Robust scheduling in an uncertain environment

M. Wilson
Robust scheduling
in an uncertain environment

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft
op gezag van de Rector Magnificus prof.ir. K.Ch.A.M. Luyben;
voorzitter van het College van Promoties,
in het openbaar te verdedigen op
maandag 1 februari 2016 om 15:00 uur

door

Michel WILSON
informatica-ingenieur
geboren te Rozenburg (ZH), Nederland
Dit proefschrift is goedgekeurd door de promotor:
Prof.dr. C. Witteveen

Samenstelling van de promotiecommissie:

Rector Magnificus voorzitter
Prof.dr. C. Witteveen Technische Universiteit Delft, promotor
Dr. T.B. Klos Technische Universiteit Delft, copromotor

Onafhankelijke leden:
Prof.dr. K.I. Aardal Technische Universiteit Delft
Prof.dr.ir. G.J.J.A.N. van Houtum Technische Universiteit Eindhoven
Prof.dr.ir. L.A.M. van Dongen Universiteit Twente
Dr. J. Boerkoel Harvey Mudd College, Claremont, CA, USA
Prof.dr.ir. H.J. Sips Technische Universiteit Delft, reservelid

Overig lid:
B. Huisman MSc NedTrain, Utrecht

Published and distributed by: Michel Wilson
e-mail: m.wilson@tudelft.nl
ISBN: 978-94-6299-278-8 NUR 980

The research reported in this thesis has been funded by
NedTrain in the framework of the Applied Research &
Development Program ‘Rolling Stock Life Cycle Logistics’.
SIKS Dissertation Series No. 2016-06.
The research reported in this thesis has been carried out
under the auspices of SIKS, the Dutch Research School for
Information and Knowledge Systems.

Copyright © 2016 by Michel Wilson.
This work is licensed under the Creative Commons Attribution-NonCom-
mercial 3.0 Netherlands License.

You are free to share (to copy, distribute and transmit) and to remix (to adapt) this work under
the following conditions: (i) attribution — you must attribute the work in the manner specified
by the author or licensor (but not in any way that suggests that they endorse you or your use
of the work), (ii) noncommercial — you may not use this work for commercial purposes; and
with the understanding that (i) any of the above conditions can be waived if you get permission
from the copyright holder, (ii) where the work or any of its elements is in the public domain
under applicable law, that status is in no way affected by the license, and (iii) in no way are your
fair dealing or fair use rights, or other applicable copyright exceptions and limitations, the
author's moral rights or rights other persons may have either in the work itself or in how the
work is used, such as publicity or privacy rights, affected by the license.
This is a human-readable summary of the Legal Code, which can be found in full at
http://creativecommons.org/licenses/by-nc/3.0/nl/legalcode.

Printed in The Netherlands
Preface

With great satisfaction, and, admittedly, more than a little relief at being finished, I am writing this preface to my thesis. It has been a long and at times difficult journey resulting in something I am very proud of. I’ve sometimes heard the finishing of a thesis or the defence of one being referred to as giving birth to one’s baby. Being a young father, I can certainly see some of the parallels. But the differences are many, not in the least the fact that delivering a thesis is (fortunately!) physically much less demanding than delivering a baby. Another striking difference is that the delivery of this thesis marks the end of an academic journey, whereas the true journey only begins after one’s child is born.

Many people assisted me during this journey, for which I am grateful. I want to start by thanking Arjan van Gemund, the supervisor for my master’s thesis, who started me on this journey by steering me in the direction of a vacancy for a PhD student at the Algorithmics group. This is where I met Cees Witteveen, to which I owe many thanks. Cees was always available for advice, and we had many in-depth meetings discussing papers and chapters in progress, which benefited greatly from his unwavering attention to detail. I also want to thank Tomas, who joined in supervising me as co-promotor, at a slightly later stage. I enjoyed our long discussions on the experimental analysis of algorithm and on the proper design of experiments. I may have sighed at the extra experimental work his critical eye caused, but at the end it is clear that the results I have obtained are much improved. I also enjoyed our discussions on teaching, lecturing and education in general.

The other colleagues in the Algorithmics group made my time as PhD student very pleasant. I want to thank my office mates over the years: Pieter, Renze, Adriaan and Jeroen, for giving me lots of survival tips for beginning PhD students, some of which I’ve followed with good results, others which I’ve ignored to my own detriment; Joris, for his happy moods and his love for his beautifully ugly car, but also for ensuring that “our room” was ranked at the top of the indoor karting competition; Gieb, for the interesting discussions on language, and for all the odd discussions on Dutch culture and life in general; and of course Simon, more or less my “successor” for this research project, with which I’ve had lots of very interesting and deeply technical discussions on scheduling topics, after each
of which the whiteboard looked even more arcane, but also on operating systems, desktop user interfaces, and typesetting software. And of course I want to thank all the other group members, Chetan, Hans, Léon, Marijn, Matthijs, Matthijs, Peter, Shruti, Sicco, Yinqian, for all the engaging conversations that took place during lunch or over coffee. I especially want to thank Peter for all the extended coffee breaks, over which we discussed science, academia, life after academia, and life in general. And of course I want to thank Ronald, Simon and Jonathan for working with me on their Master thesis, contributing to my research greatly, and Erik, Jan, Erwin and Wilco for their work on the scheduling demonstrator tool for their bachelor project.

I also am indebted the many people at NedTrain who helped me in my research: Leo van Dongen for heading the research program as board member of NedTrain, but also Bob Huisman for his day-to-day leadership in the research program, for the many insightful discussions we had on my research, his useful feedback on my writing, and his social insight into some of the people I interacted with. The atmosphere among the group of people doing research at NedTrain was very warm and cooperative, and I want to thank Joachim and Jorge, the other PhD students in the group, as well as all the Master students which worked at NedTrain, for the many enjoyable lunch walks we took, and for the broad ranging discussions on all the differing fields of research that were presented in our little group. The ideas and suggestions by Fred van Houten, Geert-Jan van Houtum, Leo Kroon, and Rob Basten, prompted by my presentations during the meetings of the steering group committee, also helped me in my research. My work with Pim Op ’t Land and Mariëlle ten Have on data analysis of the on-board diagnostic system of the VRM train series gave me interesting insight into the workings of the maintenance procedures at NedTrain. I enjoyed being able to visit some of the NedTrain workshops, and I want to thank the engineers I met there there, for being patient enough to explain their work to another one of those ‘theoretical guys’ visiting them.

I am glad to have friends that did not grow too bored with me when I talked about what occupied my mind, and I’m also glad for all the times when they succeeded in getting my mind otherwise involved, be it by going to concerts, having dinner, playing a game, or just by having a good talk. I want to thank my parents as well, for supporting me all this time, and for bearing with me during the explanations of what, exactly, I was doing all this time. And of course, I want to express my gratitude to Simone, for her love, for always being at my side and for believing in me at all times: even when I was in doubt myself you never lost your trust in me. And, I want to thank Emma, our dear daughter, for bringing joy in my life, and for teaching me lots of new things about flexibility in planning.
Contents

List of Figures · vii

List of Tables · ix

List of Algorithms · xi

1 Introduction · 1
1.1 A bit of historical context · 1
1.2 Railway organizations · 3
1.3 Research program at NedTrain · 7
1.4 Research goals · 13
1.5 Outline and contributions · 15

2 Existing work · 17
2.1 Resource-constrained project scheduling · 17
2.2 Mixed Integer Linear Programming solutions · 22
2.3 Branch and bound methods · 24
2.4 Schedule Generation Schemes · 26
2.5 Precedence constraint posting · 28
2.6 Simple temporal networks · 31
2.7 Multi-agent scheduling · 38
2.8 Conclusions and research questions · 39

3 Measuring flexibility · 41
3.1 The need for flexibility · 42
3.2 Flexibility of an STN · 43
3.3 Applications · 54
3.4 Conclusion and discussion · 62

4 Distributing flexibility to improve robustness · 65
4.1 Motivation · 66
4.2 Methods for distributing flexibility · 69
4.3 Experiments · 72
4.4 Conclusion and discussion · 83

