beneath its associated clearing point and denoted by the block section M_i.

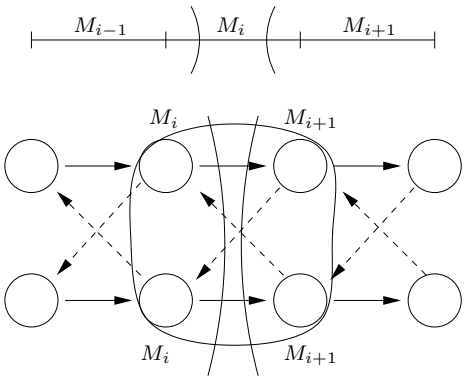

Figure 38: A border situation with two slow trains travelling in the same direction.

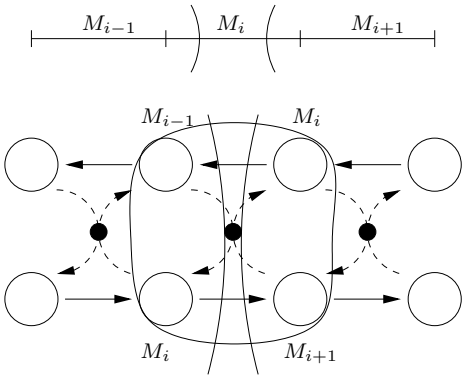

Figure 39: A border situation with two slow trains travelling in opposite direction.

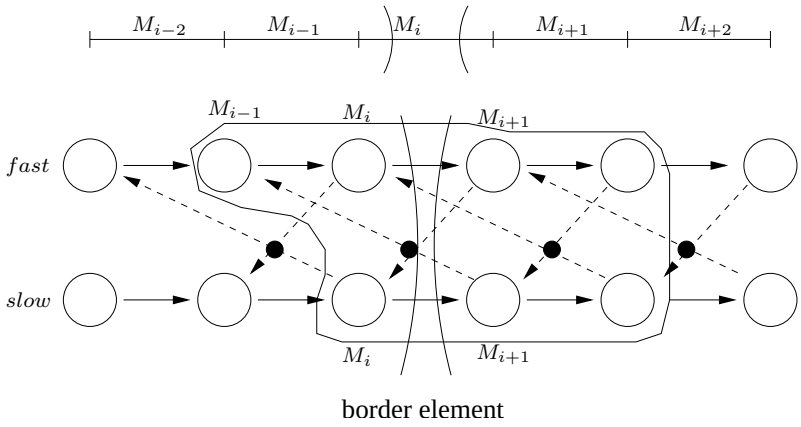

Figure 40: A border situation with a slow and a fast train travelling in the same direction.

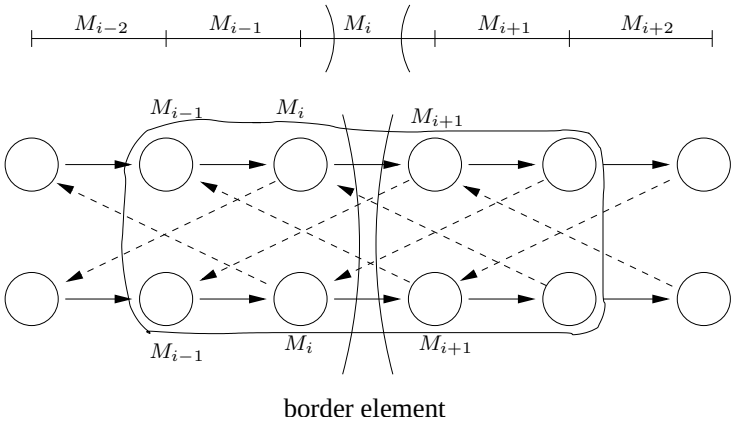

Figure 41: A border situation with two fast trains travelling in the same direction.

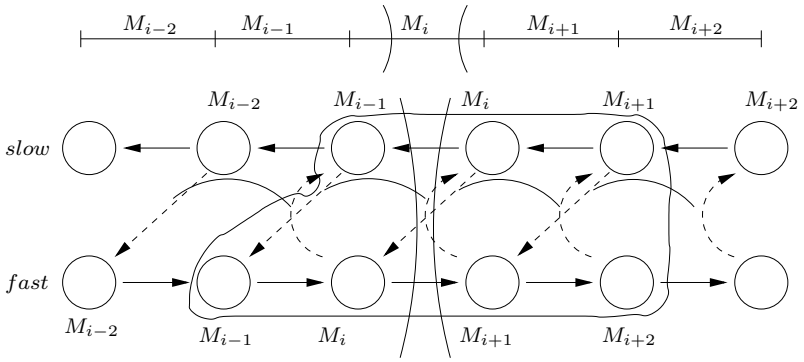

Figure 42: A border situation with a slow and a fast train travelling in opposite direction.

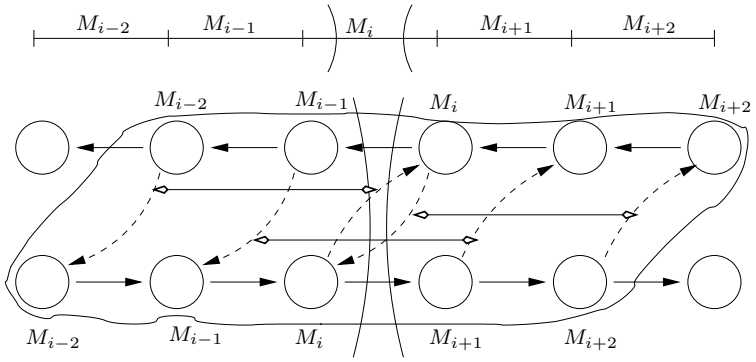

Figure 43: A border situation with two fast trains travelling in opposite direction.

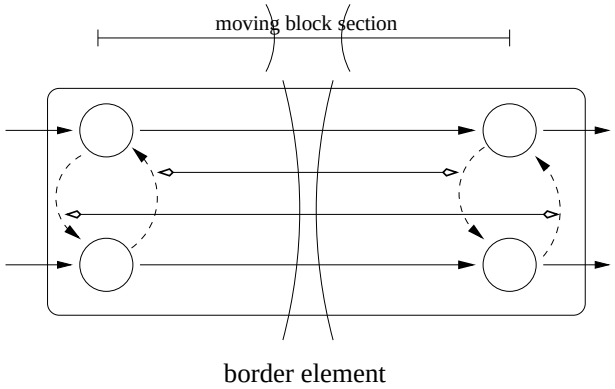

Figure 44: A border situation with two moving block trains.

The following assumptions on the considered physical railway networks and the corresponding graphs can be made, as they can be fulfilled in most real-world problems easily.

Assumption 7.1 *1. Each border element contains no nodes (and no arcs) belonging to a junction, i.e. borders are not located at junctions.*

2. Border elements of different borders are disjoint.

Nodes belonging to a border element are called **border nodes**. Additionally the source 0 and the sink $*$ belong to the set of border nodes. In Figure 45 the alternative graph corresponding to the example above (Figure 37) is depicted (the source and the sink node are not depicted). The border element which models the physical border contains four nodes of the graph.

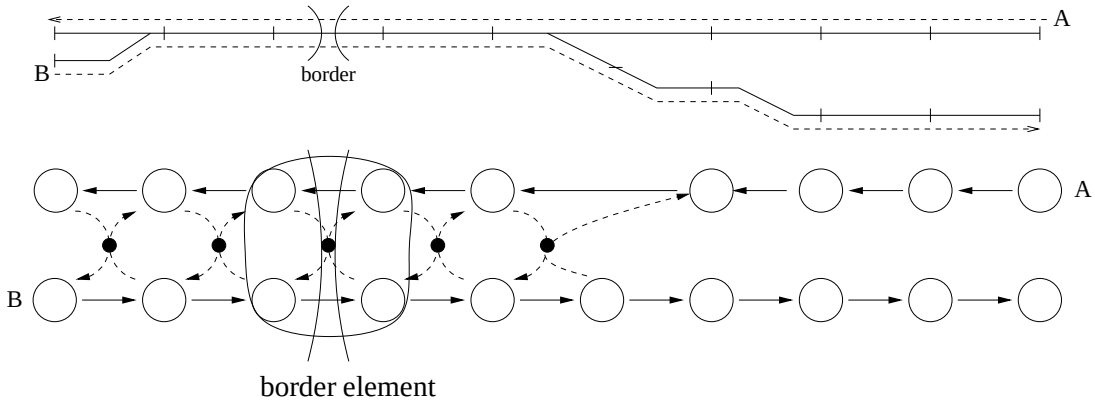

Figure 45: The alternative graph corresponding to the problem from Figure 37.

Based on the structure of the considered (railway) problems the following property can be stated.

Property 7.1 *Each fixed arc or alternative pair which is incident with nodes of different basic local graphs belongs to (exactly) one border element. In particular all nodes of the arc or pair are border nodes of the same border element. (In case of an alternative arc the whole pair belongs to the border element.)*

In the next Step of the decomposition approach border elements are integrated in the basic local graphs.

A **local graph** $G_L = (V_L, C_L, A_L)$ represents traffic in a corresponding local network L (and at its adjacent borders) and is defined as follows:

- V_L contains all nodes belonging to the corresponding basic local graph and all border nodes belonging to border elements which are incident with this graph. (A border element is **incident** with a basic local graph iff it contains at least one node of this basic local graph.) Additionally V_L contains the source and the sink.
- C_L contains all fixed arcs connecting nodes of V_L.
- A_L contains all pairs of alternative arcs where both arcs connect nodes of V_L.

Note that each border element is duplicated as is belongs to the local graphs of both local networks which are incident with this border. The two local graphs and border elements corresponding to Figure 45 are shown in Figure 46.

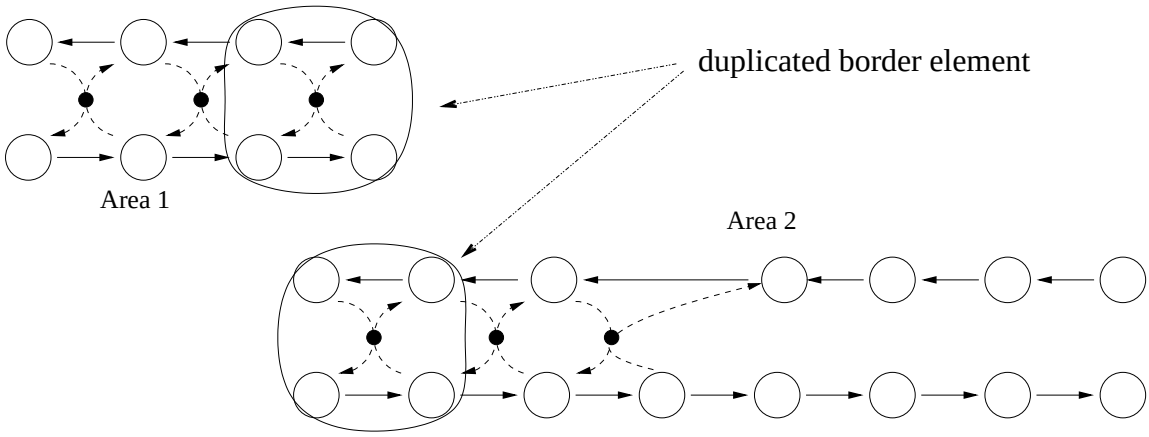

Figure 46: The graph from Figure 45 divided into two local graphs.

Having defined the local graphs above it is clear, that any node, fixed arc, and alternative pair of the graph for the whole problem is contained in at least one of the local graphs.

Now local solutions can be computed for all local graphs (and thus for all local networks) independently using the techniques described in Section 6. A local feasible solution corresponds to a complete consistent selection for the corresponding local alternative graph (see 4.1.2). The graph $G_L = (V_L, C_L \cup S_L)$, where S_L is a complete consistent selection, is called a **local solution graph** for the local network L.

In Figure 47 local solution graphs corresponding to the local graphs in Figure 46 are depicted. Now the source and the sink are depicted for further considerations.

