Heuristics for Duty Scheduling in Public Transit

Diplomarbeit
bei Prof. Dr. Martin Grötschel

vorgelegt von Fabian Stöffler
am Fachbereich Mathematik der
Technischen Universität Berlin

April 2006

Hiermit versichere ich die selbständige und eigenhändige Anfertigung dieser Diplomarbeit an Eides statt.

Fabian Stöffler
Berlin, 05. April 2006

Acknowledgments

This thesis would not have been possible without the great infrastructure at Konrad Zuse Institute (ZIB) and without the help and strong support of numerous people.
First of all I am very thankful toward Dr. Ralf Borndörfer and Dipl. math. oec. Steffen Weider for being the task force behind this thesis. Thank you for many fruitful discussions, valuable advice and for contributing countless helpful, inspiring ideas.
Further thanks go to Dr. Andreas Löbel for all his technical support and all the advice he gave me during my work at ZIB. I also thank Prof. Dr. Konrad Polthier for providing JavaView and for adding features whenever I needed them.
I thank Prof. Dr. Martin Grötschel and Dr. Frank Lutz for bringing me into contact with all the wonderful people at ZIB. I thank them for always providing answers to my many mathematic related questions. Special thanks go to Prof. Grötschel for supervising this thesis as well as for his strong support throughout my studies.
Personally, I thank my fellow companions, my friends Mikis Rolke, Nicole Vigh and Dominik Piesker for sharing my problems, for being such effective co-learners and for correcting so many grammatical errors in this thesis.
Finally I thank my mate Falko Jeske and my lovely girlfriend Iris Folgner for being such accepting and understanding people, for keeping me going on and for being there for me whenever I needed it.

German Summary

Die effiziente Verplanung von Fahrzeugen und Fahrern zählt zu den wichtigsten Aufgaben im öffentlichen Personennahverkehr. Besonders die Zuweisung von Fahrern hat große ökonomische Bedeutung, da fast die Hälfte des Budgets eines durchschnittlichen deutschen Verkehrsunternehmens auf Personalkosten entfällt. Gesetzliche Bestimmungen für den Fahrereinsatz, wie zum Beispiel Pausen- und Lenkzeitregelungen, existieren weltweit und erschweren eine solche Zuteilung. Die Konstruktion einer Menge von täglichen Diensten, die sämtlichen Vorschriften genügen und die alle vordefinierten Fahrzeugtätigkeiten abdecken, nennt man Dienstplanung.
Aufgrund seiner Problemstruktur ist das Dienstplanungsproblem $\mathcal{NP}$-schwer. In Deutschland umfassen durchschnittliche Dienstplanungsprobleme aufgrund der komplizierten und umfangreichen Gesetzgebung mehr als zehntausend abzudeckende Fahrzeugtätigkeiten und mehrere Millionen möglicher Dienste.
Aktuell werden zur Lösung vor allem Column-Generation Techniken eingesetzt, die auf einer Set-Partitioning Definition des Dienstplanungsproblems beruhen. Ein Beispiel hierfür ist der Solver DS-Opt, der in den Softwarepaketen BERTA und MICROBUS integriert ist.
Obwohl die Qualität der Lösungen zufriedenstellend ist und meist große Kosteneinsparungen ermöglicht, ist die Anwendung solcher Lösungsverfahren sehr zeitintensiv und kann bei hinreichender Komplexität der Instanzen und Regelungen scheitern.
In dieser Arbeit beschreiben wir FAST, eine neue Heuristik die eine geeignete Relaxierung des Dienstplanungsproblems verwendet, um die Laufzeit entscheidend zu reduzieren während die Qualität der Lösung nahezu konstant bleibt.
Zuerst formulieren wir das Dienstplanungsproblem als graphentheoretisches Modell auf einen Digraphen $\mathcal{D} = (\mathcal{V}, \mathcal{A})$. Die Knoten $\mathcal{V}$ stellen Fahrzeugtätigkeiten dar und die Bögen $\mathcal{A}$ repräsentieren mögliche Verknüpfungen dieser Tätigkeiten. Jeder potentielle Dienst kann durch einen Pfad in $\mathcal{D}$ dargestellt werden. Nicht jeder Pfad entspricht jedoch einem zulässigen Dienst. Gesucht ist also eine Menge von knotendisjunkten Pfaden, die Dienste repräsentieren und die zusammen alle Knoten abdecken.
Anschließend stellen wir einige interessante Dienstplanungsheuristiken vor und passen diese an unser Modell an. Eine dieser Heuristiken,

HASTUS-Micro, betrachten wir genauer, da diese eine Kernidee für FAST liefert.
Im Hauptteil der Arbeit beschreiben wir FAST im Detail und präsentieren sehr zufriedenstellende Rechenergebnisse. Im Vergleich zu DS-Opt konnte auf einer Menge von 23 Probleminstanzen unterschiedlicher Größe eine durchschnittliche Laufzeiteinsparung von 95.41% erreicht werden. Die Zielfunktionswerte verschlechterten sich dabei durchschnittlich um weniger als 3.50 %.
Mit diesen Ergebnissen erreicht FAST das gesetzte Ziel, schnelle, qualitativ gute Lösungen für Dienstplanungsprobleme zu ermöglichen.
Abschließend zeigen wir weiteres Verbesserungspotential in FAST auf, sowohl bezüglich der Laufzeit als auch bezüglich der Lösungsqualität.

Contents

Introduction

In public transit one of the most important operative planning decisions is the effective routing of drivers and vehicles. The allocation of drivers has enormous economic significance as the average German traffic company spends nearly half of its budget on driver salaries. Legal requirements concerning the scheduling of driver shifts, e.g. break- and driving-time restrictions, complicate this task. The process of building a set of daily duties which satisfy all regulations and together cover all timetabled vehicle activities is called duty scheduling.
The Plauener Straßenbahn GmbH located in the small German city of Plaue uses a very simple process to construct legal duty schedules for their regional tram network.
First all vehicle work which has to be covered by driver shifts is rounded in respect to a four hour time pattern. For each of the resulting time frames a number of tasks, i.e. vehicle work which has to be covered, is determined. Assume that the traffic company starts its working day at four o'clock in the morning and ends operating at midnight. Before driver duties are constructed, the number of required duties of different duty groups, e.g. early duties, afternoon duties and evening duties, is resolved. The number of evening duties equals the number of tasks in the last time frame ending at midnight. These evening duties also cover some of the tasks in the second last time frame The remaining tasks indicate the number of afternoon duties. The first three time frames are occupied similarly determining the number of early and split duties. Afterwards duties are formed as indicated by the former grouping while optimization of the schedule takes place during this construction.
Simple heuristics like the one described above, which has come to be known as the Treiber-Heuristic, have been used for duty scheduling for more than thirty years. Regional characteristics are exploited to reduce the size of the problem and to construct duty schedules. It is obvious that such methods discard many details of the problem.
As reductions have been unavoidable for early schedulers, recent mathematical research and increase in computing power overcomes this necessity. Up to date solution methods for duty scheduling mostly keep all details of the problem to build huge models, consisting of more than ten thousand tasks and many million possible duties, which can then be solved using mathematical methods. An example is the solver DS-Opt which uses a column generation approach based on a set par-

titioning definition of the Duty Scheduling Problem to generate good duty schedules.
Under this circumstances it is even more remarkable that the planers in Plaue report very satisfying results for their planing network. Furthermore their simple heuristic eliminates the problem of longer runtimes which occurs for approximation solvers such as DS-Opt. Plaue serves as an example that clever saturation of the problem structure can still match modern mathematical approaches.
In this thesis we study the question if a general problem reduction approach can be adapted to speed up approximation solvers while almost maintaining the solution quality. We examine some older duty scheduling heuristics for tricks and hints which can be used to reduce the problem to a critical extract before the resulting problem is solved with DS-Opt. As a result we describe a new heuristic which uses a suitable relaxation of the problem to reduce the runtime.

Main Results

The duty scheduling heuristic FAST has been developed, implemented and tested on more than 50 instances of real world duty scheduling problems of different sizes. Complete results are presented for 35 of these instances, while 23 instances of a sufficient size are relevant for our main results.
For these 23 relevant instances FAST achieved **runtime savings of 95.40%** in average compared to DS-Opt.
Of the same instances FAST averaged only a **3.50% increase in the objective value**.
A total of 12 instances have been solved in less than 5% of the DS-Opt runtime, while 5 instances could even be solved in less than 2%. The best result was reported on a large instance which could be solved in 0.06% of the original runtime. Further 13 of the 23 instances reported an objective value increase of less than 2%, while FAST even produced slightly better solutions then DS-Opt for 4 instances. Furthermore, FAST solved 3 instances which could not be solved by DS-Opt in the maximum computation time of three days, set up for this benchmark.

Thesis-Overview

This thesis is subdivided into six chapters. In the first chapter we describe the Duty Scheduling Problem in terms of an acyclic digraph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$. The vertices $\mathcal{V}$ identify vehicle activities and the arcs $\mathcal{A}$ represent possible connections between these activities. Each duty corresponds to a path in $\mathcal{D}$. The aim is to find a set of node-disjunct paths in $\mathcal{D}$ covering all vertices.
Chapter two gives a literature overview on solution approaches toward the problem. We provide a general overview and sort the methods into

three categories. Further we give detailed descriptions for some of these heuristics.
Chapter three solely presents the duty scheduling heuristic HASTUS-Micro. This heuristic outlines the general approach used in our new heuristic FAST.
All details of FAST are covered in the next two chapters. Chapter four gives a complete description and provides algorithms used in FAST, while all computational results and a short analysis of the parameters are discussed in chapter five.
In the last chapter we evaluate the results and suggest some issues for further research.
Finally Appendix A lists complete computational results and Appendix B includes some Duty Scheduling Graphs.

Required Background

In this thesis we assume that the reader is familiar with basic terms concerning graph theory, more precisely digraphs, paths, matchings and flows, as well as linear and integer programming, most importantly with the concepts of set covering, set partitioning and column generation.
Nice surveys on these terms are the books *Linear Programming* by Văsek Chvátal [13] and *Theory of Linear and Integer Programming* by Alexander Schrijver [44]. The notation used in this thesis is based upon the lecture notes *Graphen- und Netzwerkalgorithmen* and *Lineare Optimierung* by Martin Grötschel [29, 30].
Some parts of this thesis also discuss genetic algorithms and tabu search. Overviews on tabu search are found in many books of Fred Glover [26, 27]. Genetic algorithms are covered in the books of Lawrence Davis [19] and David E. Goldberg [28].
Finally we assume that the reader is familiar with the economic problem of duty scheduling and the associated mathematical methodology. [31] provides a wonderful overview on alternative terms in duty scheduling while [51] provides a short summary on duty scheduling heuristics.
The notations and terms used in this thesis are based on publications of Ralf Borndörfer, Martin Grötschel, Andreas Löbel, et. al. [7, 8]. Another similar description is given in [43]. [7] also proposes a column generation technique adapted to the Duty Scheduling Problem and describes the solver DS-Opt which is only outlined in this thesis, but is used as a module in FAST.

Notations

The following is an overview on the notations used in this thesis. Most of the symbols are introduced to model the Duty Scheduling Problem and are defined in chapter 1.

Base and Resource Constraint Modeling

$\mathcal{K}$	set of duty types
$\mathcal{U}$	set of duty resources
$\mathcal{W}$	set of base resources
$\hat{u}_k \in \mathbb{Q}^{\mathcal{U}}$	duty resource limits for every duty type $k \in \mathcal{K}$
$\hat{w} \in \mathbb{Q}^{\mathcal{W}}$	base resource limits
$\tilde{w}_k \in \mathbb{Q}^{\mathcal{W}}$	base resource consumption of duty type $k \in \mathcal{K}$

Duty Scheduling Graph

$\mathcal{D} = (\mathcal{V}, \mathcal{A})$	Duty Scheduling Graph
$\mathcal{V}$	tasks
$\mathcal{T} \subseteq \mathcal{V}$	timetable tasks
$\mathcal{S} \subseteq \mathcal{V}$	supplementary tasks
$\mathcal{I} \subseteq \mathcal{V}$	artificial tasks
$s \in \mathcal{I}$, $e \in \mathcal{I}$	start and end artificial tasks
$\mathcal{A}$	links
$\mathcal{A}_i \subseteq \mathcal{A}$	type i links, $i \in \{1, 2, 3, 4\}$

Task and Link Labels for $\mathcal{D} = (\mathcal{V}, \mathcal{A})$

$\bar{s}_t$	start time of task $t \in \mathcal{V}$
$\bar{e}_t$	end time of task $t \in \mathcal{V}$
$\bar{v}_t$	block id of task $t \in \mathcal{V}$
$\bar{c}_t$	cost for task $t \in \mathcal{V}$
$\bar{u}_{tu}$	consumption of duty resource $u \in \mathcal{U}$ of task $t \in \mathcal{V}$
$\bar{w}_{tw}$	consumption of base resource $w \in \mathcal{W}$ of task $t \in \mathcal{V}$
$\hat{c}_l$	cost for link $l \in \mathcal{A}$
$\hat{u}_{lu}$	consumption of duty resource $u \in \mathcal{U}$ of links $l \in \mathcal{A}$
$\hat{w}_{lw}$	consumption of base resource $w \in \mathcal{W}$ of links $l \in \mathcal{A}$

Duties and Duty Schedules for $\mathcal{D} = (\mathcal{V}, \mathcal{A})$

$\mathcal{P}^{\mathcal{D}}, \mathcal{P}$	set of all possible duties in $\mathcal{D}$
c_p	cost for duty $p \in \mathcal{P}^{\mathcal{D}}$
u_p	duty resource consumption of duty $p \in \mathcal{P}^{\mathcal{D}}$
w_p	base resource consumption of duty $p \in \mathcal{P}^{\mathcal{D}}$
$\tilde{\mathcal{C}}^{\mathcal{D}}$	all duty schedules in $\mathcal{D}$
$\mathcal{C}^{\mathcal{D}}$	all feasible duty schedules in $\mathcal{D}$
$C(C)$	cost of a duty schedule $C \in \tilde{\mathcal{C}}^{\mathcal{D}}$
$\mathcal{O}^C \subset \mathcal{T}$	set of overcovered timetable tasks in $C \in \mathcal{C}^{\mathcal{D}}$

Blocks and Duty Pieces for $\mathcal{D} = (\mathcal{V}, \mathcal{A})$

$\mathcal{F}(\mathcal{D})$	set of all blocks in $\mathcal{D}$
$\mathcal{N}^m$	set of all possible duty pieces in a block $m \in \mathcal{F}(\mathcal{D})$
$\mathcal{N}(\mathcal{D})$	set of all possible duty pieces in $\mathcal{D}$
$\tilde{\mathcal{N}}^p$	set of the pieces of work contained in duty $p \in \mathcal{P}^{\mathcal{D}}$

FAST Preparations

$\mathcal{D}_{MC} = (\mathcal{V}, \mathcal{A}, \tilde{z})$	Multi Cover Duty Scheduling Graph
$\tilde{S}(t) \subset \mathcal{S}$	set of associated supplementary tasks for $t \in \mathcal{T}$
$\mathcal{T}_d \subset \mathcal{T}$	set of driving tasks
$\mathcal{T}_b \subset \mathcal{T}$	set of break tasks
$\mathcal{T}_w \subset \mathcal{T}$	set of working tasks
$\mathcal{S}_{-1} \subset \mathcal{S}$	set of supplementary tasks with indication -1
$\mathcal{S}_{+1} \subset \mathcal{S}$	set of supplementary tasks with indication $+1$
$\mathcal{S}_0 \subset \mathcal{S}$	set of supplementary tasks with indication 0

FAST Heuristic

$\bar{r} \in \mathbb{N}$	rounding parameter in seconds
$\mathcal{D}^{\bar{r}}$	$\bar{r}$-Rounding of $\mathcal{D} = (\mathcal{V}, \mathcal{A})$
$\mathcal{D}^{\bar{r}}_{pos}$	$\bar{r}$-*pos*-Relaxation of $\mathcal{D} = (\mathcal{V}, \mathcal{A})$
$\mathcal{O}(l) \subset \mathcal{A}$	set of original links for $l \in \mathcal{A}^{\bar{r}}_{pos}$
$\hat{q} \in \mathbb{Q}^8$	fix parameter vector for FAST
$\mathcal{T}^{start} \subset \mathcal{T}$	set of timetable task starting fixed pieces
$\mathcal{T}^{end} \subset \mathcal{T}$	set of timetable task ending fixed pieces
$\mathcal{D}^{(\hat{x},\hat{q})}_{fix}$	$\hat{x}$-$\hat{q}$-FAST-Fix of $\mathcal{D} = (\mathcal{V}, \mathcal{A})$

1. The Duty Scheduling Problem

Abstract: We define the Duty Scheduling Problem (DSP) and associated terms such as blocks, duty pieces and link types. Further we introduce a set covering and a set partitioning model of the DSP.

Duty Scheduling is an important part of planning in public transit. Its objective is to cover a set of predetermined tasks of vehicle work with a set of legal driver shifts, minimizing the total cost of the needed shifts. Due to labor agreements all shifts must respect certain constraints, mostly regarding driving time, length of meal breaks and total duration of service. Furthermore, the schedule formed by the selected shifts sometimes has to satisfy certain global restrictions negotiated between traffic companies and worker unions, see [8].

The definition of the Duty Scheduling Problem given in section 1.1 is similar to a description used for airline crew scheduling in [11] adapted to meet the needs of duty scheduling. Sections 1.2 and 1.3 introduce some additional terms related to the problem, while sections 1.4 and 1.5 propose two integer program formulations.

Unfortunately, scheduling terminology differs between countries, between organizations and even within a single organization. Thus mathematical notation is far from being standardized. A widely accepted glossary on alternative terms is [31]. In this thesis, we do not revise terminology, but follow the set of scheduling terms used by Borndörfer, Grötschel, Löbel, et. al. [7, 8].

1.1 Definition

We describe the Duty Scheduling Problem as the problem of covering a set of vertices in an acyclic digraph using vertex-disjunct paths. The vehicle work is represented by the vertices of this digraph. Duties are represented by paths. Constraints are modeled using labels on the vertices and arcs.

Let $\mathcal{K}$ indicate a set of *duty types* which have to satisfy different constraint sets. Common examples for such types are

- part time duties, i.e. short shifts without a long meal break,
- split duties, i.e. shifts consisting of two parts separated by a very long break

- and straight duties, i.e. shifts of normal length containing a meal break.

Different labor agreements apply to different duty types.
Further, let $\mathcal{U}$ be a set of *duty resources*, e.g. driving time, break time or working time, used to construct duty constraints. Let $\mathcal{W}$ be a set of *base resources* used to formulate global restrictions. Restrictions regarding base resources are called *base constraints*. Examples for base constraints are limits on the maximum number of duties of a specific duty type and duty mix restrictions. Such mix restrictions manage the fraction of specific duty types in the total set of duties needed to cover all vehicle work. The sets $\mathcal{U}$ and $\mathcal{W}$ usually contain different resources to satisfy the requirement of different traffic companies.
For all duty types we define duty resource limits, which may not be exceeded by a duty of this type. We define base resource limits, respectively.

Definition 1.1.1 *Let $\mathcal{K}$ be a set of duty types, $k \in \mathcal{K}$ and let $\mathcal{U}$ be a set of duty resources. The vector $\hat{u}_k \in \mathbb{Q}^{\mathcal{U}}$ is called the duty resource limit for the duty type k.*

Definition 1.1.2 *Let $\mathcal{W}$ be a set of base resources. The vector $\hat{w} \in \mathbb{Q}^{\mathcal{W}}$ is called the base resource limit.*

Furthermore, we define base resource consumptions for each duty type $k \in \mathcal{K}$ as follows.

Definition 1.1.3 *Let $\mathcal{K}$ be a set of duty types, $k \in \mathcal{K}$ and let $\mathcal{W}$ be a set of base resources. The vector $\tilde{w}_k \in \mathbb{Q}^{\mathcal{W}}$ is called the base resource consumption of duty type k.*

For example, to formulate a base constraint which limits the number of duties of a specific duty type $k \in \mathcal{K}$ to a maximum of five duties, we simply introduce a base resource $L \in \mathcal{W}$ and set $(\tilde{w}_k)_L = 1$ and $\hat{w}_L = 5$.
Given duty types, resource sets and limits as defined above, we define a digraph on which the Duty Scheduling Problem can be described.

Definition 1.1.4 *Let $\mathcal{K}$ be a set of duty types and let $\mathcal{U}$, $\mathcal{W}$ be duty and base resource sets, respectively. A digraph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ constructed as follows is called a Duty Scheduling Graph. Its nodes $t \in \mathcal{V}$ are called tasks and are subdivided into timetable tasks $\mathcal{T} \subset \mathcal{V}$ representing vehicle work, which has to be assigned to a driver, supplementary tasks $\mathcal{S} \subset \mathcal{V}$ modeling additive activities such as sign-on and sign-off and into artificial tasks $\mathcal{I} \subseteq \mathcal{V}$. Tasks are labeled with start and end times $\bar{s}, \bar{e} \in \mathbb{N}^{\mathcal{V}}$, block id's $\bar{v} \in \mathbb{N}^{\mathcal{V}}$, costs $\bar{c} \in \mathbb{Q}^{\mathcal{V}}$, duty resource consumptions $\bar{u} \in \mathbb{Q}^{\mathcal{V}\times\mathcal{U}}$ and base resource consumptions $\bar{w} \in \mathbb{Q}^{\mathcal{V}\times\mathcal{W}}$. The arcs $\mathcal{A}$ are called links and connect tasks which can be performed consecutively by the same driver. Links are labeled with costs $\hat{c} \in \mathbb{Q}^{\mathcal{A}}$, duty resource*

consumptions $\hat{u} \in \mathbb{Q}^{\mathcal{A}\times\mathcal{U}}$ *and base resource consumptions* $\hat{w} \in \mathbb{Q}^{\mathcal{A}\times\mathcal{W}}$. $\mathcal{D}$ *contains at least two artificial tasks* $s \in \mathcal{I}$ *and* $e \in \mathcal{I}$ *serving as start and end for all shifts.*

In any Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ the set of supplementary tasks $\mathcal{S} \subset \mathcal{V}$ can be empty. We assume, that $\bar{s}_t = \bar{e}_t = \bar{v}_t = \bar{c}_t = 0$, $\bar{u}_t = (0, \ldots, 0)$ and $\bar{w}_t = (0, \ldots, 0)$ for any artificial task $t \in \mathcal{I}$. It would be possible to spare the task labels $\bar{c}, \bar{u}$ and $\bar{w}$ by setting $\hat{c}_l = \hat{c}_l + \bar{c}_{t_1}$ for any link $l = (t_1, t_2) \in \mathcal{A}$, $\hat{u}, \hat{w}$ respectively. For modeling reasons we keep all task labels as defined above. In this thesis we usually indicate tasks by $t, \tilde{t}, t_1$ or t_k and links by $l, \tilde{l}, l_1$ or l_k. Because links only connect tasks, which can be performed consecutively by the same driver, we assume that any Duty Scheduling Graph $\mathcal{D}$ is acyclic.
Whenever we use a Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$, we do not explicitly mention the corresponding duty types $\mathcal{K}$ and resource sets $\mathcal{U}$ and $\mathcal{W}$, but assume that $\mathcal{K}$, $\mathcal{U}$ and $\mathcal{W}$ as well as the corresponding limits and duty type base resource consumptions are also defined [1].
A very small Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ with $|\mathcal{V}| = 11$, $|\mathcal{A}| = 20$, $\mathcal{I} = \{s, e\}$ and $\mathcal{S} = \emptyset$ is shown in Figure 1.1. $\mathcal{D}$ is acyclic and timetable tasks are connected by links.

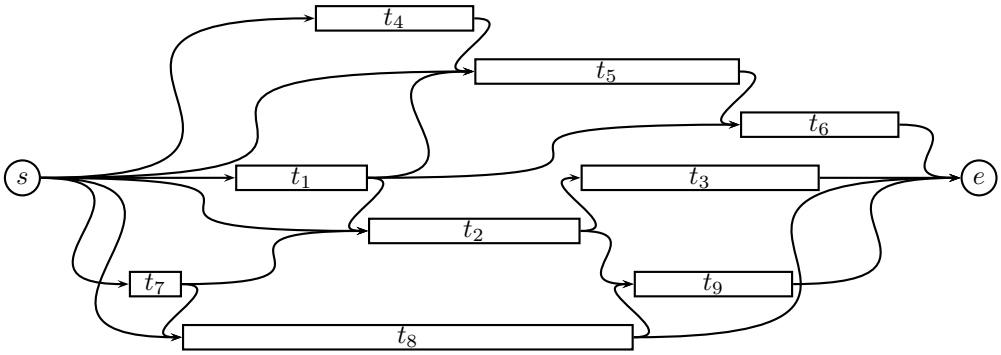

Figure 1.1: Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$

Figure 1.2 displays another small Duty Scheduling Graph using the tool DSVis, see Appendix B. This graph also contains supplementary tasks $t \in \mathcal{S}$. It is clearly visible that all supplementary tasks, represented by small dots, are associated to a single timetable task $\tilde{t} \in \mathcal{T}$. The different colors of the links are described in section 1.3. Further, Figure 1.3 enlarges a small extract of the Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ shown in Figure 1.2 to highlight the surroundings of a timetable tasks $t \in \mathcal{T}$. More Duty Scheduling Graphs for real world instances are part of Appendix B.

[1] The number of duty types, duty resources and base resources is implicitly given by the length of the labels in $\mathcal{D}$.

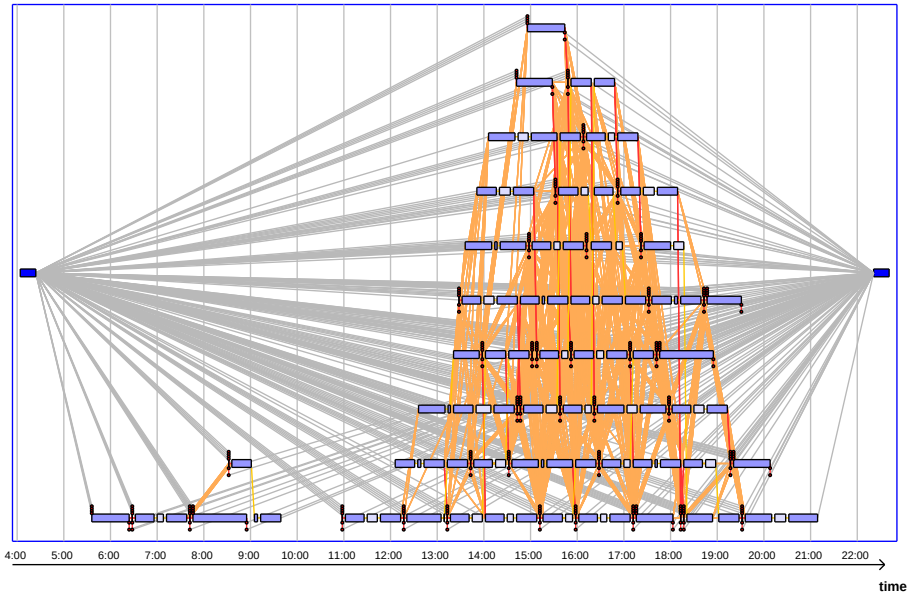

Figure 1.2: Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$, $|\mathcal{V}| = 423$, $|\mathcal{A}| = 3168$

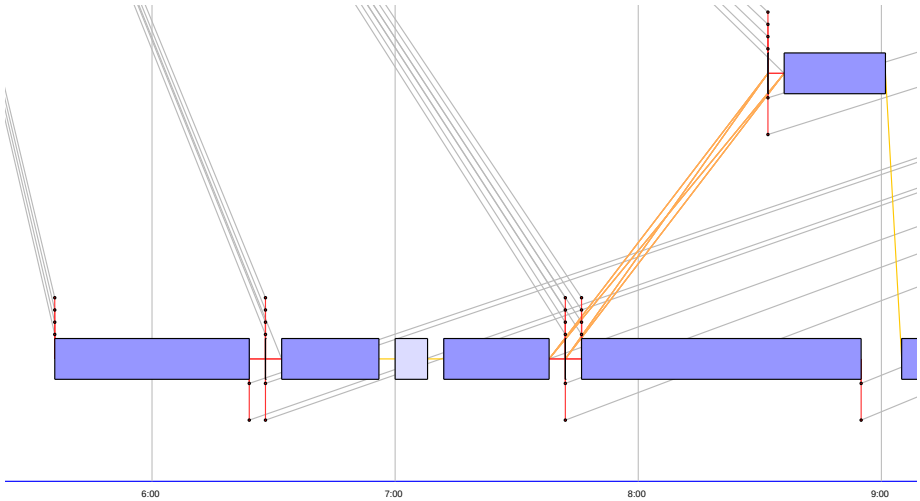

Figure 1.3: Extract of the Duty Scheduling Graph in Figure 1.2

Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph and let p be a path in $\mathcal{D}$ consisting of vertices $V(p)$ and arcs $A(p)$. The path p has cost

$$c_p := \sum_{l \in A(p)} \hat{c}_l + \sum_{t \in V(p)} \bar{c}_t \quad \in \mathbb{Q} \tag{1.1.1}$$

and consumes duty resources

$$u_p := \sum_{l \in A(p)} \hat{u}_{l.}^T + \sum_{t \in V(p)} \bar{u}_{t.}^T \ \in \mathbb{Q}^{\mathcal{U}} \ .$$

Instead of writing $t \in V(p)$ and $l \in A(p)$ we also write $t \in p$ when dealing with a task and $l \in p$ when dealing with a link. We define a duty in $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ as follows.

Definition 1.1.5 *Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph, let $\mathcal{K}$ be the set of duty types and let $\hat{u}_k \in \mathbb{Q}^{\mathcal{U}}$ $\forall$ $k \in \mathcal{K}$ be the corresponding duty resource limits. Further, let p be any (s,e)-path in $\mathcal{D}$. If $(u_p)_L \leq (\hat{u}_k)_L$, $\forall$ $L \in \mathcal{U}$ for any duty type $k \in \mathcal{K}$, p is called a duty of duty type k. The set of all possible duties in $\mathcal{D}$ is denoted by $\mathcal{P}^{\mathcal{D}}$ or $\mathcal{P}$.*

For simplicity and because it applies to real situations, we further assume that any duty $p \in \mathcal{P}$ is of at most a single duty type. Figure 1.4 displays a duty $p \in \mathcal{P}$.
A duty $p \in \mathcal{P}$ of duty type $k \in \mathcal{K}$ has cost c_p as defined in 1.1.1 and consumes base resources

$$w_p := \sum_{l \in A(p)} \hat{w}_{l.}^T + \sum_{t \in V(p)} \bar{w}_{t.}^T + \tilde{w}_k \ \in \mathbb{Q}^{\mathcal{W}} \ .$$

We now define a set of duties which covers all timetable tasks $t \in \mathcal{T}$.

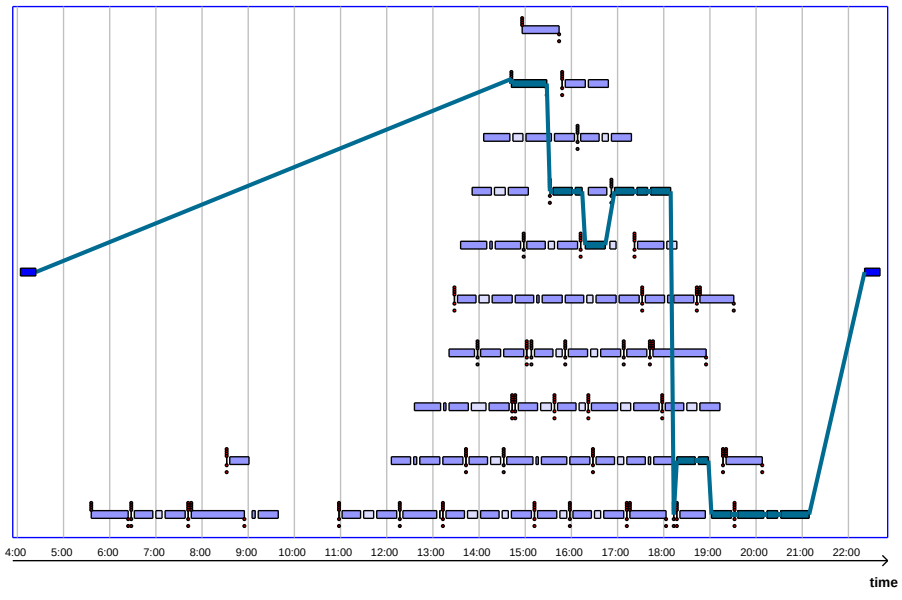

Figure 1.4: Example of a duty in $\mathcal{D} = (\mathcal{V}, \mathcal{A})$

Definition 1.1.6 *Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph and let $C \subset \mathcal{P}$ be a set of duties in $\mathcal{D}$.*
If $\forall\ t \in \mathcal{T} : |\{p \in C \mid t \in p\}| \geq 1$, i.e. each timetable task is covered by at least one duty, the set C is called a duty schedule. Its cost is $C(C) := \sum_{p \in C} c_p$. The set of all duty schedules in $\mathcal{D}$ is denoted by $\tilde{\mathcal{C}}^{\mathcal{D}}$.

A duty schedule as defined above must not be legal, because timetable tasks may be covered more than once and potential base constraints are not satisfied. Thus we define the following.

Definition 1.1.7 *Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph, let $\hat{w} \in \mathbb{Q}^{\mathcal{W}}$ be the corresponding base resource limits and let C be a duty schedule in $\mathcal{D}$. C is called feasible duty schedule, if and only if all the following conditions are satisfied:*

(i) $\forall\ t \in \mathcal{T} : |\{p \in C \mid t \in p\}| = 1$

(ii) $\forall\ L \in \mathcal{W} : (w^C)_L := \sum_{p \in C} (w_p)_L \leq (\hat{w})_L$

The set of all feasible duty schedules in $\mathcal{D}$ is denoted by $\mathcal{C}^{\mathcal{D}}$.

Of course $\mathcal{C}^{\mathcal{D}} \subset \tilde{\mathcal{C}}^{\mathcal{D}}$ for any Duty Scheduling Graph $\mathcal{D}$. Each feasible duty schedule covers each timetable task exactly once and respects all base resource limits. We often say schedule instead of duty schedule or feasible duty schedule when the circumstances do not matter. The Duty Scheduling Problem can now be defined as follows.

Definition 1.1.8 *Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph and let $\mathcal{C}^{\mathcal{D}}$ be the set of all feasible duty schedules. The problem of finding a feasible duty schedule $C \in \mathcal{C}^{\mathcal{D}}$ satisfying*

$$C(C) = \min\{C(\tilde{C}) \mid \tilde{C} \in \mathcal{C}^{\mathcal{D}}\}$$

is called the Duty Scheduling Problem.

In this thesis, we refer to a Duty Scheduling Problem given a Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ as $\min\{C(C) \mid C \in \mathcal{C}^{\mathcal{D}}\}$. The set of base resources $\mathcal{W}$ is empty for many duty scheduling instances.

1.2 Blocks and Duty Pieces

Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph. Further assume that $\mathcal{D}$ is simple, i.e. no parallel arcs are contained. The timetable tasks $t \in \mathcal{T}$ represent work determined by a set of moving traffic vehicles, e.g busses or trams. In real life situations these vehicles follow daily round trips starting and ending at a depot and perform many of the timetable tasks consecutively in-between. Each timetable task $t \in \mathcal{T}$ is part of one such trip, indicated by its block id $\bar{v}_t$. Thus timetable tasks $t, \tilde{t} \in \mathcal{V}$ sharing the same block id, $\bar{v}_t = \bar{v}_{\tilde{t}}$ are part of the same trip.
We now partition $\mathcal{T}$ according to the underlying vehicle round trips. A longest possible path containing only timetable tasks $t \in \mathcal{T}$ which share the same block id $\bar{v}_t$ is called a *block*.

Definition 1.2.1 *Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph. A (t_1, t_k)-path m in $\mathcal{D}$ is called a block with id $i \in \mathbb{N}$, if and only if m suffices the following two conditions:*

(i) $\forall\ t \in V(m) \cap \mathcal{V} : \quad t \in \mathcal{T} \wedge \bar{v}_t = i$

(ii) $\forall\ (\tilde{t}, t_1), (t_k, \tilde{t}) \in \mathcal{A} : \quad \tilde{t} \in \mathcal{T} \Rightarrow \bar{v}_{\tilde{t}} \neq i$

We denote the set of all blocks in $\mathcal{D}$ with $\mathcal{F}(\mathcal{D})$.

We assume that each timetable task $t \in \mathcal{T}$ is contained in exactly one block, i.e. $\mathcal{F}(\mathcal{D})$ is a partitioning of $\mathcal{T}$. This assumption suffices the fact that no two overlapping timetable tasks can have the same block id. Furthermore, each supplementary task $\tilde{t} \in \mathcal{S}$ can be allocated to a single block by its block id $\bar{v}_{\tilde{t}}$. We now define a *duty piece* as a subpath of a block.

Definition 1.2.2 *Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph, let $\mathcal{F}(\mathcal{D})$ be the set of all blocks and let $m \in \mathcal{F}(\mathcal{D})$. A subpath n of m in $\mathcal{D}$ is called a duty piece. The set of all duty pieces in m is denoted by $\mathcal{N}^m$. The set of all duty pieces in the Duty Scheduling Graph $\mathcal{D}$ is denoted by $\mathcal{N}(\mathcal{D})$.*

This definition implies that each block can be subdivided into a set of duty pieces in numerous ways. The number of possibilities to perform such a subdivision on a block m depends on the number of timetable tasks contained in the block $|V(m) \cap \mathcal{T}|$. We define the following.

Definition 1.2.3 *Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph, let $\mathcal{F}(\mathcal{D})$ be the set of all blocks and let $m \in \mathcal{F}(\mathcal{D})$. A set of duty pieces $n_1, \ldots n_k \in \mathcal{N}^m$ with $\bigcup_{i=1}^{k} V(n_i) = V(m)$ and $V(n_i) \cap V(n_j) = \emptyset, i \neq j$ is called a duty piece partition of the block m.*

Duty pieces and duty piece partitions as defined above are not to be confused with *pieces of work*. Pieces of work are maximal subpaths of a duty which are also subpaths of a block, i.e. parts of a duty which are performed on the same vehicle, see [8]. Each piece of work is also a duty piece, but not each duty piece is necessarily a piece of work in a duty schedule. We denote the set of all pieces of work contained in any duty $p \in \mathcal{P}$ by $\tilde{\mathcal{N}}^p$.
In case the Duty Scheduling Graph $\mathcal{D}$ is not simple, blocks and duty pieces are defined similarly, but parallel arcs are not considered during the definition. Blocks and duty pieces are then only identified by the contained tasks, i.e. it does not matter which of the parallel arcs is used in the block or duty piece. Duty pieces and blocks play a very important role in the following chapters of this thesis.

1.3 Link Types

Links connect tasks which can be performed consecutively. We partition the set of links into four subsets according to which tasks they

connect. Type 1 links must be contained in any duty schedule. Type 2 links can either connect successive timetable tasks of the same block or timetable tasks and supplementary tasks representing work associated with the same block id. A type 3 link connects tasks associated with different blocks, while type 4 links connect tasks to the artifical tasks s and e. The four sets are introduced in Definition 1.3.1.