5 Sequential flexibility · 85
5.1 Motivation · 87
5.2 Constraint posting · 88
5.3 Task grouping · 95
5.4 Execution of grouped schedules · 100
5.5 Experiments · 101
5.6 Conclusion and discussion · 108

6 Discussion · 111
6.1 Research questions · 111
6.2 Research goals and further research · 114

Bibliography · 117

Summary in English · 123

Summary in Dutch · 127

Curriculum vitae · 131

SIKS dissertation series · 133
List of Figures

1.1 Railway organizations in the Netherlands · 4
1.2 Map of service and maintenance locations · 5
1.3 An example of the interchange method · 10

3.1 Computing an interval schedule · 52

4.1 Influence of delay length on violations · 76
4.2 Influence of the number of delays on violations · 77
4.3 Influence of delay length on tardiness · 77
4.4 Influence of the number of delays on tardiness · 77
4.5 Violations versus flexibility loss for equalized and predecessor-based distributions · 79
4.6 Violations versus flexibility loss for equalized and successor-based distributions · 79
4.7 Violations versus tardiness for equalized, maximal-flexibility and predecessor-based distributions · 80
4.8 Violations versus tardiness for equalized, maximal-flexibility and successor-based distributions · 80
4.9 Number of violations with guaranteed flexibility · 82
4.10 Tardiness with guaranteed flexibility · 82

5.1 Performance of task grouping for different values of $\gamma$ · 104
5.2 Commitment versus robustness · 105
5.3 Lateness versus number and average group task size · 107
List of Tables

1.1 Classification of scheduling situations · 11
4.1 Flexibility loss of the different flexibility distributions · 74
4.2 Average performance for the different flexibility distributions · 75
5.1 Summary of the \texttt{PSPLIB} benchmark instances. · 102
5.2 Characteristics of grouped schedules for different parameters. · 103
List of Algorithms

2.1 Serial schedule generation · 27
2.2 Parallel schedule generation · 27
2.3 Floyd-Warshall’s all-pairs shortest-path algorithm · 35
2.4 Bellman-Ford’s single-source shortest-path algorithm · 36

5.1 Operation to group tasks together. · 97
5.2 The task grouping algorithm · 99
Introduction

In this chapter, an introduction will be given to the research presented in this thesis. The research is part of the rolling stock life cycle logistics applied research and development program funded by Dutch railway industry companies. We will start this introduction with a bit of historical overview, to illustrate some of the reasons why such a program is needed in what seems to be a mature and well-developed industry. After this, we will discuss the general structure and operation of the railway organizations in the Netherlands. Having knowledge of the context in which the research is performed is important, because this context has a significant influence on the direction of the research, and on the choices which are made.

After discussing the railway organizations, the research program of which this research forms a part will be described, along with the problems and context inspiring the research. These problems will then be used to formulate a set of research goals that are used to guide the research in this thesis.

1.1 A bit of historical context

Before turning to an explanation and motivation of the exact research presented in this thesis, we start by taking a step back, to examine some of the historical aspects of railway organizations, in general, and applied to the Netherlands in particular. We do this to answer a question the reader might have about the why of research on railway asset maintenance operations: what is the necessity of performing this research, besides the obvious one of wanting to continually improve upon the state of the art?

This is a valid question, since there is a long and rich history of railway operations. The earliest known evidence of railway operations dates back to the 6th century BC in ancient Greece, in the form of a 6 km stretch of railway used to transport boats in Corinth (Lewis, 2001). Starting in the 16th century, narrow-gauge railways started to see a lot of use in the mining industry (Agricola, 1556). But the real breakthrough came in the form of the steam engine. Early steam engines used low-pressure steam, which made them large, bulky, and unsuitable for non-stationary use. The beginning of the 19th century saw the
invention of high-pressure steam engines in Great Britain by Richard Trevithick, among others (Kirby et al., 1956, p. 175–177). This led to a rapidly developing railway industry: in the 1950s more than 7,000 miles of track had been laid in Britain alone (Wolmar, 2009, p. 99).

The railway sector is a mature industry, in part due to its age, and this would indeed suggest that the problem of maintaining railway assets has been studied extensively. But if we examine the more recent history of, in this case, Dutch railway operations, an important change in operations is currently occurring. In the past, the Dutch national railways developed rolling stock in very close cooperation with Dutch industry. If we restrict ourselves to train sets, this already started with the development of the first electrical train set in 1924 (Mat’24, 259 coaches produced in total), and continued with the development of the first streamliner train sets (the diesel-electrical Mat’34, as well as the electrical Mat’35, Mat’36 and Mat’40, totalling 508 coaches). For all of these train sets, the Dutch railways collaborated closely with the Dutch companies of Werkspoor, Allan and Beijnes in both design and construction (van Gestel et al., 1992, 1989).

This trend continued after the war, which saw a lot of engines and train sets being confiscated, not to be returned, or outright destroyed. The new train series Mat’46 was developed to replenish the supply of rolling stock (418 coaches total), together with an effort to electrify a major part of the Dutch railway network. Again, Werkspoor, and to a lesser extent, Allan and Beijes, were involved. Both 1954 and 1964 saw the introduction of newly designed train sets (Mat’54, 428 coaches, and Mat’64, 616 coaches), with Werkspoor as the main supplier. Due to capacity problems, part of the production at Werkspoor was outsourced to Düwag (sold to Siemens in 1989), in Germany. In 1972, Werkspoor ceased its activity in the construction of railway stock, and its activities were taken over by Talbot (taken over by Bombardier in 1995) in Germany (van Gestel et al., 1997a,b).

More recent train sets developed in close cooperation with the Dutch railways are the series SGM (Talbot, 1975, 240 coaches), ICM (Talbot, 1977, 482 coaches), DD-AR/DDZ (Talbot, 1991, 308 coaches) and V-IRM (Talbot/Bombardier, 1994 and later, 872 coaches). Note that all these series, from Mat’24 all the way to V-IRM, were developed in cooperation with, and exclusively for, the Dutch railway. The year 2009 therefore marked an important change, with the introduction of the SLT (Bombardier/Siemens, 648 coaches). At first glance, this seems to be a train set developed by familiar companies: Bombardier acquired Talbot, which was the company to take over the activities of Werkspoor in ’72, and Siemens acquired Düwag, which was tasked with producing a part of the Mat’64 train series. A key difference however is that the SLT series is an “off the shelf” train set, so to speak: it is based, in large part, on the German train set BR 425, developed by Siemens and Bombardier (van Gestel et al., 1997a, RailWiki (DDAR, VIRM, SLT)).

This change in acquiring rolling stock—from actively cooperating with industry in the specification of requirements and design, towards a role as buyer of an existing design, albeit with adaptations—also brings important changes towards maintenance operations (van Dongen, 2011). An evident one is the gaining of technical knowledge required to perform maintenance. Cooperating
in the development of a train set leads to a large build-up of technical knowledge, whereas, when buying an existing design, one has to rely on the manufacturer of that design to impart technical knowledge on maintenance engineers. A somewhat more subtle point is the interaction between train design and maintenance capabilities: the capabilities of the maintenance organization can have influence on design choices. Familiarity with certain procedures will be reflected in design choices, which will ease the complexity of maintenance operations.

Compounding the situation above is the increase in technical complexity of trains. Due to technical progress, trains are computerized more and more, and contain highly complex electrical systems for functions like traction, braking and current conversion and control—systems that used to be largely mechanical in nature in the past. In addition, more and more systems of a facilitating nature are present in a train: HVAC systems, passenger information systems, on-board internet.

In light of the trends of buying existing train designs, and the increasingly complex technical nature of these systems, it is evident that an applied research and development program focused on the changing maintenance operations surrounding these trains has merit, for more than the usual reason of a continuous improvement upon the current state of the art—there is a clear change in the environment in which maintenance operations take place.

1.2 Railway organizations

Another important part of the environment of a maintenance organization is the system of railway organizations around it. To gain a better understanding of how NedTrain operates, we look at how the system of railway organizations in the Netherlands is structured. In Figure 1.1, the major railway organizations in the Netherlands are depicted, together with their relationships.