In order to check whether local feasibility provides global feasibility a **coordinator graph** induced by the local feasible solutions (local solution graphs) is defined. The idea is to

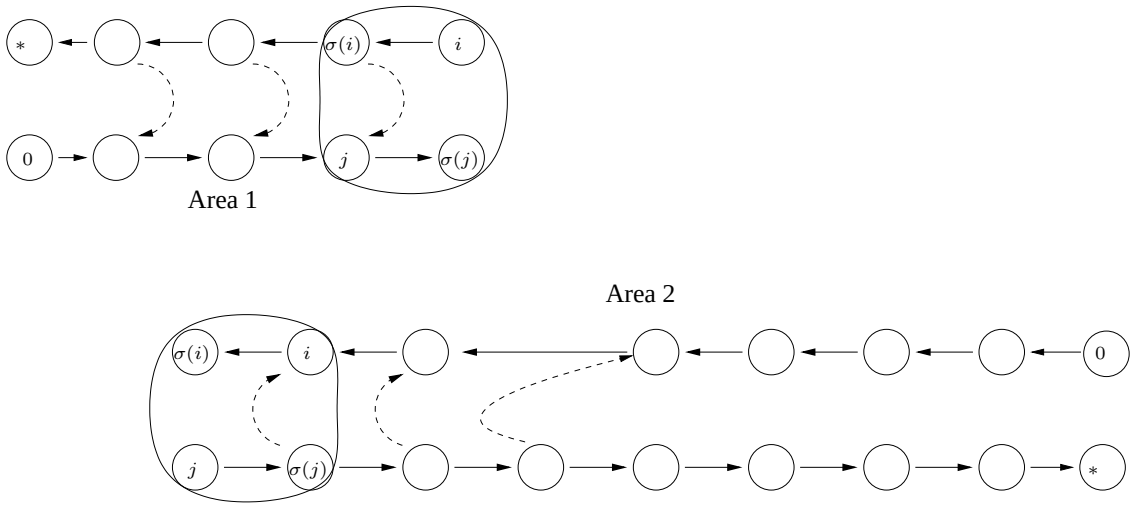

Figure 47: Feasible solutions for the local graphs from Figure 46.

have a condensed graph where paths between border nodes in local solution graphs are represented by fixed arcs. Later it will be shown that this condensed information is sufficient for an coordinator to control and force global feasibility. Starting from local feasible solutions (local complete consistent selections) for all local graphs the coordinator graph $G_B = (V_B, A_B)$ is defined as follows:

- V_B is the set of all border nodes (nodes which belong to border elements and nodes 0 and $*$).
- A_B is the set of so-called **border arcs**. The set of border arcs is defined as follows. Let i, j be two different border nodes (having a local graph in common). The arc $i \to j$ belongs to A_B iff in at least one of the local solution graphs a directed path from i to j exists. Then the weight of $i \to j$ in the coordinator graph is the length of a corresponding longest directed path among all such directed local paths. (There may be two or more local solution graphs including such a corresponding directed path).

Which directed paths are present between border nodes in the local solution graphs and (if present) the lengths' of corresponding longest directed paths can be investigated by longest path calculations. Those can be done for example by applying a longest-path-version of the Floyd-Warshall algorithm (see Ahuja et al. [4]) to each local solution graph.

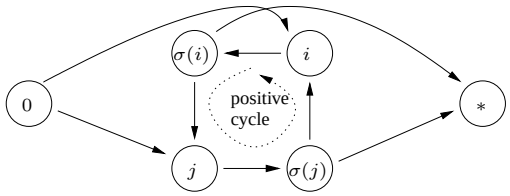

Figure 48: Coordinator graph corresponding to local solution in Figure 47.

The coordinator graph associated with the local solutions graphs in Figure 47 is depicted in Figure 48. Note, that due to more clarity not all border arcs and no weights are shown.

Especially the fixed arcs of border elements, locally chosen alternative arcs between border nodes and accompanying transitively induced arcs belong to the coordinator graph as they are special directed paths between border nodes in local solution graphs. Note, that possibly many arcs which are transitively induced by others may belong to the coordinator graph. In order to check local solutions for global feasibility the following theorem can be used.

Theorem 7.1 *Let be given a feasible local solution for each local network (i.e. a complete consistent selection for each local graph). Then these local feasible solutions are feasible on a global level (globally feasible) if and only if the corresponding coordinator graph contains no positive cycle.*

Before proving Theorem 7.1, some lemmata are given. G_{all} denotes the union of all local solution graphs, i.e. it is the union of all nodes and all fixed arcs of the local graphs and all alternative arcs which are contained in one of the local complete consistent selections.

Lemma 7.1 *Let $(u_1 \rightarrow u_2 \rightarrow \ldots \rightarrow u_n)$ with $n \geq 3$ be a directed path in G_{all} and u_1 and u_n be the only border nodes of this path. Then there exists exactly one local solution graph where all nodes and arcs (and thus the whole path) are present.*

Proof: *As u_2 is a non-border-node it is contained in exactly one local (solution) graph. Thus, even the arcs $u_1 \rightarrow u_2$ and $u_2 \rightarrow u_3$ can only be contained in the same local solution graph. For $n = 3$, this proves the lemma. If $n > 3$, then the non-border-node u_3 is only contained in exactly one local solution graph and this must be the same as for u_2, as the arc $u_2 \rightarrow u_3$ can only exist there. Repeating this argument proves the lemma.* □

In this thesis a cycle which contains no subcycles is called **minimal cycle**. Obviously a graph contains a positive cycle if it contains a minimal positive cycle. The opposite direction is stated by

Lemma 7.2 *If a graph contains a positive cycle, it contains a minimal positive cycle.*

Proof: *A minimal positive cycle can be found by iteratively looking for minimal subcycles of a positive cycle and deleting them if they have non-positive length. This method always terminates with a minimal positive cycle either when a minimal subcycle with positive length is found or when no more (minimal) subcycles exist and all deleted subcycles had non-positive length. Then the minimal cycle at the end must have positive length.* □

Lemma 7.3 *Let C be a minimal positive cycle in G_{all}. Then C contains at least two border nodes.*

Proof: *If C would contain at most one border node it would be a local cycle which follows from the same argumentation as in the proof of Lemma 7.1. Then the corresponding local solution graph would contain this positive cycle C which contradicts the feasibility of all local solutions.* □

Note, that especially in the case in which an arc of a border element is included in such a positive cycle, this cycle contains two border nodes (as an arc of an border element connects two border nodes).

Now the proof of Theorem 7.1 is given.

Proof: *It is assumed that local feasible solutions (feasible local solution graphs) are given for all local networks.*

Consider the graph G_{all}, which is derived by sewing together all local solution graphs (see above).

If G_{all} contains no positive cycle it represents a globally feasible solution. Note, that any node, fixed arc, and alternative pair of the graph for the whole problem is contained in at least one local graph and each alternative pair is chosen in at least one of the local selections. From each alternative pair exactly one arc is chosen, and thus, the global selection is complete. Otherwise an alternative pair which is chosen differently in different local selections results in a positive cycle in G_{all}. This global complete selection is consistent as no positive cycle occurs in G_{all}.

Thus, if the given local feasible solutions are infeasible on a global level, the graph G_{all} must contain a positive cycle. Such a positive cycle may on one hand be caused by alternative pairs which exist in different local alternative graphs but are not chosen equally there. On the other hand inconsistencies in the temporal constraints may lead to such a positive cycle.

W.l.o.g the cycle may be assumed to be a minimal one, as such a minimal one then must exist and can be computed from a given positive cycle (see Lemma 7.2). In any case a positive cycle C in G_{all} must consist of parts located in different local solutions graphs. Otherwise the conflict would be a local one, which contradicts the premises of the theorem. From Lemma 7.3 it follows that C must contain at least two border nodes.

Now let the positive cycle be $C = (u_1, \ldots, u_2, \ldots, \ldots, u_n = u_1)$, where $(u_1, u_2, \ldots, u_n = u_1)$ is the sequence of all its border nodes. For each pair u_i and u_{i+1} $(i = 1, \ldots, n-1)$ two cases can be identified:
Case 1: $u_i \rightarrow u_{i+1}$ is an arc of C. Then $u_i \rightarrow u_{i+1}$ is contained in at least one local solution graph. This follows directly from the definition of G_{all}.
Case 2: There is a path from u_i to u_{i+1}. Then there exists exactly one local solution graph where all nodes and arcs (and thus the whole path) are present. (see Lemma 7.1).
For both cases it is known that the coordinator graph G_B contains an arc $u_i \rightarrow u_{i+1}$, since a corresponding local directed path exists. The length of such an arc in the coordinator graph is at least the length of the corresponding arc or path in C (in the corresponding local solution graph).
It follows that $(u_1 \rightarrow u_2 \rightarrow \ldots \rightarrow u_n = u_1)$ is a cycle in G_B with at least the length of C. Thus, G_B contains a positive cycle!

Now consider the coordinator graph G_B derived from the local feasible solution graphs.
Let $C = (u_1, u_2, \ldots, u_n = u_1)$ be a positive cycle in G_B.
Each arc $u_i \rightarrow u_{i+1}$ in C corresponds to a directed path of the same length in one of the local solution graphs.
Therefore each arc $u_i \rightarrow u_{i+1}$ corresponds to a directed path of the same length in G_{all}. and the paths $u_1 \rightarrow \ldots \rightarrow u_2 \rightarrow \ldots, \ldots \rightarrow u_n = u_1$ build a cycle in G_{all} with the length of C.
Thus, $u_1 \rightarrow \ldots \rightarrow u_2 \rightarrow \ldots, \ldots \rightarrow u_n = u_1$ is a positive cycle in G_{all}.
Then by definition of G_{all} the local solutions are globally infeasible.

Summarizing the results above the following has been shown:

Local solutions are globally infeasible if and only if the coordinator graph G_B contains a positive cycle.

This is equivalent to the statement of Theorem 7.1.

□

Using a longest-path-version of the Floyd-Warshall algorithm (see Ahuja et al. [4]) the coordinator graph can be checked for the existence of positive cycles with polynomial effort and even positive cycles can be identified.

For the example above a positive cycle can be identified in the coordinator graph from Figure 48. Thus, the local solutions are unfeasible on a global level.

With a similar argumentation as used in the proof of Theorem 7.1 a statement concerning an optimal globally feasible solution can be proved. More precisely one can show

Theorem 7.2 *Let be given a globally feasible solution composed from local feasible solutions for all local graphs and let G_B be the corresponding coordinator graph. If this global solution is not optimal, then in an optimal solution (with coordinator graph G_B^{opt}) one of the following conditions is satisfied:*

(i) At least one border arc on a critical path in G_B is shorter in G_B^{opt}.

(ii) At least one border arc on a critical path in G_B does not exist in G_B^{opt}.

The next section describes methods in order to get global feasible solutions for the decomposed problem.

7.2 Methods to solve the decomposed problem

Having introduced a suitable decomposition model, now methods in order to deal with conflicts on a global level are presented. At first different types of conflicts on a global level are classified and basic methods in order to solve these conflicts are presented. In a second part strategies in order to control a global search procedure for finding global feasible solutions are developed.

7.2.1 Solving global conflicts - general techniques

Before starting with the detailed description a sketch of the general approach for computing a global feasible solution is drawn. The idea of this procedure is the following. In each step local feasible solutions (if existent) are computed for all local problems. Then the coordinator graph is build and checked for global feasibility. If the local solutions provide a global infeasible solution artificial constraints are imposed to the local problems in order to lead them to global feasibility. If for some local problems local feasible solutions cannot be found or do not exist one may modify or relax the artificial constraints. The process is repeated until a globally feasible solution is reached or a stopping criterion is fulfilled. As a possible stopping criterion a maximal number of coordination steps may be used.

```
1  begin
2  for all local alt. graphs do
3     Compute local solutions;
4     if a local solution is unfeasible then
5        stop;
6  L_Feas = 1;
7  Build the coordinator graph G_b;
8  if G_b contains a pos. cycle then
9     G_Feas = 0;
10 else G_Feas = 1;
11 while G_Feas == 0 and Stopcriterion is not fulfilled do
12    begin
13    Impose artificial constraints to loc. alt. graphs;
14    for all local alt. graphs do
15       Compute (new) local solutions;
16       if a local solution is unfeasible then
17          L_Feas = 0;
18       else L_Feas = 1;
19    while L_Feas == 0 and Stopcriterion is not fulfilled do
20       begin
21       Relax/Modify some artificial constraints;
22       for all local alt. graphs do
23          Compute (new) local solutions;
24          if a local solutions is unfeasible then
25             L_Feas = 0;
26          else L_Feas = 1;
27       end
28    if L_Feas == 0 then
29       stop;
30    Eliminate the artificial constraints from the local
       solution graphs;
31    Rebuild the coordinator graph G_b;
32    if G_b contains a pos. cycle then
33       G_Feas = 0;
34       Reinsert the artificial constraints to the local
          graphs;
35    else G_Feas = 1;
36    end
```

```
37 if G_Feas == 0 then
38    UNSUCCESS (no global feasible solution is found);
39 else
40    SUCCESS (global feasible solution is found);
41 end
```

Listing 5: Sketch of coordination procedure

In the following techniques in order to impose artificial constraints to local alternative graphs are developed. Different strategies how to choose these artificial constraints are presented in the next subsection.

As mentioned above conflicts between solutions for different local areas result in positive (directed) cycles in the coordinator graph. In the following one may always assume that positive (global) cycles are minimal, i.e. they do not contain a node more than once and thus do not contain subcycles. This is no restriction as any positive cycle must contain a minimal positive cycle. Such a minimal positive cycle can be found by iteratively looking for minimal subcycles of a positive cycle and deleting them if they have non-positive length. This method always terminates with a minimal positive cycle either when a minimal subcycle with positive length is found or when no such subcycle exists and all deleted subcycles had non-positive length. Then the minimal cycle at the end must have positive length. Obviously a minimal positive cycle can be computed from a general positive cycle with polynomial computational effort.

If a positive cycle is detected in the coordinator graph some actions in order to eliminate this cycle have to be undertaken. As the coordination level has no detailed information about local solutions, it has to use information about the structure of paths in local areas which are represented by border arcs in the coordinator graph. For example a border arc in a cycle may represent a path (in a local area graph) which consists only of fixed arcs. Then the coordination level knows that it is not possible to eliminate this part of the cycle. On the other hand a border arc representing a path with alternative arcs may be forced to become shorter or to disappear.