Definition 1.3.1 *Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph. We define the following subsets of $\mathcal{A}$:*

(i) The set of type 1 links $\mathcal{A}_1 \subset \mathcal{A}$ is defined as

$$\mathcal{A}_1 = \{(t_1, t_2) \in \mathcal{A} \mid t_1, t_2 \in \mathcal{T} \wedge |\delta^{out}(t_1)| = 1 = |\delta^{in}(t_2)|\}$$

(ii) The set of type 2 links $\mathcal{A}_2 \subset \mathcal{A}$ is defined as

$$\mathcal{A}_2 = \{(t_1, t_2) \in \mathcal{A} \mid (t_1 \in \mathcal{S} \vee t_2 \in \mathcal{S}) \wedge \bar{v}_{t_1} = \bar{v}_{t_2}\} \cup$$
$$\{(t_1, t_2) \mid t_1 \in \mathcal{T} \wedge t_2 \in \mathcal{T} \wedge (t_1, t_2) \notin \mathcal{A}_1 \wedge \bar{v}_{t_1} = \bar{v}_{t_2} \wedge \bar{e}_{t_1} = \bar{s}_{t_2}\}$$

(iii) The set of type 3 links $\mathcal{A}_3 \subset \mathcal{A}$ is defined as

$$\mathcal{A}_3 = \{(t_1, t_2) \in \mathcal{A} \mid t_1 \neq s \wedge t_2 \neq e \wedge \bar{v}_{t_1} \neq \bar{v}_{t_2}\}$$

(iv) The set of type 4 links $\mathcal{A}_4 \subset \mathcal{A}$ is defined as

$$\mathcal{A}_4 = \{(t_1, t_2) \in \mathcal{A} \mid t_1 = s \vee t_2 = e\}$$

Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph and let $C \in \tilde{\mathcal{C}}^{\mathcal{D}}$ be a not necessarily feasible duty schedule. Obviously

$$\forall\, l = (t_1, t_2) \in \mathcal{A}_1\ \exists\, p \in C : l \in A(p)\ ,$$

i.e. each type 1 link is contained in one of the duties in the duty schedule. Further each duty $p \in \mathcal{P}$ starts and ends with a type 4 link.
We assume that for any Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$

$$\mathcal{A} = \mathcal{A}_1 \,\dot\cup\, \mathcal{A}_2 \,\dot\cup\, \mathcal{A}_3 \,\dot\cup\, \mathcal{A}_4\ ,$$

i.e. the set of links can be partitioned into the four sets introduced in Definition 1.3.1. In Figure 1.2 type 1 links are colored yellow, type 2 links are colored red, type 3 links are colored orange and type 4 links are colored gray. This distinction is useful in chapter 4.

1.4 A Set Partitioning Model

We describe the Duty Scheduling Problem (DSP) defined in section 1.1 as a set partitioning problem (SPP) with side constraints. We further relax the problem by softening the base constraints, i.e. the limit $\hat{w}$, for feasible duty schedules, see definition 1.1.7.
Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph and let $\mathcal{P}$ be the set of all duties regarding $\mathcal{D}$. For any $p \in \mathcal{P}$ introduce a binary variable x_p to decide whether the duty is contained in the final feasible duty schedule. To soften the base resource limits, further define slack variables $s_b \in \mathbb{Q}$

Model 1.1 (**DSP-SPP**)

$$\min \quad \sum_{p\in\mathcal{P}} c_p x_p + \sum_{b\in\mathcal{W}} c_b s_b$$

$$\text{s.t.} \quad \sum_{\{p\in\mathcal{P}|t\ \in\ p\}} x_p = 1 \qquad \forall\, t \in \mathcal{V} \tag{1.4.2}$$

$$\sum_{p\in\mathcal{P}} (w_p)_b x_p - s_b \leq \hat{w}_b \qquad \forall\, b \in \mathcal{W} \tag{1.4.3}$$

$$x_p \in \{0,1\} \quad \forall\, p \in \mathcal{P} \tag{1.4.4}$$

$$s_b \geq 0 \qquad \forall\, b \in \mathcal{W} \tag{1.4.5}$$

and penalty cost $c_b \in \mathbb{Q}$ for any base resource $b \in \mathcal{W}$. The resulting integer program (**DSP-SPP**) is shown in Model 1.1.
Each solution $x \in \{0,1\}^{\mathcal{P}}$ of Model 1.1 corresponds to a set of duties $\{p \in \mathcal{P} \mid x_p = 1\}$. Constraint set 1.4.2 ensures that each timetable task is covered exactly once, i.e. $C = \{p \in \mathcal{P} \mid x_p = 1\} \subset \mathcal{P}$ is a duty schedule and also respects the first condition for feasible duty schedules. If $\mathcal{W}$ is empty, C is a feasible duty schedule. Otherwise if very high penalties c_b are choosen, constraint set 1.4.3 guarantees that C is nearly feasible. Still C may be infeasible with respect to some base constraints. However we tolerate this in our model to guarantee feasibility, i.e. the base constraints in this model are soft constraints. The duty schedule $C \in \mathcal{C}^{\mathcal{D}}$ which corresponds to the optimal solution of Model 1.1 is the duty schedule with minimal cost.

1.5 A Set Covering Model

We now propose a set covering model similar to Model 1.1, which generates a duty schedule in which tasks may be covered more than once, i.e. the duty schedule must not be feasible. Such a model is interesting because set covering concerns many less columns than set partitioning and can be solved effectively.
For any not necessarily feasible duty schedule we define the following.

Definition 1.5.1 *Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph and let $C \subset \mathcal{P}$ be any duty schedule. A timetable task $t \in \mathcal{T}$ is called overcovered with respect to C if*

$$|\{p \in C|\ t \in p\}| > 1\ .$$

We denote the set of all overcovered tasks regarding C by $\mathcal{O}^C$.

Obviously $\mathcal{O}^C = \emptyset$ for any feasible duty schedule C. A duty schedule with $\mathcal{O}^C \neq \emptyset$ has to be adjusted manually to achieve feasibility. Fortunately, $|\mathcal{O}^C|$ is likely to be small as the objective is to minimize total

cost and if timetable tasks are covered more than once, the total cost increases.
Model 1.2 is a set covering model (SCP) generating a duty schedule with $|\mathcal{O}^C| \geq 0$.

Model 1.2 (**DSP-SCP**)

$$\min \quad \sum_{p \in \mathcal{P}} c_p x_p + \sum_{b \in \mathcal{W}} c_b s_b$$

$$\text{s.t.} \quad \sum_{\{p \in \mathcal{P} | t \in p\}} x_p \geq 1 \qquad \forall\, t \in \mathcal{V} \tag{1.5.6}$$

$$\sum_{p \in \mathcal{P}} (w_p)_b x_p - s_b \leq \hat{w}_b \qquad \forall\, b \in \mathcal{W} \tag{1.5.7}$$

$$x_p \in \{0,1\} \qquad \forall\, p \in \mathcal{P} \tag{1.5.8}$$

$$s_b \geq 0 \qquad \forall\, b \in \mathcal{W} \tag{1.5.9}$$

Solutions of Model 1.2 are treated similarly as solutions of Model 1.1, but have to be adjusted to form feasible duty schedules. Model 1.2 does not solve the Duty Scheduling Problem, but generates a duty schedule which is very likely near the optimum. A general solution approach toward the Duty Scheduling Problem is, to first solve Model 1.2 obtaining an optimal solution which defines a duty schedule C. Then all duties covering tasks $t \in \mathcal{O}^C$ are either shortened or the corresponding work is substituted by breaks in all but one of the duties.

2. Solution Methods for the Duty Scheduling Problem

Abstract: We give a literature overview on solution methods for the Duty Scheduling Problem. Such methods can be grouped into three main categories. Mathematical programming heuristics, set partitioning and set covering algorithms and meta heuristics. We cover methods from all three categories, and point out some general approaches.

For a Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ the number of duties $|\mathcal{P}|$ is potentially exponential in the number of tasks. Further the pricing problem, see section 2.4, and the Duty Scheduling Problem (DSP) itself are $\mathcal{NP}$-hard, see [7, 43]. Currently no exact solution algorithm for instances of relevant size is available and as a result, all methods presented in this thesis are heuristic. It is useful to examine heuristics to find a start solution or to improve known solutions to accelerate lp-based algorithms such as DS-Opt, see 2.4.
Because of the economic importance of good duty schedules, many different heuristic approaches have been studied and have been implemented in various software planning packages.
Such approaches include mathematical programming heuristics based on tricky relaxations and decompositions. Also optimized set covering and set partitioning algorithms based on static or dynamic column generation have been implemented. Finally, meta heuristics such as genetic algorithms and tabu search procedures have been adapted to suffice the DSP.
This chapter provides an overview on some efficient duty scheduling heuristics, most of which marked the state-of-the-art during their development.
Most duty scheduling heuristics are part of commercial scheduling modules. Detailed documentation is not necessarily available. Nevertheless we outline the algorithms and point out general ideas of such methods in the following sections.
Furthermore, recent enhancements and unpublished adaptions may have been made to some of the heuristics, which are not discussed in this thesis.

2.1 Overview

In general, duty scheduling heuristics relate to either one of three different mathematical fields as can be seen in Figure 2.1. In this study

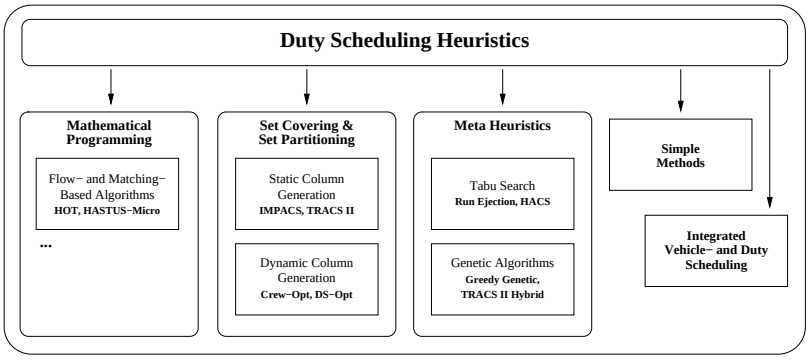

Figure 2.1: Duty Scheduling Heuristics

we further classify heuristics into seven different functional categories.

Simple methods include scheduling systems, where duties of a specific length were cut from long blocks, while short blocks were combined with the remaining slices to form additional duties. Due to their simplicity, use of such methods was spread wide. We do not discuss such methods in this thesis.

Many duty scheduling heuristics are based on mathematical programming methods. Widely used procedures include matching- and flow-based algorithms. Such approaches are covered in detail as we provide successful examples in section 2.2 and in chapter 3.

Section 2.3 discusses another heuristic approach involving static column generation algorithms to solve the set covering definition of the Duty Scheduling Problem. First a huge number of potential duties is generated in advance and all other duties are discarded. Of course the set of potential duties must be chosen smartly. In the second step the set covering model (**DSP-SCP**), see Model 1.2, is formulated and solved regarding only the subset of potential duties.

Dynamic column generation is an iterative method similar to the former approach. Potential duties are not just generated in advance, but throughout the solution process. In each iteration a pricing routine is used to determine new potential duties. These methods mark the state-of-the-art in duty scheduling and are described in many recent papers [7, 20, 22, 23, 48]. In this thesis we only outline dynamic column generation and give references to detailed descriptions, see section 2.4.

Meta heuristics have been used in many fields of applied combinatorial mathematics for fast search of large solution spaces to solve $\mathcal{NP}$-hard optimization problems, see e.g. [2, 39]. Many authors claim, that better solutions than other solvers could be constructed. Some of the most frequently used meta frameworks for duty scheduling are genetic algorithms and tabu search which have been successfully adapted to suffice duty scheduling by some authors. We give a detailed description of these methods in section 2.5 and section 2.6.

It is worth mentioning that some recent developments focus on integrated approaches of vehicle and duty scheduling. During such processes, both the vehicle and duty schedule are obtained simultaneously, allowing to adapt the schedules after every iteration. Such approaches are the next step toward a total goal oriented solution process for operative planning in public transit. However, we do not discuss integrated aspects in this thesis. Some recent publications are [9], [10] and [32].

2.2 Flow- and Matching-Based Algorithms

Most early approaches to solve the DSP were heuristics based on tricky decompositions exploiting characteristics of the problem structure to generate good solutions. Mathematical programming methods such as matching- and flow algorithms or assignment methods were used to assist in the planning process.

An example of such a strategy is implemented in the scheduling heuristic described in [36], where the authors use a matching algorithm to exploit a set of very strict lunch break rules. Further examples are the dynamic programming based algorithm proposed in [4] and the assignment heuristic contained in the scheduling software HOT II which is discussed in section 2.2.1.

Of course it is not possible to mention all interesting methods in this thesis. An overview is given in the proceedings of the conferences on computer-aided scheduling [16, 17, 18, 21, 40, 49, 50]. Nevertheless we outline a very widely used decomposition of the Duty Scheduling Problem into a flow- and a matching problem which serves as base idea for some heuristics, see also chapter 3.

Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a duty scheduling graph. In a first step the timetable tasks $\mathcal{T} \subset \mathcal{V}$ of each block $m \in \mathcal{F}(\mathcal{D})$ are sorted into sets such that each block is partitioned into duty pieces, i.e. a duty piece partition for each block is determined, see section 1.2. A flow algorithm can be used to choose a good partition for each block with respect to the other block partitions. These duty pieces are then regarded as the smallest planning units, i.e. all links connecting other tasks to inner tasks of these duty pieces are dropped. In a second step duties are formed by connecting these pieces. Though it is very restrictive in practice, many older heuristics bound the number of pieces of work contained in a duty, e.g. $\forall\, p \in \mathcal{P}\ |\tilde{\mathcal{N}}^p| \leq 3$. In this case a matching or assignment procedure can be used to form the duties by linking the duty pieces. Otherwise a subset $\tilde{\mathcal{P}} \subset \mathcal{P}$ is generated concerning only remaining links and the DSP is solved on this subset of duties.

A very successful example of this approach is HASTUS-Micro, described in the next chapter. It outlines the general idea used in our new heuristic FAST.

2.2.1 HOT II

HOT II is a commercial scheduling package consisting of modules covering all planning operations of a public mass transit company. It has been developed in Hamburg and was widely used in Germany. Descriptions of HOT II can be found in [15] and [47]. An early description of the assignment procedure used for duty scheduling described below can be found in [37]. Unfortunately there is no exact description of the current duty scheduling algorithm used in this package.
Given a duty scheduling graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$, Algorithm 2.1 describes the HOT II assignment heuristic used to build a feasible duty schedule $C \in \mathcal{C}^{\mathcal{D}}$. We assume that the set of base resources $\mathcal{W}$ is empty.

Algorithm 2.1 (HOT II)

Require: $\mathcal{D} = (\mathcal{V}, \mathcal{A})$
1: $\mathcal{F}(\mathcal{D}) = \mathcal{F}_M \dot\cup \mathcal{F}_A \dot\cup \mathcal{F}_E$
2: $A \leftarrow \emptyset$; $B \leftarrow \emptyset$
3: **for** $F \in \{\mathcal{F}_M, \mathcal{F}_A, \mathcal{F}_E\}$ **do**
4: **for** $m = (t_1, \ldots, t_k) \in F$ **do**
5: **if** $\exists\, p \in \mathcal{P}, (t_1, \ldots, t_j) \in \mathcal{N}^m : \{t_1, \ldots, t_j\} = V(p) \cap \mathcal{T}$ **then**
6: $C \leftarrow C \cup \{p\}$
7: **if** $V(m) \setminus V(p) \neq \emptyset$ **then**
8: $B \leftarrow B \cup \{m\}$
9: **end if**
10: **else**
11: $A \leftarrow A \cup \{m\}$
12: **end if**
13: **end for**
14: Solve an assignment problem on the sets A and B to form more duties
15: **end for**
16: Solve another assignment problem using the remaining duty pieces of morning, afternoon and evening blocks to form split duties
17: **return** $C \in \mathcal{C}^{\mathcal{D}}$

The set of blocks $\mathcal{F}(\mathcal{D})$ is partitioned into subsets according to the start time of the first and the end time of the last contained timetable task. The set $\mathcal{F}_M \subset \mathcal{F}(\mathcal{D})$ contains all morning blocks, beginning very early and ending around midday. $\mathcal{F}_A$ and $\mathcal{F}_E$ contain all afternoon and evening blocks, respectively.
A feasible duty schedule $C \in \mathcal{C}^{\mathcal{D}}$ is build up in three stages. In the first stage early duties are built covering only morning blocks. In the second and third stages afternoon and evening blocks are formed similarly covering only afternoon and evening blocks. Finally, all duty pieces not contained in any of the duties constructed in the three stages are used to form additional split duties. We describe the construction of morning duties.
First, each morning block $m = (t_1, \ldots, t_k) \in \mathcal{F}_M$ is examined. If a

duty $p \in \mathcal{P}$ exists, consisting of a single piece of work $n \in \mathcal{N}^m \cap \tilde{\mathcal{N}}^p$, the duty p is added to the duty schedule[1]. The duty p must cover the first j timetable tasks of m to be legal. Thus at most one uncovered maximal duty piece $\tilde{n} = (t_{j+1}, \dots t_k) \in \mathcal{N}^m$ remains. If the duty p does not cover the complete block m, m is added to a set B containing all morning blocks which are just partly covered. The information which duty piece $n \in \mathcal{N}^m$ is already contained in a duty is also stored.
The remaining morning blocks $m \in \mathcal{F}_M$ for which no such duty can be found due to insufficient break time or because the block is too short, are stored in a set A.
An assignment procedure is now used to construct more morning duties from the unused blocks and duty pieces of partly covered blocks. Blocks contained in the set A are matched with residual duty pieces of the blocks in B. The cost for the assignment of any two blocks $m \in A$ and $\tilde{m} \in B$ is determined by the existence of a duty p covering the complete block m and the remaining duty piece of block $\tilde{m}$. If such a duty p exists, the cost c_p is used in the assignment procedure. If no such duty exists the cost is set to ∞. This is an unbalanced assignment procedure and there are remaining duty pieces not yet covered by any duty p chosen in this first stage.
Afternoon and evening duties are formed similarly from the sets $\mathcal{F}_A$ and $\mathcal{F}_E$. All remaining duty pieces not yet covered by the duties constructed in the three stages are collected and split duties are formed covering all these duty pieces. These split duties are also added to the final schedule, which then covers all timetable tasks. HOT II does not support base constraints. Any constraints regarding the duty mix must be implicitly concerned throughout the procedure or need to be adjusted manually afterwards by the planner.
The authors do not present computational data for HOT II.

2.3 Static Column Generation

Recall from section 1.5 the set covering model (**DSP-SCP**) of the DSP. Unfortunately for real world instances $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ of average size the number of possible duties $|\mathcal{P}|$ exceeds many million. As the model contains a column for each possible duty, it is almost impossible to explicitly solve it, because even the construction of the matrix is an enumeration of all possible duties. Without any further reduction the model has no practical use.
The development of static column generation techniques to construct a reduced version of the model started alongside the constant improvement of mathematical programming heuristics. Given a Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$, static column generation approaches feature the generation of a huge set of useful duties $P \subseteq \mathcal{P}$. The set covering

[1] Each piece of work is a duty piece; $\tilde{\mathcal{N}}^p \subset \mathcal{N}(\mathcal{D})$.

Algorithm 2.2 (IMPACS)

Require: $\mathcal{D} = (\mathcal{V}, \mathcal{A})$, $\mathcal{P}^{\mathcal{D}}$

1: **for all** $m \in \mathcal{F}(\mathcal{D})$ $t_1, t_2 \in m$ **do**
2: **if** $\exists (t_1, t_2) \in \mathcal{A}_2$ and t_1, t_2 are not likely to be in different duties **then**
3: $\mathcal{A} \leftarrow \mathcal{A} \setminus \{\{(t_1, \tilde{t}) \in \mathcal{A} | \tilde{t} \neq t_2\} \cup \{(\tilde{t}, t_2) \in \mathcal{A} | \tilde{t} \neq t_1\}\}$
4: **end if**
5: **end for**
6: Let $\tilde{\mathcal{P}} \subset \mathcal{P}$ be the set of duties in the modified Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$
7: $\forall\ k \in \mathcal{K}$ choose $\hat{u}_k$ stricter such that only favorable duties are possible
8: Generate $P \subset \tilde{\mathcal{P}}$, $|P| \sim 10,000 \wedge \forall\ k \in \mathcal{K}\ \exists\ p \in P : u_p \leq \hat{u}_k$
9: Delete inefficient duties $p \in P$
10: Solve (**DSP-SCP**) regarding only the remaining duties $\hat{P}$ (ZIP Code Ryan 1980)
11: **return** $C \in \mathcal{C}^{\tilde{\mathcal{D}}}$ where $C \subset \hat{P}$

model (**DSP-SCP**) is then constructed only regarding this subset of duties rather than all of them.
Two commercial systems using such approaches are IMPACS [38, 46, 52] and TRACS II [25, 33, 34], both developed and commonly used in Great Britain. We discuss both systems in this section.

2.3.1 IMPACS

IMPACS is the crew scheduling component of the BUSMAN program suite. It uses a static column generation method to generate a feasible duty schedule.
Given a set of duties $\mathcal{P}$ corresponding to an original Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ it consists of two reduction phases where a subset $\hat{P} \subset \mathcal{P}$ is generated. A feasible duty schedule is then constructed by solving the set covering model (**DSP-SCP**) regarding the subset $\hat{P}$. Algorithm 2.2 outlines this procedure.
The first reduction phase is described in the first loop in lines 1-5. All blocks $m \in \mathcal{F}(\mathcal{D})$ are examined. If any two consecutive timetable tasks $t_1, t_2 \in V(m) \cap \mathcal{T}$ exist which are not likely to be covered by different duties, these timetable tasks are fixed to be part of the same duty piece in the solution. All outgoing links of t_1 and all incoming links of t_2 are discarded. Only the type 2 link $(t_1, t_2) \in \mathcal{A}$ is maintained. This link is now a type 1 link. This link reduction removes all duties p from $\mathcal{P}$ where t_1 and t_2 are not contained in the same piece of work. Thus the number of possible duties decreases as this reduction takes place. Let $\tilde{\mathcal{P}} \subset \mathcal{P}$ be the set of remaining duties.
The second reduction phase generates a final set of potential duties $\hat{P}$, see lines 7-9. First the duty resource limits $\hat{u}_k$ are made stricter for any duty type $k \in \mathcal{K}$. Thus, inefficient duties are dropped from the set $\tilde{\mathcal{P}}$. A huge number of favorable duties is then generated according to these stricter limits, which is afterwards reduced as the least efficient duties

Algorithm 2.3 (TRACS II)

Require: $\mathcal{D} = (\mathcal{V}, \mathcal{A})$

/* Phase I */

1: Generate $P \subset \mathcal{P}$ such that P contains only suitable duties with respect to some parameters.
2: Remove from P all duties which do not play a significant role in covering all timetable tasks $\mathcal{T}$. Let $\tilde{P}_0$ be the set of remaining duties
3: Eventually generate more such sets $\tilde{P}_1, \ldots \tilde{P}_k$ using different parameters
4: $\tilde{P} \leftarrow \tilde{P}_0 \cup \ldots \cup \tilde{P}_k$

/* Phase II */

5: Use a heuristic to generate a set of duties $C \subset \tilde{P}$ which together cover all timetable tasks
6: Solve the lp-relaxation of (**DSP-SCP**) concerning only the duties in C; Use a column generation routine or steepest edge approach to add new duties $p \in \tilde{P}$ if $|\tilde{P}|$ is large
7: The solution $x \in \mathbb{Q}^{\tilde{P}}$ provides a target number of duties $N = |\{p \in \tilde{P} | x_p > \delta\}|$
8: Generate a feasible schedule $\tilde{C}$, $|\tilde{C}| = N$ using branch and bound; If necessary increase N
9: **return** $\tilde{C}$

are discarded. The resulting set of duties is $\hat{P}$.
Finally the integer linear program (**DSP-SCP**) is formulated regarding all remaining duties $p \in \hat{P}$. It is solved by a variant of the RYAN ZIP Code, see [42], to generate a duty schedule which is manually adjusted to form a feasible duty schedule.
Due to a high competitive environment the authors do not present useful computational results in any of the references [38, 46, 52].

2.3.2 TRACS II

The TRACS II scheduling suite is a commercial package for duty scheduling, which also supports the scheduling of train drivers. It has been developed as successor of IMPACS. We give a short overview in this section.
Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph and let $\mathcal{P}$ be the corresponding set of duties. TRACS II is based upon a two phase approach as outlined in Algorithm 2.3.
During the first phase, a huge number of feasible duties $P \subset \mathcal{P}$ is generated. All of these duties satisfy a set of previously defined requirements in order to be suitable for a final schedule. Such constraints may be a specified minimum spread time, a minimum driving time or a maximum meal break length. Duties, which do not satisfy all constraints are prevented and not added to P. After this generation, P contains a huge number of duties and is still too large. Thus P is reduced as duties, which are not significant to cover all timetable tasks are dropped. In this sense a duty $p \in P$, which covers only a part of the timetable tasks covered by duty $\tilde{p} \in P$ having similar cost can be removed. All remaining duties are stored in a set $\tilde{P}_0 \subset \mathcal{P}$.

Eventually more such duty sets $\tilde{P}_1, \ldots \tilde{P}_k$ are constructed regarding different constraint sets, see line 3. All duties contained in any of these sets are potential candidates for the final schedule and are added to the duty set $\tilde{P}$. This set is significantly smaller, i.e. $|\tilde{P}| \ll |\mathcal{P}|$.
At the beginning of the second phase a heuristic is used to generate a subset $C \subset \tilde{P}$ containing duties, which together cover all timetable tasks. This set of duties is used to define the lp-relaxation of the set covering model (**DSP-SCP**) which is solved as new duties from the set $\tilde{P}$ are added to the problem in a column generation manner. If possible, a steepest edge approach is used instead of a column generation technique. A fractional solution is obtained indicating a target number of duties to be contained in the final duty schedule, see line 7. An integer solution is generated using a Branch&Bound technique. This integer solution corresponds to the final duty schedule $C \in \mathcal{C}^{\mathcal{D}}$. Obviously C uses only duties contained in the prior generated set $\tilde{P}$.
[34] claims that TRACS II outperformed many established bus driver scheduling systems by year 2000 in the generation of efficient schedules. However the sizes of the presented problem instances are not provided.

2.4 Dynamic Column Generation

Dynamic column generation, mostly referred to as column generation, is a very efficient and widely accepted iterative process to solve linear relaxations of general set partitioning models consisting of many columns. Theoretical and practical descriptions of dynamic column generation algorithms are found throughout the literature. An example is the Branch&Price algorithm described in [3].
In every iteration, a so called restricted master linear problem (RMLP) is solved along with one or more subproblems from which new columns are generated. Each of the restricted master linear problems contains only a subset of the original set of columns which changes dynamically after each iteration as new columns are added and others are dropped. Dynamic column generation can easily be adapted to suffice additional side constraints and thus it can be used to tackle the linear relaxation of (**DSP-SPP**), see section 1.4.
Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph and let $\mathcal{P}$ be the corresponding set of duties. Algorithm 2.4 describes a general dynamic column generation procedure to generate a feasible near optimal duty schedule $C \in \mathcal{C}^{\mathcal{D}}$.
First, an initial subset $\tilde{\mathcal{P}} \subset \mathcal{P}$ is generated similarly to static column generation procedures. $\tilde{\mathcal{P}}$ must contain at least one feasible duty schedule to ensure feasibility. The duties contained in $\tilde{\mathcal{P}}$ can either be known duties or they can be generated a priori as described in IMPACS or TRACS II. A good set $\tilde{\mathcal{P}}$, i.e. a set of duties containing a good feasible duty schedule, speeds up the solution process. The master linear problem is the linear relaxation of (**DSP-SPP**). The current

Algorithm 2.4 (Dynamic Column Generation)

Require: Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$, $\mathcal{P}$, $\lambda \in \mathbb{Q}$
1: Define an initial subset $\tilde{\mathcal{P}} \subset \mathcal{P}$ which contains at least a duty schedule $C \subset \tilde{\mathcal{P}}$
2: **while** TRUE **do**
3: $x \leftarrow$ Solve the (RMLP) regarding the subset $\tilde{\mathcal{P}}$
4: $lb \leftarrow$ Calculate a lower bound on the objective using duality
5: $ub \leftarrow$ Calculate the objective regarding x. (upper bound)
6: **if** $ub \leq (1 + \lambda)lb$ **then**
7: break
8: **else**
9: Add duties $p \in \mathcal{P} \setminus \tilde{\mathcal{P}}$ with negative reduced cost to $\tilde{\mathcal{P}}$
10: Eventually remove some unused duties from $\tilde{\mathcal{P}}$
11: **end if**
12: **end while**
13: Generate a feasible duty schedule $\tilde{C} \in \mathcal{C}^{\mathcal{D}}$ from the current fractional solution x
14: **return** $\tilde{C}$

restricted master linear problem contains only columns associated with duties which are contained in $\tilde{\mathcal{P}}$. An optimal solution is found using an lp-solver such as CPLEX 10.0. The optimal solution x obtained, corresponds to a fractional duty schedule which is optimal for the given set of duties $\tilde{\mathcal{P}}$. The objective value regarding x is an upper bound for the optimum of the master linear program. Subproblems are called to generate duties with negative reduced costs. These routines are called pricing problems. Depending on the kind of the pricing routine, a suitable method to generate such duties can be modelled using a resource constrained shortest path problem, see for example [7, 43]. If no duties with negative reduced cost exist, the current fractional solution x is also optimal for the master linear problem. Otherwise, new columns are added, i.e. the duties with negative reduced cost are added to $\tilde{\mathcal{P}}$ and the next iteration is performed. Theoretically, dynamic column generation always finds the optimal solution of the master linear problem. However for problems related to real world data, optimal solutions are seldom achieved in a satisfying time frame, because the pricing problem in $\mathcal{NP}$-hard. Thus a parameter $\lambda \in [0, 1]$ is introduced to abort the iterative process when the current solution is provably near the optimum. Fortunately many methods exist to generate a feasible duty schedule $\tilde{C} \in \mathcal{C}^{\mathcal{D}}$ given a fractional schedule.

Some duty scheduling optimizers using the dynamic column generation approach are the PROB1 solver in the CARMEN system [48], the CREW-Opt optimizer contained in the HASTUS system [20, 22, 23] and DS-Opt [7, 8, 43].

CREW-Opt was one of the first solvers which implemented dynamic column generation to solve duty scheduling problems. It replaced the former heuristic HASTUS-Micro in the HASTUS scheduling package.

The DS-Opt optimizer developed at the Konrad Zuse Institute Berlin

is integrated in the BERTA and MICROBUS systems, see [8]. It is also used as an optimizer in our new heuristic FAST, see chapter 4. Currently, dynamic column generation solvers as listed above mark the state-of-the-art in duty scheduling. Some results for DS-Opt are listed in Appendix A.

2.4.1 Dynamic Aggregation of Set Partitioning Constraints

Many duty scheduling heuristics are based on the approach to group timetable tasks into duty pieces to reduce the size of the problem as all associated links can be discarded. The resulting problems usually can be solved much faster. This approach is backed by the fact that most efficient duties consist of just a few pieces of work as a vehicle change is costly and thus many consecutive timetable tasks of the same block are contained in the same duty. A new approach described in [24] exploits this characteristic to accelerate the dynamic column generation process.
Recall, that large duty scheduling problems generally contain more than ten thousand timetable tasks and specify more than one trillion feasible duties. Thus the corresponding (**DSP-SPP**), see Model 1.1, consists of equally huge numbers of set partitioning constraints and variables. Furthermore, a feasible duty usually covers in average only around twenty to thirty of the timetable tasks resulting in columns which contain only this number of non-zero elements in the set partitioning section of the model. Consequently, the restricted master linear problems associated with the lp-relaxation of (**DSP-SPP**), i.e. the master linear problem, are highly degenerated. This fact considerably slows down the column generation process.
The dynamic constraint aggregation algorithm described in [24] overcomes this degeneracy by grouping set partitioning constraints according to an equivalence relation. Two timetable tasks $t, \tilde{t} \in \mathcal{T}$ are said to be equivalent, if each duty in the current set $p \in \tilde{\mathcal{P}}$ contains either both tasks $t, \tilde{t}$ or none of them. For each equivalence class a single partitioning constraint is added to represent all timetable tasks contained in this set. The so declared groups of timetable tasks identify fixed duty pieces, as timetable tasks of different blocks should never be contained in the same equivalence class if the set $\tilde{\mathcal{P}}$ is chosen reasonably.
Throughout the dynamic column generation process the restricted master linear problems are substituted by aggregated master linear problems which are much easier and faster to solve as they contain much less partitioning constraints.
To guarantee optimality, the aggregation is modified dynamically in every iteration of the procedure.
The authors claim, that this method significantly reduces the size of the master problem, degeneracy, and solution times, especially for larger instances. In general for integrated vehicle and duty scheduling instances they state a reduction of 39% in average of the number of constraints

and 90% in average of the master problem solving time.
A detailed description of this recent approach, whose efficiency has not yet been validated by other authors, is found in [24]. Certainly future research in the area of duty scheduling will focus strongly on designing and analyzing more sophisticated strategies using aggregation and to adapt similar methodology to interior point methods.

2.5 Genetic Algorithms

Genetic algorithms are a general purpose solving strategy capable of searching highly complex solution spaces because they do not get stuck in local optima. Their analogy is with population genetics and evolution as proposed by Charles Darwin's theory. Often near optimal solutions are found very fast while optimal solutions are almost never reached.
Due to the complexity of the corresponding solution space, duty scheduling instances have been effectively solved by genetic algorithms. Several authors have adapted and implemented genetic procedures to support other methods or to serve as standalone duty scheduling heuristics. Some publications regarding genetic algorithms and duty scheduling are [14, 53, 34, 5].
[14] proposes a tailored genetic algorithm for duty scheduling. We discuss this algorithm in section 2.5.2.

2.5.1 Basics

Some basic terms on genetic algorithms are reviewed in this section. Useful overviews are [19] and [28].
A genetic algorithm is an iterative solving strategy. Each iteration consists of a set of members forming a population $P = \{M_1, \ldots, M_n\}$. Each member $M_i \in P$ of the population represents a possible solution in the solution space and has a specified value $f(M_i)$ representing its fitness, i.e. the quality of the solution. The fitness indicates how likely the member is to survive and to recombine. It represents the competition for survival and competition to mate. The size of each generation $|P|$ is bounded by a value n representing the scarcity of the population. For two members $M_i, M_j \in P$ we define the generation of all possible children as $C_{ij} = C(M_i, M_j) = \{C_1, \ldots, C_l\}$. The general strategy of any genetic algorithm is to replace the weakest members in the population with the offspring of the fittest, while raising the average fitness level.
Let P_k be the population of the current iteration. Two members $M_i, M_j \in P_k$ are chosen to mate with respect to the fitness values of all members. Fitter members are more likely to be chosen than others. Let C_{ij} indicate the offspring. Some weak members of P_k are chosen randomly and compared to members of C_{ij}. Assume $C_l \in C_{ij}$ is a chosen child and $\tilde{M} \in P_k$ a chosen weak member of the current population. The standard replacement procedure is to compare the fitness value of

C_l to those of $\tilde{M}$, i.e. if $f(C_l) > f(\tilde{M})$, $\tilde{M}$ is discarded and replaced by C_l resulting in the new population $P_{k+1} = P_k \backslash \tilde{M} \cup C_l$.
One advanced method to choose members for mating and discarding from a population in a probabilistic manner is the roulette wheel selection described in [19]. In this method each member is assigned a probability based on its fitness. A population member is being selected for mating or for being replaced with respect to this probability.
After a predefined number of iterations the best member of the current population is determined. The associated solution is returned.
During the iterative process, fitter population members are more likely to survive and to mate. Thus the average fitness level increases, as do the chances that the best member represents a usable solution.

2.5.2 Greedy Genetic Algorithm

This section outlines one of the first implemented genetic duty scheduling algorithms where a greedy procedure is used to choose population members for mating and recombining. Duty schedules generated with this algorithm are typically within the range of two to three duties near the best known solution. The algorithm does not construct a feasible duty schedule. A detailed description of the algorithm is found in [14].
Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph and let the corresponding set of base resources $\mathcal{W}$ be empty. Each population member $M_i \in P$ is a duty schedule $C \in \mathcal{C}^{\mathcal{D}}$, i.e. we call each duty schedule a population member. Members are represented by a chromosome structure, i.e. a chain of 0 and 1, whose total size equals the number of duties $p \in \mathcal{P}$. Assume that $\mathcal{P} = \{d_i | i = 1, \dots 12\}$ and that the population member $M_i \in P$ is a duty schedule $C = \{d_2, d_6, d_9, d_{12}\}$, then M_i would be represented by the chromosome **010001001001**.
The length of each chromosome is equal, but the number of bits taking value '1' may differ. The fitness value $f(M_i)$ for each member $M_i \in P$ is the cost of the underlying duty schedule, $C(C)$. We assume, that a duty schedule which contains less duties is always fitter than duty schedules containing more duties.
Given a target number of generations K and a value indicating the size of each generation n, Algorithm 2.5 is an outline of the greedy genetic algorithm.
Let $P_k = \{M_1, \dots, M_n\}$ indicate the population in iteration $k \in \mathbb{N}$, i.e. the k-th generation. The initial population P_0 is obtained by generating n random duty schedules, see lines 3-6. The roulette wheel method is used in each iteration to choose two members M_i and M_j with rather high fitness value for recombining, see lines 8 and 9. Each member is given a probabilistic value indicating how likely the member is to be picked depending on the actual fitness of the member relating to the total fitness of the population. Similarly, a weak member with a low fitness value is chosen to eventually be substituted by a child. The currently best member is never chosen to be replaced and thus the best

Algorithm 2.5 (Greedy Genetic Algorithm)

Require: Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$, $K \in \mathbb{N}, n \in \mathbb{N}$
Ensure: $K > 0$
1: $k \leftarrow 0$
2: $P_0 \leftarrow \{\}$
3: **for** $i = 0$ to n **do**
4: $M_i \leftarrow$ Generate random duty schedule $C \in \mathcal{C}^{\mathcal{D}}$
5: $P_0 \leftarrow P_0 \cup M_i$
6: **end for**
7: **while** $k < K$ **do**
8: $M_i \leftarrow$ Choose random member $M_i \in P_k$
9: $M_j \leftarrow$ Choose random member $M_j \in P_k \setminus \{M_i\}$
10: $C_i \leftarrow$ Generate a child $C_i \in C_{ij}$
11: $\hat{M} \leftarrow$ Choose a random weak member $\hat{M} \in P_k$
12: **if** $f(C_i) > f(\hat{M})$ **then**
13: $P_{k+1} = P_k \setminus \{\hat{M}\} \cup C_i$
14: **end if**
15: If P_k indicates convergence, replace most members $M \in P_k$ with new random schedules.
16: $k \leftarrow k + 1$
17: **end while**
18: **return** $\tilde{M}$ where $f(\tilde{M}) = min\{f(M)|M \in P_{k-1}\}$

duty schedule is always maintained in the population. This is known as an elitist treatment.