It is important to keep in mind that this organizational structure has been subject to some large changes throughout history. As in many European countries, the railway industry initially consisted of several different pioneering companies. Gradually, these companies consolidated through a series of mergers and take-overs, leading to the establishment of the Nederlandse Spoorwegen (NS, the Dutch Railways), starting as a rate harmonization collective, which transformed to an interest group in 1917, and to a state-owned company in 1938. This process started to reverse again in the period between 1995 and 2002, when the European trend of privatization emerged, ending in the current situation.

In this situation, the railway infrastructure is organizationally separated from the companies using it. All infrastructure is owned by ProRail, which is still a state-owned company. ProRail is responsible for ensuring railway safety, managing the railway timetables for all combined traffic, and for infrastructure maintenance and construction. Actual maintenance and construction work is not performed by ProRail itself, but by several subcontractors working under directions of ProRail. Examples of subcontractors used are Strukton, BAM and VolkerRail.
NedTrain is a part of NS Group, which is the largest railway operator in the Netherlands. NS Group can be decomposed in three important parts: railway station exploitation, train maintenance and of course passenger transportation. The railway station exploitation is performed by NS Stations. This division is tasked with the design and construction of new railway stations, the renovation and modernisation of existing railway stations, and with the managing of, for example, shop concessions in the larger railway stations.

The majority of passenger trains running on the Dutch railway network are operated by NS Reizigers (NSR). Other NS Group subsidiaries active in this field are NS International, which operates international and high speed lines, and Abellio, which operates some railways in the United Kingdom and in Germany. Other passenger train operators active in the Netherlands are Veolia Transport, Arriva (part of Deutsche Bahn), Connexxion, Syntus (partially owned by NS) and Euro-Express Treincharter (EETC, operating several seasonal international trains). Other users of the railway network are the various railfreight operators, such as Captrain (owned by SNCF, the national railway company of France), DB Schenker Rail (owned by Deutsche Bahn, of which the former NS Cargo is now part), ERS Railways, and Rotterdam Rail Feeding.

The other company in this group is NedTrain, which performs rolling stock maintenance. NedTrain is currently contracted to perform maintenance for the rolling stock used by NSR and NS International. The fleet used by these two divisions has a size of around 3000 passenger carriages, of which some 250 are under some form of maintenance at any one time. These maintenance activities are scheduled with a time horizon of roughly two weeks, in which somewhere between 20 and 30 trains are serviced in a single workshop, with approximately 40 tasks per train, resulting in a schedule containing between 800 and 1200 tasks.

Maintenance operations are spread out over 37 locations throughout the Netherlands (see Figure 1.2 for a map). A distinction can be made between service locations and workshop locations; both perform different types of maintenance.
Figure 1.2
A map of the Netherlands, showing all (passenger) railway connections. All service and maintenance locations of NedTrain are marked on the map.
1. Introduction

- At a service location (‘servicebedrijf’ (SB) in Dutch) day-to-day maintenance is performed. Such maintenance consists of cleaning, safety checks and minor repairs, which are usually performed overnight in an single shift of eight hours. Service locations are marked with small circles in Figure 1.2.

- At a workshop location (‘onderhoudsbedrijf’ (OB) in Dutch) periodic maintenance is performed. Roughly every three months, a train undergoes a mostly fixed set of preventive and corrective maintenance actions. The usual duration of such maintenance is approximately three days. Trains that break down in service and cannot be repaired at location are also sent to service locations. In Figure 1.2, workshop locations are marked with small squares.

Workshop locations are also used to perform refurbishments. Such refurbishments are only executed once or twice in the lifetime of a train, to significantly extend its usable life.

The core task of NedTrain is guaranteeing fleet availability in the broadest sense of the word. Firstly, this means that trains should have as few breakdowns during service as possible. To ensure this, NedTrain has to focus on the quality of their maintenance work. The time at which maintenance is performed is important as well: a train should not be left running too long without maintenance, since this raises the risk of a breakdown, lowering availability. On the other hand, frequent maintenance has a negative impact on fleet availability as well. A second aspect impacting availability is the number of trains undergoing maintenance at the same time, and the length of the maintenance visits. If maintenance can be performed quickly, fewer trains will be out of service for maintenance simultaneously, increasing availability for passenger transport. Such an increase might mean that fewer trains are needed to guarantee a certain passenger transport capacity. But, note that short maintenance times are only worthwhile if they can be relied on: otherwise, the schedule of NS will be disrupted, or insufficient passenger transport capacity will be available, resulting in fines for NedTrain.

Maintenance activities are largely scheduled by hand. This currently is a feasible approach, since (scheduled) maintenance takes place only every three months, and it is possible to schedule these activities with a lot of slack. In such a three-monthly visit, a lot of tasks are clustered in a single visit. Durations of the individual tasks that need to be performed in these visits are uncertain, but the number of tasks is large, which reduces the variance. Combined with the allowed slack in scheduling, this results in a feasible mode of operation. If this slack is not enough, as a last resort it is sometimes possible to omit certain maintenance activities.

However, there is a push for higher efficiency, caused by various desires in the organization with respect to maintenance operations:

- There is a desire to fragment maintenance visits into multiple, smaller, visits, which can then be scheduled in off-peak periods, so that more rolling stock is available at peak periods.
• There is a desire to increase the efficiency of maintenance operations by reducing slack, so that a train in the workshop has less idle time.

• And lastly, there is pressure to avoid the last resort option of omitting activities.

To be able to support these wishes, it is important to have automated systems that can support the scheduling of maintenance activities.

1.3 Research program at NedTrain

The previous two sections outline some of the changes NedTrain is facing with regard to its maintenance operations. The acquisition procedure of rolling stock has changed—from a cooperation between operator and constructor, where knowledge was transferred in both directions, towards a buyer/seller-relationship in which knowledge is shared only reluctantly. This change introduces new challenges in the field of maintenance: the structure, tooling or knowledge of a certain maintenance organization might be ill-suited for the maintenance procedures as devised by the constructor. The scheduling of maintenance activities is also faced with new challenges due to the changing environment: the desire to split the large maintenance visits into multiple smaller visits increases the importance of having accurate and flexible schedules.

In an effort to address these problems NedTrain has initiated a broad research effort. The goal of the research program is to enable and support NedTrain in being a leading rolling stock maintenance company (Huisman, 2009). The direction is governed by the key performance indicators for the fleet:

• total cost of ownership, or life cycle cost;
• availability of sufficient transport capacity;
• reliability of transport; and
• quality of transport.

The program is divided into three separate studies, which are closely interrelated. The division is made using the various time scales of the processes involved in the procuring, supporting and maintaining of rolling stock, and separate studies are performed on strategic, tactical and operational aspects:

**strategic** At the strategic level, the decisions with the longest time scales are made. These are decisions on the buying of rolling stock and their specification—typically, rolling stock lasts about three to four decades, so that decisions on this level can have a long-lasting impact on performance levels. Another long-term strategic decision concerns the location of maintenance workshops and the available equipment.

**tactical** With a time scale of about one to five years—between the strategic
and operational level—the tactical level concerns the allocation and management of spare parts, human resources and maintenance tasks. With the introduction of new rolling stock comes the option of contracting out some of the maintenance work to the manufacturer. To evaluate the use of such an option, decision makers need to have a good understanding of how such changes in the supply chain influence the key performance indicators.

**operational** The individual jobs where the actual maintenance work on the rolling stock is performed lie on the operational level. The scheduling of these jobs is highly dynamic, owing to the dominant policy of condition-based maintenance: parts are only repaired or replaced once they are defective or once their condition has deteriorated below a certain threshold. This creates unforeseen fluctuations in workload and spare parts requirements, which are currently dealt with by having sufficient buffer stock, and by having a fair amount of slack in the maintenance schedule.

The strategic aspects are investigated at the University of Twente, where research is performed on assessing the supportability, in terms of maintenance and logistics, when acquiring new rolling stock (Parada Puig et al., 2011), and on the decision to repair certain components of the train in the workshop itself, or to replace them, and repair them in another more specialized shop. Tactical aspects are investigated at the Technical University of Eindhoven, in the form of a framework for spare parts planning and control (Arts, 2013).