The question now is, what the coordination level can do in order to force some local areas to produce shorter longest paths in the next round. The idea is to do this by introducing new (constraining) arcs with suitable weights. Detailed techniques are described later.

Firstly different types of global conflicts resulting from different types of border arcs are described. In order to classify different types of border arcs firstly a graph G_{min} is introduced. G_{min} is the graph for the whole railway network which contains only fixed arcs. If now all longest paths between border nodes are computed in the restrictions of this graph to the corresponding local networks, i.e. in the local alternative graphs with only

fixed arcs, a set of necessary conditions for the starting times of all pairs (i,j) of border nodes is generated.

Let l_{ij} be the length of border arc $i \rightarrow j$ resulting from a solution for the local graphs and l_{ij}^{min} be the length of a corresponding longest path in the restrictions of G_{min} to the local networks, if such paths exist there. It is distinguished between the following types of border arcs $i \rightarrow j$:

- $i \rightarrow j$ is **relaxable**, iff there is no directed path from i to j in the restriction of G_{min} to the corresponding local areas.
- $i \rightarrow j$ is **non-relaxable**, iff there is a directed path from i to j in the restrictions of G_{min} to the corresponding local areas and $l_{ij} = l_{ij}^{min}$. (The length of the paths from i to j represented by the border arc $i \rightarrow j$ cannot be decreased as it is independent from the choice of a specific solution.)
- $i \rightarrow j$ is **time-relaxable**, iff there is a directed path from i to j in the restrictions of G_{min} to the corresponding local areas and $l_{ij} > l_{ij}^{min}$. (The lengths of paths corresponding to $i \rightarrow j$ in the considered solution may be decreased.)

Note, that even for relaxable arcs it may not be possible to eliminate them. For example in a local area graph two arcs of the same alternative pair may result in the same longest path between two border nodes i and j. If no other alternative arcs are located on this longest path, the corresponding relaxable border arc $i \rightarrow j$ cannot be eliminated.

Clearly, the structure of a global conflict depends on the types of border arcs in the corresponding positive cycle C in the coordinator graph. Let

- N_C be the set of all non-relaxable arcs in C,
- T_C be the set of all time-relaxable arcs in C, and
- R_C be the set of all relaxable arcs in C.

The following types of global conflicts indicated by a positive cycle C in the coordinator graph may occur:

1. If $T_C = \emptyset$, and $R_C \neq \emptyset$ there is a possibly relaxable conflict.
2. If $R_C = \emptyset$, and $T_C \neq \emptyset$ there is a possibly time-relaxable conflict.

3. A Combination of Type 1 and 2 is the general case of a possibly solvable global conflict.

4. If $T_C = \emptyset$, and $R_C = \emptyset$ there is a non-relaxable conflict. All border arcs represent paths which are already present when only considering fixed arcs. Thus, the positive cycle is contained in any solution and no global feasible solution exists.

In the following it is discussed in detail which new constraining arcs may be introduced by the coordinator in order to force some local areas to modify their solutions in a suitable way. Knowing different types of conflicts different techniques to eliminate them may be proposed, i.e. different constraining arcs which are introduced by the coordination level to the local areas (local alternative graphs). Generally speaking the coordination level may impose new arcs between the end-node and the start-node of a local path corresponding to an arc in the coordinator graph. Note, that two border nodes, which are connected by a border arc in the coordinator graph, must have at least one local graph in common, as a corresponding directed path in one of the local solution graphs exists.

Firstly techniques in order to solve conflicts of Type 1 are proposed, i.e. when having detected a positive cycle C in the coordinator graph with $R_C \neq \emptyset$. Let $i \rightarrow j$ be in R_C with length e. The idea is to introduce a constraining arc $j \rightarrow i$ with weight x in one or even all corresponding local graphs, such that $x + e > 0$. When introducing this arc ($j \rightarrow i$ with weight $x > -e$) a corresponding local area is forced to make $i \rightarrow j$ shorter or even to disappear in the next round. Otherwise a positive cycle would occur in the local graph. This type of constraint imposed to a local graph is called $CS(i, j)$. An example of this technique is drawn in Figure 49.

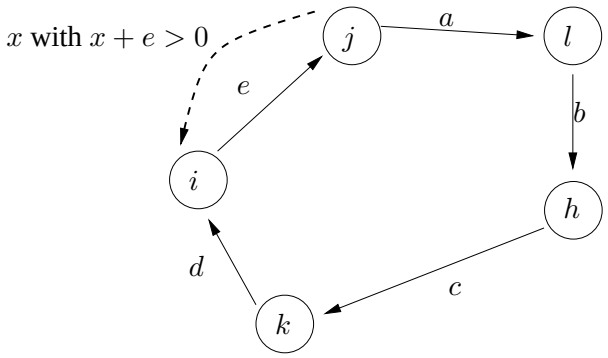

Figure 49: Introducing a constraining arc of Type $CS(i, j)$.

Another technique closely related to $CS(i, j)$ is to impose more than one constraint of this type at the same time. Coming back to the example of Figure 49 one could additionally impose $CS(h, k)$, if $h \rightarrow k$ is relaxable.

A constraint of type $CS(i,j)$ is quite general as it does not specify the length of the imposed constraining arc in detail. In order to specify this length one may choose the length of the path from j to i in the considered cycle. Let $C = (j = v_0, v_1, ..., v_n = i, j)$ be a positive cycle in the border graph and $l_{k,k+1}$ be the length of arc $v_k \rightarrow v_{k+1}$. An arc $j \rightarrow i$ with length $L = \sum_{k=0}^{n-1} l_{k,k+1}$ may be introduced in one or even all local graphs where corresponding paths are present. This type of constraint is called $CS_{min}(i,j)$. The length of a path corresponding to $i \rightarrow j$ cannot exceed $-L = -\sum_{k=0}^{n-1} l_{k,k+1}$ in the next round of computing local solutions, if such a path is still present there. If such paths have disappeared, then (with regard to the involved local graphs) the positive cycle is eliminated in any case.

Otherwise with $CS_{min}(i,j)$ cycle C is obviously eliminated in the next round if the lengths of all other border arcs in C do not increase. Additionally all other arcs may be imposed not to increase in the next round by introducing arcs $v_{k+1} \rightarrow v_k$ with length $-l_{k,k+1}$ for all $k = 1, ..., n-1$ in all areas where corresponding paths are present. This type of constraint is called $CS_{bound}(v_k, v_{k+1})$.

In Figure 50 an example of a positive cycle in the coordinator graph with two relaxable arcs $i \rightarrow j$ and $h \rightarrow k$ is depicted. The constraint $CS_{min}(i,j)$ together with the additional constraints $CS_{bound}(j,l), CS_{bound}(l,h), CS_{bound}(h,k)$ and $CS_{bound}(k,i)$ is imposed.

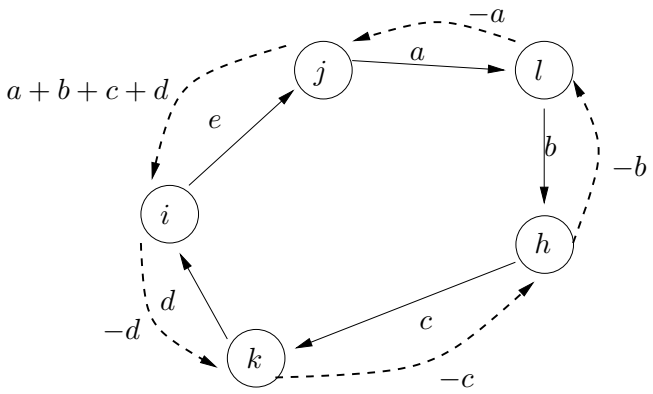

Figure 50: Introducing constraining arcs of Type $CS_{min}(i,j)$ and $CS_{bound}(v_k, v_{k+1})$.

Clearly it may happen that the local areas where $CS_{min}(i,j)$ was imposed cannot fulfill this new constraint. In this case one must try to impose other constraints in the next round. In the given example (Figure 50) one could try to impose the arc $h \rightarrow k$ to become shorter, i.e. to impose $CS_{min}(h,k)$.

In Table 4 the proposed constraints are summarized which may be imposed to local graphs in order to eliminate a global positive cycle $C = (j = v_0, v_1, ..., v_n = i, j)$ of Type 1.

($l_{k,k+1}$ are the lengths of arcs $v_k \to v_{k+1}$ and e is the length of $i \to j$).

Constraint	Idea	Constr. arc
$CS(i,j)$	force arc $i \to j$ to become shorter	$j \to i$ with weight $x > -e$
$CS_{min}(i,j)$	force arc $i \to j$ to become short enough to avoid the pos. cycle	$j \to i$ with weight $\sum_{k=0}^{n-1} l_{k,k+1}$
$CS_{bound}(v_k, v_{k+1})$	force arc $v_k \to v_{k+1}$ not to become longer	$v_{k+1} \to v_k$ with length $-l_{k,k+1}$

Table 4: Different types of constraining arcs.

If R_C contains more than one border arc one may apply another method. More than one local graph may be involved to produce shorter paths, i.e. the task of shortening or destroying paths which result in global positive cycles is divided to different local graphs. Coming back to the example from Figure 50 one could try to impose both arc $i \to j$ and arc $h \to k$ to become shorter (or to disappear) by introducing arcs in opposite direction with weights $a + b + c + d - \Delta$ and $-c + \Delta$, where Δ is positive. This technique is illustrated in Figure 51.

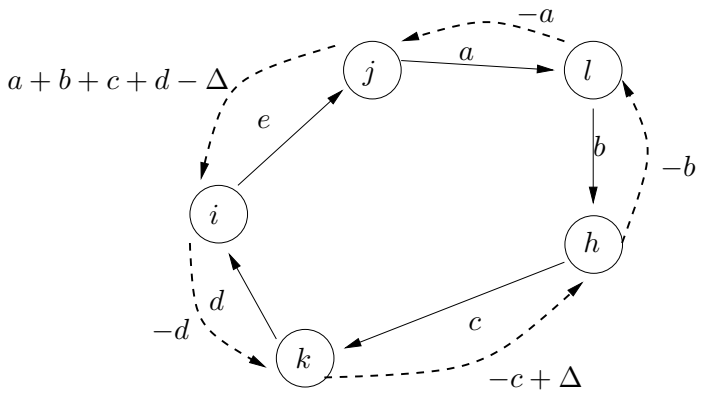

Figure 51: Dividing the task of eliminating a global positive cycle.

The described techniques may also be applied to conflicts of Type 2. Then time-relaxable arcs are imposed to become shorter in the next round with a similar strategy in order to eliminate a positive cycle. Of course, the weight of a constraining arc should not exceed the negative of the length of a corresponding path in the restrictions of G_{min} to the local graphs. This would obviously lead to a local conflict immediately.

In the general situation a global conflict is a combination of the described situations (Type 3). Thus, one may apply a combination of the described techniques. Constraints $CS(i,j)$ or $CS_{min}(i,j)$ for border arcs $i \rightarrow j \in R_C \cup T_C$ and constraints $CSbound(i,j)$ for all border arcs $i \rightarrow j$ may be imposed.

Obviously constraining arcs may lead to infeasibility for local areas (local alternative graphs). Some of these infeasibilities may be anticipated by investigating the restrictions of G_{min} to the local areas.

If for a local problem no feasible solution can be found some artificial constraints could be relaxed or modified by the coordination level.

Note, that constraining arcs are only introduced in order to force the local areas to compute different local solutions. Thus, they are not considered when checking new local feasible solutions for global feasibility, i.e. they are deleted from the local solution graphs before computing longest paths for building the corresponding new coordinator graph.

Using the criteria from Theorem 7.2 one could introduce constraining arcs even to shorten border arcs (local paths) lying on a critical path in the coordinator graph. Thus, these techniques theoretically can also be used to optimize global feasible solutions.

Generally the introduction of constraining arcs can be interpreted as the definition of allowed moves in a neighbourhood for a global (not necessarily feasible) solution. For such a global solution composed from local feasible solutions one can define

Neighbourhood N_{GL}: *For a decomposed railway problem as introduced above let be given a global (not necessarily feasible) solution S composed from local feasible solutions. Additionally let the corresponding coordinator graph be given.*
The neighbourhood $N_{GL}(S)$ is defined to be the set of all global solutions derived by the following steps:

(i) A set of constraining arcs is introduced (in order to impose certain constraints to local areas).

(ii) Local feasible solutions respecting all constraining arcs exist and are computed. Then the new global solution is composed from those.

Of course, this neighbourhood is of exponential size and thus mainly of theoretical interest.

Having described general strategies and techniques in order to get rid of global conflicts (or certain local paths), next detailed information on the methods applied in this work is given. This is done in the next section.

7.2.2 Choosing constraining arcs

In this section it is described how the general techniques from Section 7.2.1 are applied in order to solve global conflicts (positive cycles in the coordinator graph) occurring within the coordination procedure.

An idea of designing a coordination procedure is to build a kind of a greedy algorithm for the decomposed problem. As long as no global feasible solution is found or another stopping criterion is fulfilled new constraining arcs are introduced in each coordination round of the procedure. If for a local problem no feasible solution can be found the procedure is stopped.