A child C_i is generated by a chromosome crossover method. The bits for the child chromosome are chosen randomly from both parents. A 0 is possible though both parents have bit 1 at the position. Thus the child can have less duties than either of the parents. The possible child chromosomes are implicitly given by the possible combinations of M_i and M_j. Two possible child schedules are shown in Figure 2.2.

Parent Schedules
01000100100011010 0
10001000100101100 1
0100100010001101 00
1000100100101100 1
Child Schedules

Figure 2.2: Possible crossovers of two chromosomes

To generate a child by crossover, the authors propose some approaches with the aim to pick a good child, rather than any. Assume, that the set of timetable tasks $\mathcal{T}$ is ordered, and that $D_t \subset \mathcal{P}$ is the set of all duties containing timetable task $t \in \mathcal{T}$ which are contained in either one of the parent schedules.

- **random crossover** - Run through the ordered set $\mathcal{T}$ and for each yet uncovered timetable task $t \in \mathcal{T}$ choose a random duty $p \in D_t$.
- **greedy crossover** - Run through the ordered set $\mathcal{T}$ and for each yet uncovered timetable task $t \in \mathcal{T}$ choose the duty $p \in D_t$, which covers the most of yet uncovered timetable tasks.
- **ip crossover** - Generate and solve an integer program to generate the optimal child schedule.
- **permutated greedy crossover** - Same as greedy crossover, but rearrange the timetable tasks $\mathcal{T}$ after each call.
- **variable greedy crossover** - Diversify the greedy. Choose either $p \in D_t$ as in greedy crossover or choose $p \in D_p$ according to how much the remaining work is fractured.

The random crossover generates a different child for each different run, while the greedy crossover always produces the same child. Choosing a random child thus keeps the complexity of the search space, while the greedy crossover simplifies it rapidly. As a result using the greedy crossover creates good solutions very fast, but if the number of generations is high, the random crossover method catches up and overtakes the greedy solutions. Integer linear program crossover is overtaken even faster as it gets stuck very early on good solutions. The last two options are attempts to randomize the greedy crossover to produce good solutions in a short time, but keep the possibility to improve the solution in later iterations. These two provided the best results.
Furthermore, whenever a convergence of the population is detected, most population members are replaced by random schedules. The next iteration continues with this modified population, see line 15. Because the few remaining schedules are much fitter than the rest, the fitness values are normalized. Therefore the new random members have the chance to pass their genetic information onto later generations and thus new duties are generated. To normalize, the fitness values are ordered and the member with the highest fitness value gets a new fitness value three times as high as the lowest member.
When the predefined number of generations is reached, the currently best schedule is returned.
Results of the greedy genetic algorithm using different crossover strategies are listed in [14]. However, the authors only state the number of duties in the final schedule but do not describe the problem instances.

2.5.3 TRACS II - Hybrid Genetic Algorithm

Although TRACS II, outlined in section 2.3.2, provides good solutions, it sometimes fails to generate an integer solution in the Branch&Bound process. In such cases, parameters must be changed to modify the set of potential duties $\tilde{P}$. This requires a lot of insight and advanced knowledge which is not always available.

In [34] the authors propose a hybrid genetic algorithm (GA) to overcome this disadvantage. If Branch&Bound fails to generate an integer solution in the second phase of TRACS II, the GA constructs a duty schedule out of the fractional set of duties $\{p_1, \dots p_n \in \mathcal{P} | x_{p_i} > 0\}$, obtained by solving the lp-relaxation of the set covering model. The set of base resources must be empty for this algorithm. This procedure never fails to generate a feasible duty schedule. Algorithm 3.4, more precisely lines 10 to 18 show the embedment of the GA in TRACS II. After TRACS II finished, the GA is used to find a feasible duty schedule C_{GA}. If $|C_{GA}| > N$, the Branch&Bound process to determine an integer solution in TRACS II is repeated with the new target number of duties $|C_{GA}|$. As long as feasible solutions are generated by this process, the target number is reduced. Finally, the best of all generated duty schedules is returned. We describe the GA in this section.

Algorithm 2.6 (TRACS II Hybrid GA)

Require: $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ and $\mathcal{P}$ set of all feasible duties.

1: Generate $P \subset \mathcal{P}$ such that P contains only suitable duties.
2: Remove from P all duties which do not play a significant role in covering all timetable tasks $\mathcal{T}$. Let $\tilde{P}_0$ be the set of remaining duties.
3: Eventually generate more such sets $\tilde{P}_1, \dots \tilde{P}_k$ using different parameters.
4: $\tilde{P} \leftarrow \tilde{P}_0 \cup \dots \cup \tilde{P}_k$
5: Use a heuristic to generate a set of duties $C \subset \tilde{P}$ which together cover all timetable tasks.
6: Solve the lp-relaxation of (**DSP-SCP**) concerning only the duties in C. Use a column generation routine or steepest edge approach to add new duties $p \in \tilde{P}$ if $|\tilde{P}|$ is large.
7: The solution $x \in \mathbb{Q}^{\tilde{P}}$ provides a target number of duties $N = |\{p \in \tilde{P} | x_p > \delta\}|$.
8: Generate a feasible schedule $\tilde{C}$, $|\tilde{C}| = N$ using branch and bound. If necessary increase N.
9: $\hat{C} \leftarrow \{p \in \tilde{P} |\ x_p > 0.2\}$
10: $C_{GA} \leftarrow$ duty schedule generated by the GA regarding the fractional duty set $\hat{C}$.
11: **if** $|C^0_{GA}| > N$ **then**
12: $\quad N \leftarrow |C^0_{GA}|$
13: $\quad i \leftarrow 1$
14: $\quad$ **while** $\tilde{C}^i \leftarrow$ Generate feasible schedule $\tilde{C}$, $|\tilde{C}| = N$ using Branch&Bound **do**
15: $\quad\quad N \leftarrow N - 1$
16: $\quad\quad i \leftarrow i + 1$
17: $\quad$ **end while**
18: **end if**.
19: **return** Best solution found out of $C_{GA}, \tilde{C}^1, \dots \tilde{C}^i, \tilde{C} \subset \mathcal{P}$.

Once again let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph and let $\mathcal{P}$ be the set of all corresponding feasible duties. We indicate the set of potential duties, i.e. the set of duties generated in phase one of TRACS

II by $\tilde{P} \subset \mathcal{P}$. Furthermore, let the set of preferred duties $\hat{C} \subseteq \tilde{P}$ be the set of duties corresponding to the fractional solution $x \in [0,1]^{\tilde{P}}$ as mentioned above.
Let n be the size of the population in each iteration and let g indicate the number of generations. The length of chromosomes representing the population members is $L \in \mathbb{N}$, where

$$L = |\{p \in \hat{C} | x_p > 0.2\}| \ ,$$

i.e. each preferred duty with a fractional value greater 0.2 is represented by a gene. Each chromosome represents a partial duty schedule $\tilde{S}_i \subset \hat{C}$, where gene i stores value 1 if the duty i is contained in this partial schedule, 0 otherwise. The so defined partial duty schedules must not cover all timetable tasks $t \in \mathcal{T}$.
Each partial schedule $\tilde{S}_i$ can be augmented to form a full schedule $C_i \in \mathcal{C}^{\mathcal{D}}$ by first adding duties $p \in \tilde{P} \setminus \tilde{S}_i$ in a greedy manner and discarding redundant duties after all timetable tasks $t \in \mathcal{T}$ have been covered.
Each population member $M_i \in P$ is a partial schedule and represented by a chromosome. The fitness of $M_i \in P$ is indicated by

$$f(M_i) = \sum_{p \in S_i} c_p + M \cdot |S_i| \quad M \in \mathbb{N} \quad M \gg 0 \ .$$

Furthermore, for each member a set of so called trait duties, which are always passed to the child generation, is indicated. These duties are contained in the full schedule and are considered important, because they are a cost effective combination and should be part of any child schedule.
The initial population $P_0 = \{M_1, \ldots, M_n\}$ is constructed as n random chromosomes are generated as follows. A value $r \in \mathbb{Q}$, $r \sim 0.25 \cdot L$ is chosen randomly and for each chromosome r random genes are given value 1, the rest is given value 0. The corresponding full schedules $C_1, \ldots, C_n \in \mathcal{C}^{\mathcal{D}}$ are constructed, the fitness values $f(M_1), \ldots f(M_n)$ are calculated and the trait duties are identified.
The roulette wheel method is used to choose population members for mating. Assume that two rather fit members $M_i \in P$ and $M_j \in P$ have been chosen so. A single point crossover strategy on the parent chromosomes, see Figure 2.3, is used to form two child chromosomes, which again indicate partial schedules. These are augmented to form full schedules by first adding all trait duties of one of the parents and then executing the procedure mentioned above.
Everytime parents are chosen to mate, a random number r between 1 and g is generated. The number of generations the fittest member survived is indicated by s. In the case of $r \leq s$, mutation occurs before the crossover. The requirement $r \leq s$ implies that mutation is more likely to occur in later generations and thus discourages premature convergence.

Parent Schedules

0100010010001101 00

1000100010010110 01

0100010010010110 01

1000100010001101 00

Child Schedules

Figure 2.3: A single point crossover of two chromosomes

If mutation occurs, some of the genes which have equal values for both parents are changed. Selection and the number of genes which are changed are random.
To form a new generation P_i, the current parent population P_{i-1} is merged with all generated children and all members are ranked by their fitness value. A proportion of this combined population, which is selected by deterministically choosing fit members, survives, all other members are discarded. New random members are added to the population till the number of individuals equals n again.
The algorithm either stops at the predetermined number of generations g or if the fittest member survives a certain number of generations.
TRACS II together with the hybrid GA always finds a feasible duty schedule[2]. Results of TRACS II with GA are slightly lower than TRACS II without GA. [34] also reports computational results showing that the GA outperforms TRACS II for the biggest instances in the test set. However, exact sizes of these instances are not provided.

2.6 Tabu Search

Tabu search is an iterative procedure designed for the solution of optimization problems. It is a neighborhood search strategy which searches the solution space while avoiding entrainment in cycles by maintaining a list of previously visited solutions in a so called tabu list. An efficient memory scheme is necessary to deal with such a list. Each solution has an associated set of neighbored solutions which can be reached by a specific move. These moves have to be defined for each tabu search scheme and identify the neighborhood. Iteration moves which lead to solutions stored in the tabu list are prohibited or penalized. Thus tabu search ensures that new regions of the solution space are investigated throughout the process, ultimately finding the desired solution. Tabu search was invented by Glover [26, 27] and has been used to solve a wide range of hard optimization problems such as job shop scheduling, graph coloring, the traveling salesman problem and the capacitated arc routing problem. Recently it has also been used to solve duty scheduling problems.
[12] proposes two tabu search based heuristics for the duty scheduling

[2] $\mathcal{W} = \emptyset$.

Algorithm 2.7 (Run-Ejection)

Require: Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$

1: Build an initial solution $C_1 \in \mathcal{C}^{\mathcal{D}}$, by first grouping the timetable tasks of each block into duty pieces and afterwards forming a feasible duty schedule only consisting of duties using at most two of these pieces
2: Build a non-bipartite graph $G_1 = (V_1, A_1)$ where each node corresponds to a duty piece in the initial duty schedule C_1 and each feasible two-piece-duty $p \in C_1$ is represented by an edge between two such nodes
3: $i \leftarrow 1$
4: **while** TRUE **do**
5: Construct G_{i+1} from G_i using the ejection chain method
6: Solve the Maximum Cardinality Matching Problem for G_{i+1} using a greedy procedure resulting in the next feasible duty schedule $C_{i+1} \in \mathcal{C}^{\mathcal{D}}$
7: **if** Current schedule C_{i+1} is OK **then**
8: BREAK
9: **else**
10: $i \leftarrow i + 1$
11: **end if**
12: **end while**
13: **return** C_{i+1}

problem. The first, tabu crew, uses a tabu search framework to improve an initial solution constructed by a standard run-cutting procedure. In each improvement step the most expensive duties are destroyed and the resulting pieces are used to form new duties. This procedure may pass through a sequence of incomplete solutions and thus represents a form of strategic oscillation. The second and more successful heuristic is called run ejection; discussed in section 2.6.1.
Another heuristic also based on a tailored tabu search framework, HACS, is described in [45] and is also outlined in section 2.6.2. Both algorithms do not support base constraints, i.e. $\mathcal{W} = \emptyset$.

2.6.1 Run Ejection Algorithm

The run ejection algorithm is proposed in [12] and achieved good solutions for instances of the Lisbon Metro Company. In these scenarios feasible duties consist of either one or two pieces of work. Thus the set of duties $\mathcal{P}$ is restricted to duties consisting of at most two pieces of work. Furthermore the objective is not to minimize the total cost, but the number of duties contained in the final schedule. Algorithm 2.7 outlines the run ejection algorithm.
It is a tabu search procedure where in each iteration a duty piece partition of a Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$, i.e. for all blocks $m \in \mathcal{F}(\mathcal{D})$, is determined and a good duty schedule using these duty pieces is constructed. An initial solution is generated using a simple heuristic, which also provides a duty piece partition for each block. Each duty piece partition of $\mathcal{D}$ corresponds to a digraph G_i where the duty pieces are vertices and all feasible two piece duties are arcs con-

necting these vertices. The iterations are performed in a tabu search manner. The tabu search neighborhood structure is embedded and each move between two solutions consists of two steps. First the current digraph $G_i = (V_i, A_i)$ is transformed into $G_{i+1} = (V_{i+1}, A_{i+1})$ using a so called ejection chain method. New duty pieces are generated and existing ones are replaced, resulting in a slightly different duty piece partition for $\mathcal{D}$. Then a feasible duty schedule, represented by a set of arcs in G_i, is determined by solving a maximum cardinality matching problem on the current digraph G_{i+1}.
We now describe the operations performed in the ejection chain method to transform the digraph. Let $\tilde{N}_i \subset \mathcal{N}(\mathcal{D})$ be the current set of duty pieces and let G_i be the current digraph having a vertex for each duty piece $n \in \tilde{N}_i$. The ejection chain method consists of levels $l = 1, \ldots, L$, where in each level l a subgraph $\tilde{G}_i^l$ in G_i^l is deleted and replaced with another subgraph $\hat{G}_i^l$ resulting in $G_i^{l+1} = G_i^l \backslash \tilde{G}_i^l \cup \hat{G}_i^l$. We set $G_i^1 = G_i$. Each delete/replace combination can be split into the following four basic operations:

- **shift-left**: Two consecutive duty pieces of the same block $n_i = (t_1, \ldots, t_l)$, $\tilde{n}_i = (t_{l+1}, \ldots, \tilde{t}_k) \in \tilde{N}_i$ are replaced by two duty pieces $n_{i+1} = (t_1, \ldots, t_{l+1})$, $\tilde{n}_{i+1} = (t_{l+2}, \ldots, t_k) \in \tilde{N}_{i+1}$, transferring the timetable task $t_{l+1} \in \mathcal{T}$ from the second to the first duty piece.
- **shift-right**: Similar to shift-left, transferring a task from the first to the second duty piece.
- **cut**: Cut a single duty piece $n_i = (t_1, \ldots, t_l)$ into two pieces $n_{i+1} = (t_1, \ldots, t_k)$, $\tilde{n}_{i+1} = (t_{k+1}, \ldots, t_l) \in \tilde{N}_{i+1}$.
- **merge**: Combine two adjacent duty pieces of the same block $n_i = (t_1, \ldots, t_l)$, $\tilde{n}_i = (t_{l+1}, \ldots, \tilde{t}_k) \in \tilde{N}_i$ to form a single duty piece $n_{i+1} = (t_1, \ldots, t_k) \in \tilde{N}_{i+1}$.

The operations in a chain are chosen randomly to secure a diversification of the search. Further each duty piece $n \in \tilde{N}_i$ may only be modified once in each ejection chain to propagate a search over the entire duty schedule. Finally the objective for evaluation of a transmission from $G_i^l = (V_i^l, A_i^l)$ to $G_i^{l+1} = (V_i^{l+1}, A_i^{l+1})$ is

$$F'(S_i) = |A_i^{l+1}| - |A_i^l|$$

which favors the cut move and thus increases the number of arcs and possible duties.
A feasible solution $C_{i+1} \in \mathcal{C}^{\mathcal{D}}$ is obtained by solving the maximum cardinality matching problem on $G_{i+1}^{l^*}$ where l^* represents the level of the chain where the best value for $F'(S_i)$ has been found. $G_{i+1}^{l^*}$ also identifies the digraph which is further modified in the next iteration. The matching problem is solved using a greedy procedure as follows. In each iteration the duty piece with the lowest number of possible links which is not yet contained in a duty, is linked together with another duty piece generating a feasible two piece duty. This feasible duty

exists, because the corresponding arc has been contained in the digraph G_i. These duties together with the remaining trippers, see [31], form the next feasible duty schedule.
A tabu list is kept of all reversing operations, i.e. if a cut operation is performed, the reversing merge operation is stored and vice versa. Each reverse move is kept in the tabu list for Θ iterations, where Θ is chosen randomly from within an interval for each move.
The authors do not state when a current feasible duty schedule is satisfying and thus the loop in lines 4-12 exits and the best solution is returned. One method would be to indicate an upper bound on the number of iterations of run ejection.
[12] states computational results for the run ejection algorithm. The tested instances contained up to 26 blocks and up to 718 timetable tasks. Compared to manual schedules, the run ejection algorithm reduced the number of duties for all instances.

2.6.2 HACS - Heuristic for Automatic Crew Scheduling

HACS is a driver scheduling heuristic based on tabu search developed by the authors of TRACS II. First, an initial, not necessarily feasible, duty schedule is constructed in a simple manner. The schedule is modified using a tabu search routine to establish feasibility and to minimize the cost. Below we take a closer look at the model and at the tailored tabu search method used in this heuristic. However we do not discuss the tabu tenure and the memory scheme of HACS here, but remark that both issues are important for efficient runtimes of HACS. A detailed description is given in [45]. Compared to TRACS II, HACS reports slightly weaker solutions, but much better runtimes.
Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph. As mentioned, HACS uses infeasible duties and infeasible duty schedules while processing. Thus we define $\mathcal{P}$ not as the set of all duties but as the set of all $[s, e] - paths$ in $\mathcal{D}$. Furthermore, we define the set of all feasible duty schedules $\mathcal{C}^{\mathcal{D}}$ based on this modified set $\mathcal{P}$. Thus, a duty schedule is a set of $[s, e] - paths$ covering each timetable task exactly once, but the duties must not respect any duty resource constraints.
For any duty $p_i \in \mathcal{P}$, let $\tilde{\mathcal{N}}^{p_i} = \{n_{ij} | j = 1, \dots m\}$ be the set of contained pieces of work. i.e. the contained maximal duty pieces.. Further, let $L_{p_i} = \{l_{ij} | j = 1, \dots m + 1\} \subset \mathcal{A}$ identify the type 3 and type 4 links contained in p_i [3]. The links l_{i1} and l_{im+1} represent type 4 links, i.e. the sign on and sign off activity.
An example of three duties $p_1, p_2, p_3 \in \mathcal{P}$ covering two blocks $m, \tilde{m} \in \mathcal{F}(\mathcal{D})$ is shown in Figure 2.4.
Associated with each duty $p \in \mathcal{P}$ are functions $f(p)$ representing the penalty and $g(p)$ representing its cost. If $p \in \mathcal{P}$ is indeed a duty, $f(p) = 0$.

[3] $|L_{p_i}| = |\tilde{\mathcal{N}}^{p_i}| + 1$.

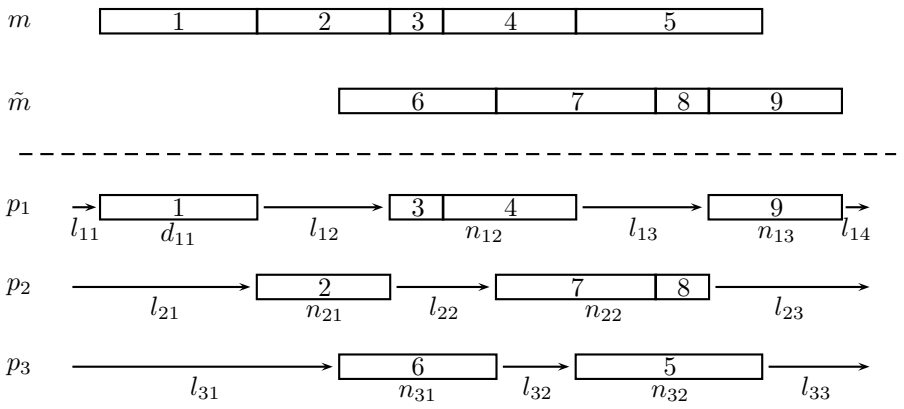

Figure 2.4: Example of 3 duties covering 2 blocks

A duty p with $f(p) = 0$ satisfies all duty resource constraints and thus is feasible with respect to some duty type $k \in \mathcal{K}$. For simplicity, the penalty and cost of a duty p are stored on all its links $l \in L_p$, i.e. $f(l) = f(p)$ and $g(l) = g(p)$.
The aim of HACS is to find a duty schedule $C \in \mathcal{C}^{\mathcal{D}}$ with $f(C) = \sum_{p \in C} f(p) = 0$ [4] and with minimal cost $g(C) = \sum_{p \in C} g(p)$ between all such duty schedules. HACS is outlined in Algorithm 2.8.
HACS does not aim for a good initial solution and instead a simple and quick method is used to obtain this first duty schedule. Let $N = V/(T - M)$ where $V = \sum_{t \in \mathcal{T}} \bar{e}_t - \bar{s}_t$ indicates the total duration of all timetable tasks and T and M identify the maximum spread time and the minimum meal break time of a single duty. N is a lower bound on the target number of duties. Further $P = 2N$ and $D = V/P$ are assumptions for the number and average duration of duty pieces in a duty schedule.
All timetable tasks of each block $m \in \mathcal{F}(\mathcal{D})$ are grouped to form duty pieces of total duration D, resulting in at least P such pieces. P is increased if necessary. Finally the duty pieces are paired with respect to their starting time, i.e. the starting time of the first piece has to be lower than the starting time of the second piece and the corresponding duties containing only these pieces are added to the initial duty schedule. If the number of pieces is odd, the last remaining duty piece will be included in the initial schedule as a duty with a single piece of work. Duties may be time infeasible and may have violated constraints.
The multi-neighborhood of the HACS tabu search procedure is identified indirectly by defining the following four swapping moves. Each of the following moves modifies two duties $p, \tilde{p}$ in the current duty schedule C_i to obtain a new schedule C_{i+1}.

- **swapping links:** Two links $l_1 = (t, \tilde{t}) \in L_p$ and $l_2 = (t_1, t_2) \in L_{\tilde{p}}$ are removed and replaced by the crossover counterparts $\tilde{l}_1 = (t, t_2)$

[4] C is feasible because $\mathcal{W}$ is empty.

Algorithm 2.8 (HACS)

Require: Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$
1: Construct an initial (infeasible) duty schedule $C \in \mathcal{C}^{\mathcal{D}}$.
2: $P \leftarrow |C|$
3: **while** TRUE **do**
4: **while** $f(C) < X$ **do**
5: $X \leftarrow f(C)$
6: $C \leftarrow$ Perform tabu swapping move to modify C
7: **end while**
8: $\tilde{C} \leftarrow C$
9: **while** $g(C) < X$ **do**
10: $X \leftarrow g(C)$
11: $C \leftarrow$ Perform tabu swapping move reducing the cost $g(C)$
12: **end while**
13: **if** $C \neq \tilde{C}$ **then**
14: NEXT;
15: **else**
16: **if** $f(C) = 0$ **then**
17: BREAK;
18: **else**
19: $P \leftarrow P + 1$
20: $C \leftarrow C \cup \tilde{p}$ where $\tilde{p}$ is a new duty minimizing the penalty
21: **end if**
22: **end if**
23: **end while**
24: **return** $C \in \mathcal{C}^{\mathcal{D}}$

and $\tilde{l}_2 = (t_1, \tilde{t})$, see Figure 2.5.

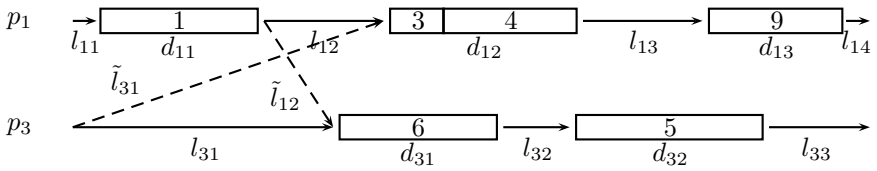

Figure 2.5: Swapping links $l_{12} \in L_{p_1}$ and $l_{31} \in L_{p_3}$

- **swapping pieces**: Two pieces are swapped between two duties, i.e. two swapping links moves are performed simultaneously.
- **inserting piece**: A piece is taken from a duty p and inserted in a duty $\tilde{p}$ by replacing $(t_1, t_2) \in L_p$, $(t_3, t_4) \in L_p$ and $(t_5, t_6) \in L_{\tilde{p}}$ with the links (t_1, t_4) (t_5, t_2) and (t_3, t_6), see Figure 2.6.
- **modifying pieces**: The pieces as built in the initial schedule can only be modified by this move. Assume that pieces n_{ik} and n_{jl} are succeeding pieces in a block $m \in \mathcal{F}(\mathcal{D})$ and timetable task t_j is the last task contained in piece n_{ik}, then t_j can be removed from n_{ik} and inserted in n_{jl}. The same applies if t_j is the first task

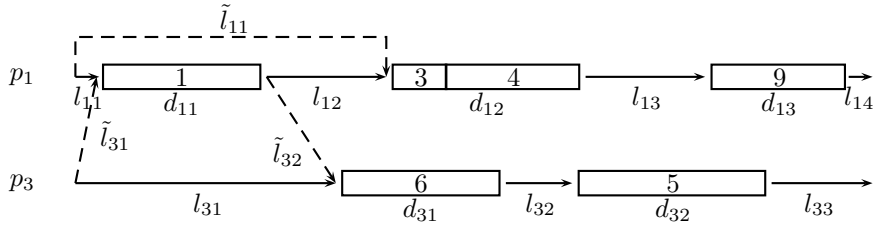

Figure 2.6: Inserting n_{11} into D_3

contained in n_{jl}. See Figure 2.7.

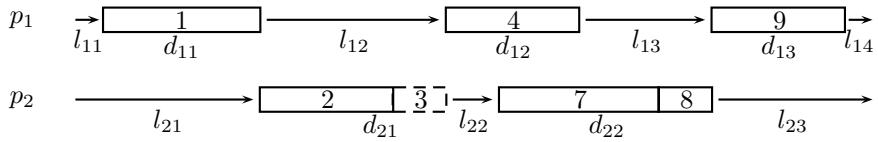

Figure 2.7: Modified n_1 and n_2 after recutting block m_1

The value of any move is defined as either $f(S) - f(\tilde{S})$ or $g(S) - g(\tilde{S})$ depending on the objective of the current minimization procedure. Note that $f(S) - f(\tilde{S}) = f(l_{1i}) + f(d_{2j}) - f(\tilde{l}_{1i}) - f(\tilde{l}_{2j}))$ and $g(S) - g(\tilde{S}) = g(l_{1i}) + g(d_{2j} - g(\tilde{l}_{1i}) - g(\tilde{l}_{2j}))$ respectively.
Using these swapping moves, first the feasibility is reduced as far as possible. Afterwards swapping operations are used to reduce the cost as far as possible. If any change has been made during the cost reduction, again, $f(C)$ is reduced. Otherwise, if the current schedule is feasible, it is returned. If and only if no feasible schedule is found with a specific number of duties $P \in \mathbb{N}$, P is increased, see Algorithm 2.8. A good duty, minimzing the penalty, is added to the procedure and the process is repeated.
Computational data for HACS is listed in [45]. HACS has been used to solve instances with up to 999.101 potential duties and about 860 timetable tasks. For instances of this size HACS provided solutions near the best known solutions.

2.7 Conclusions

In this chapter we discussed some duty scheduling heuristics from three different mathematical fields. Many different ideas have been implemented by various authors. However, only a few of the heuristics presented in this thesis can manage the whole complexity of the duty scheduling problem. Often a lot of details of the problem are discarded or the number of tasks and pieces of work contained in a duty is restricted. In most cases the set of duties $\mathcal{P}$ is reduced. Further many of the heuristics do not support base constraints, i.e. the set $\mathcal{W}$ has to be

empty.
It is hard to compare the heuristics from a mathematical point of view because most authors do not provide computational data and exact descriptions of the evaluated instances. Still we summarize some basic results.
HOT II and similar older heuristics such as HASTUS-Micro, see chapter 3, reduce the size of the Duty Scheduling Graph as far as possible. Due to the lack of computational power at that time, such a reduction was the only way to solve the problem. Related heuristics are still used by some companies because with modern computing they provide fast and often satisfying solutions. An example is the Treiber-Heuristic described in the introdution of this thesis.
Static column generation approaches such as IMPACS and TRACS II focus on a reduction of $\mathcal{P}$. The set covering formulation is then solved on the subset. For the methods mentioned in this thesis, no computational data is available. Dynamic column generation approaches, see section 2.4, are likely to be superior as more duties can be considered.
The authors of meta heuristics claim that genetic algorithms and tabu search are efficient in generating good duty schedules. Still most of the presented results only consider small duty scheduling instances and do not support base constraints. Hopefully future research enhances these methods.
State-of-the-art solvers like DS-Opt and CREW-Opt are able to solve duty scheduling instances with more than 10.000 timetable tasks, see Appendix A. Furthermore by solving the set partitioning formulation (**DSP-SPP**) the whole complexity of the problem is considered. For today's large duty scheduling instances however, column generation procedures are time consuming.
One attempt to speed up the column generation process is done with the dynamic constraint aggregation algorithm described in section 2.4.1. The authors claim huge runtime savings. However no detailed description of this algorithm is available.
Further research in duty scheduling should focus on methods to speed up the column generation process.
In chapter 4 we use a problem reduction approach based on ideas collected from some of the heuristics described in this section. The aim is to construct a duty scheduling heuristic able to solve large instances faster then column generation solvers but supporting the whole complexity of the problem.

3. HASTUS - A General Approach to Duty Scheduling

Abstract: We describe the duty scheduling heuristic HASTUS-Micro, which uses the two phase solution approach introduced in section 2.2. HASTUS-Micro provides the basic idea for our heuristic FAST presented in the next chapter.

The first version of HASTUS, a package containing both vehicle and duty scheduling modules, was developed in the early 80's. For 25 years its algorithms have been enhanced and improved, features have been added and flexibility has been increased. Today HASTUS is one of the world's leading scheduling packages, used by more than 250 scheduling companies from all over the world, see [1].
In this chapter we discuss HASTUS-Micro, a duty scheduling heuristic, which has been part of the early HASTUS package. It is described in [35] and [41]. Due to increased computational power and further mathematical research, HASTUS-Micro has been substituted by the column generation method CREW-Opt in later versions of HASTUS. Still, it provides the general idea for our new duty scheduling heuristic FAST. We take a closer look at HASTUS-Micro in this chapter.

3.1 The Approach

HASTUS-Micro follows the basic idea of splitting a Duty Scheduling Problem $\min\{C(C) \mid C \in \mathcal{C}^{\mathcal{D}}\}$ into two subproblems. First the size of the Duty Scheduling Graph $\mathcal{D}$ is reduced. For each block $m \in \mathcal{F}(\mathcal{D})$ a duty piece partition $n_1, \ldots, n_k \in \mathcal{N}^m$ is determined. All links $l \in \mathcal{A} \setminus \mathcal{A}_1$ connecting other tasks to inner tasks of these duty pieces are discarded. As a result some of the type 2 links of the block m turn into type 1 links. After this reduction, the Duty Scheduling Problem $\min\{C(C) \mid C \in \mathcal{C}^{\tilde{\mathcal{D}}}\}$ is solved, where $\tilde{\mathcal{D}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{A}})$ indicates the reduced Duty Scheduling Graph.
The difficulty arising from the procedure described above is to select a good duty piece partition for each block. To perform a smart partition, all details of the Duty Scheduling Graph are dropped and a relaxation of the Duty Scheduling Problem, called HASTUS-Macro, is solved on this reduced graph. The pieces of work used in the solution of HASTUS-Macro are then used as a blueprint for a good duty schedule. The blocks of the original problem are partitioned with respect to these

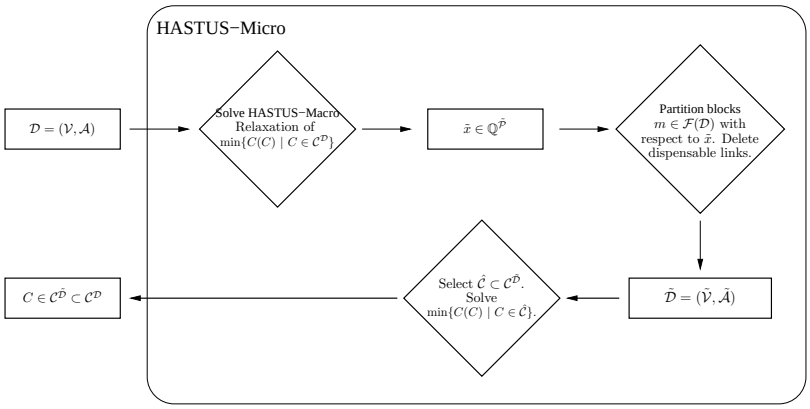

Figure 3.1: HASTUS-Micro

pieces of work[1]. Figure 3.1 gives an overview on HASTUS-Micro. The next section describes HASTUS-Macro as a standalone procedure. In section 3.3 we discuss HASTUS-Micro in more detail.

3.2 HASTUS-Macro

During negotiations with labor unions regarding new service rules for drivers, cost-intense decisions must be made by transportation companies. Under these circumstances the companies need information on the economical impact of possible rule changes.
In this context HASTUS-Macro was developed as a tool to evaluate changing duty constraints and global restrictions for duty schedules.
The idea is to solve a time effective relaxation of the Duty Scheduling Problem on a modified Duty Scheduling Graph multiple times regarding different duty and base resource limits $\hat{u}$ and $\hat{w}$. Thus the cost of different duty schedules can be compared providing valuable arguments during negotiations.
Given a Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$, HASTUS-Macro first rounds all tasks with respect to a value $\bar{r} \in \mathbb{N}$. Then a linear model is defined, which is a modification of $\min\{C(C) \mid C \in \mathcal{C}^{\mathcal{D}}\}$. This modification ignores the position of all tasks and thus reduces the size of the problem. The model can be solved in a very short time.

3.2.1 Rounding a Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$

We now discuss the rounding used in HASTUS-Macro. Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph. In this section we assume that $\mathcal{S} = \emptyset$. Let $\bar{r} \in \mathbb{N}$ be a value satisfying $\frac{86400}{\bar{r}} \in \mathbb{N}$ and set $k = \frac{86400}{\bar{r}} - 1$. The set

$$\mathcal{X} = \{[0, 1 \cdot \bar{r}], [1 \cdot \bar{r}, 2 \cdot \bar{r}], \ldots, [k \cdot \bar{r}, 86400]\} \tag{3.2.1}$$

is a partition of the 24-hour day into intervals of $\bar{r}$ seconds, the set

$$\tilde{\mathcal{X}} = \{0, 1 \cdot \bar{r}, 2 \cdot \bar{r}, \ldots, k \cdot \bar{r}, 86400\} \tag{3.2.2}$$

[1]Pieces of work are maximal duty pieces contained in a duty, see section 1.2.

contains all relevant second values.
A rounding of $\mathcal{D}$ with respect to $\bar{r}$ is constructed as follows. For all tasks $t \in \mathcal{V}$, the start time $\bar{s}_t$ and the end time $\bar{e}_t$ are rounded to the closest value contained in $\tilde{\mathcal{X}}$. Thus each task covers some succeeding intervals $[i, j] \in \mathcal{X}$. A task t may cover zero intervals, i.e. $\bar{s}_t = \bar{e}_t$ after the rounding. The duty and base resource consumption of tasks and links is adjusted to match the new start and end times. Let $\tilde{\mathcal{D}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{A}})$ be the resulting Duty Scheduling Graph, then $|\mathcal{V}| = |\tilde{\mathcal{V}}|$ and $|\mathcal{A}| = |\tilde{\mathcal{A}}|$. However, the set of possible second values for start and end times has been reduced to $|\tilde{\mathcal{X}}|$. This fact is exploited by the HASTUS-Macro relaxation.
Figure 3.2 displays a part of a Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ and the corresponding rounded Duty Scheduling Graph $\tilde{\mathcal{D}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{A}})$. The set of original timetable tasks $t_1, \ldots t_{10} \in \mathcal{T} = \mathcal{V} \setminus \mathcal{I}$ has been rounded with respect to a rounding value $\bar{r} = 600$. Clearly $\bar{s}_{\tilde{t}_2} = \bar{e}_{\tilde{t}_2} = \bar{s}_{\tilde{t}_{10}} = \bar{e}_{\tilde{t}_{10}} = 1800$ and $\bar{s}_{\tilde{t}_8} = \bar{e}_{\tilde{t}_8} = 1200$.