This thesis will treat the last part, the operational aspects of maintenance. First, an overview is given on the current situation and practices, since these (partially) dictate some of the constraints on any approach used to solve the problem.

### 1.3.1 Operational aspects of maintenance

In the current situation, the maintenance workload for each workplace is set in advance in cooperation with NSR, and depends on the schedule of the rolling stock. NSR decomposes its schedule into three different levels:

**rolling stock unit** At the lowest level are the schedules of the rolling stock units themselves (‘materieeleenheid’). Each such unit is (for practical purposes) indivisible, and uniquely identified by a number.

**path** A path, sometimes (confusingly) called train, is the smallest indivisible task a unit can carry out. A *regular path* connects two major nodes in the railway network together and is used for the actual transport of passengers. There are also *maintenance paths*, these lead from and to maintenance facilities. Each path is to be executed by a rolling stock unit of a specified type. Note also that only one unit is to be assigned to each path. If passenger flow dictates higher capacity than a single unit can provide, the schedule contains additional paths departing and arriving at the same time in the same locations.
circulation The circulation (‘omloop’) is at the top of the scheduling structure. They connect paths together, such that they can be used to allocate the rolling stock units. Two paths can be connected to each other in a circulation if the arrival station of one path is identical to the departure station of the other, and if the arrival time precedes the departure time. Each circulation has a length of roughly three months, and has a maintenance path at the start and at the end, such that timely maintenance is automatically performed.

As an example, take the 1723 train; this is the 07:08 intercity service departing at The Hague Central Station, arriving at Enschede at 09:32. In the train schedule, this train is connected to some other train, departing from Enschede after 09:32. Such a pair of trains \((t_1, t_2)\) is called a (regular) transition (Maróti and Kroon, 2007): both \(t_1\) and \(t_2\) are planned to be carried out by the same rolling stock unit, and \(t_2\) is the earliest task after \(t_1\). In this fashion, a chain of trains is formed into a circulation. The rolling stock type of these trains (and hence of the circulation) is ICm (‘Intercity-materiaal’). The 4011 is an example of such a unit, and could be assigned to execute the train services in this circulation. Since the usual length of a circulation is about three months, ending in a visit to a maintenance location, each rolling stock unit visits a maintenance location every three months if the schedule is executed without any disturbances.

In practice however, this rarely happens. Delays cause disturbances, which then might cause material units to be switched to a different circulation. Therefore, it happens often that a unit that needs maintenance has no maintenance path scheduled in its circulation at that time. Additionally, when breakdowns occur, a unit might need to undergo extra maintenance at the earliest possible opportunity. To schedule maintenance visits, the circulation is modified with interchanges. For this, a small number of regular transitions \(n\) is considered, of the form \((t_i, t'_i)\) with \(1 \leq i \leq n\), where the arrival stations of all trains \(t_i\) is the same, such that the time intervals between arrival and departure of all trains overlap. Two train units \(i\) and \(j\) participating in these transitions can be interchanged, by removing their regular transitions and adding two custom transitions \((t_i, t'_j)\) and \((t_j, t'_i)\). By extending this idea to more than two trains, and by applying it in more than one location, interchanges can be added to divert rolling stock units needing maintenance to a maintenance path in time. An example is depicted in Figure 1.3: Some paths (the thick lines) on (parts of) three circulations (the thin lines) are displayed. The rolling stock unit assigned to the top left path has developed a defect, and custom interchanges (the arrows) are added, to move the unit to the path on the lower right, which leads to a maintenance location. It must be noted that this approach only works for certain non-fatal defects, since the rolling stock unit must still be capable of executing the passenger service of the path leading towards the interchange to the maintenance path.

While the intricacies of how a rolling stock unit arrives at the maintenance location are not the main focus of this research, it is important to keep some of the details in mind. They impose some additional constraints on how the problem is solved. Predictions on when units will visit a maintenance location
are based on the circulation they are currently running on. This applies to unscheduled maintenance work as well (EbK, ‘extra binnenkomst’ in Dutch), and in this sense, unscheduled maintenance is scheduled as well, but only at a very late moment. The circulation method also highlights the importance of meeting the deadlines: if the maintenance is not finished in time, the unit cannot depart the maintenance location to resume its service. If maintenance is not ready in time, an interchange can be performed by switching the unit scheduled for departure with an equivalent unit of the same time which might be finished earlier than planned.

1.3.2 Scheduling of maintenance

The actual scheduling of maintenance jobs is currently a labor-intensive task. The base schedule is the same for every week. It contains arrival and departure time of rolling stock units of a certain type, dictated by the circulations which pass through a maintenance location. Using the type of the rolling stock unit, a generic set of tasks is included for each unit. Roughly two weeks in advance, the exact unit which will arrive in a slot is known: around this time a unit needing maintenance is routed to a circulation sending it to a maintenance location in one or two weeks time, using the interchange method described above. Additionally, it is ‘flagged’; this means that this unit should not partake in any interchanges, since this would disturb its path towards the maintenance facility. A disturbance this late could cause difficulty adjusting the path back using interchanges, to prevent this from happening the flagging method is used.

If problems have been reported for this unit, or if part of the maintenance of last time was not completed, some maintenance tasks are known in advance. The rest of the tasks to be performed is usually not known in advance. This might seem odd, since there is obviously a maintenance plan for each rolling stock series describing which tasks need to be performed at which interval. The difficulty however is that the tasks in this plan are often loosely defined in terms of either time or resource requirements. The time requirements might change with the amount of resources allocated to the tasks, or with the type of resources allocated. Time requirements might also be location-dependent: at one workshop, there might be more experience with some task, leading to lower time requirements. But the biggest problem is that most maintenance tasks are condition-based, and the condition of the rolling stock unit is generally

Figure 1.3
An example of the interchange method, where one rolling stock unit is routed to a maintenance path (adapted from Maróti and Kroon, 2007).
not known in advance. These tasks generally have the form of an inspection, followed by some (conditional) corrective action. Consequently, this leads to uncertainty with respect to the time needed to execute the task, and uncertainty with respect to the resources required, in terms of personnel, equipment and often parts as well. Collectively, the issues described above can be referred to as task uncertainty, and form one part of the scheduling problem.

Schedulers currently allocate workplace resources manually (i.e., tracks, possibly of special type, and personnel of different types) to the various (high-level) tasks in the base schedule. Any information on the actual tasks to be performed is of course used, but often, little is known and the schedulers have to rely on their own expertise to know how many maintenance engineers are needed to ensure a task is completed on time. Problems are compounded by the fact that the spare capacity that is theoretically available (per the base schedule dictated by the train paths) is usually fully used by unexpected extra arrivals. The schedulers must also contend with unexpected capacity changes, for example due to personnel taking sick leave. These issues form the second part of the scheduling problem, resource uncertainty. Note that the task uncertainty also contains an uncertainty with respect to resources. The difference is that the resource uncertainty caused by task uncertainty pertains to the resource usage, as opposed to resource availability.

### Table 1.1

<table>
<thead>
<tr>
<th></th>
<th>No uncertainty</th>
<th>Uncertainty</th>
</tr>
</thead>
<tbody>
<tr>
<td>No human recovery</td>
<td>Smooth shop optimize</td>
<td>Stress shop support reactive scheduling</td>
</tr>
<tr>
<td>Human recovery</td>
<td>Social shop schedule as advice</td>
<td>Sociotechnical shop schedule as framework</td>
</tr>
</tbody>
</table>

#### 1.3.3 A classification of scheduling situations

With the knowledge of some of the practical aspects of maintenance scheduling in the NedTrain workshop, we now turn to the work of Wiers (1997), in which the interaction between (automated) scheduling systems and human scheduling is analyzed. Wiers uses the dimensions of uncertainty and human recovery are used to distinguish between four stereotypical scheduling situations. Here, uncertainty refers to task uncertainty as discussed above: incomplete information about the tasks that need to be performed. Human recovery refers to the ability of the teams executing the schedule in the workshop to compensate for this uncertainty, by utilizing flexibility present in the schedule. Wiers makes a distinction between the ability of operators to correct for uncertainty (human recovery) and the authority they have to do so (autonomy).