Priority rules in order to introduce constraining arcs to local graphs are needed. For describing more details let C (length $L > 0$) be a positive cycle found in the coordinator graph after having computed local feasible solutions for all local graphs. The priority rules for the coordination step are the following:

- $CFRTS(\delta)$ constrains the first relaxable or time-relaxable border arc $i \to j$ with weight l in C slightly, i.e. it introduces the constraining arc $j \to i$ with weight $-l + \delta$ in the corresponding local graph. Here $\delta > 0$ is a parameter of the priority rule. Thus, the technique used is a version of $CS(i, j)$. In the next round the length of the local path from i to j cannot exceed $l - \delta < l$, since otherwise a positive cycle occurs in the corresponding local graph.

- $CFRTM$ sets the maximal suitable constraint for the first relaxable or time-relaxable border arc $i \to j$ with weight l in C. That means it introduces the constraining arc $j \to i$ with weight $k = \min\{L - l, -l_{ij}^{min}\}$. This technique is a version of $CS_{min}(i, j)$ which takes care of the minimal length of time-relaxable arcs. For relaxable arcs $k = L - l$ holds, as for such arcs l_{ij}^{min} is set to a large negative value. In the next round the length of the local path from i to j cannot exceed $-\min\{L - l, -l_{ij}^{min}\} = \max\{l - L, l_{ij}^{min}\} < l$, since otherwise a positive cycle occurs in the corresponding local graph.

- $CLRTM$ sets the maximal suitable constraint to the longest relaxable or time-relaxable border arc $i \to j$ with weight l in C. That means it introduces the constraining arc $j \to i$ with weight $k = \min\{L - l, -l_{ij}^{min}\}$. In the next round the length of the local path from i to j cannot exceed $-\min\{L - l, -l_{ij}^{min}\} = \max\{l - L, l_{ij}^{min}\} < l$, since otherwise a positive cycle occurs in the corresponding local graph.

- $CSRTM$ sets the maximal suitable constraint to the shortest relaxable or time-relaxable border arc $i \to j$ with weight l in C. That means it introduces the

constraining arc $j \rightarrow i$ with weight $k = \min\{L - l, -l_{ij}^{min}\}$. In the next round the length of the local path from i to j cannot exceed $-\min\{L - l, -l_{ij}^{min}\} = \max\{l - L, l_{ij}^{min}\} < l$, since otherwise a positive cycle occurs in the corresponding local graph.

- $CLDRTM$ sets the maximal suitable constraint for a border arc, too. If C contains no relaxable arcs, it sets the maximal suitable constraint to that time-relaxable border arc $i \rightarrow j$ with weight l in C, where $l - l_{ij}^{min}$ is maximal. Otherwise the maximal suitable constraint is set to the first relaxable arc in C. In the next round the length of the local path from i to j cannot exceed $-\min\{L - l, -l_{ij}^{min}\} = \max\{l - L, l_{ij}^{min}\} < l$, since otherwise a positive cycle occurs in the corresponding local graph.

- $CLSRTM$ sets the maximal suitable constraint for a border arc, too. If C contains time-relaxable arcs, it sets the maximal suitable constraint to that time-relaxable border arc $i \rightarrow j$ with weight l in C, where $l - l_{ij}^{min}$ is minimal. Otherwise the maximal suitable constraint is set to the last relaxable arc in C. In the next round the length of the local path from i to j cannot exceed $-\min\{L - l, -l_{ij}^{min}\} = \max\{l - L, l_{ij}^{min}\} < l$, since otherwise a positive cycle occurs in the corresponding local graph.

A direct generalization of the first two rules are the following.

- $CnRTS(\delta)$ applies the first rule to the n-th arc of C.
- $CnRTM$ applies the second rule to the n-th arc of C.

If a constraining arc, which is chosen by a priority rule, already exists, the associated weight is updated to the new value. The proposed priority rules can be applied in both simple heuristics and more sophisticated procedures.

Of course, other types of constraining arcs are possible. Then many other techniques of introducing constraining arcs may be applied, too. Even procedures, which allow already introduced constraining arcs to be modified or deleted could be developed. For all types of problems and conflicts arising computational tests have to show which are suitable strategies in order to derive global feasible solutions.

7.3 Reachability of global feasible solutions

In this section it will be proved that for any problem instance of a decomposed railway problem (see Section 7.1) having a global feasible solution one can force the local areas

to compute such a global feasible solution using the techniques described above, namely by applying the coordination procedure and introducing suitable constraining arcs of type $CS(i,j)$ within the procedure. Based on a corresponding theorem, in the next section an enumeration approach is proposed which terminates with a global feasible solution if one exists.

Theorem 7.3 *Let be given an instance of a decomposed railway problem which has a global feasible solution. Let S be a global solution to the problem which may be unfeasible but is derived from local feasible solutions for the local areas. Then a set of constraining arcs of Type $CS(i,j)$ exists which leads from S to a global feasible solution.*

Proof: *Let S^g be a global feasible solution of a given problem instance. (W.l.o.g. S^g may be assumed to be given in terms of an ESS.) A set of constraining arcs which leads to S^g or another global feasible solution is constructed iteratively. Let s_i be the starting time associated to border node i in S^g.*

The global solution S may be unfeasible but is derived from local feasible solutions for the local areas. By Theorem 7.1 this global solution is feasible iff the coordinator graph contains no positive cycle. If S is feasible no further action is required. Otherwise the coordinator graph must contain a positive cycle $C = (u_1, \ldots, u_n = u_1)$ with arc lengths $l_{u_i,u_{i+1}}$ $\forall\, i = 1, \ldots, n-1$.

The following operation is applied to the global unfeasible solution S.

1. *A suitable constraining arc is chosen and inserted in the corresponding local graphs.*
2. *New local feasible solutions are computed. (The constraining arcs are chosen in a way that such local feasible solutions always exist and thus can be found at least by complete enumeration.)*
3. *At the end the corresponding new global solution $S^{'}$ is checked for feasibility.*

If $S^{'}$ is unfeasible, the operation is repeated until feasibility is reached. All constraining arcs chosen within the procedure stay in the local solutions graph. As explained above they are not considered when checking for global feasibility.

Next it is described how constraining arcs have to be chosen. Then it is proved that the procedure above terminates.

A constraining arc in Step 1 is chosen with respect to the given global feasible solution S^g. For the starting times in the global feasible solution S^g one has:

$$\sum_{i=1}^{n-1}(s_{u_{i+1}} - s_{u_i}) = 0 < \sum_{i=1}^{n-1} l_{u_i,u_{i+1}}$$

and thus

$$s_{u_{i+1}} - s_{u_i} < l_{u_i,u_{i+1}}$$

for at least one i. Thus, $s_{u_{i+1}} = s_{u_i} + l_{u_i,u_{i+1}} - \Delta < s_{u_i} + l_{u_i,u_{i+1}}$ with $\Delta > 0$ for at least one i.

If no path corresponding to $u_i \rightarrow u_{i+1}$ is present in the restrictions of G_{min} to the local areas then $u_i \rightarrow u_{i+1}$ is a relaxable border arc. Otherwise (if a path corresponding to $u_i \rightarrow u_{i+1}$ exists in the restrictions of G_{min} to the local areas) $s_{u_{i+1}} - s_{u_i} \geq l^{min}_{u_i,u_{i+1}}$ must hold, as $l^{min}_{u_i,u_{i+1}}$ represents a necessary condition even for the starting times in S^g. Together with $s_{u_{i+1}} - s_{u_i} < l_{u_i,u_{i+1}}$ one has $l_{u_i,u_{i+1}} > l^{min}_{u_i,u_{i+1}}$ and thus, the arc must be time-relaxable.

In this situation a constraining arc of Type $CS(i,j)$ is introduced, namely the arc $u_{i+1} \rightarrow u_i$ with weight $L_{u_{i+1},u_i} = s_{u_i} - s_{u_{i+1}} > -l_{u_i,u_{i+1}}$ in the corresponding local graphs. (Thus, in the next round the length l of a border arc $u_i \rightarrow u_{i+1}$ derived from these local graphs cannot exceed $-L_{u_{i+1},u_i} < l_{u_i,u_{i+1}}$. Otherwise $l + L_{u_{i+1},u_i} > -L_{u_{i+1},u_i} + L_{u_{i+1},u_i} = 0$ would be the (positive) length of a local cycle.) The chosen constraining arc cannot be already present in the corresponding local graphs, as otherwise $l_{u_i,u_{i+1}} \leq -L_{u_{i+1},u_i}$ would already hold in S or no such corresponding path would be present there.

If the constraining arcs described above are introduced then for all local graphs always local feasible solutions exist (local complete consistent selections) satisfying the imposed constraints, since the global feasible solution S^g already satisfies them ($s_{u_i} \geq s_{u_{i+1}} + L_{u_{i+1},u_i} = s_{u_{i+1}} + s_{u_i} - s_{u_{i+1}} = s_{u_i}$ is obviously true). Such local complete consistent selections can be found in any case by using complete enumeration for the local problems (see Section 6.3).

Finally it has to be shown that the procedure terminates with a global feasible solution. Let n_B be the number of border nodes. One can introduce at most $n_B * (n_B - 1)$ different constraining arcs of the type described above. More precisely one can introduce at most two constraining arcs of this type for every pair of border nodes in each local graph where both nodes are present (,since local paths and therefore border arcs between pairs of border nodes can only appear in local graphs which contain both nodes).

Possibly a global feasible solution is reached before having introduced all suitable constraining arcs. Otherwise suppose all these constraining arcs have been introduced. Then

the local feasible solutions induced by the constraining arcs are feasible on a global level since for any cycle $C = (u_1, \ldots, u_n = u_1)$ *in the coordinator graph with arc lengths* $l_{u_i,u_{i+1}}\ \forall\ i = 1, \ldots, n-1$ *one has the following. As all constraining arcs have been introduced the relation*

$$l_{u_i,u_{i+1}} \leq -L_{u_{i+1},u_i} = s_{u_{i+1}} - s_{u_i}$$

holds for all i*. Then*

$$\sum_{i=1}^{n-1} l_{u_i,u_{i+1}} \leq \sum_{i=1}^{n-1} (s_{u_{i+1}} - s_{u_i}) = 0$$

holds, indicating that the length of the cycle must be non-positive. Thus, the coordinator graph does not contain a positive cycle and the induced global solution is globally feasible. □

For the neighbourhood N_{GL} (see Section 7.2.1) Theorem 7.3 especially leads to

Property 7.2 *Consider neighbourhood* N_{GL}*, where only constraining arcs of the type from the proof above are introduced iteratively in order to eliminate positive cycles. Let* ALG *be an arbitrary algorithm which leads to a feasible local solution for a local problem if it exists in a finite number of computational steps. Then the neighbourhood* N_{GL}^{ALG}*, which applies the strategy from above for introducing constraining arcs and applies algorithm* ALG *in step (ii), is feasibility-connected.*

Now let $N_{GL,O}^{ALG}$ be an extension of N_{GL}^{ALG}, where for a global feasible solution constraining arcs can be introduced in order to shorten global critical paths. This is done with regard to a given optimal global solution analogously to the way presented in the proof of Theorem 7.3. Then a similar argumentation leads to

Property 7.3 *Neighbourhood* $N_{GL,O}^{Alg}$ *is opt-connected.*

As in the case of the neighbourhoods in Section 6 even Properties 7.2 and 7.3 are mainly of theoretical interest. The next section exploits Theorem 7.3 with regard to enumerative procedures.

7.4 Enumeration techniques for decomposed problems

Based on the considerations for Theorem 7.3 an enumerative algorithm on the coordinator level can be constructed. As already stated above it may be assumed that all relevant data - and thus all arc weights - are integer. In case of rational numbers one can easily satisfy this assumption by multiplying all data by a suitable number $M \in \mathbb{N}$.

The idea of the procedure is to enumerate all suitable sets of constraining arcs. The number of these sets has shown to be finite. Then for problems having a global feasible solution Theorem 7.3 ensures that such a solution is found.

Let $C = \max\{|c_{ij}|\}$ be the maximal absolute arc weight for a given problem instance and m the number of all arcs in the graph. Then the length of a longest directed path between two arbitrary nodes in the graph corresponding to a global feasible solution (a global complete consistent selection) is bounded by $-mC$ and mC. W.l.o.g. one may assume that a global solution is given by an ESS (Earliest start schedule), since otherwise such an ESS can be derived from the solution by a left-shift of starting times in the schedule. Thus, the maximal difference between the starting times of two different operations in a global feasible solution is also bounded by $-mC$ and mC since starting times in an ESS are associated with longest paths in the corresponding solution graph.

Then for constraining arcs weights are only useful in the interval from $-mC$ to mC. Since one can only decide between introducing a constraining arc with one of the useful weights in $\{-mC, -mC+1, \ldots, 0, \ldots, mC\}$ or not introducing it, there are $2(mC+1)$ possibilities for each possible constraining arc (as all arc weights are integer). Possible constraining arcs are arcs between two border nodes which have at least one local graph in common. Thus, there are at most $n_B(n_B-1)$ possible constraining arcs, where n_B is the number of border nodes. The number of local alternative graphs where a constraining arc can be introduced is obviously bounded, too.

Using Theorem 7.3 (Reachability theorem) leads to the following: If a global feasible solution exists also a set of constraining arcs exists, which leads to a global feasible solution. Of course, this depends on the use of suitable methods to solve the local problems. One can apply complete enumeration to each of the local problems. Then it is possible either to find a local complete consistent selection for a local alternative graph or to prove that no such local complete consistent selection exists (see Section 6.3).