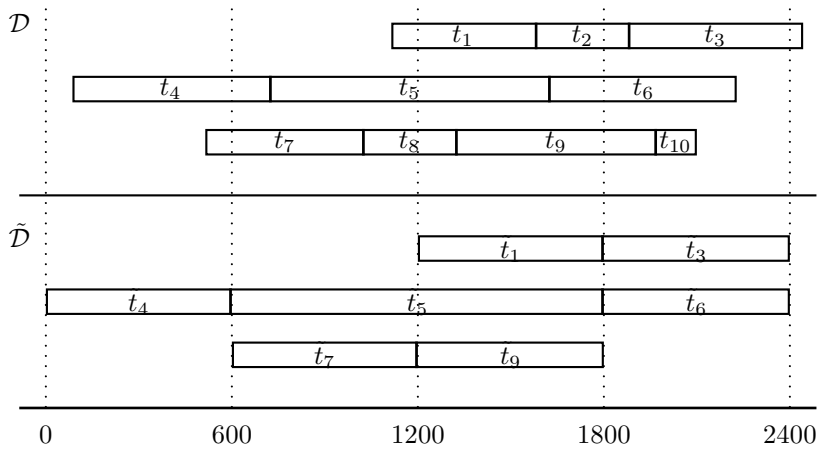

Figure 3.2: HASTUS-Macro: Modification of $\mathcal{D} = (\mathcal{V}, \mathcal{A})$

3.2.2 The HASTUS-Macro Relaxation

Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph with $\mathcal{S} = \emptyset$ and let $\bar{r} \in \mathbb{N}$ be a rounding value. Further let $\tilde{\mathcal{D}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{A}})$ be the rounded Duty Scheduling Graph and let $\mathcal{P}^{\tilde{\mathcal{D}}}$ be the set of corresponding duties.
For each interval $[i, j] \in \mathcal{X}$, we define the number of covering timetable tasks $z^{[i,j]} \in \mathbb{N}$ as

$$z^{[i,j]} = \left|\{t \in \tilde{\mathcal{T}} | \bar{s}_t \leq i \wedge \bar{e}_t \geq j\}\right| . \tag{3.2.3}$$

As mentioned, the HASTUS-Macro relaxation discards the requirement that each timetable task $t \in \tilde{\mathcal{T}}$ is covered directly by a duty $p \in \mathcal{P}^{\tilde{\mathcal{D}}}$, i.e. the position of the tasks. However each interval $[i, j] \in \mathcal{X}$ must be

covered by at least $z^{[i,j]}$ duties. Thus in a relaxed duty schedule the number of duties matches the number of covering timetable tasks for each interval, but the duties are not assigned to timetable tasks.
To model this requirement, we introduce another Duty Scheduling Graph $\hat{\mathcal{D}} = (\hat{\mathcal{V}}, \hat{\mathcal{A}})$ which contains a single timetable task $t^{[i,j]}$ for each interval $[i,j] \in \mathcal{X}$. Timetable tasks $t^{[i,j]}$ and $t^{[k,l]}$ are connected by a link if they either represent succeeding intervals, i.e. $j = k$, or if an original link $(t_1, t_2) \in \tilde{\mathcal{A}}$ exists with $\bar{e}_{t_1} = j \wedge \bar{s}_{t_2} = k$.
Let $\mathcal{P}^{\hat{\mathcal{D}}}$ be the set of duties in $\hat{\mathcal{D}}$. HASTUS-Macro only considers duties $p \in \mathcal{P}^{\hat{\mathcal{D}}}$ which contain no more than three pieces of work, i.e. $|\tilde{\mathcal{N}}^p| \leq 3$. Such duties may use no more than two type 3 links. Let

$$\hat{\mathcal{P}} = \{p \in \mathcal{P}^{\hat{\mathcal{D}}} \mid |l \in \hat{\mathcal{A}}_3 \cap A(p)| \leq 2\} .$$

indicate the subset of all such duties. To speed up runtime, HASTUS-Macro only considers duties $p \in \hat{\mathcal{P}}$.
Finally, we do not use binary decision variables to indicate if a duty is part of the resulting duty schedule. Instead, non-negative variables $x_p \in \mathbb{Q}$ are introduced for each duty $p \in \hat{\mathcal{P}}$.
Model 3.1 describes the HASTUS-Macro relaxation of a Duty Scheduling Problem $\min\{C(C) \mid C \in \mathcal{C}^{\mathcal{D}}\}$ regarding a set of intervals $\mathcal{X}$.

Model 3.1 (**HASTUS-Macro**)

$$\min \quad \sum_{p \in \hat{\mathcal{P}}} c_p \cdot x_p$$

$$\text{s.t.} \quad \sum_{\{p \in \hat{\mathcal{P}} \mid t^{[i,j]} \in p\}} x_p \geq z^{[i,j]} \quad \forall\, [i,j] \in \mathcal{X} \tag{3.2.4}$$

$$\sum_{p \in \hat{\mathcal{P}}} (w_p)_b x_p \leq \hat{w}_b \quad \forall\, b \in \mathcal{W} \tag{3.2.5}$$

$$\sum_{p \in P^{ij}} x_p \geq f^{ij} \quad \forall\, i\ \forall\, j :\ f^{ij} > 0 \wedge j - i \leq \hat{f} \tag{3.2.6}$$

$$\sum_{p \in P^{i}} x_p \geq f^{i} \quad \forall\, i\ \exists j : f^{ij} > 0 \tag{3.2.7}$$

$$x_p \geq 0 \quad \forall\, p \in \hat{\mathcal{P}} \tag{3.2.8}$$

Constraint set 3.2.4 ensures that each timetable task $t^{[i,j]} \in \hat{\mathcal{T}}$ is covered by as many duties as indicated by the value $z^{[i,j]}$. Restrictions 3.2.5 ensure that the fractional duty schedule $C = \{p \in \hat{\mathcal{P}} \mid x_p > 0\}$ satisfies all base constraints.
To disallow abnormal duties and to ensure usability of the solution, constraints 3.2.6 and 3.2.7 have been appended to the problem as follows.

Let $\mathcal{F}(\tilde{\mathcal{D}})$ be the set of blocks in the rounded Duty Scheduling Graph $\tilde{\mathcal{D}}$ and let

$$F^{ij} = \{(t_1, \ldots, t_k) \in \mathcal{F}(\tilde{\mathcal{D}}) \mid \bar{s}_{t_1} = i \wedge \bar{e}_{t_k} = j\} \subset \mathcal{F}(\tilde{\mathcal{D}}) \tag{3.2.9}$$

be the subset of all blocks starting at $i \in \tilde{\mathcal{X}}$ and ending at $j \in \tilde{\mathcal{X}}$. Further let $f^{ij} = |F^{ij}|$ indicate the number of blocks contained in F^{ij} and let

$$f^i = \sum_{\{j \in \tilde{\mathcal{X}} | f^{ij} > 0\}} f^{ij} \tag{3.2.10}$$

indicate the number of blocks starting in $i \in \tilde{\mathcal{X}}$.
Given a parameter $\hat{f} \in \mathbb{N}$, a block $m \in F^{ij}$ is called a *short block* if $j - i \leq \hat{f}$. We now introduce two subsets of $\hat{\mathcal{P}}$. The set

$$P^i = \{p \in \hat{\mathcal{P}} | \exists\ (t_k, \ldots, t_l) \in \tilde{\mathcal{N}}^p\ \bar{s}_{t_k} = i\} \subset \hat{\mathcal{P}} \tag{3.2.11}$$

indicates the set of duties containing a pieces of work starting at $i \in \tilde{\mathcal{X}}$. The subset

$$P^{ij} = \{p \in P^i | \exists\ (t_k, \ldots, t_l) \in \tilde{\mathcal{N}}^p\ \bar{s}_{t_k} = i \wedge \bar{e}_{t_l} = j\} \subseteq P^i \tag{3.2.12}$$

contains all duties containing pieces of work also ending at $j \in \tilde{\mathcal{X}}$.
Using the notation introduced above, the constraint set 3.2.6 requires that for any combination $i, j \in \tilde{\mathcal{X}}$ with $j - i \leq \hat{f}$, the fractional schedule contains at least as many pieces of work starting at i and ending at j as short $[i, j]$-blocks are contained in the rounded Duty Scheduling Graph $\tilde{\mathcal{D}}$.
Similarly, constraint set 3.2.7 ensures that for any $i \in \tilde{\mathcal{X}}$ the fractional schedule contains more duty pieces starting in i than blocks that start at i in $\tilde{\mathcal{D}}$.
The objective of HASTUS-Macro is to minimize the total cost of all fractional duties used to cover all intervals with respect to $z^{[i.j]}$.
More general information on HASTUS-Macro is found in [6].

3.3 Hastus-Micro

In the context of HASTUS-Micro, a HASTUS-Macro solution $\hat{x} \in \mathbb{Q}^{\hat{\mathcal{P}}}$ is used as an outline for a good duty schedule in the original problem. Each block is partitioned into a set of duty pieces with respect to $\hat{x}$. The objective is to find a partition that minimizes the total difference between the pieces of work indicated by $\hat{x}$ and the duty pieces used to partition the blocks.
Recall, that HASTUS-Micro can be subdivided into two parts. First, a flow algorithm is used to determine good partitions for the blocks. All redundant links are discarded. Afterwards the Duty Scheduling Problem is solved heuristically regarding the reduced Duty Scheduling Graph.
This procedure is outlined by Algorithm 3.1. All steps are discussed in more detail below.

Algorithm 3.1 (HASTUS-Micro)

Require: $\mathcal{D} = (\mathcal{V}, \mathcal{A})$

1: Solve (**HASTUS-Macro**) to attain a solution $\hat{x} \in \mathbb{Q}^{\hat{\mathcal{P}}}$
2: Use a flow algorithm to partition the blocks into duty pieces with respect to $\hat{x}$
3: Construct duties from the duty pieces using a matching procedure
4: **return** The best solution $C \in \mathcal{C}^{\mathcal{D}}$

Throughout this section let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph and let $\bar{r}$, $\mathcal{X}$, $\tilde{\mathcal{X}}$ and $\tilde{\mathcal{D}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{A}})$ be indicated as in section 3.2. Further let $\hat{x} \in \mathbb{Q}^{\hat{\mathcal{P}}}$ be a fractional solution of the HASTUS-Macro relaxation of the Duty Scheduling Problem $\min\{C(C) \mid C \in \mathcal{C}^{\mathcal{D}}\}$.

3.3.1 Duty Piece Partitions for the Blocks

We now describe the flow algorithm used in HASTUS-Micro to partition the blocks into duty pieces.
The problem to partition a single block into a set of duty pieces can be modeled as a flow problem on a digraph $D = (V, A)$ as follows. Assume that $m = (t_1, \ldots, t_k) \in \mathcal{F}(\mathcal{D})$ is a block consisting of k timetable tasks, $t_i \in \mathcal{T}$. Define a set of vertices

$$V = \{s, e\} \cup \{v_{ij} | t_i, t_j \in m \wedge i + 1 = j\}$$

where s and e identify the start and the end of the block m and $v_{ij} \in V$ identifies a type 1 or type 2 link (t_i, t_j) contained in the block. We define the set of arcs as

$$A = \{(t_1, t_2) | t_1, t_2 \in V \wedge t_1 \neq t_2\} \ ,$$

i.e. any two vertices are connected.
Figure 3.3 displays the relevant digraph $D = (V, A)$ regarding a block $m = (t_1, \ldots, t_4) \in \mathcal{F}(\mathcal{D})$. Each (s, e)-flow in D identifies a partition of m into duty pieces. Each arc used in the flow corresponds to one of the duty pieces contained in the partition. The costs of the arcs identify the costs of the corresponding duty pieces and are chosen with respect to the HASTUS-Macro solution.

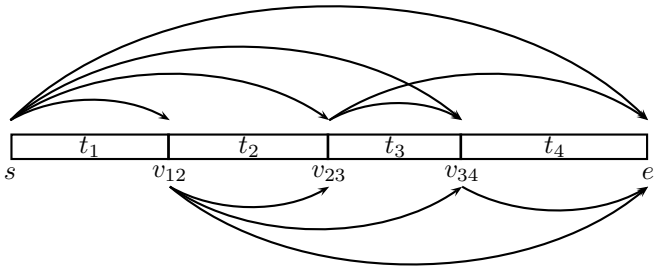

Figure 3.3: Partitioning a block into pieces using a flow algorithm

We now describe a non-linear model to generate a partition of all blocks using a flow algorithm. Each solution corresponds to a simultaneous

partition of all blocks into duty pieces. The optimal solution identifies the duty piece partitions with the minimum difference to the given HASTUS-Macro solution $\hat{x}$.

Let $\mathcal{F}(\tilde{\mathcal{D}})$ be the set of all blocks contained in the rounded Duty Scheduling Graph $\tilde{\mathcal{D}}$ and let

$$I = \{(i,j) \in \mathbb{N}^2 | i,j \in \tilde{\mathcal{X}}\ i \neq j\}$$

be a set containing all possible start-end combinations for duty pieces. Further let P^i and P^{ij} be defined as in 3.2.11 and 3.2.12. Finally, for each original block $\tilde{m} \in \mathcal{F}(\mathcal{D})$ define a set

$$T^{\tilde{m}} = \{i \in \mathbb{N} | \exists\ t \in \tilde{\mathcal{T}} \cap V(\tilde{m}) :\ \bar{s}_t = i \vee \bar{e}_t = i\} \tag{3.3.13}$$

containing the time values of the corresponding rounded block $\tilde{m} \in \mathcal{F}(\tilde{\mathcal{D}})$. Obviously $T^{\tilde{m}} \subset \tilde{\mathcal{X}}$ for all blocks $\tilde{m} \in \mathcal{F}(\tilde{\mathcal{D}})$.

For any original block $\tilde{m} \in \mathcal{F}(\mathcal{D})$ and any $(i,j) \in I$ we introduce binary variables $y_{ij}^m \in \{0,1\}$ to decide if a duty piece starting at i and ending at j is used in the partition of block $\tilde{m}$.

The times $i,j \in \tilde{\mathcal{X}}$ are not the original starting and ending times of the timetable tasks used in the block $\tilde{m}$. Introduce penalty costs $d_{ij}^{\tilde{m}} \in \mathbb{Q}$ to represent the adjustment that has to be made to use a duty piece starting at $i \in \tilde{\mathcal{X}}$ and ending in $j \in \tilde{\mathcal{X}}$ to partition the original block $\tilde{m}$. Set $d_{ij}^{\tilde{m}} = \infty$ if $i \notin T^{\tilde{m}}$ or $j \notin T^{\tilde{m}}$.

Model 3.2 generates a partition of all original blocks into duty pieces with respect to the HASTUS-Macro solution $\hat{x}$.

Model 3.2 (**HASTUS-Micro Flow**)

$$\min \quad \sum_{(i,j)\in I} ((\sum_{p\in P^{ij}} \hat{x}_p - \sum_{m\in\mathcal{F}(\tilde{\mathcal{D}})} y_{ij}^m)^2 \quad + \sum_{m\in\mathcal{F}(\tilde{\mathcal{D}})} d_{ij}^m y_{ij}^m)$$

$$\text{s.t.} \quad \sum_{i\in T^m} y_{ik}^m - \sum_{j\in T^m} y_{kj}^m = \begin{cases} -1 & k = \bar{s}_{t_1} \\ +1 & k = \bar{e}_{t_l} \\ 0 & \text{else} \end{cases} \forall\ m = (t_1..t_l) \in \mathcal{F}(\tilde{\mathcal{D}}) \forall\ k \in T^m \tag{3.3.14}$$

$$y_{ij}^m \in \{0,1\} \qquad \forall\ m \in \mathcal{F}(\tilde{\mathcal{D}}), (i,j) \in I \tag{3.3.15}$$

Constraint set 3.3.14 represent flow constraints for each original block. The objective is a non-linear function. HASTUS-Micro uses a heuristic procedure to solve the model. Assume that all variables y_{ij}^m not associated with a specific block $\tilde{m}$ are fixed. The objective can then be reduced as follows.

$$
\begin{aligned}
& \sum_{(i,j)\in I} (\sum_{p\in P^{ij}} \hat{x}_p - \sum_{m\in\mathcal{F}(\mathcal{D})} y_{ij}^m)^2 + \sum_{(i,j)\in I} \sum_{m\in\mathcal{F}(\mathcal{D})} d_{ij}^m y_{ij}^m \\
= & \sum_{(i,j)\in I} (\sum_{p\in P^{ij}} \hat{x}_p - \sum_{m\neq\tilde{m}} y_{ij}^m - y_{ij}^{\tilde{m}})^2 \\
& + \sum_{(i,j)\in I} \sum_{m\neq\tilde{m}} d_{ij}^m y_{ij}^m + \sum_{(i,j)\in I} d_{ij}^{\tilde{m}} y_{ij}^{\tilde{m}} \\
= & \sum_{(i,j)\in I} ((\sum_{p\in P^{ij}} \hat{x}_p - \sum_{m\neq\tilde{m}} y_{ij}^m)^2 - 2y_{ij}^{\tilde{m}} (\sum_{p\in P^{ij}} \hat{x}_p - \sum_{m\neq\tilde{m}} y_{ij}^m) + (y_{ij}^{\tilde{m}})^2) \\
& + \sum_{(i,j)\in I} \sum_{m\neq\tilde{m}} d_{ij}^m y_{ij}^m + \sum_{(i,j)\in I} d_{ij}^{\tilde{m}} y_{ij}^{\tilde{m}}
\end{aligned}
$$

Because $y_{ij}^{\tilde{m}} \in \{0,1\}$, condition $(y_{ij}^{\tilde{m}})^2 = y_{ij}^{\tilde{m}}$ holds. Removing all fixed parts of the objective we get a new objective

$$\min \sum_{(i,j)\in I} c_{ij}^{\tilde{m}} y_{ij}^{\tilde{m}}, \quad c_{ij}^{\tilde{m}} = -2(\sum_{p\in P^{ij}} \hat{x}_p - \sum_{m\neq\tilde{m}} y_{ij}^m) + 1 + d_{ij}^{\tilde{m}} \ . \quad (3.3.16)$$

Assuming that all other blocks are fixed we can define a flow problem to determine a partition of block $\tilde{m} = (t_1, \ldots, t_l)$, see Model 3.3.

Model 3.3 ($FP^{\tilde{m}}$)

$$\min \quad \sum_{(i,j)\in I} c_{ij}^{\tilde{m}} y_{ij}^{\tilde{m}}$$

$$\text{s.t.} \quad \sum_{i\in T^{\tilde{m}}} y_{ik}^{\tilde{m}} - \sum_{j\in T^{\tilde{m}}} y_{kj}^{\tilde{m}} = \begin{cases} -1 & k = \bar{s}_{t_1} \\ +1 & k = \bar{e}_{t_l} \\ 0 & \text{else} \end{cases} \quad \forall\, k \in T^{\tilde{m}} \qquad (3.3.17)$$

$$y_{ij}^{\tilde{m}} \in \{0,1\} \qquad \forall\, (i,j) \in I \qquad (3.3.18)$$

Model 3.3 is used to formulate a sub-optimal algorithm to solve Model 3.2. Let $K^{max} \in \mathbb{N}$ be an upper bound on the number of improvement iterations and let $\hat{x} \in \mathbb{Q}^{\hat{\mathcal{P}}}$ be a fractional HASTUS-Macro solution. Algorithm 3.2 is a heuristic to generate a good solution for (**HASTUS-Micro Flow**) .

For initialization, any feasible solution of (**HASTUS-Micro Flow**) is used. In each iteration of the loop in lines 4-16 a sub-optimal solution is generated. Each block $m \in \mathcal{F}(\mathcal{D})$ is partitioned into the optimal set of duty pieces assuming that all other block partitions are fixed. The loop exits if no improvement has been made or if the upper bound on the number of iterations K^{max} is reached.

Algorithm 3.2 (MINSG)

Require: $\mathcal{D} = (\mathcal{V}, \mathcal{A})$, $\hat{x} \in \mathbb{Q}^{\hat{P}}$, $K^{max} \in \mathbb{N}$
/* Initial Solution */
$k \leftarrow 1$
$\tilde{y}_{ij}^m \leftarrow$ Any Solution of (**HASTUS-Micro Flow**)
$Z_0 \leftarrow \sum_{(i,j)\in I}((\sum_{p\in P^{ij}} \hat{x}_p - \sum_{m\in\mathcal{F}(\mathcal{D})} \tilde{y}_{ij}^m)^2 + \sum_{m\in\mathcal{F}(\mathcal{D})} d_{ij}^m \tilde{y}_{ij}^m)$
/* Improve Solution */
while $k < K^{max}$ **do**
  $k \leftarrow k + 1$
  for $m \in \mathcal{F}(\mathcal{D})$ **do**
    $y_{ij}^m \leftarrow$ Solve (FP^m) regarding $\hat{x}$ and $\tilde{y}_{ij}^{\tilde{m}}$
  end for
  for $m \in \mathcal{F}(\mathcal{D})$ **do**
    $\tilde{y}_{ij}^m \leftarrow y_{ij}^m \forall (i,j) \in I$
  end for
  $Z_k \leftarrow \sum_{(i,j)\in I}((\sum_{p\in P^{ij}} \hat{x}_p - \sum_{m\in\mathcal{F}(\mathcal{D})} \tilde{y}_{ij}^m)^2 + \sum_{m\in\mathcal{F}(\mathcal{D})} d_{ij}^m \tilde{y}_{ij}^m)$
  if $Z_k = Z_{k-1}$ **then**
    BREAK
  end if
end while
return $\tilde{y}_{ij}^m$

Let $\{\tilde{y}_{ij}^m \mid (i,j) \in I, m \in \mathcal{F}(\mathcal{D})\}$ be the solution returned by 3.2. For each block $m \in \mathcal{F}(\mathcal{D})$, the variables $\{\tilde{y}_{ij}^{\tilde{m}} \mid (i,j) \in I\}$ define a partition of m into duty pieces. Assume that $\tilde{m} = (t_1, \ldots, t_h, \ldots, t_n) \in \mathcal{F}(\mathcal{D})$ is a block, that $\tilde{y}_{ij}^{\tilde{m}} = y_{jv}^{\tilde{m}} = 1$ for fixed $i, j, v \in \tilde{\mathcal{X}}$ and that $\tilde{y}_{kl}^{\tilde{m}} = 0 \; \forall \; k, l \in \tilde{\mathcal{X}} \; (k,l) \neq (i,j), (j,v)$. Further, assume that $\bar{s}_{t_1}$ has been rounded to i, $\bar{e}_{t_h}$ has been rounded to j and $\bar{e}_{t_n}$ has been rounded to v. Then the block $\tilde{m}$ partitions into the two duty pieces $(t_1 \ldots, t_h) \in \mathcal{N}^{\tilde{m}}$ and $(t_{h+1}, \ldots, t_n) \in \mathcal{N}^{\tilde{m}}$. Let

$$L^{\tilde{m}} = \{(t, \tilde{t}) \in \mathcal{A} \setminus \mathcal{A}_2 | (t \in \tilde{m} \wedge t \neq t_h \wedge t \neq t_n) \vee (\tilde{t} \in \tilde{m} \wedge \tilde{t} \neq t_1 \wedge \tilde{t} \neq t_{h+1})\} \quad (3.3.19)$$

be the set of links connecting to inner tasks of these pieces of block $\tilde{m}$. Each link $l \in L^{\tilde{m}}$ is redundant and is removed.
Thus the set of links $\mathcal{A}$ in the reduced original problem is

$$\tilde{\mathcal{A}} = \mathcal{A} \setminus \bigcup_{m\in\mathcal{F}(\mathcal{D})} L^m \; . \quad (3.3.20)$$

Assume that $\tilde{\mathcal{D}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{A}})$ is the reduced Duty Scheduling Graph, where $\tilde{\mathcal{V}} = \mathcal{V}$, then we indicate the reduced Duty Scheduling Problem by $\min\{C(C) \mid C \in \mathcal{C}^{\tilde{\mathcal{D}}}\}$.

3.3.2 Solving the Reduced Problem

Let $\tilde{\mathcal{D}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{A}})$ be the reduced Duty Scheduling Graph obtained after the first phase of HASTUS-Micro. Further let $\min\{C(C) \mid C \in \mathcal{C}^{\tilde{\mathcal{D}}}\}$ be the reduced Duty Scheduling Problem.

In [35] the authors propose a maximum-weighted-matching formulation and an assignment procedure to solve the reduced Duty Scheduling Problem $\min\{C(C) \mid C \in \mathcal{C}^{\hat{\mathcal{D}}}\}$. In both cases only duties with at most two pieces of work are considered, i.e. all duties contained in the set

$$\hat{\mathcal{P}} = \{p \in \mathcal{P}^{\hat{\mathcal{D}}} \mid |l \in \hat{\mathcal{A}}_3 \cap V(p)| \leq 1\}$$

This is an enormous restriction and therefore we do not discuss the exact procedure here. A general description can be found in [35].
The authors do not list computational results for HASTUS-Micro.

4. FAST - A Macro-Based Heuristic for Duty Scheduling

Abstract: We dicuss the new heuristic FAST which is based upon a two phase approach. In the first phase a relaxation of the Duty Scheduling Problem is solved. In the second phase the original problem is reduced as links are removed according to the obtained relaxed solution. Finally the reduced problem is solved by DS-Opt.

The aim of this thesis was to construct a new duty scheduling heuristic to tackle large instances of the Duty Scheduling Problem and generate good feasible duty schedules in a short time. After facing many difficulties, the new heuristic FAST has been developed. FAST can be understood as a macro-based heuristic, relaxing all details of a Duty Scheduling Problem while maintaing all essential structures.
The approach used in FAST is similar to the HASTUS approach described in the previous chapter. In a first phase the Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ is modified and a relaxed version of the Duty Scheduling Problem is solved on this modified graph. The solution obtained is then used to reduce the size of the original Duty Scheduling Graph to a critical extract such that the Duty Scheduling Problem $\min\{C(C) \mid C \in \mathcal{C}^{\mathcal{D}}\}$ can be solved much faster.
In the context of this work, FAST has been implemented and has achieved very satisfying results. Solutions for all tested instances have been compared to solutions generated by the approximation solver DS-Opt, stating huge runtime savings while almost maintaining solution quality.
We now discuss FAST in detail, while computational results and statistical data are listed in the next chapter and in Appendix A.

4.1 Outline

In this section we outline the two phase approach of FAST. Detailed descriptions of the single steps are discussed in the later sections of this chapter.
Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph. For large real world instances the number of links is extremely large, i.e. $|\mathcal{A}| > 10^6$. The number of possible duties which can be defined on a Duty Scheduling Graph with so many links usually exceeds many million. Thus the number of feasible duty schedules is also rather enormous, $|\mathcal{C}^{\mathcal{D}}| \gg 10^9$.

Given so many possibilities, the generation of near optimal solutions using current solvers like DS-Opt can take several hours or even days. The basic idea of FAST is, to first reduce the size of $\mathcal{D}$ to a very small but critical fraction allowing to generate much faster solutions of similar quality using DS-Opt. For each block $m \in \mathcal{F}(\mathcal{D})$, a duty piece partition is chosen and all links and supplementary tasks connecting to the inner tasks of these duty pieces are discarded.
As mentioned earlier, such a block partition to reduce the size of the problem is further backed by the fact that good duty schedules usually contain few type 3 links and therefore most of the type 3 links and supplementary tasks contained in the problem are dispensable.
Let $\tilde{\mathcal{D}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{A}})$ be the resulting Duty Scheduling Graph, then $\tilde{\mathcal{T}} = \mathcal{T}$ but $\tilde{\mathcal{S}} \subset \mathcal{S}$ and $\tilde{\mathcal{A}} \subset \mathcal{A}$. Therefore, $\mathcal{C}^{\tilde{\mathcal{D}}} \subset \mathcal{C}^{\mathcal{D}}$ and the Duty Scheduling Problem $\min\{C(C) \mid C \in \mathcal{C}^{\tilde{\mathcal{D}}}\}$ is a reduction of the original problem.
Unfortunately, the number of possible partitions into duty pieces is quite large for each block. Further, it is hard to identify critical type 3 links which are important to build efficient duties and should be retained in the problem. Clearly we need more information on the problem structure to perform a valuable partition of the blocks into duty pieces. To gain such information, a relaxed version of the problem, maintaining all characteristics while disposing of all details, is constructed and solved. Its solution serves as a delineation of a good duty schedule and all original blocks are partitioned into duty pieces with respect to this relaxed solution.
Figure 4.1 sketches the two phases of FAST. In phase one, a modified Duty Scheduling Graph $\mathcal{D}^{\bar{r}}_{pos}$ is constructed by rounding all timetable tasks with respect to a value $\bar{r}$. On this modified graph, a relaxation of the set covering formulation of the Duty Scheduling Problem, see Model 1.5 is solved. This relaxation does not require a direct covering of the timetable tasks but a multiple covering of a set of periods identified by $\bar{r}$.
The relaxed solution is used in phase two to reduce the size of the original Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ as duty piece partitions, see Definition 1.2.3, are selected for each block and all corresponding inner links are discarded. Finally, DS-Opt is used to solve the resulting Duty Scheduling Problem.

4.2 Reducing a Duty Scheduling Graph

We now describe a modification of a Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ which retains all essentials such as peaks and block duration while all details are discarded. This modification can be understood as a reduced Duty Scheduling Graph, which uses a decreased level of complexity and detail to describe the same problem.
First, a rounding parameter $\bar{r} \in \mathbb{N}$ satisfying $\frac{86400}{\bar{r}} \in \mathbb{N}$ is introduced and for any task $t \in \mathcal{V}$ the corresponding start and end times $\bar{s}_t, \bar{e}_t$

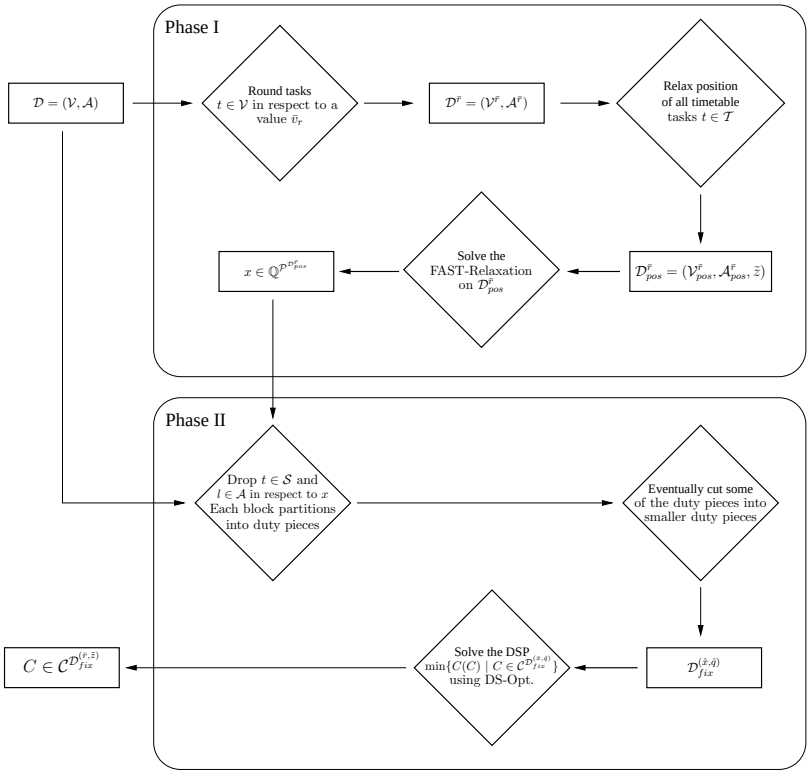

Figure 4.1: The FAST-Heuristic

are rounded with respect to $\bar{r}$. Any timetable task $t \in \mathcal{T}$ with $\bar{s}_t = \bar{e}_t$ after the rounding operation, is deleted along with all adjacent links, reducing the size of the graph $\mathcal{D}$. Thus after the rounding we can assume

$$\forall\, t \in \mathcal{T} : \bar{s}_t = i \cdot \bar{r} \wedge \bar{e}_t = j \cdot \bar{r} \wedge i \neq j\ . \tag{4.2.1}$$

The composition of blocks is maintained by adding new links whenever a timetable task in a block is deleted. If $t_i \in \mathcal{T}$ is deleted and

$$m = (t_1, \ldots, t_{i-1}, t_i, t_{i+1}, \ldots, t_n) \in \mathcal{F}(\mathcal{D})$$

is the block containing t_i then a new link $\tilde{l} = (t_{i-1}, t_{i+1})$ is added to preserve the block m. All links $\tilde{l}$ added in this manner are either type 1 or type 2 links. The exact construction of the rounding is described in section 4.2.1.
Further, the parameter $\bar{r}$ defines intervals

$$[0, 1 \cdot \bar{r}], [1 \cdot \bar{r}, 2 \cdot \bar{r}], \ldots, [k \cdot \bar{r}, 86400]$$

of length $\bar{r}$, where $k = \frac{86400}{\bar{r}} - 1$. The intervals are a partitioning of the 86400-seconds day. Note that $\bar{r}$ must be chosen as a divisor of 86400 in order to define intervals of equal length. Let

$$T^{[i,j]} = \{t \in \mathcal{T} \ \mid\ \bar{s}_t \leq i \wedge \bar{e}_t \geq j\} \tag{4.2.2}$$

indicate the set of all timetable tasks corresponding to an interval $[i, j]$.

For each interval $[i,j]$ with $|T^{[i,j]}| > 0$ the modified Duty Scheduling Graph contains a single timetable task t along with the value $z_t = |T^{[i,j]}| \in \mathbb{N}$ which indicates the total number of corresponding timetable tasks. Finally, unnecessary links and supplementary tasks are discarded. This construction is described in section 4.2.5, using new partitions of $\mathcal{T}$ and $\mathcal{S}$ introduced in section 4.2.3 and 4.2.4.
The rounded Duty Scheduling Graph is a Duty Scheduling Graph $\tilde{\mathcal{D}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{A}})$ along with a vector $\tilde{z} \in \mathbb{Q}^{\tilde{\mathcal{V}}}$,

$$\tilde{z}_t = \begin{cases} z_t & \text{if } t \in \tilde{\mathcal{T}}, \\ 0 & \text{otherwise,} \end{cases}$$

which is defined in section 4.2.2 and which is the result of the algorithm described in section 4.2.5.
We now discuss the construction outlined above in more detail.

4.2.1 The $\bar{r}$-Rounding of a Duty Scheduling Graph

Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph. For each timetable task $t \in \mathcal{T}$ we define the set of associated supplementary tasks as

$$\begin{aligned} \tilde{S}(t) = \{\ \tilde{t} \in \mathcal{S} \mid \exists\ (\tilde{t}, t)\text{-path } p \text{ in } \mathcal{D} \vee \exists\ (t, \tilde{t})\text{-path } p \text{ in } \mathcal{D} : \\ (l \in A(p) \Rightarrow l \in \mathcal{A}_2) \wedge \\ (t_1 \in V(p) \setminus \{t\} \Rightarrow t_1 \in \mathcal{S})\} \ . \end{aligned} \tag{4.2.3}$$

We assume, that each supplementary task is associated to exactly one timetable task, i.e.

$$\forall\ \tilde{t} \in \mathcal{S} : |\{t \in \mathcal{T} \mid \tilde{t} \in \tilde{S}(t)\}| = 1 \ .$$

Given a Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ and a rounding parameter $\bar{r} \in \mathbb{N}$, Algorithm 4.1 constructs a modified Duty Scheduling Graph $\tilde{\mathcal{D}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{A}})$ which is a rounding of $\mathcal{D}$ with respect to $\bar{r}$ [1]. All time parameters such as $\bar{r}$ and $\bar{s}_t$ are given in seconds.
The first loop in lines 2-15 runs through all blocks. All contained timetable tasks $t \in \mathcal{T}$ are rounded with respect to the rounding value $\bar{r}$. Each timetable task is contained in exactly one of the blocks $m \in \mathcal{F}(\mathcal{D})$, see section 1.2. If $\bar{s}_t \neq \bar{e}_t$, the rounded timetable task t is kept, otherwise, it is dropped. Whenever a timetable task t, which is not the first or the last task contained in a block, is dropped, a new link connecting adjacent timetable tasks is created to maintain the structure of the block, see line 8.
Any supplementary task associated to one of the remaining timetable tasks, is also rounded and added to the modified Duty Scheduling Graph, see loop in lines 16-21. Finally, all links adjacent to a task, which is not contained in the resulting graph, are discarded.
As a result we define the following

[1] We use conventions of the programming language C.

Definition 4.2.1 *Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph and let $\bar{r} \in \mathbb{N}$, $\frac{86400}{\bar{r}} \in \mathbb{N}$. The Duty Scheduling Graph $\mathcal{D}^{\bar{r}}$ = FAST-Round($\mathcal{D}$,$\bar{r}$) is called the $\bar{r}$-Rounding of $\mathcal{D}$.*

Figures 4.2 and 4.3 show an original Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ and its corresponding 1800-Rounding $\mathcal{D}^{1800} = (\mathcal{V}^{1800}, \mathcal{A}^{1800})$. Start and end times of tasks $t \in \mathcal{V}^{1800}$ have been rounded to intervals of 1800 seconds. The number of tasks and links in the 1800-Rounding $\mathcal{D}^{1800}$ is notably smaller.

4.2.2 The Multi-Cover Duty Scheduling Graph

An important part of FAST is the use of timetable tasks, which represent simultaneous work of more than one vehicle. We now introduce another digraph, the *Multi-Cover Duty Scheduling Graph*, which is similar to the original Duty Scheduling Graph but supports multiple covering of timetable tasks. Because the only difference toward Definition

Algorithm 4.1 (FAST-Round)

Require: Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$, $\bar{r} \in \mathbb{N}$
Ensure: $\frac{86400}{\bar{r}} \in \mathbb{N}$

1: $\tilde{\mathcal{T}} \leftarrow \emptyset$; $\tilde{\mathcal{S}} \leftarrow \emptyset$; $\tilde{\mathcal{I}} \leftarrow \mathcal{I}$; $\tilde{\mathcal{A}} \leftarrow \emptyset$
2: **for** $(t_1, \ldots, t_k) \in \mathcal{F}(\mathcal{D})$ **do**
3: $\quad \hat{t} \leftarrow nil$; $missed \leftarrow$ FALSE
4: $\quad$ **for** $i = 1, \ldots, k$ **do**
5: $\quad\quad \bar{s}_{t_i} \leftarrow \bar{r} \cdot \lfloor \frac{\bar{s}_{t_i}}{\bar{r}} + 0.5 \rfloor$; $\bar{e}_{t_i} \leftarrow \bar{r} \cdot \lfloor \frac{\bar{e}_{t_i}}{\bar{r}} + 0.5 \rfloor$
6: $\quad\quad$ **if** $\bar{s}_{t_i} \neq \bar{e}_{t_i}$ **then**
7: $\quad\quad\quad$ **if** $\hat{t} \neq nil \wedge missed =$ TRUE **then**
8: $\quad\quad\quad\quad \tilde{\mathcal{A}} \leftarrow \tilde{\mathcal{A}} \cup \{(\hat{t}, t_i)\}$
9: $\quad\quad\quad$ **end if**
10: $\quad\quad\quad \tilde{\mathcal{T}} \leftarrow \tilde{\mathcal{T}} \cup \{t_i\}$; $\hat{t} \leftarrow t_i$; $missed \leftarrow$ FALSE
11: $\quad\quad$ **else**
12: $\quad\quad\quad missed \leftarrow$ TRUE
13: $\quad\quad$ **end if**
14: $\quad$ **end for**
15: **end for**
16: **for** $t \in \mathcal{S}$ **do**
17: $\quad \bar{s}_t \leftarrow \bar{r} \cdot \tilde{f}(\frac{\bar{s}_t}{\bar{r}})$; $\bar{e}_t \leftarrow \bar{r} \cdot \tilde{f}(\frac{\bar{e}_t}{\bar{r}})$
18: $\quad$ **if** $t \in \tilde{S}(\hat{t}) \wedge \hat{t} \in \tilde{\mathcal{T}}$ **then**
19: $\quad\quad \tilde{\mathcal{S}} \leftarrow \tilde{\mathcal{S}} \cup t$
20: $\quad$ **end if**
21: **end for**
22: $\tilde{\mathcal{V}} \leftarrow \tilde{\mathcal{T}} \cup \tilde{\mathcal{S}} \cup \tilde{\mathcal{I}}$
23: **for** $\tilde{l} \in \mathcal{A}$ **do**
24: $\quad$ **if** $\tilde{l} = (t_1, t_2) \wedge t_1 \in \tilde{\mathcal{V}} \wedge t_2 \in \tilde{\mathcal{V}}$ **then**
25: $\quad\quad \tilde{\mathcal{A}} \leftarrow \tilde{\mathcal{A}} \cup \tilde{l}$
26: $\quad$ **end if**
27: **end for**
28: **return** $\tilde{\mathcal{D}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{A}})$

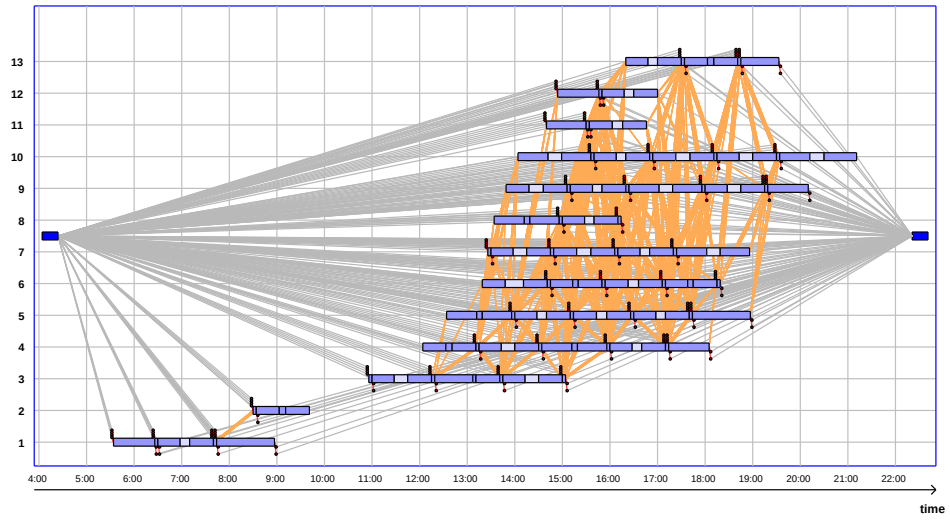

Figure 4.2: $\mathcal{D} = (\mathcal{V}, \mathcal{A})$, $|\mathcal{V}| = 423$, $|\mathcal{A}| = 3168$

1.1.4 is the labeling of the vertices, we simply adapt some of the task labels for the new digraph.