In Table 1.1, the four stereotypical scheduling situations as described by Wiers (1997) are shown. The names of these situations refer to the distribution of autonomy between the (central) scheduler, and the teams executing that schedule.
in the workshop. Each of the possible distributions has certain consequences for the type of scheduling information system employed.

In the case of NedTrain, there is both a high uncertainty in the tasks to be executed, as well as the opportunity for human recovery: given the autonomy, the teams executing maintenance tasks are often able to adjust the schedule based on information about the actual situation in the workshop. The sociotechnical shop is a good fit for this scheduling situation. In such a shop, schedules are used as framework for execution. They are not executed exactly, so expensive optimization of the schedule is to be discouraged.

1.3.4 The future of maintenance operations: some requirements

The system as described in this section of the introduction is currently in operation at NedTrain, and the research program of which this thesis forms a part is intended to provide methods for future improvement in maintenance operations. A reason for having a research program devoted to this improvement is the change from in-house participation in the design phase of new train sets to the procurement of train sets based on existing designs, as outlined in Section 1.1. The impact on the operational aspect of maintenance operations is expressed in higher task uncertainty. Separately, another important change to the operational aspect of maintenance operations currently being discussed at NedTrain is off-peak maintenance. Instead of taking a train out of revenue service for two days every three months, it might be more advantageous to take the train out of service for several shorter periods during those three months. If these periods are restricted to fall within off-peak times, this would lead to more trains being available at peak hours. This, in turn, would enable NSR to get by with fewer total trains, which could be a substantial cost saving.

For this concept to be able to work, tight and quick scheduling is of the essence. The schedules need to be tight, because the off-peak period is short. To make good use of such a short period for maintenance, there should be little slack in the schedule. Scheduling should also be quick: if disturbances cause a different set of trains to be at the workshop, it must be possible to adapt the maintenance schedule for that day quickly.

Current scheduling systems employed at NedTrain are not quick and flexible enough to adequately deal with the added requirements of off-peak maintenance operations. Assimilating new trains into the maintenance schedule is a time-consuming process, which is currently made to work by utilizing the fact that there is a lot of slack available to correct any deviations from the assumed work load of each train. Hence, there is need for a software system based on smart algorithms supporting the schedulers in a workshop. The system should be able to quickly form a work plan based on given information. The scheduler should be able to modify the resulting work plan, to take additional, implicit, preferences and constraints into account. A complicating factor is the potentially large processing time caused by the size and complexity of the problem. A maintenance schedule usually covers two weeks of maintenance work, in which between 20 and 30 trains are scheduled. Per train about 40 tasks are executed,
An important requirement is that the work plan should be flexible, in the sense that there is room for adaptation—it is unreasonable to assume that exact task information is present before commencing work on a train. When this information becomes available, for example if the need for extra work becomes apparent after an inspection has been performed on a certain component of the train, we want to be able to integrate this task into the existing work plan, without altering it. However, if an extra task, such as discussed above, or even an extra train, needs to be added to an existing work plan, it is clear that the desire for the plan to be unaltered will often be an unreasonable assumption. This leads us to the requirement of robustness: if the flexibility we have available is insufficient to integrate the extra tasks, we want the necessary adaptations to the work plan to be as minimal as possible.

A last requirement is that the execution of the work plan by independently working teams should be facilitated. Different teams operate in the same workplace concurrently—each team might be performing maintenance operations on a different train. The requirements of flexibility and robustness come into play here as well. A team desires a certain measure of independence in executing its part of the schedule, flexibility in the schedule is a property which can facilitate this. Robustness, to counter the effects of the uncertainty in the schedule, is especially important in a setting with multiple teams, as it is undesirable for delays in one team to affect the other teams.

### 1.4 Research goals

The problems above sketch a dynamic environment in which a schedule continuously evolves as new information comes in. Currently, the schedules contain a fair amount of slack to cope with uncertain information and changing circumstances. Trains visit a workplace for maintenance for a longer period than strictly necessary, which results in a sizeable number of trains being at a workshop at any one time on which no maintenance work is performed. Workplace utilization is therefore sub-optimal, but more importantly, fewer trains are available to transport passengers. To satisfy a given transportation demand, more rolling stock units are required. If, by reducing slack in the maintenance schedule, even one train less has to be bought, this represents a substantial saving. In some cases the available slack is still not enough to deal with an incident. In this case, a workshop can opt to omit certain tasks. It depends on the type of task and on safety regulations if this is possible. Schedulers start at a high level, with a baseline schedule based on the schedule used by nsr. As more information becomes available, this schedule is updated and refined, and resources are allocated to the tasks to be performed.

While this approach was satisfactory in the past, there is a demand to increase the efficiency of the scheduling process. Shorter maintenance times lead to higher availability of the rolling stock, which in turn allows a smaller fleet to
serve the same demand. As noted above, the cost saving of having a smaller fleet is significant. In the same vein, pilot studies are being conducted in which the current, long, maintenance visit is replaced by several smaller visits to the workshops—timed in such a way to avoid the rush hour, in which fleet availability demand is at its highest.

The scheduling process such as currently employed at NedTrain is based on the requirement of having a fairly large amount of slack in the schedule to be able to adapt to unforeseen circumstances. Additionally, it is a relatively labor-intensive procedure, which involves a lot of manual work. Such a process is well-suited if the tasks to be scheduled are of a static nature, but even in the current situation this is not the case. The timing of the depot visits is sometimes uncertain, but also the exact set of tasks resulting from a visit can be uncertain: inspection tasks can result in additional work. Frequent unexpected visits caused by breakdowns create additional uncertainty. The changing requirements to the scheduling process will increase the load even more.

This leads us to the first research goal, on the scheduling process itself:

**RG1** The scheduling process employed in the NedTrain workshops relies on a large amount of slack to keep unexpected events under control. In addition, it is labor-intensive and inflexible. In what way can this process be at least partially automated? Note that automation alone is not sufficient: the process of constructing and adapting a schedule has to be quick enough to make it part of an interactive process, and it has to be flexible, to be able to adapt to the incomplete and changing nature of the environment.

Automation of at least part of the scheduling process is an obvious solution to decrease the labor-intensive nature of the process. This in itself can be a part of increasing the flexibility of the scheduling process: if the process to generate a schedule is quick, and if a user is able to influence the generation process by asserting different preferences, a set of different alternative schedules can be created. This also coincides with the idea that not all of the preferences of what constitutes a “good schedule” can be encoded in a formal way—some form of user intervention will always be necessary.

Having a portfolio of automatically generated schedules gives a form of flexibility: if unexpected events occur, switching to a different schedule might still result in meeting imposed deadlines, even if the slack in the schedule is to be reduced. But this seems a rather roundabout way of solving the problem: flexibility in the process of creating schedules itself is important, but another element of improvement is the flexibility in the schedules itself. If a schedule is very rigid, any change in one of the tasks necessitates a restart of the scheduling process, to partially or completely adapt to the new situation. In contrast, if the schedule is flexible as well, adaptation to such a new situation could be much easier.

**RG2** Schedules, by their nature, prescribe a certain way to execute a set of tasks, giving rise to a form of rigidity. To be able to adapt to uncertain events it is
desirable to reduce this rigidity.

a) As a primary goal, we want to have a schedule which prescribes task execution in a *flexible* way, such that we can adapt to changing events without needing to revise the schedule, and without having to resort to using large amounts of slack.

b) As a secondary goal, we want the resulting schedule to be *robust*: if the available flexibility is not enough to be able to adapt to some events, we want the unavoidable revisions we have to make to continue executing the needed tasks to be as minor as possible.

A final challenge is the interaction between uncertainty and the structure in which the schedule is to be executed. Multiple independent teams in the workplace cooperate to execute a single schedule. Some amount of independent control for the teams operating in a workshop is desirable, especially in light of the sociotechnical shop scheduling framework as discussed in Section 1.3.3. A form of dependence between the teams is inevitable however, as they make use of shared resources to execute their tasks. This leads to situations where uncertainties in the schedule cause problems in one team to affect the other teams as well.