By enumerating all sets of suitable constraining arcs and computing corresponding (local complete) selections there are two possible results:

1. for some set a global complete consistent selection is found (local complete consistent selections for all local problems are found and no positive cycle is contained in the coordinator graph), or

2. for each such set one either does not have local complete consistent selections for some of the local problems or the local feasible solutions are not feasible on a global level. Then no global feasible solution exists.

Obviously this algorithm is only of theoretical interest as its complexity might be quite bad. Nevertheless it demonstrates the ability of the proposed decomposition approach to compute a global feasible solution if one exists.

7.5 The influence of the decomposition on computation times

The decomposition of railway scheduling problems as proposed above provides an advantage in computation times. For example the effort for computing a globally feasible solution by the proposed coordination procedure is smaller than for the greedy algorithms treating the problem as a whole.

For a more detailed discussion let n be the number of nodes and m be the number of alternative arcs of the alternative graph for the whole problem. Now let this graph be divided into k local graphs of identical size, i.e. each of the local graphs contains about $\frac{n}{k}$ nodes and $\frac{m}{k}$ alternative pairs. (For the sake of clarity it is assumed that m and n are multiples of k.) Of course, the precise numbers are slightly greater as some nodes and alternative arcs are located in more than one local graph. Nevertheless this is a good approximation for large numbers n and m.

The main effort in both, the coordination procedure and the greedy algorithms, is spend on the computation of longest paths. If the Floyd-Warshall-Algorithm is used each of these computations needs an effort of $O(z^3)$, where z is the number of nodes of the corresponding graph. A longest-path-calculation is done mainly for each alternative arc once. Thus, the computational effort for the greedy algorithms, which treat the problem as a whole, is $O(m \cdot n^3)$.

In the coordination procedure in each coordination round the local problems have to be solved by the greedy algorithm. This requires an effort of $O(\frac{m}{k} \cdot \frac{n^3}{k^3})$ for each local graph. Additionally a longest-path calculation for the coordinator graph has to be carried out. Suppose that the coordinator graph is at most as large as a local graph ($\frac{n}{k}$ nodes). Then the computational effort for one round of the coordination procedure is led by the term $O((m+1) \cdot \frac{n^3}{k^3})$. This effort is significantly smaller than the effort for the algorithm treating the problem as a whole (factor $\frac{1}{k^3}$). If only a few rounds of coordination are needed the coordination procedure should be significantly faster than the greedy algorithm.

If other solution procedures with exponential computational effort are used then the advantage of the coordination approach may be even greater.

8 Implementation and results

In this section computational results for some of the solution procedures described above are presented. The main focus lies on the decomposition approaches. It will be shown that by the methods developed in Section 7.2 feasible solutions for small instances can be derived with an acceptable effort of computation time.

In the first part of this section some implementation details of the tested methods are described. Different classes of test instances for (decomposed) railway problems are presented in the second part. Finally in the third part computational results are presented, analyzed, and compared.

8.1 Implementation details

The decomposition approach analyzed in this thesis is a kind of a greedy procedure. Local solutions are computed by the greedy algorithm from Section 6.1 with the different priority rules discussed there. If a priority rule for the local problems cannot choose a pair (e.g. because of local paths which do not exist), then an arbitrary pair is chosen. Of course, a local solution has to be re-computed during the coordination procedure, only if the local problem has changed, i.e. if a new constraining arc has been introduced.

The coordination step itself uses the priority rules from Section 7.2.2 in order to introduce suitable constraining arcs. The algorithm either terminates after a maximal number of coordination steps with or without having found a global feasible solution. Or it terminates either if one of the local problems cannot be solved or no suitable constraining arc can be introduced. All longest-path-calculations are performed by the Floyd-Warshall-Algorithm, which is simultaneously used to identify positive cycles if they exist.

The decomposition approach is compared with the greedy procedures from Section 6.1, which are used to solve the test instances as a whole. Even in this case the longest-path-calculations are performed by the Floyd-Warshall-Algorithm.

The procedures were implemented in C. The tests were run on a PC (Intel Celeron with 1.8 GHz and 640 MB memory) with operating system Fedora 5.

8.2 Test data

The algorithms proposed above are tested on a variety of instances for the railway scheduling problem. In this thesis artificial instances are used due to the lack of small real-world instances for (decomposed) railway problems.

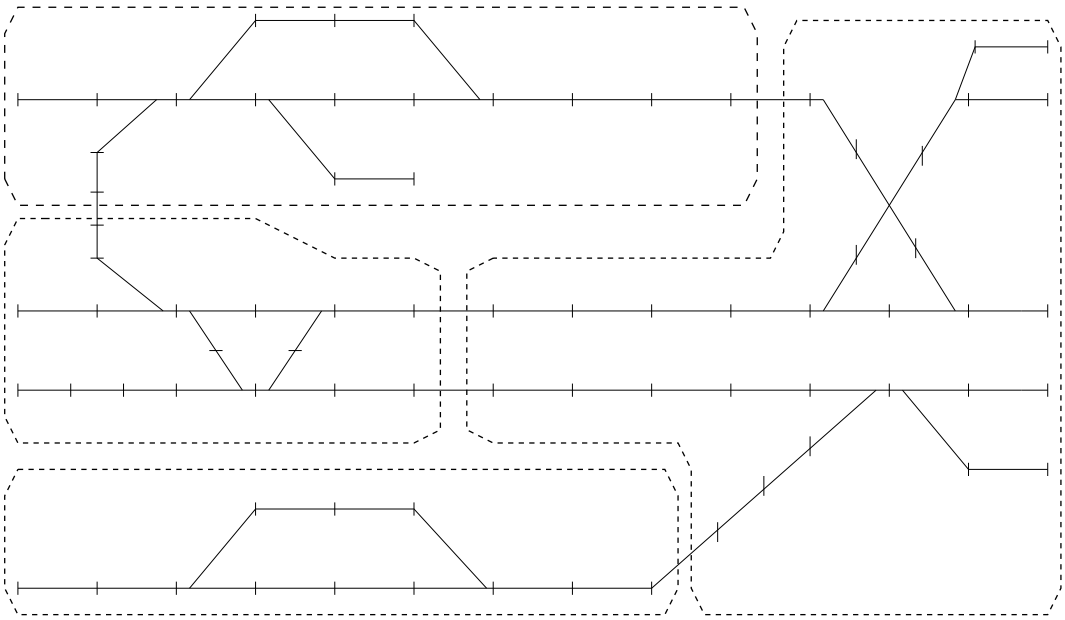

Figure 52: Example for railway network with four local areas.

The instances tested here are based on three different railway networks containing up to 73 fixed block sections. One of them is depicted in Figure 52. The design of the test networks was led by the idea of including a variety of different situations which could complicate scheduling trains. A large number of instances was created by introducing up to 16 trains moving with different travelling times on different routes through the networks. The travelling times of the trains vary in the range from 1 to 7. The crossing time ϵ is set to a small positive value. Each instance is created for the problem with and without decomposition. The decomposition of the physical network is already depicted in Figure 52 by dashed lines. Two of the networks are decomposed in three and the third in four local networks.

After preliminary tests the most interesting instances were chosen. Different classes of instances were designed. For the first class TS of instances no additional constraints are allowed. This class contains 34 instances. The second class TS_SC allows also starting constraints, i.e. for a subset of trains a starting constraint is given. The class TS_SC contains 29 instances.

For all problems the makespan objective is considered. As the main focus lies on generating feasible solutions this choice is quite unimportant. Feasible solutions for the makespan problem stay feasible even if L_{max} or other regular objective functions are considered.

8.3 Computational results

In this section computational results for the greedy procedures from Section 6.1 and the decomposition procedures from Section 7.2 are presented. These approaches are tested on the instances described above.

8.3.1 Greedy procedures

In this section computational results for the greedy procedures from Section 6.1 are described.

The first observation is that for each of the test instances the greedy procedures provide feasible solutions. In fact, creating test instances which cannot be solved seems to be very hard. For this work a large variety of different instances was created. Different ideas to derive 'hard' instances did not work. Of course ideas from the complexity proofs would lead to hard problems. But those instances would be too large to solve them in reasonable time.

Having a look at the computational results from Mascis & Pacciarelli [45] encourages the claim that creating 'hard' instances is very difficult. Even for job-shop problems with blocking and no-swap allowed feasible solutions for most of the instances are derived by the greedy procedures. It is clear that job-shop problems may include a variety of complex situations which are unnatural when transferred to railway problems. In job-shop problems a job can move from any machine to any other whereas in railway problems such moves are limited by the structure of the physical network. Thus, even the 'harder' job-shop problems are solved by greedy procedures quite satisfactory.

Additionally the constraint propagation techniques included in the greedy procedures seem to be very powerful for railway problems. If for example a train moving on a single line is chosen to be the first one in a certain block section this choice is propagated for the whole line.

Detailed results for the greedy procedures are presented in the Tables 5 and 6. Table 5 contains information on results for instances without starting constraints (TS). The second column contains for each priority rule the average relative deviation from the average values (Dev. from Avg.), i.e. the mean value of all values $100\frac{C_{max}-AV}{AV}$, where AV is the average objective value over all priority rules for the corresponding problem instance. The third column contains for each priority rule the average relative deviation from the best values (Dev. from Best), i.e. the mean value of all values $100\frac{C_{max}-B}{B}$, where B is the

best objective value over all priority rules for the corresponding problem instance. Additionally the average computation time needed for the different rules is given in seconds in the fourth column.

Priority rule	Dev. from Avg.	Dev. from Best.	Avg. Time
AMCC	0.73	17,50	116,7
SMCP	13,06	31,42	121,4
SMBP	2,77	19,55	125,1
SMSP	-5,83	8,09	120,6
FCFS	-10,73	2,84	111,7

Table 5: Results for instances from TS.

Table 6 contains the corresponding information on results for instances with starting constraints (TS_SC).

Priority rule	Dev. from Avg.	Dev. from Best.	Avg. Time
AMCC	-0,06	18,32	124,6
SMCP	14,29	34,88	131,3
SMBP	3,98	22,47	133,6
SMSP	-3,94	12,44	132,3
FCFS	-14,27	0,20	116,9

Table 6: Results for instances from TS_SC.

It can be seen that the computation times for the different priority rules are quite similar. This is clear because all alternative pairs have to be chosen and corresponding updates of longest-path-lengths' have to be made. The effort for these computations is the same for all rules. Only the effort for choosing a pair is different.

For both classes of instances, problems with or without starting constraints, the FCFS-rule provides the best results, also in terms of computational times. The rules which lead to the worst results are SMCP and SMBP for both classes of instances. The corresponding results are worse than the mean value over all tests.

Summarizing these observations and considerations it can be stated, that small instances for railway problems can be solved by greedy procedures satisfactory. The question is now, if this statement does also hold in case of decomposed problems. This question will be answered in the next section.

8.3.2 Coordination procedures for decomposed problems

In this section results for the (greedy) coordination procedures are reported. It will be shown that using the techniques from Section 7.2 railway problems can be solved effectively even if they have been decomposed.

Preliminary tests showed, that performing at most 20 coordination steps during the coordination procedure already provides good results. The coordination procedure was applied to the test data using 50 different combinations of priority rules. For the greedy procedure for solving the local problems five different priority rules AMCC, SMCP, SMBP, SMSP, FCFS were used. For the coordination step ten different priority rules, namely CFRT(1), CFRT(10), CFRTM, C2RT(1), C2RT(10), C2RTM, CLRTM, CSRTM, CLDRTM, and CLSRTM were used. For a detailed description of these rules the reader is referred to Section 7.2.2. In each step of the coordination procedure one constraining arc is introduced in one local graph.

The coordination procedure was tested on the 34 instances in TS (without starting constraints) and the 29 instances in TS_SC (with starting constraints). Table 7 contains for each combination of priority rules the average relative deviation from average and best objective values (see also the explanation for Table 5), which were found by the coordination procedures. These average values were calculated including all results where a feasible solution could be found.

	AMCC	SMCP	SMBP	SMSP	FCFS
CFRT(1)	14.4 (53.7)	3.2 (35.4)	-2.7 (30.8)	6.1 (43.6)	12.4 (50.2)
CFRT(10)	14.5 (53.5)	3.7 (36.2)	-2.9 (28.5)	10.2 (46.3)	11.4 (48.6)
CFRTM	13.8 (51.6)	3.8 (37.4)	-2.8 (28.6)	9.1 (44.8)	11.3 (48.4)
C2RT(1)	-12.3 (14.8)	-2.9 (26.5)	3.6 (36.7)	-10.4 (20.0)	-8.5 (23.8)
C2RT(10)	-9.9 (20.7)	-3.8 (25.4)	3.5 (35.6)	-9.4 (19.7)	-7.7 (23.6)
C2RTM	-10.1 (19.8)	-4.3 (24.8)	3.2 (35.3)	-10.3 (18.6)	-7.5 (22.7)
CLRTM	-0.9 (32.9)	-2.3 (27.9)	-1.6 (32.2)	-2.5 (32.6)	-2.7 (32.7)
CSRTM	1.5 (37.1)	-0.9 (28.3)	-4.2 (28.0)	-10.9 (18.8)	-1.4 (32.1)
CLDRTM	13.8 (51.6)	3.8 (37.4)	-2.8 (28.6)	9.1 (44.8)	11.3 (48.4)
CLSRTM	-7.7 (22.7)	**-12.8 (13.7)**	-3.9 (26.0)	-8.2 (21.3)	-4.97 (25.9)

Table 7: Relative deviation from average (best) values for instances in TS.