Definition 4.2.2 *Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph. The task labeling vector $\bar{v} \in \mathbb{Q}^{\mathcal{V}}$ is removed and is replaced by a new labeling vector $\tilde{z} \in \mathbb{N}^{\mathcal{V}}$, $\tilde{z}_t = 0 \ \forall \ t \notin \mathcal{T}$. $\tilde{z}$ indicates the number of vehicles associated with a specific task. We write $\mathcal{D}_{MC} = (\mathcal{V}, \mathcal{A}, \tilde{z})$ and call $\mathcal{D}_{MC}$ a Multi-Cover Duty Scheduling Graph.*

Because supplementary tasks $t \in \mathcal{S}$ and artificial tasks $t \in \mathcal{I}$ do not need to be covered, the corresponding multi cover label is set to zero, i.e. $\tilde{z}_t = 0 \ \ \forall \ t \in \mathcal{V} \setminus \mathcal{T}$.
Duties and duty schedules as defined in Definition 1.1.5 and Definition 1.1.6 are applicable for Multi-Cover Scheduling Graphs. Blocks and link types are no longer available because the block id's $\bar{v} \in \mathbb{Q}^{\mathcal{V}}$ have been removed.
Furthermore, Definitions 1.1.6, 1.1.7 and 1.1.8 can be easily adapted to define the Duty Scheduling Problem on a Multi-Cover Duty Scheduling Graph. However, in this chapter we only use an adaption of Model 1.2 which is introduced later.

4.2.3 A Partition of $\mathcal{T}$

Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph. Each timetable task $t \in \mathcal{T}$ represents work associated with a vehicle, which has already been routed and for which a driver has to be assigned. This work can be partitioned into *driving parts*, *break parts* and *working parts*.
Driving parts represent actual driving operations. Break parts represent breaks predefined by the position of the vehicle, e.g. a ten minute

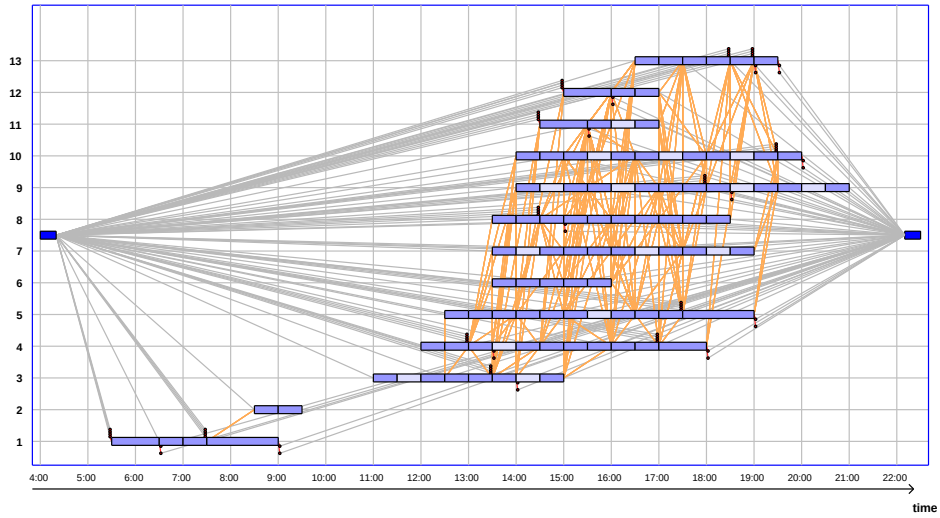

Figure 4.3: $\mathcal{D}^{1800} = (\mathcal{V}^{1800}, \mathcal{A}^{1800})$, $|\mathcal{V}^{1800}| = 179$, $|\mathcal{A}^{1800}| = 1046$

break to be taken at a terminal stop. Working parts represent additional work that has to be performed at the start or end of driving parts, such as checking functions of the vehicle and engine warm-up.
FAST uses a Multi-Cover Duty Scheduling Graph to relax the position of all timetable tasks $t \in \mathcal{T}$ by merging tasks $t_1, t_2 \in \mathcal{T}$ with $\bar{s}_{t_1} = \bar{s}_{t_2}$ and $\bar{e}_{t_1} = \bar{e}_{t_2}$. However, we need to ensure that no timetable tasks which represent different parts of work are merged. We pretend for any Duty Scheduling Graph [2], that each timetable task $t \in \mathcal{T}$ represents either a driving part, or a working part, or a break part. This assumption is justified by the fact that each timetable task $\tilde{t} \in \mathcal{T}$, which does not fulfill the requirement above, can be split into timetable tasks representing its different parts which substitute $\tilde{t}$. Links are added to connect succeeding timetable tasks resulting from such a split.
Hence we assume that for any Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ and any Multi-Cover Duty Scheduling Graph $\mathcal{D}_{MC} = (\mathcal{V}, \mathcal{A}, \tilde{z})$,

$$\mathcal{T} = \mathcal{T}_d \dot{\cup} \mathcal{T}_w \dot{\cup} \mathcal{T}_b$$

is a partition of the timetable tasks where $\mathcal{T}_d \subset \mathcal{T}$ identifies the set of *driving tasks*, $\mathcal{T}_w \subset \mathcal{T}$ identifies the set of *working tasks* and $\mathcal{T}_b \subset \mathcal{T}$ identifies the set of *break tasks*.

4.2.4 A Partition of $\mathcal{S}$

We now introduce a partition of the set of supplementary tasks. Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph, let $\mathcal{S} \subset \mathcal{V}$ and let $s, e \in \mathcal{I}$ indicate the start and end artificial tasks. For any timetable task $t \in \mathcal{T}$ let $\tilde{S}(t)$ be defined as in (4.2.3). Further recall that each supplementary task can be associated to exactly one single timetable task with respect to $\tilde{S}(.)$.

[2] And because of Definition 4.2.2 any Multi-Cover Duty Scheduling Graph.

We now decide whether a supplementary task $t \in \mathcal{S}$ is part of a $\left[s, \tilde{t}\right]$ -path or part of a $\left[\tilde{t}, e\right]$ -path, where $\tilde{t} \in \mathcal{T} : t \in \tilde{S}(\tilde{t})$. This distinction is necessary to maintain the structure of the Duty Scheduling Graph in the modification described in the next section.

Definition 4.2.3 *Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph. We define the following subsets of $\mathcal{S}$.*

(i) The set of supplementary tasks with indication -1 is defined as

$$\mathcal{S}_{-1} = \{t \in \mathcal{S} \mid \exists\ \tilde{t} \in \mathcal{T}, \left[s, \tilde{t}\right] \text{-path } p : t \in \tilde{S}(\tilde{t}) \wedge t \in p\}$$

(ii) The set of supplementary tasks with indication +1 is defined as

$$\mathcal{S}_{+1} = \{t \in \mathcal{S} \mid \exists\ \tilde{t} \in \mathcal{T}, \left[\tilde{t}, e\right] \text{-path } p : t \in \tilde{S}(\tilde{t}) \wedge t \in p\}$$

(iii) The set of supplementary tasks with indication 0 is defined as

$$\mathcal{S}_0 = \mathcal{S} \setminus \{\mathcal{S}_{+1} \cup \mathcal{S}_{-1}\}$$

We assume that for any Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ the sets $\mathcal{S}_{-1}$ and $\mathcal{S}_{+1}$ are disjunct. Definition 4.2.3 provides a partition of $\mathcal{S}$, i.e.

$$\mathcal{S} = \mathcal{S}_{-1} \dot{\cup}\ \mathcal{S}_0 \ \dot{\cup}\ \mathcal{S}_{+1}\ .$$

Each supplementary task $t \in \mathcal{S}$ has either indication 0, indication -1 or indication $+1$.

4.2.5 The $\bar{r}$-*pos*-Relaxation of a Duty Scheduling Graph

Given a Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ and a rounding parameter $\bar{r}$, we now construct a Multi-Cover Duty Scheduling Graph used to define a relaxation of the Duty Scheduling Problem in the next section. Let $\mathcal{D}^{\bar{r}} = (\mathcal{V}^{\bar{r}}, \mathcal{A}^{\bar{r}})$ be the $\bar{r}$-Rounding of $\mathcal{D}$, i.e. all tasks have been rounded with respect to $\bar{r} \in \mathbb{N}$. Algorithm 4.2 describes the construction of the Multi-Cover Duty Scheduling Graph.
A set X consisting of all intervals indicated by the rounding parameter $\bar{r}$ is constructed in line 5. For each interval $[i, j] \in X$ the number of covering driving tasks $|T_d^{[i,j]}|$ is determined. If $|T_d^{[i,j]} > 0|$, a new timetable task t_1 representing all driving tasks covering the interval $[i, j]$ is added to the returning Multi-Cover Duty Scheduling Graph, see loop in lines 6-19 of Algorithm 4.2. The corresponding value z_{t_1} identifies the number of represented driving tasks. Working tasks are treated similarly.
Break tasks $t \in \mathcal{T}_b^{\bar{r}}$ and artificial tasks $t \in \mathcal{I}$ are copied, see line 3 and line 21.
Supplementary tasks are sorted by indication and added to the resulting Multi-Cover Duty Scheduling Graph, while redundancy is prevented and thus the number of supplementary tasks is reduced, see lines 22-32.

Algorithm 4.2 (FAST-Relax)

Require: Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$, $\bar{r} \in \mathbb{N}$

$\mathcal{D}^{\bar{r}} = (\mathcal{V}^{\bar{r}}, \mathcal{A}^{\bar{r}}) \leftarrow$ FAST-Round($\mathcal{D}$, $\bar{r}$)
$\tilde{\mathcal{T}}_d \leftarrow \emptyset$; $\tilde{\mathcal{T}}_w \leftarrow \emptyset$
$\tilde{\mathcal{T}}_b \leftarrow \mathcal{T}_b^{\bar{r}} \subset \mathcal{T}^{\bar{r}} \subset \mathcal{V}^{\bar{r}}$
$\mathcal{V}^{\bar{r}} \supset \mathcal{T}^{\bar{r}} = \mathcal{T}_d^{\bar{r}} \cup \mathcal{T}_w^{\bar{r}} \cup \mathcal{T}_b^{\bar{r}}$
$X \leftarrow \{[0, 1 \cdot \bar{r}], [1 \cdot \bar{r}, 2 \cdot \bar{r}], [2 \cdot \bar{r}, 3 \cdot \bar{r}], \ldots [k \cdot \bar{r}, 86400]\}$
for all $[i, j] \in X$ **do**
    $T_d^{[i,j]} = \{t \in \mathcal{T}_d^{\bar{r}} | \bar{s}_t \leq i \wedge \bar{e}_t \geq j\}$
    $T_w^{[i,j]} = \{t \in \mathcal{T}_w^{\bar{r}} | \bar{s}_t \leq i \wedge \bar{e}_t \geq j\}$
    if $|T_d^{[i,j]}| > 0$ **then**
        $t_1 \leftarrow new(\bar{s}_{t_1} = i, \bar{e}_{t_1} = j, z_{t_1} = |T_d^{[i,j]}|)$
        $\tilde{z}_{t_1} \leftarrow z_{t_1}$
        $\tilde{\mathcal{T}}_d \leftarrow \tilde{\mathcal{T}}_d \cup \{t_1\}$
    end if
    if $|T_w^{[i,j]}| > 0$ **then**
        $t_2 \leftarrow new(\bar{s}_{t_2} = i, \bar{e}_{t_2} = j, z_{t_2} = |T_w^{[i,j]}|)$
        $\tilde{z}_{t_2} \leftarrow z_{t_2}$
        $\tilde{\mathcal{T}}_w \leftarrow \tilde{\mathcal{T}}_w \cup \{t_2\}$
    end if
end for
$\tilde{\mathcal{T}} \leftarrow \tilde{\mathcal{T}}_d \cup \tilde{\mathcal{T}}_w \cup \tilde{\mathcal{T}}_b$
$\tilde{\mathcal{I}} \leftarrow \mathcal{I}$
for $t \in \mathcal{S}$ **do**
    if $t \in \mathcal{S}_{-1} \wedge \neg$redundant$(t, \tilde{\mathcal{S}}_{-1})$ **then**
        $\tilde{\mathcal{S}}_{-1} \leftarrow \tilde{\mathcal{S}}_{-1} \cup \{t\}$
    end if
    if $t \in \mathcal{S}_0 \wedge \neg$redundant$(t, \tilde{\mathcal{S}}_0)$ **then**
        $\tilde{\mathcal{S}}_0 \leftarrow \tilde{\mathcal{S}}_0 \cup \{t\}$
    end if
    if $t \in \mathcal{S}_{+1} \wedge \neg$redundant$(t, \tilde{\mathcal{S}}_{+1})$ **then**
        $\tilde{\mathcal{S}}_{+1} \leftarrow \tilde{\mathcal{S}}_{+1} \cup \{t\}$
    end if
end for
$\tilde{\mathcal{S}} \leftarrow \tilde{\mathcal{S}}_{-1} \cup \tilde{\mathcal{S}}_0 \cup \tilde{\mathcal{S}}_{+1}$
$\tilde{\mathcal{V}} \leftarrow \tilde{\mathcal{T}} \cup \tilde{\mathcal{S}} \cup \tilde{\mathcal{I}}$
for $t \in \tilde{\mathcal{T}}_d \cup \tilde{\mathcal{T}}_w$ **do**
    for $\tilde{t} \in \tilde{\mathcal{T}}_d \cup \tilde{\mathcal{T}}_w \cup \tilde{\mathcal{T}}_b \wedge \bar{e}_t = \bar{s}_{\tilde{t}}$ **do**
        $\tilde{\mathcal{A}} \leftarrow \tilde{\mathcal{A}} \cup \{(t, \tilde{t})\}$
    end for
end for
for $t \in \tilde{\mathcal{T}}_b$ **do**
    for $\tilde{t} \in \tilde{\mathcal{T}}_d \cup \tilde{\mathcal{T}}_w \wedge \bar{e}_t = \bar{s}_{\tilde{t}}$ **do**
        $\tilde{\mathcal{A}} \leftarrow \tilde{\mathcal{A}} \cup \{(t, \tilde{t})\}$
    end for
end for
for $l \in \mathcal{A}^{\bar{r}}$ **do**
    if $l \notin \mathcal{A}_1^{\bar{r}} \wedge$ redundantL$(l, \tilde{\mathcal{A}}) = nil$ **then**
        $\tilde{\mathcal{A}} \leftarrow \tilde{\mathcal{A}} \cup \{$getL$(l)\}$
    end if
end for
return $\tilde{\mathcal{D}}_{MC} = (\tilde{\mathcal{V}}, \tilde{\mathcal{A}}, \tilde{z})$

The function *redundant(t, $\tilde{\mathcal{S}}$)*, see line 23, decides if a supplementary task $t \in \mathcal{S}$ is redundant because another supplementary task with the same labels is already contained in $\tilde{\mathcal{S}}$, i.e.

$$\exists\, \tilde{t} \in \tilde{\mathcal{S}} : \bar{s}_{\tilde{t}} = \bar{s}_t \wedge \bar{e}_{\tilde{t}} = \bar{e}_t \wedge \bar{c}_{\tilde{t}} = \bar{c}_t \wedge \bar{u}_{\tilde{t}} = \bar{u}_t \wedge \bar{w}_{\tilde{t}} = \bar{w}_t \ . \qquad (4.2.4)$$

For a further reduction of the number of supplementary tasks, statement 4.2.4 can be modified into a relaxed version

$$\exists\, \tilde{t} \in \tilde{\mathcal{S}} : \bar{s}_{\tilde{t}} = \bar{s}_t \wedge \bar{e}_{\tilde{t}} = \bar{e}_t \wedge \bar{c}_{\tilde{t}} = \bar{c}_t \wedge \bar{u}_{\tilde{t}} \sim \bar{u}_t \wedge \bar{w}_{\tilde{t}} \sim \bar{w}_t \ .$$

If a supplementary task t is redundant with respect to $\tilde{t} \in \tilde{\mathcal{S}}$, then t is dropped, but the information that $\tilde{t}$ is a placeholder for t is stored. All links which were connected to t now connect to $\tilde{t}$.
In lines 35-44, links are added to connect succeeding timetable tasks in the Multi-Cover Duty Scheduling Graph. However, break tasks are never connected to other break tasks, because the length of breaks is fixed.
Finally, all links $l \in \mathcal{A}$ are added to the resulting Multi-Cover Duty Scheduling Graph $\tilde{\mathcal{D}}_{MC} = (\tilde{\mathcal{V}}, \tilde{\mathcal{A}}, \tilde{z})$ or are discarded due to redundancy; see 46. The function *redundantL(l, $\tilde{\mathcal{A}}$)* returns a link $\tilde{l} \in \tilde{\mathcal{A}}$ if l is redundant with respect to $\tilde{l}$. A link $l \in \mathcal{A}$ is only added to $\tilde{\mathcal{D}}_{MC}$, if $redundantL(l, \tilde{\mathcal{A}}) = nil$. If a link is not redundant, the function *getL(l)* returns a copy of the link which is connected to the correct tasks in $\tilde{\mathcal{D}}_{MC}$, which is then added to $\tilde{\mathcal{A}}$.
The $\bar{r}$-*pos*-Relaxation of a Duty Scheduling Graph is now defined as follows.

Definition 4.2.4 *Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph and let $\bar{r} \in \mathbb{N}$, $\frac{86400}{\bar{r}} \in \mathbb{N}$. The Multi-Cover Duty Scheduling Graph*

$$\mathcal{D}^{\bar{r}}_{pos} = (\mathcal{V}^{\bar{r}}_{pos}, \mathcal{A}^{\bar{r}}_{pos}, \tilde{z}) = FAST\text{-}Relax(\mathcal{D}, \bar{r})$$

is called the $\bar{r}$-pos-Relaxation of $\mathcal{D}$.

Figure 4.4 displays the 1800-*pos*-Relaxation of the Duty Scheduling Graph displayed in Figure 4.2. There is at most one driving task in each interval, while all light blue colored break tasks contained in the 1800-Rounding in Figure 4.3 are maintained. Note that the number of tasks and links has been further reduced.

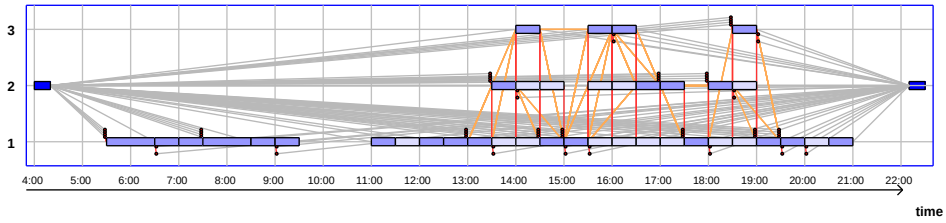

Figure 4.4: $\mathcal{D}^{1800}_{pos} = (\mathcal{V}^{1800}_{pos}, \mathcal{A}^{1800}_{pos}, \tilde{z})$, $|\mathcal{T}^{1800}_{pos}| = 111$, $|\mathcal{A}^{1800}_{pos}| = 383$

4.3 The FAST-Relaxation

Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph, $\bar{r} \in \mathbb{N}$ be a rounding parameter and let $\mathcal{D}^{\bar{r}}_{pos} = (\mathcal{V}^{\bar{r}}_{pos}, \mathcal{A}^{\bar{r}}_{pos}, \tilde{z})$ be the $\bar{r}$-*pos*-Relaxation of $\mathcal{D}$. $\mathcal{D}^{\bar{r}}_{pos}$ is a Multi-Cover Duty Scheduling Graph. Let $\tilde{\mathcal{P}}$ indicate the set of all duties in $\mathcal{D}^{\bar{r}}_{pos}$. For each $p \in \tilde{\mathcal{P}}$ let x_p indicate a non-negative variable and for each base resource $b \in \mathcal{W}$ let c_b indicate a penalty and let s_b be slack variables similar to those described in section 1.4. The *FAST-Relaxation* (FR) of the Duty Scheduling Problem $\min\{C(C) \mid C \in \mathcal{C}^{\mathcal{D}}\}$ is the linear program 4.1.

Model 4.1 (**FR**-$\mathcal{D}^{\bar{r}}_{pos}$)

$$\min \quad \sum_{p \in \tilde{\mathcal{P}}} c_p x_p + \sum_{b \in \mathcal{W}} c_b s_b$$

$$\text{s.t.} \quad \sum_{\{p \in \tilde{\mathcal{P}} \mid t \in p\}} x_p \geq \tilde{z}_t \quad \forall\, t \in \mathcal{T}^{\bar{r}}_{pos} \tag{4.3.5}$$

$$\sum_{p \in \tilde{\mathcal{P}}} (w_p)_b x_p - s_b \leq \hat{w}_b \quad \forall\, b \in \mathcal{W} \tag{4.3.6}$$

$$x_p \geq 0 \quad \forall\, p \in \tilde{\mathcal{P}} \tag{4.3.7}$$

$$s_b \geq 0 \quad \forall\, b \in \mathcal{W} \tag{4.3.8}$$

Model 4.1 is an adaption of Model 1.2. The constraint set 4.3.5 ensures that each timetable task t is covered by at least $\tilde{z}_t$ fractional duties. Constraints 4.3.6 are similar to Model 1.2. All duty variables are non-negative, see constraints 4.3.7.
A solution $\hat{x} \in \mathbb{Q}^{\tilde{\mathcal{P}}}$ of (**FR**-$\mathcal{D}^{\bar{r}}_{pos}$) is very likely to be fractional. However, FAST does not require an integer solution in the second phase.

4.3.1 RDSP - A Solver for the FAST-Relaxation

The FAST-Relaxation can be solved using RDSP$^{\hat{i}}$ (relaxed DS-Opt), which is an adaption of the column generation solver DS-Opt mentioned in section 2.4. The parameter $\hat{i} \in \mathbb{N}$ controls the number of iterations performed in the column generation process.
Let (**FR**-$\mathcal{D}^{\bar{r}}_{pos}$) be a FAST-Relaxation and let $\tilde{\mathcal{P}}$ be the set of all corresponding duties. Similar to the DS-Opt column generation approach, a subset $\tilde{P}^0 \subset \tilde{\mathcal{P}}$ is generated. A restricted lp based on 4.1 is solved by CPLEX, providing a fractional solution $x^0 \in \mathbb{Q}^{\tilde{P}^0}$. If $\hat{i} = 0$, then x^0 is already the final fractional solution which is returned by RDSP$^{\hat{i}}$.
Otherwise a number of iterations are performed and in each iteration $j \in \{1, \ldots, \hat{i}\}$ the dual variables regarding the previous solution

$x^{j-1} \in \mathbb{Q}^{\tilde{P}^{j-1}}$ are used to generate duties $p \in \tilde{\mathcal{P}} \setminus \tilde{P}^{j-1}$ with negative reduced cost. These duties are then added to $\tilde{P}^{j-1}$ while inefficient duties are dropped. The restricted master linear program of the FAST-Relaxation, see Model 4.1, is then solved regarding the modified set of duties $\tilde{P}^j$ to generate a new fractional solution $x^j \in \mathbb{Q}^{\tilde{P}^j}$
The final solution $\hat{x} = \text{RDSP}^{\hat{i}}(\mathcal{D}^{\bar{r}}_{pos}) \in \tilde{\mathcal{P}}$, where

$$\hat{x}_p = \begin{cases} x_p^{\hat{i}} & \text{if } p \in \tilde{P}^{\hat{i}}, \\ 0 & \text{otherwise,} \end{cases}$$

is returned. If $\hat{i}$ is increased to improve the quality of the final fractional solution $\hat{x} \in \tilde{\mathcal{P}}$, the runtime of RDSP also increases as more iterations are performed.

4.3.2 A Fractional Solution $\hat{x}$ of (FR-$\mathcal{D}^{\bar{r}}_{pos}$)

This section discusses some aspects of a fractional solution $\hat{x}$ of the FAST-Relaxation. Given a value $h \in \mathbb{Q}$, $0 \leq h < 1$, the fractional solution $\hat{x} \in \mathbb{Q}^{\tilde{\mathcal{P}}}$ corresponds to a set of duties

$$\mathcal{P}^h_{\hat{x}} = \{p \in \tilde{\mathcal{P}} \mid \hat{x}_p \geq h\}$$

which contains all duties with a value greater or equal h in the fractional solution.
Because the $\hat{x}_p$ variables are fractional, tasks and links are likely to be contained in more than a single duty $p \in \mathcal{P}$ with $\hat{x}_p \geq h$. We define the set of all links in the fractional solution $\hat{x}$ by

$$\mathcal{A}^h_{\hat{x}} = \{l \in \mathcal{A}^{\bar{r}}_{pos} \mid \exists\, p \in \mathcal{P}^h_{\hat{x}} : l \in A(p)\}\ . \tag{4.3.9}$$

Each link $l \in \mathcal{A}^{\bar{r}}_{pos}$ corresponds to at least one link $\tilde{l} \in \mathcal{A}$ in the original problem. We define the following:

Definition 4.3.1 *Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph, let $\bar{r}$ be a rounding parameter, let $\mathcal{D}^{\bar{r}} = (\mathcal{V}^{\bar{r}}, \mathcal{A}^{\bar{r}})$ be the $\bar{r}$-Rounding and let $\mathcal{D}^{\bar{r}}_{pos} = (\mathcal{V}^{\bar{r}}_{pos}, \mathcal{A}^{\bar{r}}_{pos}, \tilde{z})$ be the $\bar{r}$-pos-Relaxation of $\mathcal{D}$. A link $l \in \mathcal{A}^{\bar{r}}_{pos}$ represents $\tilde{l} \in \mathcal{A}$ if $\tilde{l} \in \mathcal{A}^{\bar{r}} \wedge l = redundantL(\tilde{l}, \mathcal{A}^{\bar{r}}_{pos})$.*

Given a link $l \in \mathcal{A}^{\bar{r}}_{pos}$ we define the set of *original links* as

$$\mathcal{O}(l) = \{\tilde{l} \in \mathcal{A} \mid l \ \text{ represents } \ \tilde{l}\}\ .$$

Obviously each original link $\tilde{l}$ is only represented by a single link $l \in \mathcal{A}^{\bar{r}}_{pos}$, i.e.

$$|\{l \in \mathcal{A}^{\bar{r}}_{pos} \mid \tilde{l} \in \mathcal{O}(l)\}| = 1 \quad \forall\, \tilde{l} \in \mathcal{A}\ .$$

For any $l \in \mathcal{A}^h_{\hat{x}}$ we further define the *relaxed count* $n_l \in \mathbb{N}$ and the *relaxed value* $r_l \in \mathbb{Q}$ of l as

$$n_l = |\{p \in \mathcal{P}^h_{\hat{x}} \mid l \in p\}| \quad \text{and} \quad r_l = \sum_{\{p \in \mathcal{P}^h_{\hat{x}} \mid l \in p\}} \hat{x}_p\ . \tag{4.3.10}$$

The relaxed count n_l identifies the number of duties which contain the link l, while the relaxed value r_l identifies the total value of all these duties.

4.4 Fixing Duty Pieces in $\mathcal{D} = (\mathcal{V}, \mathcal{A})$

Throughout this section let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph and let $\min\{C(C) \mid C \in \mathcal{C}^{\mathcal{D}}\}$ be a Duty Scheduling Problem defined on $\mathcal{D}$. Further let $\mathcal{D}^{\bar{r}}_{pos} = (\mathcal{V}^{\bar{r}}_{pos}, \mathcal{A}^{\bar{r}}_{pos}, \tilde{z})$ be the $\bar{r}$-*pos*-Relaxation of $\mathcal{D}$ and let $\tilde{\mathcal{P}}$ indicate the set of all duties in $\mathcal{D}^{\bar{r}}_{pos}$. Finally, let $\hat{x} \in \mathbb{Q}^{\tilde{\mathcal{P}}}$ be a fractional solution of the FAST-Relaxation of $\min\{C(C) \mid C \in \mathcal{C}^{\mathcal{D}}\}$.
In the second phase of FAST, the solution $\hat{x}$ is used to generate a subgraph $\tilde{\mathcal{D}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{A}})$ of $\mathcal{D}$ where $\tilde{\mathcal{T}} = \mathcal{T}$, but $\tilde{\mathcal{S}} \subset \mathcal{S}$ and $\tilde{\mathcal{A}} \subset \mathcal{A}$. Thus $\mathcal{C}^{\tilde{\mathcal{D}}} \subset \mathcal{C}^{\mathcal{D}}$ and the Duty Scheduling Problem $\min\{C(C) \mid C \in \mathcal{C}^{\tilde{\mathcal{D}}}\}$ is a restriction of $\min\{C(C) \mid C \in \mathcal{C}^{\mathcal{D}}\}$. As mentioned in section 4.1, each block $m \in \mathcal{F}(\mathcal{D})$ in the resulting subgraph $\tilde{\mathcal{D}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{A}})$ is partitioned into duty pieces.
Assume that for a block $m = (t_1, t_2, t_3, t_4) \in \mathcal{F}(\mathcal{D})$ we choose the two duty pieces $(t_1, t_2), (t_3, t_4) \in \mathcal{N}^m$ which identify a duty piece partition of m. Then the modified set of links $\mathcal{A}$ would be

$$\begin{aligned}\tilde{\mathcal{A}} \; = \mathcal{A} \setminus \{(t, \tilde{t}) \in \mathcal{A} | (t = t_1 \wedge \tilde{t} \neq t_2) \vee (t = t_3 \wedge \tilde{t} \neq t_4) \vee \\ (t \neq t_1 \wedge \tilde{t} = t_2) \vee (t \neq t_3 \wedge \tilde{t} = t_4)\}\end{aligned}$$

and the links (t_1, t_2) and (t_3, t_4) would be type 1 links, i.e. the two duty pieces are fixed.
These duty pieces are now the smallest units of work which have to be assigned to the same driver. Fixed duty pieces can be merged to form longer pieces of work in the final solution.
In section 4.4 we describe in detail how the restricted Duty Scheduling Graph $\tilde{\mathcal{D}}$ is constructed.

4.4.1 The Parameter Vector $\hat{q}$

The construction of the restricted Duty Scheduling Graph can be adjusted by a set of parameters which we introduce in this section. Let $\hat{x}$ be a fractional relaxed solution and let $\mathcal{A}^h_{\hat{x}}$ be the set of corresponding links as defined in 4.3.9.
To partition the blocks into duty pieces we first determine a set of critical links $\hat{L} \subset \mathcal{A}$ and then select two succeeding timetable tasks $t \in \mathcal{T}$ and $\tilde{t} \in \mathcal{T}$ of any block $(t_1, \ldots, t, \tilde{t}, \ldots t_k) \in \mathcal{F}(\mathcal{D})$ into the same duty piece if

$$\forall\, (t_i, t_j) \in \mathcal{A} : (t_i = t \vee t_j = \tilde{t}) \Rightarrow l \notin \hat{L}\ ,$$

i.e. t is not a tail and $\tilde{t}$ is not a head of a critical link $l \in \hat{L}$.
To decide whether a link $l \in \mathcal{A}$ is critical, we introduce two parameters $\hat{q}_1 \in \mathbb{N}$ and $\hat{q}_2 \in \mathbb{Q}$. A link l is critical, i.e. $l \in \hat{L}$, if $l \in \mathcal{O}(\tilde{l})$ and

$$n_{\tilde{l}} \geq \hat{q}_1 \quad \text{or} \quad r_{\tilde{l}} \geq \hat{q}_2\ .$$

Thus, the parameters $\hat{q}_1$ and $\hat{q}_2$ ensure that only links with a sufficient presence in the fractional relaxed solution $\hat{x}$ are declared critical. By choosing higher values for $\hat{q}_1$ and $\hat{q}_2$, e.g. $\hat{q}_1 = 15$ and $\hat{q}_2 = 0.3$, the

number of critical links and therefore the number of fixed duty pieces decreases.
Another parameter controls the length of the fixed duty pieces. Recall, that each duty $p \in \mathcal{P}$ has to satisfy a set of constraints. Normally these constraints declare a maximum length, e.g. d_{max}, and an optimal length, e.g. d_{opt}, for any piece of work used in a duty. Unfortunately the chosen set of critical links $\hat{L}$ does not ensure a feasible length for the fixed duty pieces. Thus to ensure feasibility in this case we eventually need to split some of the fixed duty pieces into smaller duty pieces. To increase flexibility we introduce another parameter $\hat{q}_3 \in \mathbb{Q}, 0 < \hat{q}_3 < 2$ and define the maximum length of any of the fixed duty pieces as $\hat{q}_3 \cdot d_{opt}$.
Assume that a set of duty pieces has been fixed, and that $n = (t_1, \ldots, t_k)$ is one of the fixed pieces. All links which connect one of the inner tasks of n, $t_2, \ldots t_{k-1}$ to tasks which are not contained in the duty piece are discarded. For small instances this reduction may suffice to achieve efficient runtimes, but for large instances runtimes are still high.
Normally each fixed duty piece would be connected to almost all of the following duty pieces. Only some of these connections are cost efficient. FAST uses this idea to delete some of the connections as follows.
First two parameters $\hat{q}_7 \in \mathbb{N}$ and $\hat{q}_8 \in \mathbb{N}$ are introduced which identify short, middle and long connections between two duty pieces. These parameters store time values in seconds.
Let $p_1 = (t_1, \ldots, t_k)$ and $p_2 = (\tilde{t}_1, \ldots, \tilde{t}_l)$ be two of the fixed pieces, then a connection between p_1 and p_2 has length $l_{p_1p_2} = \bar{s}_{\tilde{t}_1} - \bar{e}_{t_k}$. If $l_{p_1p_2} < \hat{q}_7$ it is a short connection, if $\hat{q}_7 \leq l_{p_1p_2} \leq \hat{q}_8$ it is a middle connection and if $\hat{q}_8 < l_{p_1p_2}$ it is a long connection. Usual parameter values are $\hat{q}_7 = 1800, \hat{q}_8 = 5400$, i.e. connections of length less than 30 minutes are short connections and connections with length up to 90 minutes are middle connections[3].
To reduce the number of connections, we introduce upper limits $\hat{q}_4$, $\hat{q}_5$, $\hat{q}_6 \in \mathbb{N}$ for the number of outgoing and incoming short, middle and long connections for all duty pieces.
All parameters are stored in a vector $\hat{q} \in \mathbb{Q}^8$ which is part of the input to FAST.

Definition 4.4.1 *The vector $\hat{q} = (\hat{q}_1, \hat{q}_2, \hat{q}_3, \hat{q}_4, \hat{q}_5, \hat{q}_6, \hat{q}_7, \hat{q}_8) \in \mathbb{Q}^8$ containing all parameters introduced in this section is called the fix-parameter vector.*

4.4.2 The Sets $\mathcal{T}^{start}$ and $\mathcal{T}^{end}$

Let again $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph and let $\hat{x}$ be a fractional solution of the FAST-Relaxation of $\min\{C(C) \mid C \in \mathcal{C}^{\mathcal{D}}\}$.
We now discuss how the set of critical links $\hat{L}$ is used to fix some of the duty pieces. As any duty piece is part of a block, it can be identified by the first and the last contained task. Thus we only generate two sets

[3] Values stored in $\hat{q}_7$ and $\hat{q}_8$ are in seconds.

$\mathcal{T}^{start} \subset \mathcal{T}$ and $\mathcal{T}^{end} \subset \mathcal{T}$ to identify all duty pieces. The set $\mathcal{T}^{start}$ contains all starting timetable tasks for the fixed duty pieces while set $\mathcal{T}^{end}$ contains all ending timetable tasks respectively.
Given the original Duty Scheduling Graph $\mathcal{D}$, the set of links indicated by $\hat{x}$, $\mathcal{A}^h_{\hat{x}}$ and a fix-parameter vector $\hat{q}$, the sets $\mathcal{T}^{start}$ and $\mathcal{T}^{end}$ are generated by Algorithm 4.3 which consists of four main loops.
In the loop in lines 3-8 two values $\hat{r}, \hat{n}$ are calculated for each link $\tilde{l} \in \mathcal{A}$ used to determine if a link is critical. More precisely, a link $\tilde{l} \in \mathcal{A}$ is critical if $\hat{n}_{\tilde{l}} \geq \hat{q}_1$ or $\hat{r}_{\tilde{l}} \geq \hat{q}_2$. Recall, that each original link $\tilde{l} \in \mathcal{A}$ is represented by exacly one link $l \in \mathcal{A}^h_{\hat{x}}$.
In the second loop in lines 10-21 all timetable tasks $t \in \mathcal{T}$ which are adjacent to a critical link $l \in \mathcal{A}$ are inserted into the correct set $\mathcal{T}^{start}$ or $\mathcal{T}^{end}$. All type 1 links and all critical type 2 links which connect two timetable tasks are ignored because the adjacent timetable tasks must be contained in the same duty piece.
Let $t, \tilde{t}$ be two succeeding timetable tasks of any block $m \in \mathcal{F}(\mathcal{D})$. It might occur, that a critical link $(t_1, \tilde{t}) \in \mathcal{A}, t_1 \neq t$ exists, but none of the outgoing links of t except $(t, \tilde{t})$ is critical. Thus $\tilde{t} \in \mathcal{T}^{start}$ but $t \notin \mathcal{T}^{end}$, which results in an incomplete cutting of the block m. The third loop in lines 22-29 overcomes such cases by adding relevant timetable tasks into the correct sets, i.e. the task t from the example above would be added into the set $\mathcal{T}^{end}$. As a result the sets $\mathcal{T}^{start}, \mathcal{T}^{end}$ represent a set of duty pieces which form a feasible and complete partitioning of all blocks.
The last loop in lines 30-47 validates the complete duty piece partition for each block. By using splitting operations, see line 40, it also ensures that the length of any of the duty pieces does not exceed the predefined value $\hat{q}_3 \cdot d_{opt}$.