**RG3** The need to use shared resources to complete tasks, combined with uncertainties in the schedule negatively affects the independence of the teams in the workshop. Our goal is to construct schedules such that there is more isolation between the teams, in terms of disturbances in the execution of the schedule in one team not affecting the other teams.

Summarizing the goals above, we want to devise a method with which schedules can be created, based on a *flexible process* (RG1), where the resulting schedules are both *flexible*, in the sense that they can easily be adapted (RG2a), and *robust*, in the sense that they are resilient to the effects of uncertain events (RG2b). Additionally, we want to improve the isolation between independent teams in terms of disturbances—a disturbance in one team should have no effect on other teams (RG3).

### 1.5 Outline and contributions

In Chapter 2, we will start by examining existing research in light of the research goals introduced above. It is shown that the scheduling problem at NedTrain can be mapped to the Resource-Constrained Project Scheduling Problem (RCPSP), a well-studied scheduling problem. Several methods for solving the RCPSP are described, and as it turns out, the Precedence Constraint Posting method (PCP), which uses the Simple Temporal Network (STN) as a flexible way of representing a set of solutions to an RCPSP instance, seems an especially good fit in light of the
first problem statement. Nevertheless, at the end of the chapter, several research questions are formulated in response to the other research goals. These research questions will serve as a basis for the following chapters.

In Chapter 3 a start is made in answering the second research goal, by examining how we can transform our intuitive notion of flexibility into a method to formally measure the flexibility of an STN. It is shown that existing methods do not take the full effect of dependencies between tasks into account, resulting in an over-estimation of the available flexibility. We present a new method to measure flexibility that does not suffer from this problem. This method also gives a partial answer to the last research goal, on isolating teams jointly executing a schedule—it turns out that our method to measure flexibility also results in an optimally decoupled schedule, which is a schedule in which constraints between tasks belonging to different teams have been replaced by constraints within the team itself, without losing flexibility. This work has been published in Wilson et al. (2013a) and has received a Distinguished Paper Award at the International Joint Conference on Artificial Intelligent, a major conference in the computer science community.

Chapter 4 examines the other half of the second research goal, on how to ensure that flexible schedules are robust as well. We examine different ways of using the available flexibility in an STN, and analyze the effect this has on the performance in terms of the robustness of the schedule. It is shown that distributing the flexibility in different ways can have a significant effect on performance. In particular, it is shown that sacrificing some total flexibility to improve the equality of distribution has a positive effect. This work has been published in Wilson et al. (2013b).

Chapter 5 looks at the second research goal in a slightly different way. The idea of being flexible in how a schedule prescribes the execution of tasks has so far been interpreted as being flexible in the temporal aspect of execution. Here, we examine if it is possible to also be flexible in the sequential aspect of execution. We show that it is possible to postpone some of the ordering decisions of creating a schedule to the time of execution, creating a new strategy of being flexible in the prescription of schedule execution. It is shown that this interpretation of flexibility also leads to schedules that are robust, as described by the second research goal. This work has been published in Wilson et al. (2012).

Finally, Chapter 6 summarizes the results presented in this thesis. The research goals introduced in this chapter are analyzed again, using the information and results from the preceding chapters. We conclude that the contributions made by this thesis constitute a theoretical framework on which a practical solution to these problems can be built. Nevertheless, while giving answers to the questions raised in this introduction, many more questions can still be raised. Therefore, we briefly discuss some of the possible avenues of future research prompted by some of those questions.
In the previous chapter, the scheduling problem present at NedTrain was introduced, and we formulated three problem questions. In this chapter we will investigate existing scheduling research, with a focus on research that might help in answering these questions. Since there is a very large body of research with scheduling as the subject, we will start with formalizing the scheduling problem such as it occurs at the NedTrain workshops. It turns out that this problem maps very well to the Resource-Constrained Project Scheduling Problem, often abbreviated as RCPSP. Therefore, it is solution methods and algorithms for the RCPSP towards which we will turn next. We will pay special attention to Simple Temporal Networks (STNs), as these can be used to represent RCPSP solutions in a flexible way. Lastly, we will review the material discussed in the light of the problem questions presented in the previous chapter, and research questions will be formulated to address unsolved problems.

### 2.1 Resource-constrained project scheduling

Let us first consider the basic scheduling problem at NedTrain. It consists of a set of trains arriving and departing at a given time, with a set of maintenance tasks to be performed on each of those trains. Some ordering constraints might be present on these tasks, such that, for example, a cleaning task is carried out last, and to complete a task one or more resources may be required, such as engineers or special tools. The aim is to find a time assignment to these tasks such that all tasks are completed before the train departs, any ordering constraints are respected, and the capacity of the available resources is never exceeded.

As already mentioned in the introduction, the Resource-Constrained Project Scheduling Problem (RCPSP, see Hartmann and Briskorn (2010) for a good overview of the basic problem and a lot of common extensions and modifications to it) closely resembles this problem. Additionally, it is a widely-studied problem, both in operations research and in other research disciplines. The basic RCPSP considers a single project to be scheduled. The basic RCPSP consists of the following elements:
tasks A set $T$ of $n$ tasks to be executed, where each task $t_i \in T$ has an associated length $l_i \in \mathbb{R}$. Each of these tasks maps to a maintenance operation to be carried out, in the case of NedTrain. The lengths will be an approximation in this case, since it is not always known in advance if a repair is needed, or how long the repair will take. Note that we will restrict ourselves to using deterministic task lengths.

precedences A precedence relation $\prec$ inducing a partial order on $T$; the intuitive interpretation of $t_i \prec t_j$ being that $t_i$ has to be finished before $t_j$ can start. The precedence relation is used to enforce certain orderings on maintenance tasks: for example, one task might specify the removal of a certain component, such that another component becomes accessible on which the second tasks has to be performed.

resources A set $R$ of $m$ renewable resources, where each resource $r_k \in R$ has integer capacity $\text{cap}(r_k) \geq 1$. These resources are used to model personnel and special tools, both with limited availability, with which the maintenance tasks are to be performed.

In some formalisms a distinction is made between renewable and consumable resources. Renewable resources become available again after a task using them has completed, and can be used to model, for example, engineers and tools. Consumable resources are lost upon completion of a task, and can be used to model materials that are used in the construction of some artefact. In our work, we assume such materials are always readily available in a warehouse, and as such we will not include them in the model—we will be dealing with renewable resources only.

resource consumption Lastly, the tasks need to be connected to the resources they require: for each $r_k \in R$ and $t_i \in T$, $\text{req}(t_i, r_k) \in \mathbb{N}$ specifies the amount that task $t_i$ requires of resource $r_k$ to be executed. If this number is zero, this particular resource is not required.

In addition to the elements of the basic rcpSP, a release time $s_i \in \mathbb{R}$, and a due time $d_i \in \mathbb{R}$ are associated with each task $t_i$, which are used to model, for example, the time at which the train enters and leaves the workshop. To model multiple trains, it can at times be convenient to use the Resource-Constrained Multi-Project Scheduling Problem (RCMPS), which is in essence a group of rcpSPs, whereby the set $R$ of resources is shared among all projects. Each of the projects then represents the maintenance work for a single train, and the individual projects have release and due times that represent the times at which the trains enter and leave the workshop.

A more formal definition of the scheduling problem at NedTrain can now be formulated as follows:

---

1 Some scheduling formalisms allow the pre-emption of a task, meaning that the execution of a task can be stopped when it is in progress, and resumed at a later time, sometimes at the cost of slightly longer total execution time for that task. We will however assume that pre-emption is not possible—once a task has been started, it can not be stopped before its completion time.
Definition 2.1 An instance of the resource-constrained multi-project scheduling problem with release and due times can be specified using a tuple \( \langle T, <, R, l, s, d, \text{req} \rangle \), where \( T = \{ t_1, \ldots, t_n \} \) is a set containing \( n \) tasks, \( < \) a precedence relationship inducing a partial order over the tasks in \( t \), \( R \) a set of \( k \) resources \( r_k \), with \( \text{cap}(r_k) \geq 1 \) specifying the capacity of that resource. The function \( l : T \to \mathbb{R} \) defines the length of each task \( t_i \in T \), and the functions \( s : T \to \mathbb{R} \) and \( d : T \to \mathbb{R} \) define release times and due times for the tasks. Lastly, \( \text{req} : (T, R) \to \mathbb{N} \) specifies the amount each task uses of the available resources in \( R \).