The best values (in bold print) are derived by the combination SMCP-CLSRTM. Comparing the results (rows) for CFRT(1), CFRT(10), and CFRTM it can be seen that setting constraining arcs with the maximal possible weights (CRFTM) leads to better results than

the other rules. The same observation can be made, when comparing C2RT(1),C2RT(10), and C2RTM. Thus, setting quite restrictive constraints, as also done by the priority rules CLRTM, CSRTM, and CLSRTM, leads to better results than setting weak constraints. Only rule CLDRTM does not provide such good results.

In Table 8 the numbers of best (feasible) solutions derived by different combinations of priority rules are depicted for the instances in TS. A column contains these numbers for a single priority rule used in the greedy procedure for the local problems. A row contains the data for a single priority rule used in the coordination procedure.

CoordPrio/Prio	AMCC	SMCP	SMBP	SMSP	FCFS
CFRT(1)	0 (28)	1 (32)	1 (30)	1 (29)	0 (33)
CFRT(10)	0 (29)	1 (32)	1 (33)	0 (**34**)	0 (**34**)
CFRTM	0 (**34**)	1 (**34**)	1 (33)	1 (**34**)	1 (**34**)
C2RT(1)	8 (25)	4 (**34**)	1 (32)	7 (30)	3 (25)
C2RT(10)	**9** (32)	4 (**34**)	1 (**34**)	7 (**34**)	6 (32)
C2RTM	**12** (33)	7 (**34**)	1 (**34**)	7 (**34**)	**9 (34)**
CLRTM	2 (25)	3 (25)	0 (29)	1 (29)	2 (24)
CSRTM	0 (26)	6 (23)	6 (27)	1 (21)	1 (30)
CLDRTM	0 (**34**)	1 (**34**)	1 (33)	1 (**34**)	1 (**34**)
CLSRTM	3 (33)	7 (**34**)	1 (**34**)	1 (**34**)	0 (**34**)

Table 8: Number of best (feasible) solution found for instances in TS.

The results from Table 7 are confirmed by Table 8. Setting more restrictive constraining arcs provides more best solutions. The number of feasible solutions increases with more restrictive constraining arcs, too. The best results are given in bold print. Many combinations of priority rules lead to feasible solutions for all 34 instances. The most best solutions are found by the combinations AMCC-C2RT(10) and FCFS-C2RTM.

The average computation times (in seconds) for the 50 different combinations of priority rules are given in Table 9. Additionally the average number of necessary coordination steps is given. This mean value is build including all instances, where a feasible solution could be found with the associated combination of priority rules.

The average computation times for instances in TS lie between 18 and 94 seconds. For some instances and combinations of priority rules the coordination procedure needs more than 1000 seconds of computation time. This shows that some of the instances can be solved very quickly, as the average values are much smaller. The average numbers of necessary coordination steps lie between 2 and 11 (see Table 9). Together with the num-

	AMCC	SMCP	SMBP	SMSP	FCFS
CFRT(1)	86.69 (8.82)	93.58 (8.28)	61.21 (4.17)	63.99 (8.24)	82.71 (8.15)
CFRT(10)	83.86 (8.10)	88.48 (5.94)	49.58 (4.64)	37.21 (6.12)	54.27 (6.29)
CFRTM	47.76 (6.29)	57.83 (4.91)	33.81 (3.82)	24.14 (4.09)	44.33 (4.62)
C2RT(1)	80.58 (9.92)	43.79 (7.15)	68.37 (5.47)	89.17 (8.20)	69.74 (10.44)
C2RT(10)	66.26 (10.28)	36.97 (6.59)	57.85 (5.00)	60.10 (6.82)	59.58 (9.81)
C2RTM	62.91 (9.82)	33.11 (6.15)	47.40 (4.59)	46.19 (5.47)	56.36 (8.88)
CLRTM	39.99 (6.24)	26.12 (4.32)	34.34 (2.76)	18.71 (2.86)	47.16 (5.54)
CSRTM	44.97 (6.31)	25.73 (4.09)	34.18 (2.85)	25.77 (3.62)	41.42 (4.57)
CLDRTM	51.94 (6.29)	57.74 (4.91)	32.79 (3.82)	22.70 (4.09)	45.72 (4.62)
CLSRTM	61.08 (9.79)	33.93 (6.65)	46.09 (4.24)	41.21 (4.91)	56.11 (9.06)

Table 9: Computation times (coordination steps) for instances in TS.

ber of feasible solutions derived by the different procedures (Table 8) this shows, that a maximal number of 20 coordination steps is sufficient to get good results.

For problem instances including starting constraints the coordination procedure behaves slightly different. The corresponding results for instances in TS_SC (instances with starting constraints) are contained in the Tables 10, 11, and 12. Again the first table (10) contains deviations from average and best results. The second table (11) contains numbers of best (feasible) solutions found by the procedure and the third table (12) contains average computation times and average numbers of necessary coordination steps.

	AMCC	SMCP	SMBP	SMSP	FCFS
CFRT(1)	15.2 (45.9)	2.0 (25.9)	-3.2 (23.9)	7.0 (38.2)	8.5 (35.7)
CFRT(10)	10.8 (38.3)	2.0 (25.9)	-3.5 (22.8)	11.5 (41.2)	8.5 (35.7)
CFRTM	13.3 (41.5)	2.0 (25.9)	-3.6 (22.6)	9.9 (39.9)	14.6 (46.0)
C2RT(1)	-12.9 (7.1)	5.8 (33.6)	6.8 (31.0)	-11.1 (7.9)	-7.7 (12.6)
C2RT(10)	-11.3 (9.2)	4.3 (31.8)	7.0 (31.1)	-9.4 (9.9)	-7.3 (13.2)
C2RTM	-10.3 (9.9)	5.0 (32.7)	6.7 (30.7)	-10.0 (9.2)	-8.3 (12.3)
CLRTM	-8.6 (12.1)	4.8 (29.4)	-5.7 (18.9)	-0.9 (27.1)	**-17.4 (1.8)**
CSRTM	6.7 (37.0)	4.3 (32.4)	-2.5 (23.9)	-4.4 (16.3)	7.2 (40.9)
CLDRTM	13.3 (41.5)	2.0 (25.9)	-3.6 (22.6)	9.9 (39.9)	14.6 (46.0)
CLSRTM	-7.2 (14.4)	-1.2 (25.0)	3.7 (27.2)	-9.2 (10.9)	-3.9 (18.5)

Table 10: Relative deviation from average (best) values for instances in TS_SC.

For problems with starting constraints the combination FCFS-CLRTM provides the best results in average (see Table 10). In contrast to the results for problems without starting constraints here more restrictive constraining arcs do not always provide better solutions. For example the combination AMCC-C2RT(1) in average leads to better results than AMCC-C2RT(10) and AMCC-C2RTM. A reason for this behavior may be, that more restrictive constraining arcs together with starting constraints do not leave enough space for solving local problems effectively.

CoordPrio/Prio	AMCC	SMCP	SMBP	SMSP	FCFS
CFRT(1)	0 (7)	0 (6)	5 (26)	1 (23)	0 (4)
CFRT(10)	0 (8)	0 (6)	5 (28)	1 (27)	0 (4)
CFRTM	0 (8)	0 (6)	6 (28)	2 (**29**)	0 (7)
C2RT(1)	8 (16)	1 (**29**)	0 (17)	6 (14)	6 (16)
C2RT(10)	**9** (19)	2 (**29**)	0 (18)	6 (16)	7 (18)
C2RTM	**9** (22)	3 (**29**)	0 (18)	6 (16)	**9** (23)
CLRTM	1 (5)	0 (6)	4 (25)	2 (26)	0 (3)
CSRTM	1 (6)	1 (9)	6 (22)	1 (11)	0 (4)
CLDRTM	0 (8)	0 (6)	6 (28)	2 (**29**)	0 (7)
CLSRTM	2 (24)	4 (**29**)	0 (18)	3 (18)	2 (25)

Table 11: Number of best (feasible) solution found for instances in TS_SC.

For the 29 instances in TS_SC (problems with starting constraints) the numbers of feasible solutions and best solutions derived by the different combinations of priority rules (Table 11) are similar to the numbers for the instances in TS. Again, setting more restrictive constraining arcs leads to more best solutions found by the algorithm. The number of feasible solutions increases with more restrictive constraining arcs, too. A closer look shows, that the quota of feasible solutions found by the procedures is smaller for instances in TS_SC than for instances in TS. Thus, also in practice the instances with starting constraints seem to be harder to solve than instances without starting constraints.

The average computation times for instances in TS_SC lie between 10 and 79 seconds. As for the problems without starting constraints for some instances and combinations of priority rules the coordination procedures needs more than 1000 seconds of computation time. The computation times are slightly smaller than for the instances in TS. The reason is, that the procedures fail in finding feasible solutions in more cases than for instances in TS. These infeasibilities occur quite early during the procedures, such that the average computation times are smaller. The average numbers of necessary coordination steps to find feasible solutions lie between 2 and 11 (see Table 12). Thus, there is nearly no

	AMCC	SMCP	SMBP	SMSP	FCFS
CFRT(1)	12.31 (8.57)	10.93 (4.83)	56.34 (4.77)	57.67 (8.22)	10.50 (8.25)
CFRT(10)	13.19 (9.00)	10.47 (3.17)	45.38 (4.93)	42.18 (6.30)	10.28 (7.25)
CFRTM	11.03 (5.75)	10.19 (2.50)	37.80 (4.32)	24.04 (4.41)	10.69 (6.14)
C2RT(1)	78.63 (7.50)	49.62 (6.48)	76.68 (6.53)	74.91 (6.71)	69.06 (10.88)
C2RT(10)	72.74 (8.16)	41.39 (5.86)	57.94 (5.61)	50.79 (6.00)	58.59 (9.06)
C2RTM	71.79 (9.32)	37.28 (5.59)	49.45 (5.11)	50.78 (5.12)	62.14 (8.74)
CLRTM	12.06 (5.60)	10.28 (3.00)	40.52 (3.68)	31.32 (3.54)	11.80 (5.33)
CSRTM	13.52 (5.00)	21.34 (3.89)	29.66 (2.77)	30.22 (2.55)	11.95 (4.50)
CLDRTM	11.22 (5.75)	10.32 (2.50)	37.47 (4.32)	24.05 (4.41)	10.76 (6.14)
CLSRTM	68.99 (9.46)	37.57 (5.72)	47.64 (4.39)	47.44 (5.17)	59.29 (9.44)

Table 12: Computation times (coordination steps) for instances in TS_SC.

difference between TS and TS_SC for this criteria.

In contrast to the strategies above one may introduce each constraining arc in all possible local graphs at the same time. Thus, in each coordination step a constraining arc is chosen according to one of the rules above and is then introduced in all possible local graphs. In this case the priority rules for the coordination step are named AllIn-rules. Tables describing the results for these procedures can be found in Appendix A (Tables 14 to 19).

Having a closer look on the results for the AllIn-rules the following main observations can be made. Some of the combinations with the AllIn-rules lead to better some to worse results than the rules above. Less feasible solutions than for the rules above can be computed. Especially for the instances with starting constraints (TS_SC) some rules do only lead to very few feasible solutions. The procedure using the AllIn-combination SMCP-CFRT(1) does not find any feasible solution. The average computation times are higher for the AllIn-rules (between 11 and 159 seconds).

Thus, the procedures where only one constraining arc is introduced per coordination step seem to work better and faster than the procedures using the AllIn-rules.

Summarizing the main observations about the performance of the coordination procedures the following can be stated:

- For all instances in TS and TS_SC feasible solutions can be found by the coordination procedure.
- A maximal number of 20 coordination steps leads to good results with an acceptable effort of computation time.

- Even in practice the instances with starting constraints (TS_SC) are harder to solve than the instances without (TS).
- Procedures where only one constraining arc is introduced per coordination step seem to work better and faster than the procedures using the AllIn-rules.
- Coordination priority rules which set more restrictive constraining arcs lead to more feasible solutions than others.
- In average more restrictive constraining arcs lead to better solutions for the instances in TS.

8.3.3 Comparison

In this section the results for the greedy procedures and the coordination procedures are compared. For the coordination procedures the results using the AllIn-rules are not taken into account.

Comparing the results presented in the last two sections one main experience is made. All problem instances can be solved by both the greedy procedures and the decomposition approach. The quality of the best solutions derived by the algorithms is better for the greedy procedures from Section 6.1.

Significant information for both the greedy algorithms and the coordination procedures is contained in Table 13. The second column contains the average deviation from the best results derived by one of the procedures for the instances in TS. The average computation times (in seconds) for instances in TS are contained in the third column. Columns 4 and 5 contain the corresponding information for instances with starting constraints (TS_SC).

	Dev. for TS	Time for TS	Dev. for TS_SC	Time for TS_SC
Greedy alg.	16,04%	119.11 sec	17,66%	127,73 sec
Coord. proc.	97,02%	51,51 sec	97,37%	36,64 sec

Table 13: Comparison of greedy algorithms and coordination procedures.