4.4.3 The $\hat{x}$-$\hat{q}$-FAST-Fix

Given a Duty Scheduling Digraph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ and the sets $\mathcal{T}^{start} \subset \mathcal{T}$ and $\mathcal{T}^{end} \subset \mathcal{T}$ identifying duty pieces partitioning all blocks, we now remove all links redundant with respect to the fixed duty piece partitions to create a subgraph of $\mathcal{D}$. The construction of the subgraph is described in Algorithm 4.4. We also reduce the number of connections between the fixed duty pieces as stated in section 4.4.1.
First, Algorithm 4.3 is called to identify starting and ending timetable tasks for all pieces, see line 2 in Algorithm 4.4. The statement in line 3 ensures that all type 1 links and all type 2 links connecting two timetable tasks belonging to the same duty piece are kept.
Lines 5-7 of Algorithm 4.4 initialize counters for short, middle and long connections for all timetable tasks. Only the counters for tasks $t \in \mathcal{T}^{start} \cup \mathcal{T}^{end}$ are needed.
The first loop in lines 9-28 runs through all connections between piece endings or artificial tasks and piece starting tasks, i.e. all $[\tilde{t}, t]$-paths where $\tilde{t} \in \mathcal{T}^{end} \cup \mathcal{I}$ and $t \in \mathcal{T}^{start}$. Assume that the path $p = (\tilde{t}, \ldots t)$

Algorithm 4.3 (Generate-Pieces)

Require: Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$, $\mathcal{A}^h_{\hat{x}}$, parameter vector $\hat{q}$

$\hat{r} \leftarrow (0, 0, \ldots, 0) \in \mathbb{Q}^{\mathcal{A}}$
$\hat{n} \leftarrow (0, 0, \ldots, 0) \in \mathbb{N}^{\mathcal{A}}$
for $l \in \mathcal{A}^h_{\hat{x}}$ **do**
    for $\tilde{l} \in \mathcal{O}(l)$ **do**
        $\hat{r}_{\tilde{l}} \leftarrow r_l$
        $\hat{n}_{\tilde{l}} \leftarrow n_l$
    end for
end for
$\mathcal{T}^{start} \leftarrow \emptyset, \mathcal{T}^{end} \leftarrow \emptyset$
for $l = (t_1, t_2) \in \mathcal{A} \setminus \mathcal{A}_1$ **do**
    if $\hat{r}_l \geq \hat{q}_1 \vee \hat{n}_l \geq \hat{q}_2$ **then**
        if $t_1 \in \mathcal{T} \wedge t_2 \notin \mathcal{T}$ **then**
            $\mathcal{T}^{end} \leftarrow \mathcal{T}^{end} \cup \{t_1\}$
        else if $t_1 \notin \mathcal{T} \wedge t_2 \in \mathcal{T}$ **then**
            $\mathcal{T}^{start} \leftarrow \mathcal{T}^{start} \cup \{t_2\}$
        else if $t_1 \in \mathcal{T} \wedge t_2 \in \mathcal{T} \wedge l \in \mathcal{A}_3$ **then**
            $\mathcal{T}^{end} \leftarrow \mathcal{T}^{end} \cup \{t_1\}$
            $\mathcal{T}^{start} \leftarrow \mathcal{T}^{start} \cup \{t_2\}$
        end if
    end if
end for
for $l = (t_1, t_2) \in \mathcal{A}_2$ **do**
    if $t_1 \in \mathcal{T}^{end} \wedge t_2 \notin \mathcal{T}^{start}$ **then**
        $\mathcal{T}^{start} \leftarrow \mathcal{T}^{start} \cup \{t_2\}$
    end if
    if $t_1 \notin \mathcal{T}^{end} \wedge t_2 \in \mathcal{T}^{start}$ **then**
        $\mathcal{T}^{end} \leftarrow \mathcal{T}^{end} \cup \{t_1\}$
    end if
end for
for $m = (t_1, \ldots, t_n) \in \mathcal{F}(\mathcal{D})$ **do**
    $in \leftarrow FALSE$
    $temp \leftarrow 0$
    for $t_1, \ldots t_n$ **do**
        if $in = FALSE$ **then**
            if $t_i \notin \mathcal{T}^{start}$ **then**
                $\mathcal{T}^{start} \leftarrow \mathcal{T}^{start} \cup \{t_i\}$
            end if
            $in \leftarrow TRUE, \quad temp \leftarrow \bar{s}_{t_i}$
        end if
        if $t_i \in \mathcal{T}^{end} \vee (\bar{e}_{t_i} - temp) \geq \hat{q}_3 \cdot d_{opt} \vee i = n$ **then**
            if $t_i \notin \mathcal{T}^{end}$ **then**
                $\mathcal{T}^{end} \leftarrow \mathcal{T}^{end} \cup \{t_i\}$
            end if
            $in \leftarrow FALSE$
        end if
    end for
end for
return $(\mathcal{T}^{start}, \mathcal{T}^{end})$

Algorithm 4.4 (FAST-Fix)

Require: Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$, $\mathcal{A}^h_{\hat{x}}$, parameter vector $\hat{q}$

/* identify start and end timetable tasks of pieces */
$(\mathcal{T}^{start}, \mathcal{T}^{end}) \leftarrow$ Generate-Pieces$(\mathcal{D}, \mathcal{A}^h_{\hat{x}}, \hat{q})$
$\tilde{\mathcal{A}} \leftarrow \mathcal{A}_1 \cup \{l = (t_1, t_2) \in \mathcal{A}_2 \mid \bar{v}_{t_1} = \bar{v}_{t_2} \wedge t_1, t_2 \in \mathcal{T} \wedge t_1 \notin \mathcal{T}^{end}\}$
$\tilde{\mathcal{S}} \leftarrow \emptyset$
$s^{short} \leftarrow (0, \ldots, 0) \in \mathbb{N}^{\mathcal{T}}$; $e^{short} \leftarrow (0, \ldots, 0) \in \mathbb{N}^{\mathcal{T}}$
$s^{middle} \leftarrow (0, \ldots, 0) \in \mathbb{N}^{\mathcal{T}}$; $e^{middle} \leftarrow (0, \ldots, 0) \in \mathbb{N}^{\mathcal{T}}$
$s^{long} \leftarrow (0, \ldots, 0) \in \mathbb{N}^{\mathcal{T}}$; $e^{long} \leftarrow (0, \ldots, 0) \in \mathbb{N}^{\mathcal{T}}$
/* add links and supplementary tasks in chosen connections */
for $t \in \mathcal{T}^{start}$ **do**
  for $[\tilde{t}, t]$-path p in $\mathcal{D} : \tilde{t} \in \mathcal{I} \cup \mathcal{T}^{end}$ **do**
    $length \leftarrow (\bar{s}_t - \bar{e}_{\tilde{t}})$
    if $\tilde{t} \in \mathcal{I} \vee length = 0$ **then**
      $\tilde{\mathcal{S}} \leftarrow \tilde{\mathcal{S}} \cup \{\hat{t} \in \mathcal{V} \cap p\}$; $\tilde{\mathcal{A}} \leftarrow \tilde{\mathcal{A}} \cup \{\hat{l} \in \mathcal{A} \cap p\}$
    else if $length \leq \hat{q}_7 \wedge (s^{short}_{\tilde{t}} \leq \hat{q}_4 \vee e^{short}_t \leq \hat{q}_4)$ **then**
      $\tilde{\mathcal{S}} \leftarrow \tilde{\mathcal{S}} \cup \{\hat{t} \in \mathcal{V} \cap p\}$
      $\tilde{\mathcal{A}} \leftarrow \tilde{\mathcal{A}} \cup \{\hat{l} \in \mathcal{A} \cap p\}$
      $s^{middle}_{\tilde{t}} \leftarrow s^{middle}_{\tilde{t}} + 1$; $e^{middle}_t \leftarrow e^{middle}_t + 1$
    else if $length \leq \hat{q}_8 \wedge (s^{middle}_{\tilde{t}} \leq \hat{q}_5 \vee e^{middle}_t \leq \hat{q}_5)$ **then**
      $\tilde{\mathcal{S}} \leftarrow \tilde{\mathcal{S}} \cup \{\hat{t} \in \mathcal{V} \cap p\}$
      $\tilde{\mathcal{A}} \leftarrow \tilde{\mathcal{A}} \cup \{\hat{l} \in \mathcal{A} \cap p\}$
      $s^{short}_{\tilde{t}} \leftarrow s^{short}_{\tilde{t}} + 1$; $e^{short}_t \leftarrow e^{short}_t + 1$
    else if $s^{long}_{\tilde{t}} \leq \hat{q}_6 \vee e^{long}_t \leq \hat{q}_6$ **then**
      $\tilde{\mathcal{S}} \leftarrow \tilde{\mathcal{S}} \cup \{\hat{t} \in \mathcal{V} \cap p\}$
      $\tilde{\mathcal{A}} \leftarrow \tilde{\mathcal{A}} \cup \{\hat{l} \in \mathcal{A} \cap p\}$
      $s^{long}_{\tilde{t}} \leftarrow s^{long}_{\tilde{t}} + 1$; $e^{long}_t \leftarrow e^{long}_t + 1$
    end if
  end for
end for
for $\tilde{t} \in \mathcal{T}^{end}$ **do**
  for $[\tilde{t}, t]$-path p in $\mathcal{D} : t \in \mathcal{I} \cup \mathcal{T}^{start}$ **do**
    $length \leftarrow (\bar{s}_t - \bar{e}_{\tilde{t}})$
    if $\tilde{t} \in \mathcal{I} \vee length = 0$ **then**
      $\tilde{\mathcal{S}} \leftarrow \tilde{\mathcal{S}} \cup \{\hat{t} \in \mathcal{V} \cap p\}$; $\tilde{\mathcal{A}} \leftarrow \tilde{\mathcal{A}} \cup \{\hat{l} \in \mathcal{A} \cap p\}$
    else if $length \leq \hat{q}_7 \wedge (s^{short}_{\tilde{t}} \leq \hat{q}_4 \vee e^{short}_t \leq \hat{q}_4)$ **then**
      $\tilde{\mathcal{S}} \leftarrow \tilde{\mathcal{S}} \cup \{\hat{t} \in \mathcal{V} \cap p\}$; $\tilde{\mathcal{A}} \leftarrow \tilde{\mathcal{A}} \cup \{\hat{l} \in \mathcal{A} \cap p\}$
      $s^{middle}_{\tilde{t}} \leftarrow s^{middle}_{\tilde{t}} + 1$; $e^{middle}_t \leftarrow e^{middle}_t + 1$
    else if $length \leq \hat{q}_8 \wedge (s^{middle}_{\tilde{t}} \leq \hat{q}_5 \vee e^{middle}_t \leq \hat{q}_5)$ **then**
      $\tilde{\mathcal{S}} \leftarrow \tilde{\mathcal{S}} \cup \{\hat{t} \in \mathcal{V} \cap p\}$; $\tilde{\mathcal{A}} \leftarrow \tilde{\mathcal{A}} \cup \{\hat{l} \in \mathcal{A} \cap p\}$
      $s^{short}_{\tilde{t}} \leftarrow s^{short}_{\tilde{t}} + 1$; $e^{short}_t \leftarrow e^{short}_t + 1$
    else if $s^{long}_{\tilde{t}} \leq \hat{q}_6 \vee e^{long}_t \leq \hat{q}_6$ **then**
      $\tilde{\mathcal{S}} \leftarrow \tilde{\mathcal{S}} \cup \{\hat{t} \in \mathcal{V} \cap p\}$; $\tilde{\mathcal{A}} \leftarrow \tilde{\mathcal{A}} \cup \{\hat{l} \in \mathcal{A} \cap p\}$
      $s^{long}_{\tilde{t}} \leftarrow s^{long}_{\tilde{t}} + 1$; $e^{long}_t \leftarrow e^{long}_t + 1$
    end if
  end for
end for
$\tilde{\mathcal{V}} \leftarrow \mathcal{T} \cup \tilde{\mathcal{S}} \cup \mathcal{I}$
return $\tilde{\mathcal{D}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{A}})$

indicates the current connection and let $\hat{l} = \bar{s}_t - \bar{e}_{\tilde{t}}$ be the length of this connection.
If $\hat{l} = 0$ or $\tilde{t} \in \mathcal{I}$, the connection is maintained to ensure feasibility and to keep the block structure. Otherwise, the connection p is identified as a long, middle or short connection and only kept if one of the associated counters has not reached the corresponding limit $\hat{q}_4$, $\hat{q}_5$ or $\hat{q}_6$.
If a connection p is chosen to be maintained, all contained links and supplementary tasks are added to the reduced Duty Scheduling Graph $\tilde{\mathcal{D}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{A}})$.
The second loop in lines 29-45 is similar to the first loop. All connections between piece endings and artificial tasks or piece starting tasks are concerned. Let $p = (\tilde{t}, \dots t)$ again indicate the current path. The path p has not been covered by the first loop if $t \in \mathcal{I}$. However, if $t \in \mathcal{T}^{start}$, p has already been covered by the first loop. Nevertheless, all connections are considered a second time to ensure that the number of outgoing connections for any fixed duty piece equals the limit if possible.
Of course the resulting Duty Scheduling Graph $\tilde{\mathcal{D}} = (\tilde{\mathcal{V}}, \tilde{\mathcal{A}})$ is a subgraph of $\mathcal{D}$. We define the following.

Definition 4.4.2 *Let $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph and let $\bar{r} \in \mathbb{N}$. Further let $\hat{x}$ be a fractional solution of the FAST-Relaxation (**FR**-$\mathcal{D}^{\bar{r}}_{pos}$) and let $\hat{q}$ be a fix-parameter vector. The Duty Scheduling Graph*

$$\mathcal{D}^{(\hat{x},\hat{q})}_{fix} = \textit{FAST-Fix}\ (\mathcal{D}, \mathcal{A}^h_{\hat{x}}, \hat{q})$$

is called the $\hat{x}$-$\hat{q}$-FAST-Fix of $\mathcal{D}$.

Let $\hat{q} = (14, 0.05, 0.7, 14, 9, 4, 5400, 1800)$ be a fix-parameter vector and let $\bar{r} = 1800$ be a rounding value. Figure 4.5 displays the corresponding $\hat{x}$-$\hat{q}$-FAST-Fix for the instance in Figure 4.2.

4.5 The Heuristic FAST($\mathcal{D}$, $\bar{r}$, $\hat{q}$)

We now merge the results from the last three sections to provide an exact description of FAST. Let again $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ be a Duty Scheduling Graph.
Recall from section 4.1 that FAST comprises of two phases. The first phase consists of the construction of the $\bar{r}$-*pos*-Relaxation of $\mathcal{D}$ and of the generation of a solution $\hat{x}$ of the FAST-Relaxation (**FR**-$\mathcal{D}^{\bar{r}}_{pos}$) using RDSP^2. The second phase builds the $\hat{x}$-$\hat{q}$-FAST-Fix, a subgraph $\mathcal{D}^{(\hat{x},\hat{q})}_{fix}$ of the former Duty Scheduling Graph $\mathcal{D}$ containing only a small but critical part. Finally the Duty Scheduling Problem

$$\min\{C(C) \mid C \in \mathcal{C}^{\mathcal{D}^{(\hat{x},\hat{q})}_{fix}}\}$$

is solved using the solver DS-Opt.

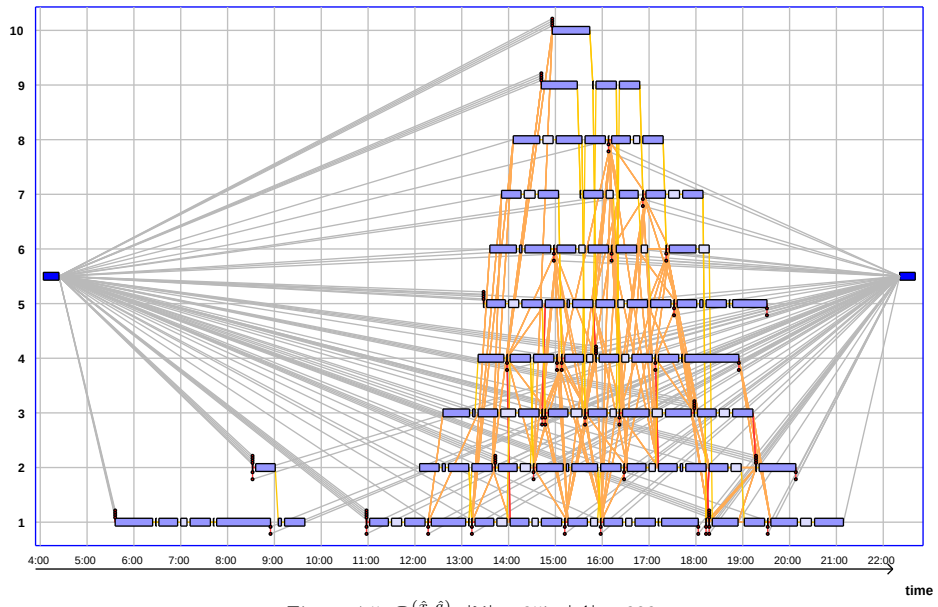

Figure 4.5: $\mathcal{D}_{fix}^{(\hat{x},\hat{q})}$, $|\mathcal{V}| = 257$, $|\mathcal{A}| = 888$

Algorithm 4.5 (FAST)

Require: Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$, rounding value $\bar{r}$, fix-parameter vector $\hat{q}$

1: /* Phase 1: create and solve the FAST-Relaxation */
2: $\mathcal{D}_{pos}^{\bar{r}} \leftarrow$ FAST-Relax$(\mathcal{D}, \bar{r})$
3: $\hat{x} \leftarrow \text{RDSP}^2(\mathcal{D}_{pos}^{\bar{r}})$
4: /* Phase 2: create and solve the FAST-Fix */
5: $h \leftarrow 0.001$
6: $\mathcal{D}_{fix}^{(\hat{x},\hat{q})} \leftarrow$ FAST-Fix$(\mathcal{D}, \mathcal{A}_{\hat{x}}^{h}, \hat{q})$
7: $C \leftarrow$ DS-Opt$(\mathcal{D}_{fix}^{(\hat{x},\hat{q})})$
8: **return** C

An outline of FAST was given in section 4.1 and in Figure 4.1. Algorithm 4.5 is a description of the FAST heuristic.
Computational results are presented in the next chapter.

5. Computational Results

Abstract: FAST has been tested on more than 50 instances of the Duty Scheduling Problem. We present detailed computational results for ten of these instances. Runtimes and objective values of FAST are compared to those of DS-Opt and state over 95% runtime savings for relevant instances. We further investigate the impact of different parameter sets on the size of the $\hat{x}$-$\hat{q}$-FAST-Fix and on the runtime.

5.1 Hardware and Software

The FAST heuristic has been implemented using the programming languages Java 1.5.1 and C++. The test and research version of DS-Opt which is used for comparison is implemented in C++.
All FAST and DS-Opt runs have been performed on an Intel(R) Pentium(R) 4 with a 3.20GHz CPU, 512 KB Cache and 2GB RAM using the system software Suse Linux 10.0.

5.2 Problem Instances $\mathcal{D} = (\mathcal{V}, \mathcal{A})$

For the FAST benchmark, a library of 60 real world instances of the Duty Scheduling Problem representing bus and tram driver scheduling problems of several German and international traffic companies has been set up.
This diversified library contains small instances, consisting of less than 100 tasks and 5,000 links, medium instances, usually Duty Scheduling Graphs with about 10,000 to 30,000 tasks and more than 200,000 links, and large instances consisting of up to 42,724 tasks and 1,776,858 links. A complete list of all 60 instances can be found in Appendix A.
Associated with each instance are different sets of duty types $\mathcal{K}$, which are also listed in the tables, and duty and base resources $\mathcal{U}$ and $\mathcal{W}$. For some instances the set of base resources $\mathcal{W}$ is empty, i.e. no base constraints are defined in this case. There are also cases were base constraints are defined, but do not significantly restrict the mix of duties, see section 1.1. Some of the instances are based upon the same bus or tram network.
For simple examples the number of different duty types is rather small, i.e. $1 \leq |\mathcal{K}| \leq 3$, while other instances correspond to up to 13 different duty types. Instances with more different duty types are not necessarily harder to solve.

DSP - Instances										
Instance	$\lvert\mathcal{K}\rvert$	$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{V}\setminus\mathcal{I}\rvert$	$\lvert\mathcal{A}_1\rvert$	$\lvert\mathcal{A}_2\rvert$	$\lvert\mathcal{A}_3\rvert$	$\lvert\mathcal{A}_4\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{F}\rvert$
ivu05	3	1561	2946	4507	714	3751	65134	3658	73257	42
ivu27	3	2191	5949	8140	1538	6578	35299	4760	48175	26
ivu58	1	694	3611	4305	236	4060	5748	2702	12746	11
ivu06	3	2187	4038	6225	1010	5164	123235	5044	134453	372
ivu07	3	3269	3269	6538	1519	7664	275251	7492	291926	423
ivu10	6	873	55443	56316	622	55688	17916	46351	120577	8
ivu08	3	5021	7673	12694	2499	10064	552733	10158	575454	500
ivu29	4	2247	23808	26055	1183	24818	231440	14659	272100	56
ivu35	4	2247	22035	24282	1183	23045	223084	14659	261971	56
ivu39	4	1182	22885	24067	299	23749	78421	14088	116557	21

Table 5.1: Set of 10 instances $\mathcal{D} = (\mathcal{V}, \mathcal{A})$

All but four of the instances in the library have been run with DS-Opt and 45 have been solved successfully. Five instances with a computation time of more than three days for our research version of DS-Opt have been aborted. For six instances DS-Opt failed to generate a feasible duty schedule. A complete list of all DS-Opt solutions and runtimes is also contained in Appendix A.

Although FAST has successfully solved three of the large instances which could not be solved in a reasonable time by DS-Opt, we only survey instances also solved by DS-Opt for the runtime analysis of FAST.

For the FAST benchmark a subset of 35 of the instances has been selected. All of these have been solved multiple times regarding different FAST fix-parameter vectors and rounding values. Complete results are again part of Appendix A, while results for selected instances are outlined in the next section.

It is further necessary to distinguish between instances. We call an instance *relevant*, if the runtime of DS-Opt exceeded 40 minutes. Of the 35 chosen instances, 23 are relevant. FAST runtimes for not relevant instances are not expected to be significantly lower than the corresponding DS-Opt runtime, as these instances can already be solved efficiently using DS-Opt.

Table 5.1 lists a set of ten instances picked from the 23 relevant instances along with the number of tasks, links and blocks. The set of artifical tasks $\mathcal{I}$ is not part of the problem data but is generated prior to the solution process. Results for these ten instances are discussed in more detail in the following sections.

Table 5.2 lists the runtime and objective value of DS-Opt solutions for the ten chosen instances. As all solutions do not contain any slacks, they indeed represent feasible duty schedules. The DS-Opt runtime for all instances exceeds 40 minutes and thus all instances are relevant.

DS-Opt Solutions						
Instance	Value	$\|C\|$	Runtime	Duty Pieces	Slacks	EstLowBound
ivu05	97.75	73	00:59:15	155	0	97.51
ivu27	69.14	49	02:51:42	125	0	68.95
ivu58	27.75	14	00:58:51	27	0	27.64
ivu06	137.72	106	07:32:55	461	0	137.37
ivu07	190.15	146	10:25:29	584	0	189.95
ivu10	27.26	19	02:38:33	61	0	27.48
ivu08	315.42	232	09:29:21	713	0	314.73
ivu29	67.16	61	19:09:34	225	0	66.89
ivu35	115.23	62	06:24:17	180	0	114.92
ivu39	32.41	27	04:07:14	49	0	32.25

Table 5.2: DS-Opt solutions for instances $\mathcal{D} = (\mathcal{V}, \mathcal{A})$

5.3 Results

We now discuss the results obtained during the FAST benchmark and present the solutions for the ten instances specified in the previous section. Complete computational results, solutions and averages are listed in Appendix A.

After an analysis of the impact of different parameters on runtime and solution quality and with respect to the experience gained from more than 500 FAST runs performed before the actual benchmark started, six different parameter sets have been chosen for the FAST benchmark. According to these parameter sets we define the following six versions of FAST used during the benchmark.

- FAST-ver1: $\bar{r} = 2400,\ \hat{q} = (14, 0.05, 0.85, 13, 8, 3, 1800, 5400)$
- FAST-ver2: $\bar{r} = 2400,\ \hat{q} = (13, 0.04, 0.7, 14, 9, 4, 1800, 5400)$
- FAST-ver3: $\bar{r} = 1800,\ \hat{q} = (15, 0.2, 0.8, 12, 8, 4, 1800, 5400)$
- FAST-ver4: $\bar{r} = 1800,\ \hat{q} = (14, 0.05, 0.7, 14, 9, 4, 1800, 5400)$
- FAST-ver5: $\bar{r} = 2400,\ \hat{q} = (15, 0.2, 0.8, 12, 8, 4, 1800, 5400)$
- FAST-ver6: $\bar{r} = 3000,\ \hat{q} = (20, 0.3, 0.9, 10, 4, 2, 1800, 5400)$

Some issues on the impact of different parameters which lead to these six final parameter sets are discussed in the next section.

All 35 instances have been solved with all six FAST versions. Also the results for non-relevant instances have been included in this analysis. Averages only regarding relevant instances are presented for the best of the six versions later. We give a short overview on the performance of the six versions compared to DS-Opt below.

FAST-ver1 achieved the second best results of all six versions using only 7.98% in average of the DS-Opt runtime while almost maintaining the objective value. More precisely FAST-ver1 reported an average 3.34% increase in the objective value. This version failed to generate feasible duty schedules for instances *ivu34* and *ivu51*.

The second version FAST-ver2 reported the best objective values, averaging just a 2.41% increase. Furthermore feasible schedules have

been found for all instances. However, the achieved runtimes were significantly weaker than those of FAST-ver1, as 12.48% of the original runtime was averaged.

FAST-ver4 uses similar fix parameters but a lower rounding parameter than FAST-ver2. It performed worse than FAST-ver2, resulting in higher runtimes, 14.20% in average, and higher objectives, 2.65% increase in average. It also failed to find a feasible duty schedule for instance *ivu51*.

FAST-ver6 has been chosen using extreme values for all parameters. As expected, this resulted in the highest reduction, the average size of the fixed Duty Scheduling Graph $\mathcal{D}_{fix}^{(3000,(20,0.3,0.9,10,4,2,1800,5400))}$ was only 14.80% of the original size, and in very fast average runtimes of 5.98%. Furthermore unexpectly all but two instances could be solved and the average objective value increase was only 4.52%.

The last two versions FAST-ver3 and FAST-ver5 reported average results compared to the versions discussed above. For more details please refer to Appendix A.

Recall, that the objective of FAST is the generation of very fast solutions and that thus FAST-ver6, which reported significantly faster solutions than the other versions, has achieved the best results.

On the set of relevant instances, all versions provided far better results. The average runtime of FAST-ver6 on the 23 relevant instances was 4,59% while the average objective increase could be lowered to 3.48%. The size of the fixed graph was only 12.72% of the original size. Obviously FAST performs much better on large instances.

Figure 5.1 and Figure 5.2 visualize the runtimes and the objective values of all six FAST versions and DS-Opt.

Of the 23 relevant instances twelve could be solved by FAST-ver6 using less than 5% of the DS-Opt runtime, while five could even be solved by less than 2%. Instance *ivu28* with a DS-Opt runtime of more than 19 hours could be solved in 48 seconds, i.e. in 0.06% of the DS-Opt runtime. For some instances FAST did not achieve a runtime saving as huge as indicated by the average. Instance *ivu30* needed 59.48% of the DS-Opt runtime. A total of five of the 23 instances needed more than 20%. However, only two of these needed more than 23%.

Although maintaining the objective value is only the second goal, it is neat that 13 of the 23 instances report an objective increase of less than 2%.

To summarize these results, we list the FAST-ver6 solution for the ten selected instances in Table 5.3. Our statements are always based on the geometrical average of all percentages. The solution tables presented in this thesis also include the arithmetic average and the total percentage of size, value and runtime of all listed instances. However the geometrical average is the best choice to compare the results in our case.

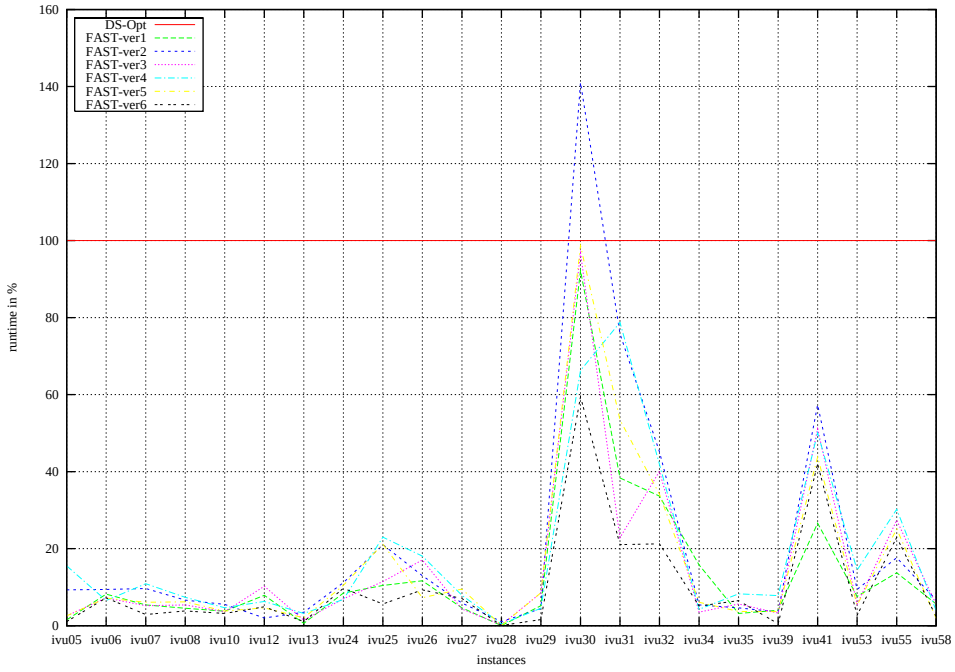

Figure 5.1: Runtimes for all FAST versions

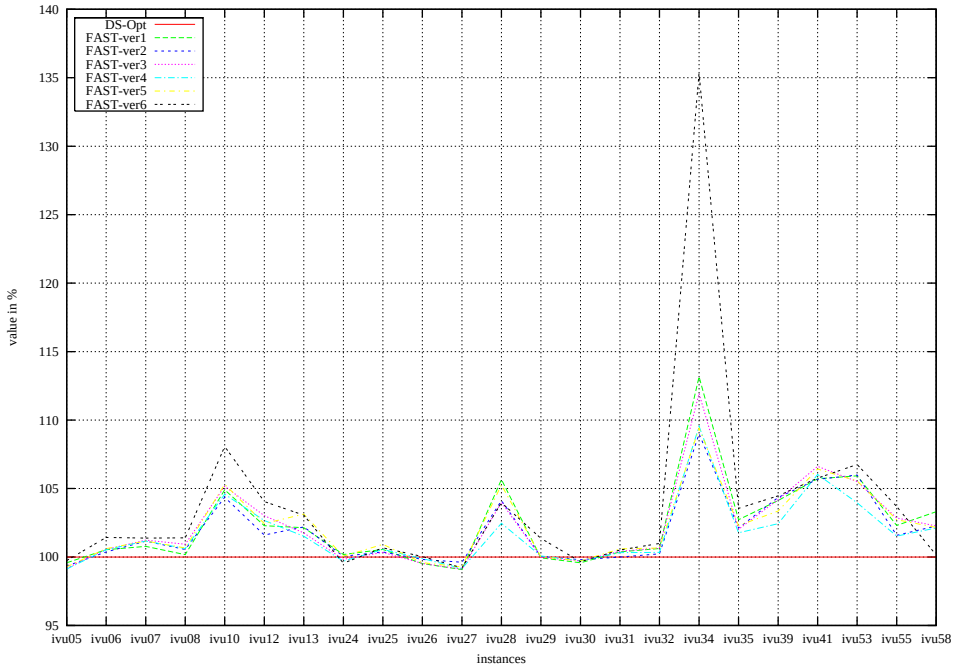

Figure 5.2: Objective values for all FAST versions

FAST - Solutions - $\bar{r} = 3000$ $\hat{q} = (20, 0.3, 0.9, 10, 4, 2, 1800, 5400)$

	$\mathcal{D} = (\mathcal{V}, \mathcal{A})$		$\mathcal{D}^{\bar{r}}_{pos}$		$\mathcal{D}^{(\bar{r},\hat{q})}_{fix}$			Solution					
Instance	$\lvert\mathcal{V} \setminus \mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{V} \setminus \mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{V} \setminus \mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$	Size%	$\lvert C\rvert$	Slacks	Value	Value%	Runtime	Runtime%
ivu05	4507	73257	267	1049	2235	5966	8.14	71	0	97.5	99.74	00:00:38	1.06
ivu27	8140	48175	209	752	4721	10728	22.26	48	0	68.6	99.21	00:12:08	7.06
ivu58	4305	12746	269	567	1539	2459	19.29	14	0	27.79	100.14	00:01:23	2.35
ivu06	6225	134453	295	1105	3953	17429	12.96	108	0	139.69	101.43	00:32:50	7.24
ivu07	6538	291926	316	1322	5456	23249	7.96	149	0	192.78	101.38	00:18:14	2.91
ivu10	56316	120577	1065	2356	24938	46691	38.72	21	0	29.46	108.07	00:05:03	3.18
ivu08	12694	575454	408	2145	7724	32638	5.67	238	0	319.85	101.40	00:21:51	3.83
ivu29	26055	272100	592	2954	10972	25002	9.18	62	0	68.08	101.36	00:18:15	1.58
ivu35	24282	261971	711	3227	10343	24421	9.32	64	0	119.33	103.55	00:25:03	6.51
ivu39	24067	116557	1579	4037	5011	9208	7.89	28	0	33.86	104.47	00:01:18	0.52
Arith-Ave							14.14				102.07		3.63
Geom-Ave							11.82				102.04		2.75
Total %							10.37				101.56		3.52

Table 5.3: FAST solutions for 10 chosen instances

5.4 Parameter Analysis

The parameter sets for the FAST versions used in the benchmark for which results are presented in the previous section have been selected after an analysis of the impact of different parameter sets on the runtime. During this analysis more than 1,000 FAST runs have been performed and compared to each other. We discuss the results of this parameter analysis in this section.

5.4.1 The Rounding Parameter $\bar{r}$

The rounding parameter $\bar{r}$ is used to round starting and ending times of tasks. It directly affects the number of tasks and links contained in the $\bar{r}$-*pos*-Relaxation of a Duty Scheduling Graph $\mathcal{D} = (\mathcal{V}, \mathcal{A})$. Obviously, choosing a high value for the parameter $\bar{r}$ should result in a small $\bar{r}$-*pos*-Relaxation and thus the FAST-Relaxation should be solved faster. On the other hand the $\bar{r}$-*pos*-Relaxation needs to retain the essential structures of the problem. Choosing a high rounding parameter $\bar{r}$ obviously discards much of this information. Thus $\bar{r}$ must be selected in a manner which allows fast runtimes but still keeps enough information on the problem structure.
In our analysis we used roundings from five minutes up to more than one hour. The value is given in seconds, i.e. $\bar{r} = 300$ means that all tasks are rounded to the nearest five minute mark. All tested instances showed similar results for different rounding parameters.
Table 5.4 shows different FAST solutions and the corresponding runtimes regarding different rounding parameters $\bar{r}$ for *ivu05* which is a medium size bus driver scheduling instance. All other parameters are equal for all runs.
The runtime for small values $\bar{r} < 900$ does not satisfy the target of less than 7% of the DS-Opt runtime. For a rounding parameter of $\bar{r} = 300$ the size of the $\hat{x}$-$\hat{q}$-FAST-Fix is still 22.21% of the original problem. Recall that the average size achieved by FAST-ver6 on relevant instances was 12.72%. A further increase of $\bar{r}$ is necessary. Although it is not our aim to keep the objective value of DS-Opt, it is welcome that the solutions for this instance are even better than the solutions provided by DS-Opt. As we further increase the rounding parameter, the size of the fix levels off at 10% of the original size. This is due to the fact that each link contained in the relaxation represents many more original links and thus more links may be fixed because they attain the critical value or count. Runtimes are satisfying for values $\bar{r} = 1,800$ and $\bar{r} = 3,000$. In general also a rounding parameter $\bar{r} = 2,400$ produced great runtimes, though for instance *ivu05* the runtime was 13.33% with the used parameter set.
Figure 5.3 shows the size of the $\bar{r}$-*pos*-Relaxation and the $\hat{x}$-$\hat{q}$-FAST-Fix regarding different rounding parameters $\bar{r}$. It is obvious that the size of the $\bar{r}$-*pos*-Relaxation steadily declines while the size of the $\hat{x}$-$\hat{q}$-FAST-

FAST - ivu05													
	$\mathcal{D} = (\mathcal{V}, \mathcal{A})$		$\mathcal{D}^{\bar{r}}_{pos}$		$\mathcal{D}^{(\bar{r},\hat{q})}_{fix}$						Solution		
$\bar{r}$	$\lvert\mathcal{V} \setminus \mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{V} \setminus \mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{V} \setminus \mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$	Size%	$\lvert C\rvert$	Slacks	Value	Value%	Runtime	Runtime%
300	4507	73257	2071	18822	3065	16273	22.21	71	0	96.76	98.98	00:18:06	30.54
600	4507	73257	1137	9817	2673	11331	15.46	72	0	97.07	99.30	00:13:06	22.10
900	4507	73257	825	8290	2689	11456	15.63	72	0	96.82	99.04	00:04:30	7.59
1200	4507	73257	635	5875	2617	10880	14.85	72	0	97.01	99.24	00:02:55	4.92
1500	4507	73257	530	4575	2467	9382	12.80	72	0	97.13	99.36	00:03:11	5.37
1800	4507	73257	439	3425	2371	8592	11.72	72	0	96.91	99.14	00:02:45	4.64
2100	4507	73257	384	2663	2369	8663	11.82	71	0	97.08	99.31	00:01:22	2.30
2400	4507	73257	311	1932	2299	8008	10.93	72	0	97.18	99.41	00:07:54	13.33
2700	4507	73257	288	1469	2289	7816	10.66	72	0	97.33	99.57	00:03:59	6.72
3000	4507	73257	268	1049	2281	7844	10.70	72	0	97.4	99.64	00:02:28	4.16
3300	4507	73257	236	733	2281	7823	10.67	72	0	97.29	99.52	00:01:35	2.67
3600	4507	73257	226	586	2281	7903	10.78	72	0	97.21	99.44	00:00:60	1.68

Table 5.4: Analysis of the rounding parameter $\bar{r}$ for problem ivu05

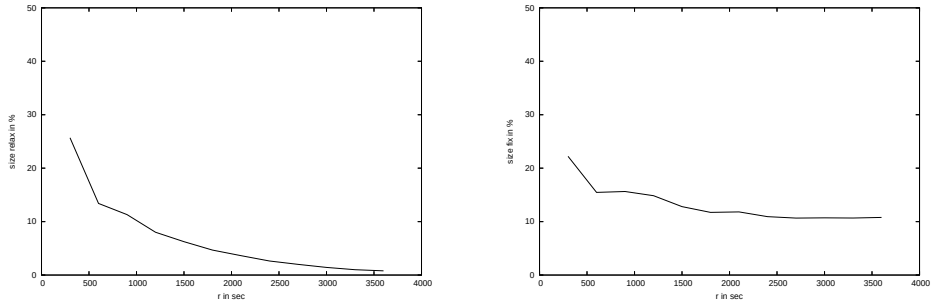

Figure 5.3: Size of $\bar{r}$-*pos*-Relaxation and $\hat{x}$-$\hat{q}$-FAST-Fix of instance ivu05 for different rounding parameters $\bar{r}$

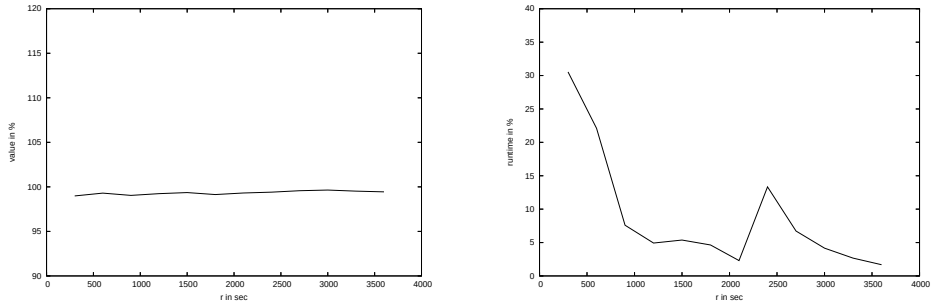

Figure 5.4: FAST runtimes and objective values for instance ivu05 and different rounding parameters $\bar{r}$

Fix closes in on the 10% mark.
The development of runtime and objective values for *ivu05* and different rounding parameters is displayed in Figure 5.4. The objective values do not change much. Rounding parameters in the range of $1,200 - 2,100$ and $2,700 - 3,600$ produced satisfying runtime savings.
Although rounding parameters $\bar{r} > 3,000$ show very fast runtimes for this instance, some instances could not be solved because too much details have been dropped. Oddly no substantial increase in the objective value occurs in any of the instances using higher rounding parameters. A rather high rounding still carries enough information on the structure of the problem to generate an efficient fix of the problem.
Best overall results have been achieved using values $\bar{r} = 1800$ and $\bar{r} = 2400$, i.e. rounding all task with respect to a 30 or 40 minute time pattern. Therefore these values have been chosen for the FAST benchmark.