Example 2.1 Consider the (fictional, and very short) maintenance visit of train 8604. This train will arrive in the workshop on Monday morning, at 08:00, and needs to return to revenue service at 12:00, after the tasks that are described below are performed:

- An exchange of the air compressor, due to air pressures being out of tolerance. Besides an engineer, a winch is required to do the heavy lifting. Roughly one hour of work is required.

- A routine check of the brake pad thickness, which takes 45 minutes if two engineers work together.

- A verification of the proper operation of the ATB system (“automatische treinbeïnvloeding”, a train protection system), which is to be performed by an engineer by using specialized ATB testing equipment on the train. This check is expected to take around 30 minutes.

This can be modeled as an instance of the RCPSP, with a set \( T = \{ t_1, t_2, t_3 \} \) of three tasks, for, respectively, the compressor exchange, the brake check, and the ATB verification. If temporal information is specified in minutes, relative to Monday morning 08:00, we have \( s_1 = s_2 = s_3 = 0 \), and \( d_1 = d_2 = d_3 = 240 \), to model release and due times such that the tasks take place when the train is in the workshop. We set \( l_1 = 60, l_2 = 45 \) and \( l_3 = 30 \) to model the lengths.

Several resources can be identified in the example: maintenance engineers, a winch to do heavy lifting on the compressor, and ATB test equipment. To model this, we set \( r_1 = 2 \), to indicate two engineers are available, and \( r_2 = r_3 = 1 \), to indicate one winch and one piece of test equipment. The resource consumption of the three tasks is modeled by setting \( \text{req}(t_1, r_1) = 1, \text{req}(t_2, r_1) = 2 \) and \( \text{req}(t_3, r_1) = 1 \) for the engineers, and \( \text{req}(t_2, r_2) = 1 \) and \( \text{req}(t_3, r_3) = 1 \) for the winch and test equipment, respectively.

Additionally, since the ATB system is tightly coupled to the brake system, there is a precedence constraint between the brake check and the ATB check: the latter can only be performed after the brake check has been completed, so we have \( t_2 < t_3 \).
The goal of the RCPSP is to find a schedule for an instance of the problem. Such a schedule is a function \( \sigma : T \rightarrow \mathbb{R} \), which assigns start times to all the tasks \( t \in T \), which satisfy all the constraints in the problem: release and due times are respected, precedence relationships hold, and resource capacities are not exceeded.

**Definition 2.2** A solution to an RCPSP \((T, <, R, l, d, r, req)\), also called a schedule, is a function \( \sigma : T \rightarrow \mathbb{R} \) assigning a start time to each task in \( T \), such that all the constraints in the problem are satisfied: For each \( t_i \in T \), \( s_i \leq \sigma(t_i) \leq d_i + l_i \), such that release and due times are satisfied; for each tuple \( t_i < t_j \) we have \( \sigma(t_i) + l_i \leq \sigma(t_j) \), such that the precedence constraints are satisfied; and lastly, for any resource \( r_k \) at any time point \( \tau \), \( \sum_{t_i \in \text{active}(\tau, \sigma)} \text{req}(t_i, r_k) \leq \text{cap}(r_k) \), where active(\( \tau, \sigma \)) = \{ \( t_i : \tau \in [\sigma(t_i), \sigma(t_i) + l_i] \) \}, such that the capacity of the available resources is never exceeded.

**Example 2.2** Continuing the example started above, we note that the three tasks cannot be executed at the same time, as this would require four engineers, and only two are available. A solution is to start with \( t_2 \) as soon as the train enters the workshop, so we set \( \sigma(t_2) = 0 \). The other two tasks can then be started after \( t_2 \) has finished, so we set \( \sigma(t_1) = \sigma(t_3) = \sigma(t_2) + l_2 = 45 \). Now, \( \sigma \) assigns starting times to all tasks, and we have a complete solution. Note that this solution also satisfies the precedence constraint \( t_2 < t_3 \).

There are many different schedules that can solve a particular instance. Instead of simply treating the RCPSP as a constraint satisfaction problem, in which we aim to find one single solution that satisfies all constraints, the RCPSP is often turned into an optimization problem, in which an objective function is used to find an optimal solution. In nearly all scheduling models, the vector \((\sigma(t_1) + l_1, \ldots, \sigma(t_n) + l_n)\) of completion times is used as input for the objective function \( \kappa : \mathbb{R}^n \rightarrow \mathbb{R} \) that maps these vectors onto a one-dimensional scalar. Many authors enforce a regularity condition

\[
(\sigma(t_1) + l_1, \ldots, \sigma(t_n) + l_n) \leq (\sigma'(t_1) + l_1, \ldots, \sigma'(t_n) + l_n) \Rightarrow \\
\kappa(\sigma(t_1) + l_1, \ldots, \sigma(t_n) + l_n) \leq \kappa(\sigma'(t_1) + l_1, \ldots, \sigma'(t_n) + l_n)
\]

on the objective functions used (Bartusch et al., 1988) that states that the objective function must be nondecreasing in the completion times of the tasks, i.e., higher completion times can never result in a lower objective function. Many different objective functions have been used in the literature (Hartmann and Briskorn, 2010), and for the common ones this condition holds:

**makespan** The most basic and widely encountered objective function optimizes the makespan of the solution, which is usually defined as the completion

---

2If earliness is to be penalized as well, for example in just-in-time production facilities, the objective function is no longer regular (see Baker and Scudder (1990) for a review of such objective functions).
time of the latest task that is executed:

\[
\text{makespan}(\sigma, T) = \max_{t_i \in T}(\sigma(t_i) + l_i)
\]

In multi-project problems, this objective function can be modified to optimize the average makespan of the individual projects.

**lateness/tardiness** If the constraints of the problem cannot all be satisfied, a common strategy is to minimize the total lateness of the project. First, we define \(L_i = \sigma(t_i) + l_i - d_i\) as the lateness of task \(t_i\) when executing schedule \(\sigma\). Then, we can define the lateness of a solution \(\sigma\) as

\[
\text{lateness}(\sigma, T) = \sum_{t_i \in T} L_i.
\]

The tardiness objective is similar, the difference is that it cannot be negative:

\[
\text{tardiness}(\sigma, T) = \sum_{t_i \in T} \max(0, L_i).
\]

**resource-based** A different kind of objective looks at the resource usage of the solution, usually combined with makespan minimization. The idea is that the capacity of the resources is optimized as well; the lower the capacity needed to execute a certain solution, the less investment is needed to make these resources available.

The situation at NedTrain can be compared best to the tardiness objective function, with a slight difference: for each train that is delivered late, a fixed penalty is given, which does not depend on the amount of tardiness.

The **rcpsp** is an NP-complete problem, since it is a generalization of the job shop scheduling problem, for which NP-completeness has been proven (Garey et al., 1976; Blazewicz et al., 1983). Despite this fact, we will still start with a discussion of optimal methods to solve the rcpsp, since the NP-completeness has not withheld numerous researchers to come up with new and innovative optimal solution methods. After this, we will turn to non-optimal methods, using heuristics to arrive at high-quality solutions. Many of these methods however still result in a fixed schedule, which is undesirable in the highly dynamic environment at NedTrain. Therefore, we will also turn to methods using a different, more flexible representation of rcpsp solutions.

Our discussion of rcpsp solution methods will take place in light of the specific criteria of the NedTrain workshop planning problem:

- The first of these is the performance of the method, or, to be exact, its ability to arrive at a solution within a reasonable time frame. The size of the problem is an important factor for this criterion: if we are to plan ahead for two weeks, we need to plan maintenance activities for roughly 40 trains. The number of tasks per train is variable, but if we take 10 tasks as average (which is a low estimate), the number of tasks to be planned is 400—a substantial number.
Compounding this problem is the need for interactivity: as already described, it is desirable to let a planner interact with the planning software. For this to work, the method must be able to quickly solve the problem: it is unrealistic to ask the planner to wait more than a couple of minutes.