In average the coordination procedures are much faster than the greedy algorithms. But the quality of the solutions derived by the greedy algorithms is much better than the quality of the solutions derived by the coordination procedure. Obviously, computing local solutions independently leads to higher objective values. This is clear as the local procedures have no information about the global situation apart from the constraining arcs which are only introduced to achieve global feasibility.

Consider for example a quite good solution derived by the greedy algorithm. Then changing only one train sequence - such that a train must wait with starting its journey until another arrives - may nearly double the objective value. Most likely such “bad” choices are made by the local algorithms within the coordination procedures and thus, the objective values are much higher than for the greedy algorithms.

Nevertheless, in average the coordination procedures are much faster than the greedy algorithms. This is an effect of the decomposition, as the local problems are much smaller than the global problem (see also Section 7.5). Thus, for example longest-path procedures are much faster for the local problems. Even the coordinator graph is quite small compared to the alternative graph for the whole problem.

Summarizing the analysis above, one can state the following. Both, greedy algorithms as well as coordination procedures are able to get feasible solutions for all test instances. The greedy algorithm always finds feasible solutions whereas some versions of the coordination procedure (using certain combinations of priority rules) fail. The greedy procedures provide solutions of much higher quality whereas the coordination procedure is more than twice faster in average.

9 Concluding remarks

In this thesis an overview on different aspects of a class of railway scheduling problems was presented. Railway scheduling problems and related shop scheduling problems were modeled in terms of alternative graphs. Existing solutions procedures as well as some new ideas were presented.

Additionally the complexity of railway scheduling and related shop scheduling problems was analyzed. The railway problem with starting constraints was shown to be NP-complete by adjusting a complexity proof from Arbib et al. [6]. Railway scheduling problems and related shop scheduling problems with blocking constraints were classified in terms of the well-known $\alpha|\beta|\gamma$-scheme.

The main part of this thesis was dedicated to the decomposition of railway scheduling problems and corresponding solution procedures. Based on approaches developed within the EU-project COMBINE II a decomposition model was proposed. The decomposition model is based on a physical decomposition of a railway network. The scheduling problem then is decomposed into local problems and a coordination problem. Coordination procedures were developed in order to compute feasible solutions for the decomposed problem. The procedures were tested on a variety of instances and compared to simple greedy procedures, which treat the problem as a whole. It turned out that the procedures for the decomposed problem are faster than the greedy algorithms but provide solutions of worse quality.

The purpose of this thesis is to set a basis for further considerations in the field of decomposition of railway scheduling problems. Of course, there are still many topics where further research could focus on. As seen above, the quality of the solutions derived by the coordination procedures is quite bad. Thus, approaches to improve the quality of such solutions might be developed. On one hand this could be done by introducing new coordination techniques. On the other hand one could try to use other techniques for solving the local problems. Such techniques could be developed exploiting the ideas in Section 6. A diploma thesis will focus on this topic in the next months.

Another research topic could be the integration of the scheduling techniques from this thesis with variable routings and simulation techniques. For basic versions of the decomposition approach this has already been done within the EU-project COMBINE II. But in contrast to the fully automated algorithms in this thesis, the procedures in COMBINE II were developed for real-time planning systems, which support human dispatchers.

A further topic for future research could be the exploitation of the coordination techniques presented in this thesis for other types of scheduling problems. For example some supply-chain problems could be decomposed in a similar way as presented here. Furthermore

general decomposition approaches and corresponding coordination techniques based on physical decompositions of scheduling problems could be developed.

References

[1] *Computers in Railways IX*. Advances in Transport 15. C.A.Brebbia, J. Allan, R.J. Hill, G. Sciutto, S. Scone, WITpress, 2004.

[2] *Computers in Railways X*. WIT Transactions on The Built Environment, 88. J. Allan, C.A.Brebbia, A.F. Rumsey, G. Sciutto, S. Scone,C.J. Goodman, WITpress, 2006.

[3] E.H.L. Aarts, P.J.M. van Laarhoven, J.K. Lenstra, and N.L.J. Ulder. A computational study of local search algorithms for job shop scheduling. *ORSA Journal on Computing*, 6(2):118–125, 1994.

[4] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows*. Prentice Hall, Englewood Cliffs, 1993.

[5] S.B. Akers and J. Friedman. A non-numerical approach to production scheduling problems. *Operations Research*, 3:429–442, 1955.

[6] C. Arbib, G.F. Italiano, and A. Panconesi. Predicting Deadlock in Store-and-Forward Networks. *NETWORKS*, 20:861–881, 1990.

[7] U. Brännlund, P.O. Lindberg, A. Nou, and J.-E. Nilsson. Railway timetabling using lagrangian relaxation. Technical report, Royal Institute of Technology, Stockholm, Sweden, 1997.

[8] P. Brucker. An efficient algorithm for the job-shop problem with two jobs. *Computing*, 40(4):353–359, 1988.

[9] P. Brucker. *Scheduling Algorithms*. Springer-Verlag, 5th edition, 2007.

[10] P. Brucker, S. Heitmann, and J. Hurink. Flow-shop problems with intermediate buffers. *OR Spectrum*, 25:549–574, 2003.

[11] P. Brucker, B. Jurisch, and B. Sievers. A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, 49(1-3):107–127, 1994.

[12] P. Brucker and S. Knust. *Complex Scheduling*. Springer-Verlag, first edition, 2006.

[13] X. Cai and C.J. Goh. A fast heuristic for the train scheduling problem. *Computers in Operations Research*, 21(5):499–510, 1994.

[14] A. Caprara, M. Fischetti, and P. Toth. Modeling and solving the train timetabling problem. *Operations Research*, 50(5):851–861, 2001.

[15] M. Carey and S. Carville. Scheduling and platforming trains at busy complex stations. *Transportation Research*, 37A(3):195–224, 2003.

[16] M. Carey and D. Lockwood. A model, algorithms and strategy for train pathing. *Journal of the Operational Research Society*, 46:988–1005, 1995.

[17] J. Carlier and E. Pinson. An algorithm for solving the job-shop problem. *Management Science*, 35(2):164–176, 1989.

[18] J. Carlier and E. Pinson. Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research*, 78:146–161, 1994.

[19] J.-F. Cordeau, P. Toth, and D. Vigo. A survey of optimization models for train routing and scheduling. *Transportation Science*, 32(4):380–404, 1998.

[20] A. D'Ariano, D. Pacciarelli, and M. Pranzo. A branch and bound algorithm for scheduling trains in a railway network. *European Journal of Operational Research*, 183:643–657, 2007.

[21] M. Dell'Amico and M. Trubian. Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research*, 41(1-4):231–252, 1993.

[22] M. M. Dessouky, Q. Lu, J. Zhao, and R.C. Leachman. An exact solution procedure to determing the optimal dispatching times for complex rail networks. *IIE Transactions*, 38:141–152, 2006.

[23] M. J. Dorfman and J. Medanic. Scheduling trains on a railway network using a discrete event model of railway traffic. *Transportation Research B*, 38:81–98, 2004.

[24] M.R. Garey and D.S. Johnson. *Computers and intractability. A guide to the theory of NP-completeness.* W. H. Freeman and Company, San Francisco, 1979.

[25] M.R. Garey, D.S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976.

[26] M. Giannettoni and S. Savio. *The European project COMBINE 2 to improve knowledge on future rail Traffic Management Systems*, pages 695–706. Advances in Transport 15, Computers in Railways IX. C.A.Brebbia, J. Allan, R.J. Hill, G. Sciutto, S. Scone, WITpress, 2004.

[27] P.C. Gilmore and R.E. Gomory. Sequencing a one state-variable machine: A solvable case of the traveling salesman problem. *Operations Research*, 12:655–679, 1964.

[28] N.G. Hall and C. Sriskandarajah. A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research*, 44(3):510–525, 1996.

[29] A. Higgins, E. Kozan, and L. Ferreira. Optimal scheduling of trains on a single line track. *Transportation Research B*, 30(2):147–161, 1996.

[30] J. Hurink, B. Jurisch, and M. Thole. Tabu search for the job-shop scheduling problem with multi-purpose machines. *Operations-Research-Spektrum*, 15(4):205–215, 1994.

[31] S.M. Johnson. Optimal two-and-three-stage production schedules with set-up times included. *Naval Research Logistic Quaterly*, 1:61–68, 1954.

[32] Jovanović, D. and Harker, P.T. A decision support system for train dispatching: an optimization-based methodology. *Journal of the Transportation Research Forum*, 31:25–37, 1990.

[33] Jovanović, D. and Harker, P.T. Tactical scheduling of rail operations: the SCAN I system. *Transportation Science*, 25:46–64, 1991.

[34] H. Kamoun and C. Sriskandarajah. The complexity of scheduling jobs in repetitive manufacturing systems. *European Journal of Operational Research*, 70(3):350–364, 1993.

[35] T. Kampmeyer. *Cyclic scheduling problems*. PhD thesis, Fachbereich Mathematik/Informatik, University of Osnabrück, 2006.

[36] S. Knust. *Shop-scheduling problems with transportation*. PhD thesis, Fachbereich Mathematik/Informatik, University of Osnabrück, 1999.

[37] D.R. Kraay and P.T. Harker. Real-time scheduling of freight railroads. *Transportation Research B*, 29(3):213–229, 1995.

[38] S.A. Kravchenko. A polynomial algorithm for a two-machine no-wait job-shop scheduling problem. *European Journal of Operational Research*, 106:101–107, 1998.

[39] W. Kubiak. A pseudo-polynomial algorithm for a two-machine no-wait job-shop scheduling problem. *European Journal of Operational Research*, 43(3):267–270, 1989.

[40] J.K. Lenstra and A.H.G. Rinnooy Kan. Computational complexity of discrete optimization problems. *Annals of Discrete Mathematics*, 4:121–140, 1979.

[41] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[42] R. Logendran and C. Sriskandarajah. Two-machine group scheduling problem with blocking and anticipatory setups. *European Journal of Operational Research*, 69(3):467–481, 1993.

[43] S. Martinez, S. Dauzère-Pérès, C. Guéret, Y. Mati, and N. Sauer. Complexity of flow-shop scheduling problems with a new blocking constraint. *European Journal of Operational Research*, 169(3):855–864, 2006.

[44] A. Mascis and D. Pacciarelli. Machine scheduling via alternative Graphs. Report dia-46-2000, Dipartimento di Informatica e Automazione, Universita Roma Tre, Roma, Italy, 2000.

[45] A. Mascis and D. Pacciarelli. Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research*, 143:498–517, 2002.

[46] M. Mastrolilli and L.M. Gambardella. Effective Neighborhood Functions for the Flexible Job Shop. *Journal of Scheduling*, 3(1):3–20, 2000.

[47] Y. Mati, N. Rezg, and X. Xie. Scheduling problem of job-shop with blocking: A taboo search approach. In *4th Metaheuristics International Conference*, pages 643–648, Porto, Portugal, 2001. MIC2001.

[48] M. Mazzarello and E. Ottiviani. A traffic management system for real-time traffic optimisation in railways. *Transportation Research B*, 41(2):246–274, 2007.

[49] S.T. Mc Cormick, M.L. Pinedo, S. Shenker, and B. Wolf. Sequencing in an assembly line with blocking to minimize cycle time. *Operations Research*, 37(6):925–935, 1989.

[50] T. Nieberg. Tabusuche für Flow-Shop und Job-Shop Probleme mit begrenztem Zwischenspeicher. Diploma thesis, Fachbereich Mathematik/Informatik, University of Osnabrück, 2002.

[51] A. Nou. Railway timetabling - lagrangian relaxation. Technical report, Royal Institute of Technology, Stockholm, Sweden, 1997.

[52] E. Nowicki and C. Smutnicki. A fast tabu search algorithm for the job shop problem. *Management Science*, 42(6):797–813, 1996.

[53] E. Nowicki and C. Smutnicki. Some new ideas in TS for job-shop scheduling. *Operations Research/Computer Science Interfaces Series*, 30:165–190, 2005.

[54] D. Pacciarelli and M. Pranzo. A Tabu Search Algorithm for the Railway Scheduling Problem. In *4th Metaheuristics International Conference*, pages 159–164, Porto, Portugal, 2001. MIC2001.

[55] C.H. Papadimitriou and Kanellakis. Flowshop Scheduling with Limited Temporary Storage. *Journal of the Association for Computing Machinery*, 27(3):533–549, 1980.

[56] S. Plaggenborg. Ein Algorithmus zur komplexitaetsmaessigen Klassifikation von Schedulingproblemen. Diploma thesis, Fachbereich Mathematik/Informatik, University of Osnabrück, 1994.

[57] OR research group at the University of Osnabrück. http://www.mathematik.uni-osnabrueck.de/research/OR/class.

[58] H. Röck. Scheduling unit task shops with resource constraints and excess usage costs. Report 83-21, Fachbereich Informatik, Technical university of Berlin, Berlin, 1983.

[59] H. Röck. Some new results in flow shop scheduling. *Zeitschrift für Operations Research, Ser. A-B*, 28(1):1–16, 1984.

[60] H. Röck. The three-machine no-wait flow shop is NP-complete. *Journal of the Association for Computing Machinery*, 31(2):336–345, 1984.