5.4.2 The Fixing Parameters $\hat{q}$

The set of fixing parameters contains six important values. All six parameters have an impact on the choice of links contained in the $\hat{x}$-$\hat{q}$-FAST-Fix.
The first two parameters $\hat{q}_1$ and $\hat{q}_2$ control, which links are used to determine the duty pieces, see section 4.4.1. For our analysis we used lower bounds for fixing links in the ranges $0.04 \leq \hat{q}_2 \leq 0.2$ and $10 \leq$

$\hat{q}_1 \leq 20$. Obviously the possible combinations for these values are huge. In general, low values in this range ensure that less links are considered during the block partition and thus less duty pieces are fixed. Each duty piece consists of more timetable tasks and many more links are discarded. However, if the lower bounds are set too high, this may result in the loss of some critical relief opportunities and thus in much longer runtimes and higher objective values. On the other hand choosing very low bounds ends up in many duty pieces and less reduction in the fix, i.e. longer FAST runtimes. The six FAST versions defined for the benchmark use different values $\hat{q}_2$ and $\hat{q}_1$ out of the range specified above. FAST-ver6 even uses an extreme value of $\hat{q}_2 = 0.3$.
The value $\hat{q}_3$ controls the maximum duration of the fixed duty pieces. Any fixed duty piece with a longer duration as indicated by $\hat{q}_3 \cdot d_{opt}$, is prohibited and cut into smaller pieces. d_{opt} indicates the optimal duty piece duration. Thus parameter $\hat{q}_3$ controls the number and size of the duty pieces contained in the fix. We choose to set this parameter between 0.7 and 1.2. For values of $\hat{q}_3 < 0.7$ many of the fixed duty pieces needed to be cut into smaller pieces. As a result the reduction achieved by choosing high lower bounds $\hat{q}_1$ and $\hat{q}_2$ is reversed. On the other hand, if the value $\hat{q}_3$ is set too high, it is likely that the reduced problem is infeasible because very long duty pieces are fixed and a feasible duty schedule covering all duty pieces is much harder to find. Eventually the duty pieces cannot be properly matched to form duties.
The last three parameters $\hat{q}_4$, $\hat{q}_5$ and $\hat{q}_6$ define optimal values for the number of incoming and outgoing short, middle and long connections between the fixed duty pieces.
Choosing low optimal values for these parameters results in many less path connections and thus less links and supplementary tasks are contained in the $\hat{x}$-$\hat{q}$-FAST-Fix. However, if to few connections are included, the problem may become infeasible. High values on the other hand allow many more connections but also result in longer runtimes because the set of possible duties $\mathcal{P}$ is not reduced to a reasonable size. In order to attain satisfying runtimes for large instances, parameters $8 \leq \hat{q}_4 \leq 18$, $5 \leq \hat{q}_5 \leq 12$ and $2 \leq \hat{q}_6 \leq 5$ have been chosen.
Table 5.5 lists ten different fix parameter sets. For the first sets the fix parameters have been selected such that the reduction is rather small and not many details are dropped. The amount of details dropped increases through the sets one to ten. The last set should result in an enormous reduction and very fast runtimes.
Instance *ivu05* has been run in FAST with all parameter sets. Figure 5.5 displays the results. Only set ten did not result in a feasible duty schedule.

FixPar - Analyse		
	$\bar{r}$	$\hat{q}$
set 1	3000	(8.0, 0.01, 0.3, 20.0, 10.0, 8.0, 1800, 5400)
set 2	3000	(10.0, 0.03, 0.5, 16.0, 8.0, 6.0, 1800, 5400)
set 3	3000	(12.0, 0.04, 0.7, 14.0, 8.0, 6.0, 1800, 5400)
set 4	3000	(13.0, 0.05, 0.8, 14.0, 8.0, 4.0, 1800, 5400)
set 5	3000	(14.0, 0.07, 0.85, 14.0, 6.0, 4.0, 1800, 5400)
set 6	3000	(15.0, 0.1, 0.9, 12.0, 6.0, 4.0, 1800, 5400)
set 7	3000	(17.0, 0.15, 0.95, 12.0, 6.0, 2.0, 1800, 5400)
set 8	3000	(18.0, 0.2, 1.0, 12.0, 6.0, 2.0, 1800, 5400)
set 9	3000	(19.0, 0.25, 1.1, 11.0, 5.0, 2.0, 1800, 5400)
set 10	3000	(20.0, 0.3, 1.2, 10.0, 4.0, 2.0, 1800, 5400)

Table 5.5: Fix parameter sets for analyse

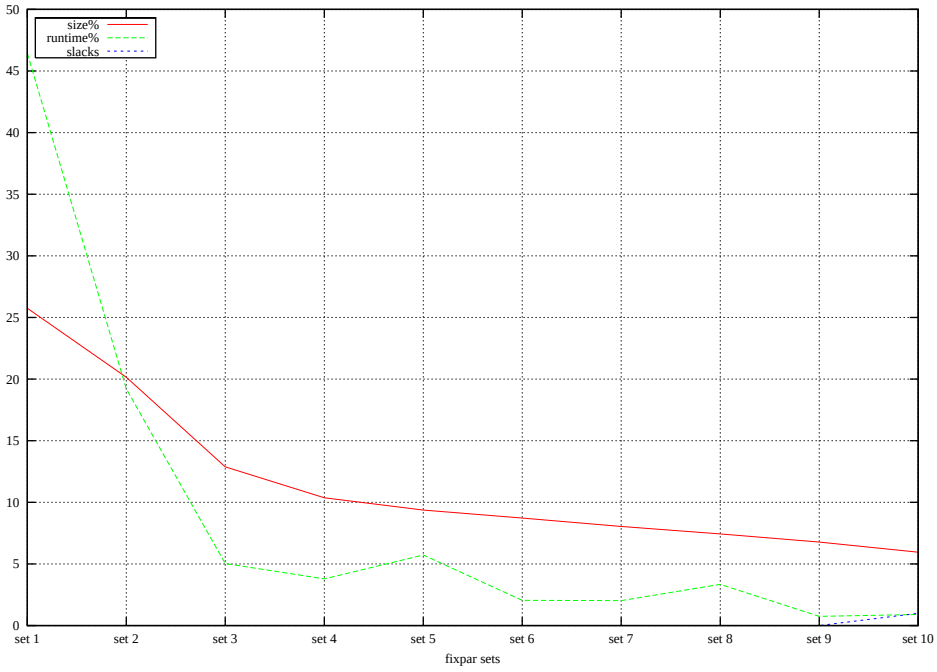

Figure 5.5: Analysis of different fix parameter sets

6. Conclusions

Abstract: We evaluate the results achieved with FAST and suggest some matters for enhancement and further research.

State-of-the-art solvers for the Duty Scheduling Problem, e.g. the approximation solver DS-Opt, produce satisfying solutions but take a long time to solve large instances and sometimes even fail to generate feasible duty schedules. The aim of this thesis was to develop a fast heuristic for duty scheduling which can be used to generate initial solutions or even as a standalone application.

FAST, the heuristic proposed in this thesis, uses a two phase approach toward duty scheduling implemented likewise in the duty scheduling heuristic HASTUS-Micro, see chapter 3. In a first phase, a relaxation of the Duty Scheduling Problem is solved on a modified Duty Scheduling Graph. Subsequently the relaxed solution is used to choose duty piece partitions, see section 1.2, for each block contained in the original Duty Scheduling Graph. A set of critical links is identified and the fixed duty pieces are chosen with respect to these links. A fixed Duty Scheduling Graph is obtained as links and supplementary tasks not associated with the selected partitions are discarded. The Duty Scheduling Problem defined on the so reduced graph can be solved decisively faster.

Six versions of FAST based upon six different parameter sets indicating different reduction levels, see section 5.3, achieved remarkable results on a large set of bus and tram scheduling instances. Compared to DS-Opt, solutions of the best FAST version report an average of 95.40% runtime savings with only a 3.48% average objective increase. These results topped the aimed percentages of about 90% and 5%.

Due to the immense problem reduction performed in the second phase of FAST, in average only 12.72% of the original links remained, currently no feasible solution can be guaranteed. However cases where no feasible duty schedule was found were uncommon. This statement holds even for high reduction levels. More precisely the best FAST version generated feasible duty schedules for all but one of the 23 relevant instances and all but two of all 35 instances used in the benchmark.

The average runtime savings of FAST were 95.40%. Still for some of the tested instances no reasonable savings could be achieved. Reasons for this observation are currently unclear, but should be found in the complexity of these instances. Eventually the reduction deletes some

critical driver relief opportunities such that the lessening has only a small effect on the runtime because good solutions are hard to be found. However, only five of the 23 relevant instances required a runtime of more than 10% while eleven even stayed below the 5% mark.
These results show that the approach used in FAST heuristic is successful and that FAST heavily outperforms DS-Opt in runtime for all instances in the test set. Clearly even for high reduction levels enough information on the problem structure is maintained in the relaxation. A fractional solution of the relaxation suffices as a blueprint for a good duty schedule. The idea to choose the set of critical links individually rather than considering whole duties in the fractional duty schedule proved to be very efficient.

6.1 Further Research

It is even more notably, that these results have been achieved although the current implementation of FAST uses only basic strategies. Further research is required to enhance FAST, to ensure feasible duty schedules and to adapt the heuristic to similar problems such as the Airline Crew Scheduling Problem.
The issues concerning infeasible duty schedules and low runtime savings for some instances can be tackled by a simple adaption. If no feasible duty schedule could be generated by FAST, then some of the fixed duty pieces cannot be covered. Such duty pieces are then split into smaller pieces and adjacent links are reinserted. DS-Opt is then run again on the resulting Duty Scheduling Graph. Inserting this *recut* operation as a loop at the end of the FAST heuristic leads to the original problem in the worst case. Thus it ensures feasible duty schedules if the original problem was feasible. Furthermore such a recut can also be executed if the approximated runtime of the DS-Opt run on the fixed Duty Scheduling Graph exceeds a specific limit.
The current version of FAST is based on a single rounding parameter used to round start and end times of all tasks. An interesting suggestion is to use additive smaller rounding parameters for morning and afternoon peaks. Such a rounding would keep more essential details in the peak hours while most of the structure in between is discarded.
The largest value used for the rounding parameter during the FAST benchmark was $3,600$. Such high rounding parameters resulted in ultimately fast runtimes but seldom feasible duty schedules could be constructed as far too many details are dropped. Too many links from the original problem are discarded. However, if a process would be executed before calling DS-Opt in the second phase of FAST, which reinserts some links into the graph such that feasibility is ensured if possible, such high rounding parameters and therefore faster runtimes would be possible.
Another improvement can be achieved during the selection of critical

links. The currently used method is very simple as all links with a sufficient presence in the relaxed solution are considered critical, see section 4.4.2. Eventually this step could be enhanced as not all but only a selected set of these links is fixed, e.g. expensive links which can be substituted by cheaper links are removed.
In the current version of FAST, the level of reduction has to be chosen in advance. However, for large instances, higher reduction levels might be necessary to achieve fast runtimes while for smaller instances such high levels result in infeasibility. Thus, it is an interesting suggestion to choose the reduction level, i.e. the rounding parameter and the fix parameters, according to the size of the instance. Hopefully such an adaption would also support the aim to ensure feasibility of the duty schedule generated by FAST.
Finally, the approach used in FAST is a general problem reduction approach which can easily be adapted to similar optimization problems. The problem of scheduling airline crews is closely related to the problem of duty scheduling. Thus we complete this chapter with the final indiction that some of the suggestions mentioned above could be implemented and that FAST could also be adjusted to suit the Airline Crew Scheduling Problem.

List of Algorithms

List of Figures

List of Tables

Bibliography

[1] GIRO Inc. website. `http://www.giro.ca`.

[2] E. Aarts and J.K. Lenstra. *Local Search in Combinatorial Optimization*. Wiley, Chichester, 1997.

[3] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H. Vance. Branch-and-Price: Column Generation for Solving Huge Integer Programs. pages 186–207. University of Michigan, 1994.

[4] J.E. Beasley and B. Cao. A dynamic programming based algorithm for the crew scheduling problem. *Computers and Operations Research*, 25(7):567–582, 1998.

[5] Michael Beck. Ein evolutionärer Algorithmus zur Dienstplanung mit mehrfacher Rückkopplung. In Manfred Boltze, editor, *Heureka '05 - Optimierung in Verkehr und Transport*, pages 513–521. Forschungsgesellschaft für Strassen- und Verkehrswesen, 2005.

[6] Jean-Yves Blais and Jean-Marc Rouseau. Hastus: A model for the ecomomic evaluation of drivers collective agreements in transit companies. *Publication 163, Centre de recherche sur les transports, Universite de Montreal*, 1980.

[7] Ralf Borndörfer, Martin Grötschel, and Andreas Löbel. Scheduling duties by adaptive column generation. *ZIB Report 01-02*, 2001.

[8] Ralf Borndörfer, Andreas Löbel, Uwe Strubbe, and Manfred Völker. Zielorientierte Dienstplanoptimierung. *ZIB Preprint SC 98-41*, 1998.

[9] Ralf Borndörfer, Andreas Löbel, and Steffen Weider. Integrierte Umlauf- und Dienstbildung im öffentlichen Nahverkehr. In *Heureka '02 - Optimierung in Verkehr und Transport*, pages 63–76. Forschungsgesellschaft für Strassen- und Verkehrswesen, 2002. available as ZIB preprint ZR 02-10.

[10] Ralf Borndörfer, Andreas Löbel, and Steffen Weider. A Bundle Method for Multi-Depot Integrated Vehicle and Crew Scheduling in Public Transport. *ZIB Preprint ZR 04-14*, 2004.

[11] Ralf Borndörfer, Uwe Schelten, Thomas Schlechte, and Steffen Weider. A Column Generation Approach to Airline Crew Scheduling. *ZIB-Report 05-37*, 2005.

[12] Luis Cavique, Cesar Rego, and Isabel Themido. New heuristic problems for the crew scheduling problem. In Stefan Voß, editor,

Meta heuristics: advances and trends in local search paradigms for optimization, pages 37–47. Kluwer, Boston, 1999.

[13] Văsek Chvátal. *Linear Programming*. Freeman, New York, 1983.

[14] R. Clement and A. Wren. Greedy Genetic Algorithms, Optimizing Mutations and Bus Driver Scheduling. In Daduna et al. [16], pages 213–235.

[15] Joachim R. Daduna and Miodrag Mojsilovic. Computer-Aided Vehicle and Duty Scheduling using the HOT Programme System. In Daduna and Wren [17], pages 133–146.

[16] J.R. Daduna, I. Branco, and J.M.P. Paixao, editors. *Computer-Aided Transit Scheduling*, volume 430 of *Lecture Notes in Economics and Mathematical Systems*. Springer, Berlin, 1995.

[17] J.R. Daduna and A. Wren, editors. *Computer-Aided Transit Scheduling*, volume 308 of *Lecture Notes in Economics and Mathematical Systems*. Springer, Berlin, 1988.

[18] S. Voß J.R. Daduna, editor. *Computer-Aided Scheduling of Public Transport*, volume 505 of *Lecture Notes in Economics and Mathematical Systems*. Springer, Berlin, 2000.

[19] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.

[20] M. Desrochers, Gilbert, Sauve, and Soumis. CREW-OPT: Subproblem modeling in a column generation approach to urban crew scheduling. In Desrochers and Rousseau [21], pages 395–406.

[21] M. Desrochers and J.-M. Rousseau, editors. *Computer-Aided Transit Scheduling*, volume 386 of *Lecture Notes in Economics and Mathematical Systems*. Springer, Berlin, 1992.

[22] Martin Desrochers and Francois Soumis. CREW-OPT: Crew Scheduling by Column Generation. In Daduna and Wren [17], pages 83–90.

[23] J. Desrosiers and J.-M. Rousseau. Results obtained with Crew-opt: A Column Generation Method for Transit Crew Scheduling. In Daduna et al. [16], pages 349–358.

[24] I. Elhallaoui, D. Villeneuve, F. Soumis, and G. Desaulniers. Dynamic Aggregation of Set Partitioning Constraints in Column Generation. *Operations Research*, 53(4):632–645, 2005.

[25] Proll Fores and A. Wren. An Improved ILP System for Driver Scheduling. In Wilson [49], pages 43–62.

[26] F. Glover. Tabu Search: Part 1. *ORSA Journal on Computing 1*, pages 190–206, 1989.

[27] F. Glover. Tabu Search: Part 2. *ORSA Journal on Computing 2*, pages 4–32, 1990.

[28] David E. Goldberg. *Genetic algorithms in search, Optimization and maschine learning*. Addison-Wesley, Massachussets, 1989.

[29] Martin Groetschel. Skriptum Graphen und Netzwerkalgorithmen. http://www.zib.de/groetschel/teaching/materials.html, 2004.

[30] Martin Groetschel. Skriptum Lineare Optimierung. http://www.zib.de/groetschel/teaching/materials.html, 2004.

[31] T. Hartley. A glossary of terms in bus and crew scheduling. In Wren [50], pages 353–359.

[32] D. Huisman, R. Freling, and A.P.M. Wagelmans. Multiple-Depot Integrated Vehicle and Crew Scheduling. *Economic Institute Report EI2003-02*, 2003.

[33] A.S.K. Kwan, R.S.K. Kwan, M.E. Parker, and A. Wren. Producing train driver schedules under different operating strategies. In Wilson [49], pages 129–154.

[34] R.S.K. Kwan, A. Wren, and A.S.K. Kwan. Hybrid genetic algorithms for scheduling bus and train drivers. In N.H.M. Wilson, editor, *Proceedings of the 2000 Congress on Evolutionary Computation*, volume 471, pages 285–292. Springer, 2000.

[35] Réjean Lessard, Jean-Marc Rousseau, and Daniel Dupuis. Hastus 1: A mathematical programming approach to the bus driver scheduling problem. In Wren [50].

[36] S. Martello and P. Toth. A heuristic approach to the bus driver scheduling problem. *European Journal of Operational Research*, 24:106–117, 1986.

[37] Miodrag Mojsilovic. Verfahren für die Bildung von Fahrzeugumläufen, Dienstplänen und Dienstreihenfolgen. In *Heureka '83 - Optimierung im Verkehr und Transport*, pages 174–191. Forschungsgesellschaft für Strassen- und Verkehrswesen, 1983.

[38] M.E. Parker, A. Wren, and R.S.K. Kwan. Modelling the Scheduling of Train Drivers. In Daduna et al. [16], pages 359–370.

[39] C.R. Reeves. *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell, Oxford, 1993.

[40] J.-M. Rousseau, editor. *Computer Scheduling of public transport 2*. North-Holland, Amsterdam, 1985.

[41] J.-M. Rousseau, R. Lessaard, and J.-Y. Blais. Enhancements to the HASTUS crew scheduling algorithm. In Rousseau [40], pages 295–310.

[42] D.M. Ryan. ZIP: A zero-one integer programming package for scheduling.

[43] Thomas Schlechte. Das Resource-Constrained-Shortest-Path-Problem und seine Anwendung in der ÖPNV Dienstplanung. Master's thesis, TU Berlin, 2003.

[44] Alexander Schrijver. *Theory of Linear and Integer Programming*. Wiley, New York, 1986.

[45] Yindong Shen and Raymond S.K. Kwan. Tabu search for driver scheduling. In Daduna [18], pages 121–135.

[46] B.M. Smith and A. Wren. A bus crew scheduling system using a set covering formulation. *Transportation Research*, 22A:97–108, 1988.

[47] Manfred Völker and Peter Schütze. Recent Developments of HOT II. In Daduna et al. [16], pages 334–348.

[48] D. Wedelin. The design of a 0-1 integer optimizer and its application in the carmen system. *Europ. J. on Operations Research*, 87:722–730, 1995.

[49] N.H.M. Wilson, editor. *Computer-Aided Transit Scheduling*, volume 471 of *Lecture Notes in Economics and Mathematical Systems*. Springer, Berlin, 1998.

[50] A. Wren, editor. *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling*. North-Holland, Amsterdam, 1981.

[51] A. Wren and J.-M. Rousseau. Bus Driver Scheduling - An Overview. In Daduna et al. [16], pages 173–187.

[52] A. Wren and B.M. Smith. Experiences with a Crew Scheduling System Based on Set Covering. In Daduna and Wren [17], pages 104–118.

[53] A. Wren and D.O. Wren. A genetic algorithm for public transport driver scheduling. *Computers and Operations Research*, 22:101–110, 1995.

Appendix A - Complete Result Tables

The following tables give a complete overview on the results achieved in the FAST benchmark and provide averages on the runtime of the different FAST versions used in the test.
Table A.1 list all 60 instances $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ contained in the library. Instance *ivu52* has been removed from the list because of a data error. Table A.2 lists the corresponding DS-Opt solutions. DS-Opt was not able to solve some instances (*ivu09, ivu17, ivu18, ivu19, ivu20, ivu21*) as not all timetable tasks could be covered with duties using a feasible duty schedule. Some runs (*ivu43, ivu45, ivu49, ivu59, trier*) have been aborted after running several days. The instances *ivu44*, *ivu46*, *ivu48* and *ivu47* have not been run with DS-Opt due to a lack of time. Thus a total of 45 instances has been solved with DS-Opt.
Table A.3 represents the runtime savings achieved by FAST. Runtimes, objective values and averages are listed for the best overall FAST version on instances of relevant size, i.e. instances solved by DS-Opt in more than 50 minutes.
The remaining tables (A.4-A.9) similarly list runtimes, objective values and averages for all FAST versions and all chosen 35 instances used in the benchmark.

DSP - Instances										
Instance	$\lvert\mathcal{K}\rvert$	$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{V}\setminus\mathcal{I}\rvert$	$\lvert\mathcal{A}_1\rvert$	$\lvert\mathcal{A}_2\rvert$	$\lvert\mathcal{A}_3\rvert$	$\lvert\mathcal{A}_4\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{F}\rvert$
ivu01	3	153	270	423	72	406	2348	342	3168	13
ivu02	3	369	740	1109	184	913	2801	925	4823	12
ivu03	3	369	370	739	184	543	2801	555	4083	12
ivu04	3	620	1254	1874	274	1904	21729	1528	25435	21
ivu05	3	1561	2946	4507	714	3751	65134	3658	73257	42
ivu06	3	2187	4038	6225	1010	5164	123235	5044	134453	372
ivu07	3	3269	3269	6538	1519	7664	275251	7492	291926	423
ivu08	3	5021	7673	12694	2499	10064	552733	10158	575454	500
ivu09	6	816	52133	52949	580	52362	15940	43583	112465	9
ivu10	6	873	55443	56316	622	55688	17916	46351	120577	8
ivu11	1	873	777	1650	622	1022	4407	389	6440	8
ivu12	6	873	8224	9097	622	8469	16402	4377	29870	8
ivu13	1	1131	9949	11080	575	10481	196537	3342	210935	26
ivu14	2	873	4839	5712	622	5084	8156	2677	16539	8
ivu15	7	143	1775	1918	70	1843	1300	1332	4545	7
ivu16	6	968	21720	22688	660	22010	128777	10852	162299	20
ivu17	6	1934	47302	49236	1253	47938	620832	23486	693509	47
ivu18	6	1934	47302	49236	1253	47938	620832	23486	693509	47
ivu19	6	1934	47302	49236	1253	47938	620832	23486	693509	47
ivu20	6	1934	47302	49236	1253	47938	620832	23486	693509	47
ivu21	6	1934	47302	49236	1253	47938	620832	23486	693509	47
ivu22	6	558	12502	13060	377	12668	40020	6178	59243	17
ivu23	6	558	12502	13060	377	12668	40020	6178	59243	17
ivu24	2	1118	2545	3663	762	2881	9792	1894	15329	22
ivu25	2	1118	2545	3663	762	2881	9792	1894	15329	22
ivu26	2	2191	4641	6832	1538	5270	34336	3452	44596	26
ivu27	3	2191	5949	8140	1538	6578	35299	4760	48175	26
ivu28	2	1888	4742	6630	594	6026	173016	4736	184372	12
ivu29	4	2247	23808	26055	1183	24818	231440	14659	272100	56
ivu30	2	1863	5172	7035	1137	5837	82839	3846	93659	63
ivu31	2	1863	5172	7035	1137	5837	82905	3846	93725	63
ivu32	2	1872	5234	7106	1137	5899	84226	3892	95154	72
ivu33	3	2251	13216	15467	1183	14226	227916	9372	252697	60
ivu34	7	34093	10887	44980	32163	12758	497750	10600	553271	61
ivu35	4	2247	22035	24282	1183	23045	223084	14659	261971	56
ivu36	9	34860	16975	51835	32883	18894	249877	16514	318168	60
ivu37	7	34860	11042	45902	32883	12961	247785	10830	304459	60
ivu38	9	34752	16734	51486	32785	18645	209817	16434	277681	58
ivu39	4	1182	22885	24067	299	23749	78421	14088	116557	21
ivu40	3	1337	4614	5951	502	5372	37878	4434	48186	79
ivu41	3	10715	51681	62396	7146	54955	219187	31644	312932	297
ivu42	3	1704	10013	11717	1008	10643	25176	6150	42977	68
ivu43	3	1337	4661	5998	501	5417	40509	4474	50901	80
ivu44	2	10712	33490	44202	7146	36763	135800	20918	200627	293
ivu45	3	10269	40938	51207	6933	44000	1671926	25224	1748083	274
ivu46	3	1248	7355	8603	486	8055	27747	6834	43122	62
ivu47	3	1049	3218	4267	870	3390	2010	1561	7831	7
ivu48	4	8840	62491	71331	5826	65254	868281	38340	977701	251
ivu49	2	656	2672	3328	0	3316	44560	2624	50500	12
ivu50	10	6093	53171	59264	3451	55644	628262	49475	736832	171
ivu51	10	2694	15374	18068	1608	16393	165237	14736	197974	69
ivu53	13	3689	39190	42879	1269	41535	361263	38314	442381	77
ivu54	13	5152	31709	36861	3324	33400	221204	29952	287880	139
ivu55	13	7333	46085	53418	4724	48510	485344	43684	582262	186
ivu56	1	471	359	830	298	527	5430	348	6603	7
ivu57	1	1220	1485	2705	484	2217	19763	1474	23938	6
ivu58	1	694	3611	4305	236	4060	5748	2702	12746	11
ivu59	3	3807	38917	42724	201	41155	1713502	22000	1776858	1368
bonn	2	1444	3289	4733	982	3724	19943	1851	26500	27
trier	3	4609	120901	125510	1742	123688	699779	92869	918078	80

Table A.1: List of Instances $\mathcal{D} = (\mathcal{V}, \mathcal{A})$, $\mathcal{V} = \mathcal{T} \cup \mathcal{S} \cup \mathcal{I}$, $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2 \cup \mathcal{A}_3 \cup \mathcal{A}_4$, $\mathcal{F} = \mathcal{F}(\mathcal{D})$

	DS-Opt Solutions					
Instance	Value	$\|C\|$	Runtime	Duty Pieces	Slacks	EstLowBound
ivu01	14.26	10	00:00:25	20	0	14.25
ivu02	30.1	23	00:02:26	56	0	29.91
ivu03	29.44	22	00:04:04	43	0	29.3
ivu04	45.04	34	00:15:39	63	0	44.82
ivu05	97.75	73	00:59:15	155	0	97.51
ivu06	137.72	106	07:32:55	461	0	137.37
ivu07	190.15	146	10:25:29	584	0	189.95
ivu08	315.42	232	09:29:21	713	0	314.73
ivu09	32.95	17	01:44:35	43	1	32.8
ivu10	27.26	19	02:38:33	61	0	27.48
ivu11	26.93	19	00:11:37	44	0	26.89
ivu12	26.13	18	00:41:57	47	0	25.86
ivu13	45.28	28	05:31:18	82	0	45.18
ivu14	26.38	18	00:17:44	48	0	26.37
ivu15	5.77	4	00:00:26	9	0	5.77
ivu16	37.57	28	00:10:48	56	0	37.32
ivu17	88.46	67	01:36:14	142	25	88.14
ivu18	81.4	59	01:27:46	103	2	81.24
ivu19	84.17	59	01:22:40	138	2	83.92
ivu20	81.81	59	02:59:20	129	2	81.59
ivu21	81.26	59	01:36:24	117	2	81.15
ivu22	26.98	20	00:02:05	47	0	26.92
ivu23	26.74	20	00:02:22	38	0	26.74
ivu24	43.17	32	00:54:04	81	0	43.1
ivu25	43.13	32	00:38:21	90	0	43.03
ivu26	69.08	49	01:51:25	124	0	68.81
ivu27	69.14	49	02:51:42	125	0	68.95
ivu28	53.11	29	19:15:12	82	0	53.09
ivu29	67.16	61	19:09:34	225	0	66.89
ivu30	113.79	83	02:47:58	183	0	113.47
ivu31	83.95	83	03:57:50	183	0	83.77
ivu32	87.58	85	01:52:35	222	0	87.34
ivu33	90.49	63	08:35:48	175	0	90.28
ivu34	136.56	120	05:41:04	229	0	136.33
ivu35	115.23	62	06:24:17	180	0	114.92
ivu36	134.14	117	04:35:14	242	0	133.75
ivu37	138.51	123	04:54:06	253	0	138.39
ivu38	146.82	113	04:34:20	201	0	146.81
ivu39	32.41	27	04:07:14	49	0	32.25
ivu40	84.8	57	04:56:15	130	0	84.7
ivu41	446.53	416	02:36:43	586	0	445.82
ivu42	101.24	90	00:08:39	141	0	101.22
ivu43			aborted			
ivu44			skipped			
ivu45			aborted			
ivu46			skipped			
ivu47			skipped			
ivu48			skipped			
ivu49			aborted			
ivu50	350.9	262	02:19:00	470	0	350.3
ivu51	146.94	108	00:30:53	174	0	146.67
ivu53	263.38	122	03:44:42	168	0	263.16
ivu54	433.67	196	02:08:06	306	0	432.4
ivu55	594.23	268	03:28:19	420	0	592.58
ivu56	23.9	14	00:00:44	15	0	23.88
ivu57	23.79	13	02:09:47	32	0	23.79
ivu58	27.75	14	00:58:51	27	0	27.64
ivu59			aborted			
bonn	64.56	53	00:15:18	92	0	64.4
trier			aborted			

Table A.2: DS-Opt Solutions for instances $\mathcal{D} = (\mathcal{V}, \mathcal{A})$. Slacks indicate uncovered duty pieces.

FAST - Solutions - $\bar{r} = 3000$ $\hat{q} = (20, 0.3, 0.9, 10, 4, 2, 1800, 5400)$

	$\mathcal{D} = (\mathcal{V}, \mathcal{A})$		$\mathcal{D}^{\bar{r}}_{pos}$		$\mathcal{D}^{(\bar{r},\hat{q})}_{fix}$			Solution					
Instance	$\lvert\mathcal{V} \setminus \mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{V} \setminus \mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{V} \setminus \mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$	Size%	$\lvert C\rvert$	Slacks	Value	Value%	Runtime	Runtime%
ivu05	4507	73257	267	1049	2235	5966	8.14	71	0	97.5	99.74	00:00:38	1.06
ivu06	6225	134453	295	1105	3953	17429	12.96	108	0	139.69	101.43	00:32:50	7.24
ivu07	6538	291926	316	1322	5456	23249	7.96	149	0	192.78	101.38	00:18:14	2.91
ivu08	12694	575454	408	2145	7724	32638	5.67	238	0	319.85	101.40	00:21:51	3.83
ivu10	56316	120577	1065	2356	24938	46691	38.72	21	0	29.46	108.07	00:05:03	3.18
ivu12	9097	29870	834	1638	4627	8361	27.99	19	0	27.19	104.05	00:02:04	4.92
ivu13	11080	210935	537	3158	3648	7697	3.64	29	0	46.67	103.06	00:03:38	1.09
ivu24	3663	15329	176	531	2338	4915	32.06	32	0	42.98	99.55	00:05:16	9.74
ivu25	3663	15329	176	531	2359	4976	32.46	32	0	43.42	100.67	00:02:10	5.64
ivu26	6832	44596	209	723	4176	9598	21.52	48	0	69.11	100.04	00:10:24	9.33
ivu27	8140	48175	209	752	4721	10728	22.26	48	0	68.6	99.21	00:12:08	7.06
ivu28	6630	184372	102	1015	2260	3390	1.83	31	0	55.22	103.97	00:00:48	0.06
ivu29	26055	272100	592	2954	10972	25002	9.18	62	0	68.08	101.36	00:18:15	1.58
ivu30	7035	93659	324	842	4890	17256	18.42	83	0	113.41	99.66	01:39:55	59.48
ivu31	7035	93725	324	842	4868	17205	18.35	83	0	84.38	100.51	00:50:08	21.07
ivu32	7106	95154	340	939	4931	17465	18.35	85	0	88.45	100.99	00:23:57	21.27
ivu34	44980	553271	82	259	36153	42786	7.73	154	4	184.86	135.36	00:17:33	5.14
ivu35	24282	261971	711	3227	10343	24421	9.32	64	0	119.33	103.55	00:25:03	6.51
ivu39	24067	116557	1579	4037	5011	9208	7.89	28	0	33.86	104.47	00:01:18	0.52
ivu41	62396	312932	1943	6143	41947	96381	30.79	418	0	472.26	105.76	01:05:51	42.01
ivu53	42879	442381	1005	6212	12179	24731	5.59	127	0	281.21	106.76	00:06:10	2.74
ivu55	53418	582262	1279	8801	31431	79890	13.72	276	0	616.21	103.69	00:47:47	22.93
ivu58	4305	12746	269	567	1539	2459	19.29	14	0	27.79	100.14	00:01:23	2.35
Arith-Ave							16.25				103.69		10.51
Geom-Ave							12.72				103.48		4.59
Total %							11.62				104.42		6.69

Table A.3: FAST-Solutions $\mathbf{FAST}(\mathcal{D}, \bar{r}, \hat{q})$ for relevant instances $\mathcal{D} = (\mathcal{V}, \mathcal{A})$

FAST - Solutions - $\bar{r} = 2400$ $\hat{q} = (14, 0.05, 0.85, 13, 8, 3, 1800, 5400)$

	$\mathcal{D} = (\mathcal{V}, \mathcal{A})$		$\mathcal{D}^{\bar{r}}_{pos}$		$\mathcal{D}^{(\bar{r},\hat{q})}_{fix}$						Solution		
Instance	$\lvert\mathcal{V} \setminus \mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{V} \setminus \mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{V} \setminus \mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$	Size%	$\lvert C\rvert$	Slacks	Value	Value%	Runtime	Runtime%
ivu01	423	3168	87	257	231	579	18.27	11	0	15.14	106.17	00:00:04	16.00
ivu02	1109	4823	153	361	741	1888	39.14	23	0	30.19	100.29	00:00:16	10.95
ivu03	739	4083	120	472	543	1409	34.50	21	0	29.77	101.12	00:00:32	13.11
ivu04	1874	25435	215	918	1000	3831	15.06	33	0	45.69	101.44	00:00:49	5.21
ivu05	4507	73257	310	1932	2283	7401	10.10	71	0	97.34	99.58	00:00:55	1.54
ivu06	6225	134453	352	2015	4031	23171	17.23	106	0	138.46	100.53	00:37:15	8.22
ivu07	6538	291926	392	2564	5522	30928	10.59	147	0	191.64	100.78	00:33:47	5.40
ivu08	12694	575454	523	3644	7784	43956	7.63	234	0	315.98	100.17	00:25:49	4.53
ivu10	56316	120577	1418	3485	26990	51348	42.58	20	0	28.6	104.91	00:05:58	3.76
ivu11	1650	6440	168	422	1264	2523	39.17	20	0	27.89	103.56	00:00:13	1.86
ivu12	9097	29870	1082	2450	4623	8927	29.88	18	0	26.73	102.29	00:03:19	7.90
ivu13	11080	210935	785	6077	3900	10145	4.80	29	0	46.25	102.14	00:02:07	0.63
ivu14	5712	16539	839	1660	3147	6176	37.34	18	0	26.79	101.55	00:01:54	10.71
ivu15	1918	4545	210	507	864	1563	34.38	5	0	6.33	109.70	00:00:16	61.53
ivu16	22688	162299	1682	4348	11473	22392	13.79	31	0	40.68	108.27	00:01:15	11.57
ivu24	3663	15329	232	826	2354	5400	35.22	32	0	43.25	100.18	00:04:35	8.47
ivu25	3663	15329	232	826	2371	5481	35.75	32	0	43.37	100.55	00:04:01	10.47
ivu26	6832	44596	254	1075	4232	11156	25.01	48	0	68.75	99.52	00:13:00	11.66
ivu27	8140	48175	254	1108	4801	12395	25.72	48	0	68.51	99.08	00:07:57	4.63
ivu28	6630	184372	137	2163	2272	3605	1.95	31	0	56.12	105.66	00:00:31	0.04
ivu29	26055	272100	817	5454	11403	30686	11.27	61	0	67.12	99.94	00:59:53	5.20
ivu30	7035	93659	445	1219	5109	23072	24.63	83	0	113.32	99.58	02:34:55	92.23
ivu31	7035	93725	445	1219	5044	22598	24.11	83	0	84.27	100.38	01:31:18	38.38
ivu32	7106	95154	473	1399	5130	23147	24.32	85	0	88.13	100.62	00:37:55	33.67
ivu34	44980	553271	97	451	36251	45926	8.30	141	2	154.55	113.17	00:53:53	15.79
ivu35	24282	261971	976	5821	10680	29638	11.31	64	0	118.37	102.72	00:12:13	3.17
ivu39	24067	116557	1868	5875	5375	10555	9.05	28	0	33.75	104.13	00:09:58	4.03
ivu41	62396	312932	2387	8579	43232	115285	36.84	417	0	472.06	105.71	00:41:45	26.64
ivu42	11717	42977	864	2108	7234	16372	38.09	88	0	108.14	106.81	00:11:35	133.91
ivu51	18068	197974	575	11548	7648	16756	8.46	121	1	172.91	117.67	00:01:03	3.39
ivu53	42879	442381	1377	12760	12194	27009	6.10	126	0	279.0	105.93	00:17:07	7.61
ivu55	53418	582262	1689	16422	33012	98471	16.91	273	0	607.98	102.31	00:28:40	13.76
ivu56	830	6603	148	351	568	875	13.25	14	0	23.95	100.20	00:00:21	47.72
ivu58	4305	12746	355	770	1487	2271	17.81	15	0	28.67	103.31	00:03:19	5.63
bonn	4733	26500	353	1602	2590	4920	18.56	56	0	70.95	109.89	00:00:38	4.13
Arith-Ave							21.35				103.42		18.10
Geom-Ave							17.40				103.34		7.98
Total %							14.20				103.64		9.32

Table A.4: FAST version 1 - **FAST**$(\mathcal{D}, 2400, (14, 0.05, 0.85, 13, 8, 3, 1800, 5400))$ Solutions and averages

FAST - Solutions - $\bar{r} = 2400$ $\hat{q} = (13, 0.04, 0.7, 14, 9, 4, 1800, 5400)$

	$\mathcal{D} = (\mathcal{V}, \mathcal{A})$		$\mathcal{D}^{\bar{r}}_{pos}$		$\mathcal{D}^{(\bar{r},\hat{q})}_{fix}$			Solution					
Instance	$\lvert\mathcal{V} \setminus \mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{V} \setminus \mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{V} \setminus \mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$	Size%	$\lvert C\rvert$	Slacks	Value	Value%	Runtime	Runtime%
ivu01	423	3168	86	257	253	897	28.31	11	0	14.57	102.17	00:00:04	16.00
ivu02	1109	4823	152	361	773	2138	44.32	23	0	30.26	100.53	00:00:25	17.12
ivu03	739	4083	119	472	559	1621	39.70	22	0	29.7	100.88	00:00:18	7.37
ivu04	1874	25435	214	918	1074	5572	21.90	33	0	45.28	100.53	00:00:46	4.89
ivu05	4507	73257	310	1932	2377	9735	13.28	72	0	97.11	99.34	00:05:32	9.33
ivu06	6225	134453	352	2015	4119	26646	19.81	107	0	138.23	100.37	00:42:45	9.43
ivu07	6538	291926	392	2564	5644	35775	12.25	149	0	192.37	101.16	00:60:22	9.65
ivu08	12694	575454	523	3644	7944	54756	9.51	237	0	317.16	100.55	00:37:50	6.64
ivu10	56316	120577	1418	3485	30542	58669	48.65	20	0	28.45	104.36	00:08:47	5.53
ivu11	1650	6440	168	422	1310	2799	43.46	20	0	27.77	103.11	00:01:51	15.92
ivu12	9097	29870	1082	2450	4986	10078	33.73	18	0	26.55	101.60	00:00:51	2.02
ivu13	11080	210935	785	6077	4476	13290	6.30	29	0	46.28	102.20	00:11:38	3.51
ivu14	5712	16539	839	1660	3324	6754	40.83	19	0	27.02	102.42	00:02:57	16.63
ivu15	1918	4545	210	507	1056	2020	44.44	4	0	5.83	101.03	00:00:16	61.53
ivu16	22688	162299	1682	4348	12224	24785	15.27	31	0	40.81	108.62	00:04:55	45.52
ivu24	3663	15329	232	826	2449	5929	38.67	32	0	43.2	100.06	00:06:03	11.18
ivu25	3663	15329	232	826	2503	6211	40.51	32	0	43.29	100.37	00:08:05	21.07
ivu26	6832	44596	254	1075	4367	12074	27.07	48	0	68.96	99.82	00:14:28	12.98
ivu27	8140	48175	254	1108	4962	13339	27.68	49	0	68.88	99.62	00:10:06	5.88
ivu28	6630	184372	137	2163	2339	4104	2.22	31	0	55.3	104.12	00:12:35	1.08
ivu29	26055	272100	817	5454	12266	35810	13.16	61	0	67.19	100.04	00:50:56	4.43
ivu30	7035	93659	445	1219	6040	33032	35.26	83	0	113.57	99.80	03:56:57	141.06
ivu31	7035	93725	445	1219	5956	32300	34.46	83	0	83.95	100.00	03:00:18	75.80
ivu32	7106	95154	473	1399	6026	32813	34.48	85	0	87.79	100.23	00:50:50	45.15
ivu34	44980	553271	97	451	36566	49191	8.89	137	0	148.86	109.00	00:18:24	5.39
ivu35	24282	261971	976	5821	11486	34605	13.20	63	0	117.68	102.12	00:17:53	4.65
ivu39	24067	116557	1868	5875	5843	12024	10.31	28	0	33.82	104.35	00:08:51	3.57
ivu41	62396	312932	2387	8579	46070	131289	41.95	420	0	471.91	105.68	01:29:55	57.37
ivu42	11717	42977	864	2108	7568	17349	40.36	88	0	108.27	106.94	00:10:54	126.01
ivu51	18068	197974	575	11548	8501	20871	10.54	115	0	162.2	110.38	00:02:17	7.39
ivu53	42879	442381	1377	12760	11931	25918	5.85	126	0	279.18	105.99	00:21:09	9.41
ivu55	53418	582262	1689	16422	37846	125858	21.61	271	0	603.62	101.58	00:36:47	17.65
ivu56	830	6603	148	351	582	1010	15.29	14	0	23.95	100.20	00:00:23	52.27
ivu58	4305	12746	355	770	1573	2483	19.48	15	0	28.38	102.27	00:03:57	6.71
bonn	4733	26500	353	1602	2879	6337	23.91	54	0	67.42	104.42	00:01:53	12.30
Arith-Ave							25.33				102.45		24.36
Geom-Ave							20.71				102.41		12.48
Total %							16.88				102.93		13.40

Table A.5: FAST version 2 - **FAST**$(\mathcal{D}, 2400, (13, 0.04, 0.7, 14, 9, 4, 1800, 5400))$ Solutions and averages

FAST - Solutions - $\bar{r} = 1800$ $\hat{q} = (15, 0.2, 0.8, 12, 8, 4, 1800, 5400)$

	$\mathcal{D} = (\mathcal{V}, \mathcal{A})$		$\mathcal{D}^{\bar{r}}_{pos}$		$\mathcal{D}^{(\bar{r},\hat{q})}_{fix}$			Solution					
Instance	$\lvert\mathcal{V} \setminus \mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{V} \setminus \mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{V} \setminus \mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$	Size%	$\lvert C\rvert$	Slacks	Value	Value%	Runtime	Runtime%
ivu01	423	3168	111	383	243	615	19.41	11	0	14.98	105.04	00:00:07	28.00
ivu02	1109	4823	197	486	765	2053	42.56	23	0	30.41	101.02	00:00:50	34.24
ivu03	739	4083	161	660	551	1466	35.90	21	0	29.76	101.08	00:00:31	12.70
ivu04	1874	25435	338	1491	1018	4143	16.28	34	0	45.42	100.84	00:01:34	10.01
ivu05	4507	73257	438	3425	2371	8275	11.29	72	0	96.95	99.18	00:01:32	2.58
ivu06	6225	134453	479	3507	4115	23642	17.58	107	0	138.48	100.55	00:32:45	7.23
ivu07	6538	291926	526	4877	5594	31278	10.71	149	0	192.49	101.23	00:32:28	5.19
ivu08	12694	575454	666	6593	7908	44676	7.76	238	0	318.37	100.93	00:30:46	5.40
ivu10	56316	120577	1998	5185	27746	52995	43.95	19	0	28.67	105.17	00:05:45	3.62
ivu11	1650	6440	247	686	1262	2556	39.68	21	0	28.46	105.68	00:01:57	16.78
ivu12	9097	29870	1533	3883	4797	9531	31.90	18	0	26.92	103.02	00:04:18	10.25
ivu13	11080	210935	1165	13270	4008	10711	5.07	28	0	46.08	101.76	00:04:49	1.45
ivu14	5712	16539	1184	2562	3069	6025	36.42	18	0	27.14	102.88	00:02:23	13.43
ivu15	1918	4545	293	762	864	1609	35.40	4	0	5.91	102.42	00:00:17	65.38
ivu16	22688	162299	2364	6196	12446	24676	15.20	31	0	40.72	108.38	00:21:41	200.77
ivu24	3663	15329	315	1564	2418	5708	37.23	32	0	43.1	99.83	00:03:47	6.99
ivu25	3663	15329	315	1564	2482	6090	39.72	32	0	43.28	100.34	00:04:24	11.47
ivu26	6832	44596	371	2178	4344	11873	26.62	49	0	68.76	99.53	00:19:01	17.06
ivu27	8140	48175	371	2227	4954	13183	27.36	49	0	68.53	99.11	00:07:27	4.33
ivu28	6630	184372	171	4494	2304	3839	2.08	31	0	55.23	103.99	00:02:53	0.24
ivu29	26055	272100	1183	10283	11910	33249	12.21	61	0	67.15	99.98	01:37:51	8.51
ivu30	?	?	?	?	?	?	?	?	?	?	0.0	?	0.0
ivu31	7035	93725	580	1686	5361	25461	27.16	83	0	84.31	100.42	00:54:24	22.87
ivu32	?	?	?	?	?	?	?	?	?	?	0.0	?	0.0
ivu34	?	?	?	?	?	?	?	?	?	?	0.0	?	0.0
ivu35	24282	261971	1413	10792	11444	33247	12.69	63	0	117.56	102.02	00:21:47	5.66
ivu39	24067	116557	2424	9325	5947	12232	10.49	28	0	33.77	104.19	00:08:09	3.29
ivu41	?	?	?	?	?	?	?	?	?	?	0.0	?	0.0
ivu42	11717	42977	1222	3351	7567	17639	41.04	87	0	109.36	108.02	00:05:53	68.01
ivu51	18068	197974	677	18410	7819	17853	9.01	118	1	181.16	123.28	00:03:35	11.60
ivu53	42879	442381	1758	23696	12744	29415	6.64	126	0	277.91	105.51	00:12:33	5.58
ivu55	53418	582262	2104	29025	34087	104300	17.91	273	0	611.19	102.85	00:57:03	27.38
ivu56	830	6603	179	675	594	1189	18.00	14	0	23.97	100.29	00:00:14	31.81
ivu58	4305	12746	464	1099	1531	2453	19.24	15	0	28.38	102.27	00:03:14	5.49
bonn	4733	26500	599	2801	2734	5511	20.79	55	0	71.89	111.35	00:02:21	15.35
Arith-Ave							22.49				103.30		21.38
Geom-Ave							18.31				103.20		10.19
Total %							13.59				103.60		6.97

Table A.6: FAST version 3 - **FAST**$(\mathcal{D}, 1800, (15, 0.2, 0.8, 12, 8, 4, 1800, 5400))$ Solutions and averages

FAST - Solutions - $\bar{r} = 1800$ $\hat{q} = (14, 0.05, 0.7, 14, 9, 4, 1800, 5400)$

	$\mathcal{D} = (\mathcal{V}, \mathcal{A})$		$\mathcal{D}^{\bar{r}}_{pos}$		$\mathcal{D}^{(\bar{r},\hat{q})}_{fix}$			Solution					
Instance	$\lvert\mathcal{V} \setminus \mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{V} \setminus \mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{V} \setminus \mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$	Size%	$\lvert C\rvert$	Slacks	Value	Value%	Runtime	Runtime%
ivu01	423	3168	111	383	257	888	28.03	11	0	14.83	103.99	00:00:08	32.00
ivu02	1109	4823	197	486	797	2313	47.95	23	0	30.31	100.69	00:01:20	54.79
ivu03	739	4083	161	660	571	1732	42.41	22	0	29.86	101.42	00:02:28	60.65
ivu04	1874	25435	338	1491	1072	5385	21.17	33	0	45.24	100.44	00:00:50	5.32
ivu05	4507	73257	438	3425	2437	10041	13.70	71	0	96.89	99.12	00:09:13	15.55
ivu06	6225	134453	479	3507	4183	27020	20.09	107	0	138.45	100.53	00:29:39	6.54
ivu07	6538	291926	526	4877	5688	36095	12.36	149	0	192.38	101.17	01:08:14	10.90
ivu08	12694	575454	666	6593	8014	55010	9.55	239	0	317.38	100.62	00:42:00	7.37
ivu10	56316	120577	1998	5185	29858	57298	47.51	19	0	28.51	104.58	00:07:24	4.66
ivu11	1650	6440	247	686	1300	2735	42.46	21	0	28.65	106.38	00:01:49	15.63
ivu12	9097	29870	1533	3883	5086	10382	34.75	18	0	26.83	102.67	00:02:39	6.31
ivu13	11080	210935	1165	13270	4404	12931	6.13	28	0	45.97	101.52	00:10:36	3.19
ivu14	5712	16539	1184	2562	3269	6595	39.87	19	0	27.08	102.65	00:01:42	9.58
ivu15	1918	4545	293	762	1080	2099	46.18	4	0	5.8	100.51	00:00:21	80.76
ivu16	22688	162299	2364	6196	12646	25889	15.95	30	0	40.06	106.62	00:03:37	33.48
ivu24	3663	15329	315	1564	2476	6080	39.66	32	0	43.04	99.69	00:03:41	6.81
ivu25	3663	15329	315	1564	2560	6544	42.69	32	0	43.37	100.55	00:08:49	22.99
ivu26	6832	44596	371	2178	4427	12470	27.96	48	0	69.01	99.89	00:20:08	18.07
ivu27	8140	48175	371	2227	5055	13865	28.78	48	0	68.56	99.16	00:13:32	7.88
ivu28	6630	184372	171	4494	2373	4459	2.41	30	0	54.42	102.46	00:05:23	0.46
ivu29	26055	272100	1183	10283	12473	36922	13.56	61	0	67.21	100.07	00:52:08	4.53
ivu30	7035	93659	580	1685	6041	33050	35.28	83	0	113.48	99.72	01:51:19	66.27
ivu31	7035	93725	580	1686	5952	32240	34.39	83	0	84.22	100.32	03:07:17	78.74
ivu32	7106	95154	602	1817	6019	32666	34.32	85	0	87.88	100.34	00:47:12	41.92
ivu34	44980	553271	140	827	36604	49878	9.01	138	0	149.6	109.54	00:14:34	4.27
ivu35	24282	261971	1413	10792	11906	36494	13.93	63	0	117.27	101.77	00:31:48	8.27
ivu39	24067	116557	2424	9325	6701	14468	12.41	27	0	33.2	102.43	00:19:21	7.82
ivu41	62396	312932	3513	13243	46287	132191	42.24	420	0	473.48	106.03	01:18:31	50.10
ivu42	11717	42977	1222	3351	7867	18552	43.16	87	0	109.11	107.77	00:06:44	77.84
ivu51	18068	197974	677	18410	8522	20985	10.59	114	1	169.49	115.34	00:03:40	11.87
ivu53	42879	442381	1758	23696	13839	34080	7.70	125	0	273.83	103.96	00:32:28	14.44
ivu55	53418	582262	2104	29025	37760	125464	21.54	271	0	603.38	101.53	01:02:58	30.22
ivu56	830	6603	179	675	600	1227	18.58	14	0	23.93	100.12	00:00:14	31.81
ivu58	4305	12746	464	1099	1685	2904	22.78	15	0	28.34	102.12	00:02:19	3.93
bonn	4733	26500	599	2801	2814	5963	22.50	56	0	70.3	108.89	00:02:34	16.77
Arith-Ave							26.05				102.70		24.34
Geom-Ave							21.51				102.65		14.20
Total %							17.25				103.11		12.35

Table A.7: FAST version 4 - **FAST**$(\mathcal{D}, 1800, (14, 0.05, 0.7, 14, 9, 4, 1800, 5400))$ Solutions and averages

FAST - Solutions - $\bar{r} = 2400$ $\hat{q} = (15, 0.2, 0.8, 12, 8, 4, 1800, 5400)$

	$\mathcal{D} = (\mathcal{V}, \mathcal{A})$		$\mathcal{D}^{\bar{r}}_{pos}$		$\mathcal{D}^{(\bar{r},\hat{q})}_{fix}$			Solution					
Instance	$\lvert\mathcal{V} \setminus \mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{V} \setminus \mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{V} \setminus \mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$	Size%	$\lvert C\rvert$	Slacks	Value	Value%	Runtime	Runtime%
ivu01	423	3168	86	257	239	632	19.94	11	0	15.1	105.89	00:00:04	16.00
ivu02	1109	4823	152	361	737	1853	38.42	23	0	30.33	100.76	00:00:27	18.49
ivu03	739	4083	119	472	547	1449	35.48	22	0	29.87	101.46	00:00:20	8.19
ivu04	1874	25435	214	918	1016	4168	16.38	34	0	45.7	101.46	00:00:21	2.23
ivu05	4507	73257	310	1932	2299	7745	10.57	72	0	97.06	99.29	00:01:31	2.55
ivu06	6225	134453	352	2015	4043	23106	17.18	107	0	138.57	100.61	00:32:30	7.17
ivu07	6538	291926	392	2564	5546	30812	10.55	148	0	192.38	101.17	00:38:02	6.08
ivu08	12694	575454	523	3644	7818	43955	7.63	237	0	317.51	100.66	00:35:24	6.21
ivu10	56316	120577	1418	3485	27338	52162	43.26	20	0	28.71	105.31	00:06:04	3.82
ivu11	1650	6440	168	422	1278	2655	41.22	20	0	27.71	102.89	00:00:45	6.45
ivu12	9097	29870	1082	2450	4658	9205	30.81	18	0	26.77	102.44	00:01:57	4.64
ivu13	11080	210935	785	6077	4098	11095	5.25	29	0	46.7	103.13	00:06:48	2.05
ivu14	5712	16539	839	1660	3174	6356	38.43	18	0	26.84	101.74	00:01:32	8.64
ivu15	1918	4545	210	507	888	1640	36.08	4	0	5.83	101.03	00:00:10	38.46
ivu16	22688	162299	1682	4348	12036	23750	14.63	30	0	40.08	106.68	00:01:24	12.96
ivu24	3663	15329	232	826	2370	5487	35.79	32	0	43.2	100.06	00:05:41	10.51
ivu25	3663	15329	232	826	2411	5677	37.03	32	0	43.53	100.92	00:08:08	21.20
ivu26	6832	44596	254	1075	4290	11490	25.76	49	0	68.8	99.59	00:08:08	7.29
ivu27	8140	48175	254	1108	4856	12694	26.34	49	0	68.64	99.27	00:16:20	9.51
ivu28	6630	184372	137	2163	2274	3586	1.94	31	0	55.91	105.27	00:00:42	0.06
ivu29	26055	272100	817	5454	11723	32389	11.90	61	0	67.18	100.02	01:38:37	8.57
ivu30	7035	93659	445	1219	5433	26071	27.83	83	0	113.47	99.71	02:45:48	98.71
ivu31	7035	93725	445	1219	5368	25554	27.26	83	0	84.45	100.59	02:07:06	53.44
ivu32	7106	95154	473	1399	5438	26014	27.33	85	0	88.16	100.66	00:38:28	34.16
ivu34	44980	553271	97	451	36326	46970	8.48	139	0	149.4	109.40	00:20:45	6.08
ivu35	24282	261971	976	5821	11010	31401	11.98	63	0	117.89	102.30	00:14:19	3.72
ivu39	24067	116557	1868	5875	5505	10970	9.41	28	0	33.5	103.36	00:09:05	3.67
ivu41	62396	312932	2387	8579	44370	121059	38.68	421	0	475.27	106.43	01:08:41	43.82
ivu42	11717	42977	864	2108	7228	16342	38.02	88	0	108.56	107.23	00:08:11	94.60
ivu51	18068	197974	575	11548	7858	18030	9.10	119	1	178.59	121.53	00:02:26	7.87
ivu53	42879	442381	1377	12760	12763	29428	6.65	126	0	278.04	105.56	00:12:04	5.37
ivu55	53418	582262	1689	16422	34217	104926	18.02	274	0	610.41	102.72	00:52:18	25.10
ivu56	830	6603	148	351	580	984	14.90	14	0	23.98	100.33	00:00:23	52.27
ivu58	4305	12746	355	770	1535	2397	18.80	15	0	28.35	102.16	00:01:08	1.92
bonn	4733	26500	353	1602	2829	5975	22.54	55	0	70.12	108.61	00:03:57	25.81
Arith-Ave							22.39				103.15		18.79
Geom-Ave							18.32				103.07		9.19
Total %							14.91				103.80		11.00

Table A.8: FAST version 5 - **FAST**$(\mathcal{D}, 2400, (15, 0.2, 0.8, 12, 8, 4, 1800, 5400))$ Solutions and averages

FAST - Solutions - $\bar{r} = 3000$ $\hat{q} = (20, 0.3, 0.9, 10, 4, 2, 1800, 5400)$

	$\mathcal{D} = (\mathcal{V}, \mathcal{A})$		$\mathcal{D}^{\bar{r}}_{pos}$		$\mathcal{D}^{(\bar{r},\hat{q})}_{fix}$			Solution					
Instance	$\lvert\mathcal{V} \setminus \mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{V} \setminus \mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{V} \setminus \mathcal{I}\rvert$	$\lvert\mathcal{A}\rvert$	Size%	$\lvert C\rvert$	Slacks	Value	Value%	Runtime	Runtime%
ivu01	423	3168	69	166	227	515	16.25	13	0	16.16	113.32	00:00:03	12.00
ivu02	1109	4823	121	280	689	1585	32.86	23	0	30.67	101.89	00:00:06	4.10
ivu03	739	4083	101	290	515	1162	28.45	22	0	29.95	101.73	00:00:12	4.91
ivu04	1874	25435	175	566	960	2991	11.75	33	0	45.89	101.88	00:00:35	3.72
ivu05	4507	73257	267	1049	2235	5966	8.14	71	0	97.5	99.74	00:00:38	1.06
ivu06	6225	134453	295	1105	3953	17429	12.96	108	0	139.69	101.43	00:32:50	7.24
ivu07	6538	291926	316	1322	5456	23249	7.96	149	0	192.78	101.38	00:18:14	2.91
ivu08	12694	575454	408	2145	7724	32638	5.67	238	0	319.85	101.40	00:21:51	3.83
ivu10	56316	120577	1065	2356	24938	46691	38.72	21	0	29.46	108.07	00:05:03	3.18
ivu11	1650	6440	134	267	1252	2328	36.14	20	0	27.96	103.82	00:00:14	2.00
ivu12	9097	29870	834	1638	4627	8361	27.99	19	0	27.19	104.05	00:02:04	4.92
ivu13	11080	210935	537	3158	3648	7697	3.64	29	0	46.67	103.06	00:03:38	1.09
ivu14	5712	16539	653	1165	3027	5507	33.29	19	0	27.4	103.86	00:01:13	6.86
ivu15	1918	4545	190	464	792	1413	31.08	4	0	5.81	100.69	00:00:08	30.76
ivu16	22688	162299	1339	3048	11452	20551	12.66	30	0	41.56	110.62	00:04:31	41.82
ivu24	3663	15329	176	531	2338	4915	32.06	32	0	42.98	99.55	00:05:16	9.74
ivu25	3663	15329	176	531	2359	4976	32.46	32	0	43.42	100.67	00:02:10	5.64
ivu26	6832	44596	209	723	4176	9598	21.52	48	0	69.11	100.04	00:10:24	9.33
ivu27	8140	48175	209	752	4721	10728	22.26	48	0	68.6	99.21	00:12:08	7.06
ivu28	6630	184372	102	1015	2260	3390	1.83	31	0	55.22	103.97	00:00:48	0.06
ivu29	26055	272100	592	2954	10972	25002	9.18	62	0	68.08	101.36	00:18:15	1.58
ivu30	7035	93659	324	842	4890	17256	18.42	83	0	113.41	99.66	01:39:55	59.48
ivu31	7035	93725	324	842	4868	17205	18.35	83	0	84.38	100.51	00:50:08	21.07
ivu32	7106	95154	340	939	4931	17465	18.35	85	0	88.45	100.99	00:23:57	21.27
ivu34	44980	553271	82	259	36153	42786	7.73	154	4	184.86	135.36	00:17:33	5.14
ivu35	24282	261971	711	3227	10343	24421	9.32	64	0	119.33	103.55	00:25:03	6.51
ivu39	24067	116557	1579	4037	5011	9208	7.89	28	0	33.86	104.47	00:01:18	0.52
ivu41	62396	312932	1943	6143	41947	96381	30.79	418	0	472.26	105.76	01:05:51	42.01
ivu42	11717	42977	716	1696	7081	14514	33.77	88	0	108.41	107.08	00:05:48	67.05
ivu51	18068	197974	433	6671	7440	14687	7.41	126	2	188.03	127.96	00:04:01	13.00
ivu53	42879	442381	1005	6212	12179	24731	5.59	127	0	281.21	106.76	00:06:10	2.74
ivu55	53418	582262	1279	8801	31431	79890	13.72	276	0	616.21	103.69	00:47:47	22.93
ivu56	830	6603	117	327	564	825	12.49	14	0	23.94	100.16	00:00:07	15.90
ivu58	4305	12746	269	567	1539	2459	19.29	14	0	27.79	100.14	00:01:23	2.35
bonn	4733	26500	319	1038	2551	4380	16.52	57	0	70.05	108.50	00:00:37	4.03
Arith-Ave							18.47				104.75		12.80
Geom-Ave							14.88				104.52		5.98
Total %							11.86				105.50		6.82

Table A.9: FAST version 6 - **FAST**$(\mathcal{D}, 3000, (20, 0.3, 0.9, 10, 4, 2, 1800, 5400))$ Solutions and averages

Appendix B - Duty Scheduling Graphs

To visualize Duty Scheduling Graphs, duties and duty schedules, the visualization tool DSVis has been developed alongside the work on this thesis. DSVis is programmed in Java and uses the JavaView library, see *www.javaview.de*.
DSVis can be used to display Duty Scheduling Graphs $\mathcal{D} = (\mathcal{V}, \mathcal{A})$, or parts of these graphs. It provides a huge set of functions to hide components of the graph, i.e. specific timetable tasks, supplementary tasks or links of a certain type. Furthermore, duties and duty schedules or parts of duty schedules regarding $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ can be visualized. The visualization of duties supports different coloring of the contained timetable tasks and links as well as fading of not contained timetable tasks.
DSVis supports functions such as zoom and rotate to analyse the components. Thus extracts of $\mathcal{D} = (\mathcal{V}, \mathcal{A})$ can be enlarged. The display of tasks and links is controlled by setting a list of configuration variables. Such variables include sizes, colors, ordering and labeling of tasks and links.
Standard coloring for links is as follows. Type 4 links connecting to artificial tasks $\mathcal{I}$ are colored gray. Type 3 links are colored orange, type 2 links are colored red and type 1 links yellow. Supplementary tasks are displayed as dots while timetable tasks are displayed as boxes of a size indicated by the starting end ending time of the task.
The following Figures B.1-B.4 have been produced with DSVis to visualize the amount of reduction achieved in the $\hat{x}$-$\hat{q}$-FAST-Fix for the instances *bonn* and *ivu05*. The timetable tasks are sorted by block and thus type 1 links and many type 2 links are invisible. Especially the number of type 3 and type 4 links is reduced as can be seen in Figure B.2 and Figure B.4, which represent the fixed Duty Scheduling Graphs of the graphs displayed in the other two figures.

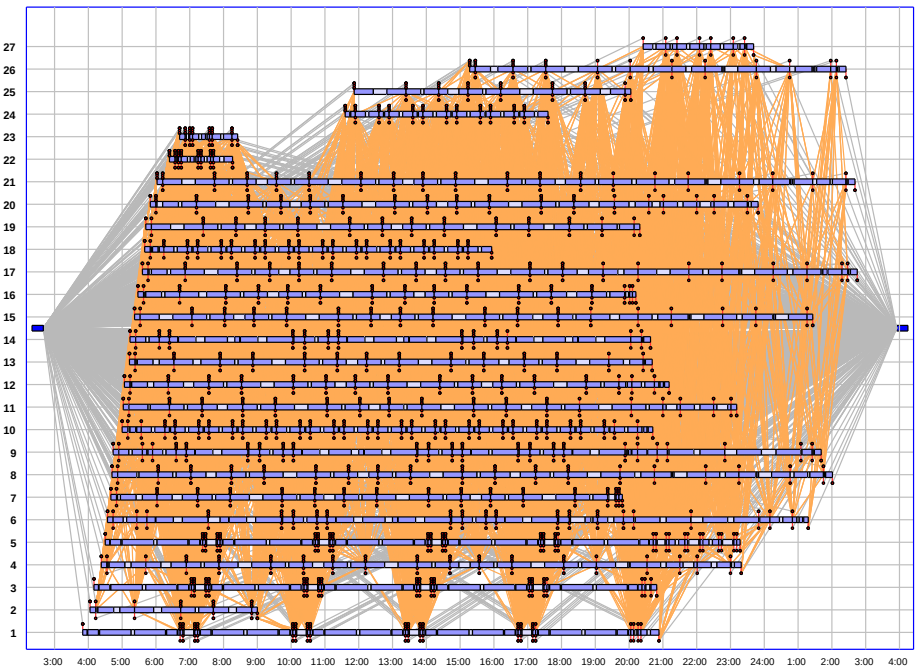

Figure B.1: Problem bonn, $|\mathcal{V}| = 4733$, $|\mathcal{A}| = 26500$

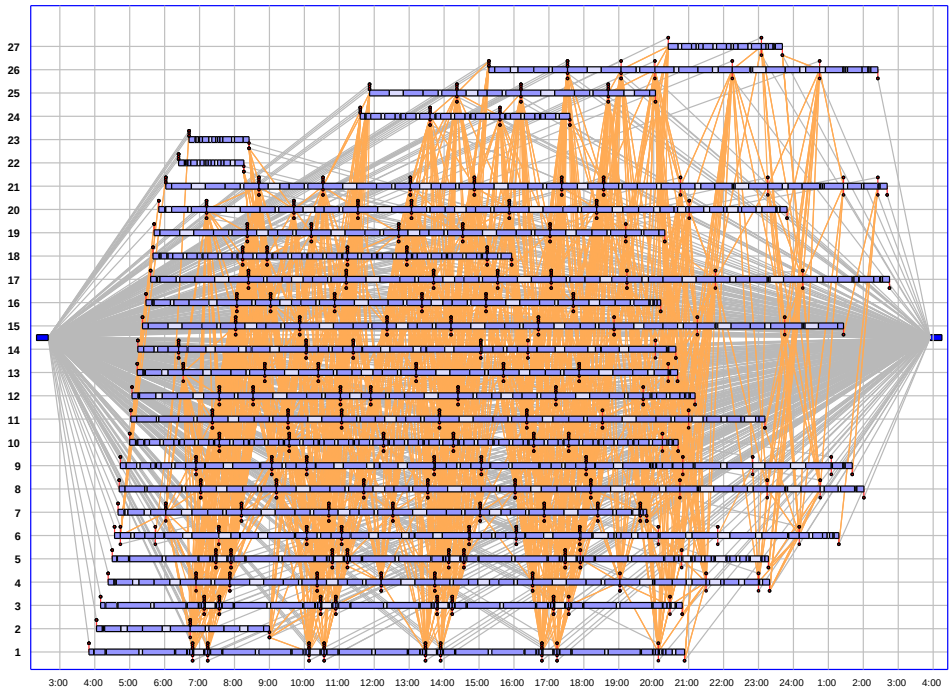

Figure B.2: Problem bonn fixed, $\bar{r} = 2400$, $|\mathcal{V}| = 2879$, $|\mathcal{A}| = 6457$

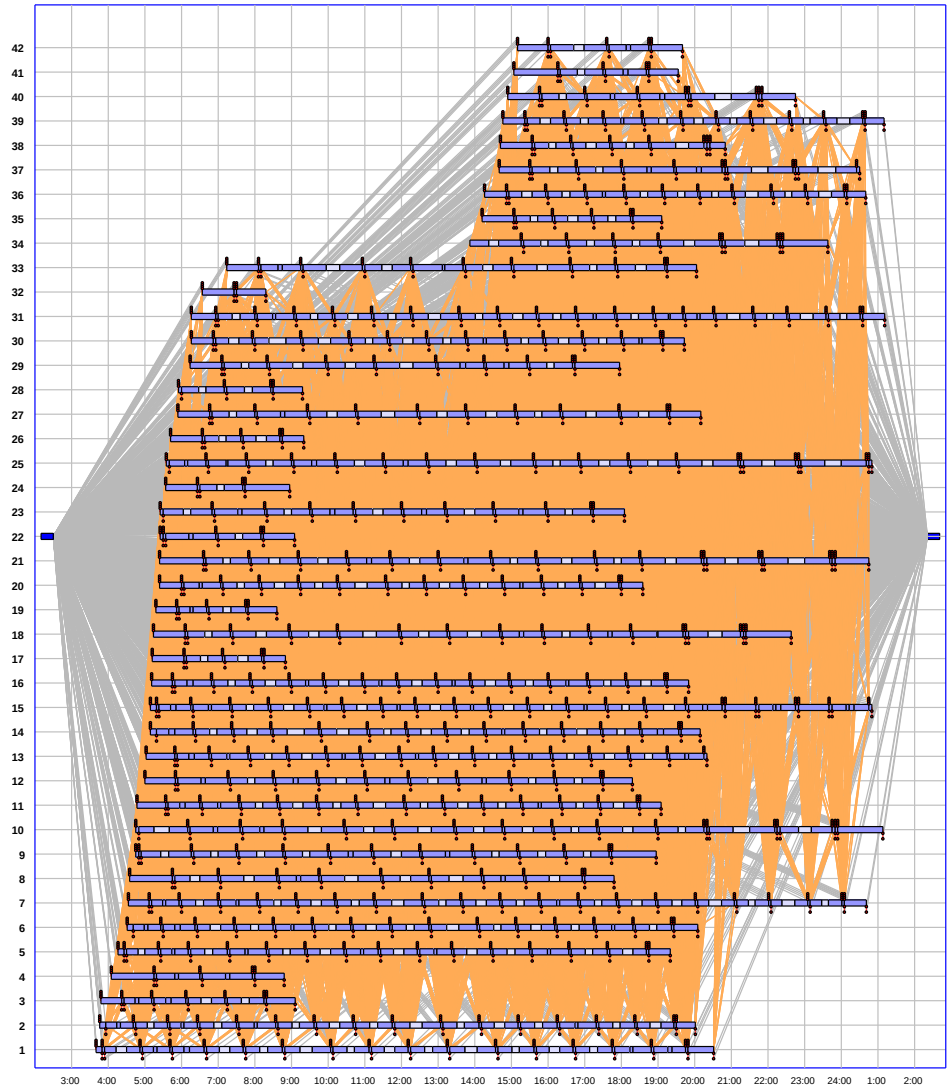

Figure B.3: Instance *ivu05*, $|\mathcal{V}| = 4507$, $|\mathcal{A}| = 73257$

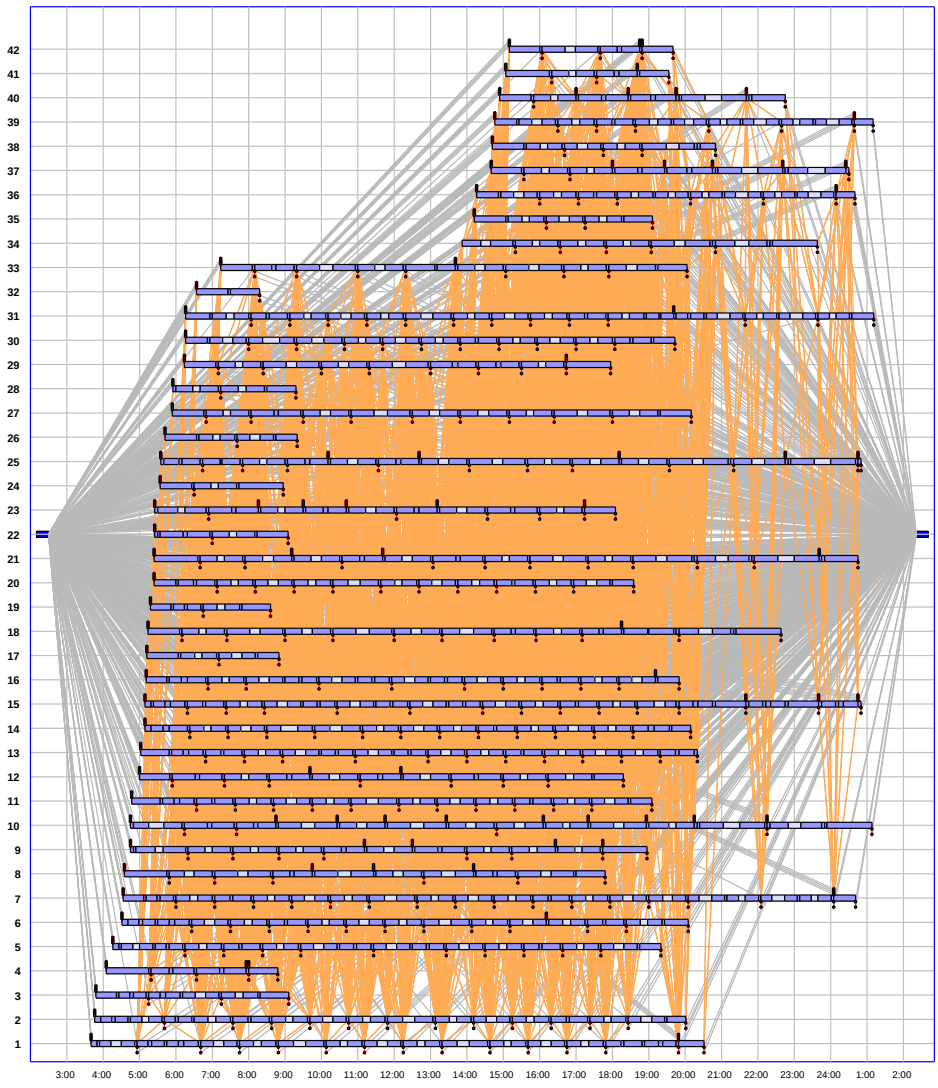

Figure B.4: Instance *ivu05* fixed - $\bar{r} = 1800$, $|\mathcal{V}| = 2587$, $|\mathcal{A}| = 13765$