[61] J. Rodriguez. A constraint programming model for real-time train scheduling at junctions. *Transportation Research B*, 41(2):231–245, 2007.

[62] B. Roy and B. Sussmann. Les problemes d'ordonnancement avec constraints disjunctives. Note DS no. 9 bis, SEMA, Paris, 1964.

[63] S. Sahni and Y. Cho. Complexity of scheduling shops with no wait in process. *Mathematics of Operations Research*, 4(4):448–457, 1979.

[64] C. Sriskandarajah and P. Ladet. Some no-wait shops scheduling problems: complexity aspect. *European Journal of Operational Research*, 24(3):424–438, 1986.

[65] C. Strotmann. Lokale Suchverfahren von Job-Shop Problemen mit Transportrobotern. Diploma thesis, Fachbereich Mathematik/Informatik, University of Osnabrück, 2001.

[66] J. Törnquist. Railway traffic disturbance management - An experimental analysis of disturbance complexity, management objectives and limitations in planning horizon. *Transportation Research A*, 41(3):249–266, 2007.

[67] P.J. Zwaneveld, L.G. Kroon, H.E. Romeijn, M. Salomon, S. Dauzere-Peres, S.P.M. van Hoesel, and Ambergen H.W. Routing trains through railway stations: model formulation and algorithms. *Transportation Science*, 30(3):181–194, 1996.

List of abbreviations

AMCC	priority rule *Avoid Maximum Current* $C_{\max}$
BNSJSP	blocking job-shop problem with no swap allowed
BWSJSP	blocking job-shop problem with swap allowed
$CFRTS(\delta)$	priority rule *Constrain first relaxable or time-relaxable slightly*
$CFRTM$	priority rule *Constrain first relaxable or time-relaxable maximally*
$CnRTS(\delta)$	priority rule *Constrain n-th relaxable or time-relaxable slightly*
$CnRTM$	priority rule *Constrain n-th relaxable or time-relaxable maximally*
$CLRTM$	priority rule (for details see Section 7.2.2)
$CSRTM$	priority rule (for details see Section 7.2.2)
$CLDRTM$	priority rule (for details see Section 7.2.2)
$CLSRTM$	priority rule (for details see Section 7.2.2)
FCFS	priority rule *First Come First Serve*
SMBP	priority rule *Select Most Balanced Pair*
SMCP	priority rule *Select Most Critical Pair*
SMSP	priority rule *Select Max Sum Pair*

Index

A Appendix: Tables

	AMCC	SMCP	SMBP	SMSP	FCFS
CFRT(1)	3.4 (21.9)	4.4 (18.9)	0.3 (26.6)	-6.7 (7.4)	1.2 (21.7)
CFRT(10)	4.5 (31.8)	5.1 (38.0)	-0.2 (26.0)	-3.5 (23.7)	1.4 (31.2)
CFRTM	15.7 (54.7)	4.7 (39.3)	-4.9 (26.5)	10.8 (47.7)	13.2 (51.5)
C2RT(1)	-14.1 (7.0)	-1.9 (26.0)	11.0 (43.0)	-5.9 (15.2)	-11.3 (6.1)
C2RT(10)	-14.0 (9.0)	-0.6 (28.9)	7.0 (39.6)	-11.2 (13.5)	-14.4 (8.8)
C2RTM	-8.8 (22.1)	-2.9 (28.0)	4.6 (38.2)	-9.0 (21.4)	-6.2 (25.4)
CLRTM	-3.1 (29.2)	-0.8 (31.3)	0.1 (33.4)	-0.9 (33.2)	-2.3 (31.2)
CSRTM	4.6 (39.9)	-4.8 (26.0)	-2.5 (28.9)	-6.9 (25.1)	1.2 (35.4)
CLDRTM	15.7 (54.7)	4.7 (39.3)	-4.9 (26.5)	10.8 (47.7)	13.2 (51.5)
CLSRTM	-6.4 (25.0)	-11.4 (16.9)	-2.5 (28.9)	-6.8 (24.1)	-3.6 (28.6)

Table 14: Relative deviation from average (best) values for instances in TS.

CoordPrio/Prio	AMCC	SMCP	SMBP	SMSP	FCFS
CFRT(1)	0 (6)	1 (4)	1 (18)	1 (5)	0 (9)
CFRT(10)	0 (14)	1 (18)	1 (18)	0 (14)	0 (15)
CFRTM	0 (34)	1 (34)	6 (34)	1 (34)	1 (34)
C2RT(1)	6 (12)	3 (24)	1 (14)	3 (8)	3 (5)
C2RT(10)	6 (15)	3 (29)	1 (28)	5 (23)	3 (13)
C2RTM	7 (33)	5 (34)	1 (34)	5 (34)	7 (34)
CLRTM	1 (30)	2 (30)	0 (31)	1 (29)	2 (26)
CSRTM	0 (33)	8 (31)	5 (29)	1 (25)	1 (34)
CLDRTM	0 (34)	1 (34)	6 (34)	1 (34)	1 (34)
CLSRTM	0 (33)	5 (34)	1 (34)	1 (34)	0 (34)

Table 15: Number of best (feasible) solution found for instances in TS.

	AMCC	SMCP	SMBP	SMSP	FCFS
CFRT(1)	26.33 (2.50)	25.04 (1.50)	22.74 (2.44)	20.95 (1.40)	23.35 (1.67)
CFRT(10)	30.80 (4.36)	27.41 (4.67)	22.80 (2.44)	21.72 (3.07)	26.57 (2.53)
CFRTM	76.69 (6.29)	76.75 (4.91)	45.50 (3.56)	47.66 (4.53)	64.47 (4.62)
C2RT(1)	30.79 (4.75)	82.18 (5.58)	40.96 (2.00)	32.44 (3.38)	26.13 (3.00)
C2RT(10)	32.05 (5.47)	95.79 (6.00)	60.98 (3.96)	34.25 (5.22)	31.06 (4.62)
C2RTM	116.40 (9.09)	91.92 (6.03)	61.98 (4.35)	49.85 (4.82)	91.83 (7.53)
CLRTM	85.43 (6.17)	77.41 (4.73)	45.14 (2.81)	27.13 (2.86)	74.22 (4.73)
CSRTM	80.10 (6.21)	68.13 (4.52)	55.70 (3.21)	52.84 (3.84)	74.77 (5.00)
CLDRTM	76.83 (6.29)	76.30 (4.91)	45.55 (3.56)	47.49 (4.53)	64.26 (4.62)
CLSRTM	115.06 (8.94)	92.60 (6.41)	61.30 (4.15)	45.14 (4.44)	91.29 (7.47)

Table 16: Computation times (coordination steps) for instances in TS.

	AMCC	SMCP	SMBP	SMSP	FCFS
CFRT(1)	31.3 (80.5)	– (–)	-3.8 (17.0)	-5.6 (11.6)	12.4 (41.0)
CFRT(10)	9.7 (35.4)	1.3 (23.6)	-2.4 (19.7)	-2.4 (20.8)	12.4 (41.0)
CFRTM	14.3 (39.6)	1.9 (22.2)	-6.2 (18.4)	10.3 (39.7)	15.6 (43.7)
C2RT(1)	-11.0 (7.6)	9.0 (34.9)	13.1 (35.5)	-11.5 (3.9)	-8.3 (7.0)
C2RT(10)	-11.4 (8.0)	6.0 (33.9)	9.8 (34.3)	-13.7 (3.6)	-9.5 (6.9)
C2RTM	-9.8 (10.8)	5.8 (33.1)	7.4 (32.7)	-9.3 (11.2)	-7.6 (13.9)
CLRTM	-8.0 (12.9)	4.7 (25.7)	-5.5 (18.1)	-0.2 (26.4)	-16.7 (3.1)
CSRTM	7.5 (31.8)	6.5 (33.4)	-0.5 (25.6)	0.8 (28.6)	7.3 (32.2)
CLDRTM	14.3 (39.6)	1.9 (22.2)	-6.2 (18.4)	10.3 (39.7)	15.6 (43.7)
CLSRTM	-6.7 (15.2)	-0.5 (25.4)	4.4 (29.1)	-8.6 (12.6)	-3.2 (19.9)

Table 17: Relative deviation from average (best) values for instances in TS_SC.

CoordPrio/Prio	AMCC	SMCP	SMBP	SMSP	FCFS
CFRT(1)	0 (1)	0 (0)	3 (15)	1 (7)	0 (2)
CFRT(10)	0 (4)	0 (3)	3 (18)	1 (13)	0 (2)
CFRTM	0 (8)	0 (6)	7 (29)	3 (29)	0 (7)
C2RT(1)	6 (10)	0 (21)	0 (8)	3 (7)	4 (6)
C2RT(10)	6 (12)	0 (25)	0 (13)	6 (12)	4 (7)
C2RTM	9 (22)	1 (29)	0 (18)	6 (16)	9 (23)
CLRTM	1 (5)	0 (6)	4 (27)	2 (26)	0 (3)
CSRTM	1 (7)	1 (10)	4 (24)	1 (21)	0 (5)
CLDRTM	0 (8)	0 (6)	7 (29)	3 (29)	0 (7)
CLSRTM	2 (24)	2 (29)	0 (18)	4 (18)	2 (25)

Table 18: Number of best (feasible) solution found for instances in TS_SC.

	AMCC	SMCP	SMBP	SMSP	FCFS
CFRT(1)	12.49 (5.00)	11.87 (–)	22.79 (2.53)	23.16 (1.57)	12.38 (3.00)
CFRT(10)	13.43 (3.25)	12.30 (2.33)	24.27 (2.78)	29.88 (2.85)	12.98 (2.50)
CFRTM	17.21 (5.75)	12.85 (2.50)	57.30 (4.17)	57.56 (5.17)	17.73 (6.14)
C2RT(1)	32.37 (4.40)	65.19 (3.86)	39.15 (2.25)	40.83 (3.43)	32.84 (3.17)
C2RT(10)	36.60 (4.83)	103.04 (5.08)	61.44 (4.00)	41.17 (4.92)	35.50 (3.86)
C2RTM	124.17 (8.68)	97.05 (5.41)	61.24 (4.72)	55.62 (4.75)	94.15 (8.04)
CLRTM	16.53 (4.60)	12.91 (2.50)	51.55 (3.67)	39.20 (3.54)	15.80 (4.67)
CSRTM	18.01 (5.43)	24.97 (4.00)	55.60 (3.29)	52.03 (3.95)	17.27 (5.20)
CLDRTM	17.23 (5.75)	12.90 (2.50)	57.50 (4.17)	65.59 (5.17)	17.71 (6.14)
CLSRTM	158.76 (8.62)	97.09 (5.55)	60.27 (4.17)	52.50 (4.61)	94.97 (8.36)

Table 19: Computation times (coordination steps) for instances in TS_SC.

Danksagung

Ich bedanke mich bei Herrn Prof. Dr. Peter Brucker. Er gab mir nach meinem bestandenen Diplom im Jahr 2001 die Möglichkeit, bei ihm als wissenschaftlicher Mitarbeiter und Doktorand zu arbeiten, und er stand mir beim Erstellen dieser Dissertation stets mit konstruktiven Vorschlägen zur Seite. Darüber hinaus ermöglichte er mir die Teilnahme an internationalen Tagungen und Workshops. Dafür und auch für die angenehme Zusammenarbeit in der Lehre gilt ihm mein herzlicher Dank.

Des Weiteren bedanke ich mich bei allen Mitwirkenden des EU-Projektes COMBINE II, das mir Inspiration für meine Dissertation gab. Insbesondere gilt mein Dank Prof. Dr. Dario Pacciarelli für die fruchtbare Zusammenarbeit im Rahmen des Projektes. Ich danke auch der Europäischen Kommission, die das COMBINE II Projekt (IST-2001-34705) finanzierte.

Bei der Implementation einiger Programmteile wirkten die studentischen Hilfskräfte Matthias Siemering und Dennis Egbers mit. Hierfür bedanke ich mich bei ihnen.

Im Rahmen seiner Diplomarbeit, die er auf der Basis dieser Dissertation erstellen wird, wirkte Andreas Haspecker an der Implementation von Programmteilen mit. Hierfür bedanke ich mich bei ihm und wünsche ihm viel Erfolg für den Abschluss seiner Diplomarbeit.

Bei Frau Jun. Prof. Dr. Sigrid Knust bedanke ich mich für die Hilfestellung bei der Verwendung des CLASS-Programmes. Ich danke auch Silvia Heitmann und Dr. Thomas Kampmeyer für viele interessante Diskussionen und Anregungen.

Für die sechs schönen Jahre mit vielen einprägsamen Erlebnissen danke ich allen Mitgliedern des Fachbereichs Mathematik/Informatik an der Universität Osnabrück und insbesondere jenen, mit denen ich im Rahmen zahlreicher Lehrveranstaltungen zusammenarbeiten durfte.

Nicht zuletzt danke ich meinen Eltern. Sie haben mir die Möglichkeit zu einer akademischen Ausbildung gegeben und mich immer darin unterstützt.

Ein ganz besonderer Dank gilt meiner Frau Julia. Sie hat mich in allen Phasen der Erstellung dieser Dissertation stets bedingungslos unterstützt. Ohne sie wäre der Weg zu dieser Dissertation in vielerlei Hinsicht schwieriger und weniger schön gewesen.

Christian Strotmann
Osnabrück, Juli 2007