- The second important criterion is the flexibility of the method. Due to the uncertain nature of the tasks to be planned, changes and incidents during execution of the schedule are to be expected. The flexibility requirement represents the desire that the schedule be easily and efficiently adaptable to overcome the consequences of such changes and incidents.

2.2 Mixed Integer Linear Programming solutions

A traditional method to provide exact solutions to the RCPSP is to encode it as a Mixed Integer Linear Programming (MILP) problem, and to use a general MILP solver to find a solution. Many different encodings have been suggested in the literature over the years. Two important and very different encoding types will be discussed below:

- (0–1) time indexed formulations, using binary decision variables $x_{it}$ such that $x_{it} = 1$ if and only if task $t_i$ starts at time $t$; and

- flow-based formulations, which involve binary decision variables $x_{ij}$ and continuous start-time variables $S_i$, such that $x_{ij} = 1$ if and only if $S_i + l_i \leq S_j$, meaning that $t_i$ precedes $t_j$.

Note that whereas the flow-based formulations use continuous time variables, the time indexed formulations are restricted to time points in $\mathbb{N}$. In the context of task scheduling this is usually not a big limitation, since the start times of tasks are usually measured at the scale of minutes. Especially in combination with uncertainty, higher accuracy is not necessary.

Many MILP-formulations are sensitive to the size of the problem horizon. Therefore, a preprocessing step is often used to find the bounds $[\text{est}(t_i), \text{lst}(t_i)]$, whereby $\text{est}(t_i)$ is the earliest start time of $t_i$, before which $t_i$ can never start, and $\text{lst}(t_i)$ is the latest start time, after which $t_i$ can no longer start. These bounds are used to limit the scope of the search for a solution, and are based on the precedence constraints and release and due times of the original problem. They can be found in polynomial time (Koné et al., 2013).

In the following sections we will give a brief overview of both time-indexed and flow-based methods.

2.2.1 Time-indexed formulations

In these formulations, decision variables are used that are indexed by discrete time. The classical formulation is due to Pritsker et al. (1969), in which there is
only one type of decision variable, $x_{i\tau}$, indexed by time and task index; $x_{i\tau} = 1$ if task $t_i$ starts at time $\tau$.

To give an impression of the structure of the constraints involved, we will discuss the constraint used to encode a precedence constraint. First, note that there is only one $\tau$ such that $x_{i\tau} = 1$, for a certain $i$—this is enforced by another constraint. A precedence constraint between $t_i$ and $t_j$ can then be formulated as

$$\sum_{\tau=est(t_i)}^{lst(t_j)} \tau x_{j\tau} \geq \sum_{\tau=est(t_i)}^{lst(t_j)} \tau x_{i\tau} + l_i \quad \text{if } t_i < t_j. \quad (2.2)$$

Since there is only one value for $\tau$ for which $\tau x_{j\tau} = 1$, the left summation is in fact equal to the currently selected start time for $t_j$. Similarly, $\tau x_{i\tau} + l_i$ is equal to the currently selected end time for $t_i$. Therefore, this constraint ensures that $t_j$ starts at or after the finish of $t_i$. Other constraints of the $\text{rcpSP}$ are encoded using a similar strategy.

The decision variables are indexed by both time and task index, this means that the number of decision variables scales with the size of the temporal horizon. As a result, this formulation is highly sensitive to the (temporal) size of the problem, since larger problems involve larger numbers of decision variables. This problem is partially mitigated by the high quality of the linear-relaxation bounds available (Artigues et al., 2008). In Möhring et al. (2003), a method based on a time-indexed formulation is used to solve well-known benchmark instances of up to 120 tasks. The time horizon size is shown to be the dominant factor in the run time, and not all instances of this size can be solved to optimality. Still, instances with 120 tasks are relatively small compared to the scheduling problems that have to be solved at NedTrain, making this class of methods impractical. Even if instances could be solved within a reasonable time frame, we still must deal with the uncertain nature of the execution environment at NedTrain. An exact solution to the scheduling problem has fixed time assignments, any change would invalidate the schedule, leading to a time-consuming re-computation.

### 2.2.2 Flow-based formulations

Formulations based on resource flows, such as described by Artigues et al. (2003), exploit the fact that, in any feasible solution, any task $t_i$ using some amount of resource $r_k$ transfers some or all of this resource to some other, unique, task $t_j$ following it, when $t_i$ is done executing. In a practical situation such as a maintenance workshop this is also very obvious: a resource such as a maintenance engineer will move from one task to a distinct next task, given by the schedule. The model uses a set of continuous start-time variables $S_i$ for each task, and to model the flow of resources, decision variables $f_{ijk}$ are introduced. They denote how many units of resource $r_k$ are transferred from task $t_i$ to $t_j$, after $t_i$ has finished executing. The flow of resources originates and terminates in two special source and sink tasks $t_0$ and $t_{n+1}$, with zero length, for which we set $\text{req}(t_0, r_k) = \text{req}(t_{n+1}, r_k) = \text{cap}(r_k)$ for all resources.
Again, we will not discuss all the constraints involved, but we will use the precedence constraint as an example. In this case, variables \( x_{ij} \) are used:

\[
x_{ij} = \begin{cases} 
1 & \text{if } t_i < t_j \\
0 & \text{otherwise.}
\end{cases}
\] (2.3)

These variables are used in the constraint

\[
S_j - S_i - M x_{ij} \geq l_i - M \quad \forall t_i \in T \cup \{t_0\}, \forall t_j \in T \cup \{t_{n+1}\},
\] (2.4)

where \( M \) is an arbitrary large integer, usually set to the scheduling horizon size. If \( t_i \neq t_j \), this reduces to \( S_j - S_i \geq l_i - M \), which is always true, given the large value of \( M \). If \( t_i < t_j \), this reduces to \( S_j - S_i \geq l_j \), ensuring that \( t_j \) starts at or after the finish of \( t_i \).

Since this formulation uses continuous-time variables for the start times of the tasks, the temporal horizon scaling problem encountered with the previous formulation does not occur. But this formulation also has its problems: it is difficult to find high-quality linear-relaxation (lower) bounds, making it hard to solve for problems that are large in the number of tasks. Applegate and Cook (1991) show some numerical results for job shop problems with 10 tasks and 10 resources (machines); different cuts are discussed, but for all of them there is still a large gap between the lower bound and the optimal value. This formulation is therefore mainly of use for instances with large time horizon, which cannot be solved within a reasonable time frame by time indexed formulations (Koné et al., 2013). Nevertheless, instances with large number of tasks still form a problem for this formulation, and as the NedTrain problems are large in both number of tasks and problem horizon, it is clear that flow-based formulations are unsuitable as well.

### 2.3 Branch and bound methods

Next, we will discuss branch and bound methods, which, in contrast to encoding the \textsc{rcpsp} as a set of constraints to be satisfied by a general \textsc{milp} solver, explicitly construct a solution in the form of a schedule. The general form of the branch and bound procedure is that of a search tree: at each point in the search procedure where multiple choices are available, branches are created for all of these choices. Using a backtracking procedure, all of these branches are eventually evaluated. The other key part of the algorithm is the bounding part: before exploring a new branch, a lower bound is computed on the solution quality that this branch will result in. If the computed bound is higher than the currently known best solution, it is certain that none of the solutions resulting from this branch can improve upon the current solution, and the branch is not explored. The combination of these mechanisms guarantees that an optimal solution will be found (given enough computational power), possibly without exploring all solutions explicitly (Demeulemeester and Herroelen, 2002).
2.3. Branch and bound methods

2.3.1 High-level algorithm

A competitive branch and bound procedure for solving the \textit{RCPSP} was proposed by Demeulemeester and Herroelen (1992), based on the time incrementing depth first search procedure proposed by Christofides et al. (1987). In these procedures, schedules are constructed incrementally: during the execution of the algorithm, a part of the tasks is assigned a starting time. Using the time assigned to the latest task, the set of tasks can be divided into three sets: finished tasks, tasks in progress, and unfinished tasks. In the set of unfinished tasks, certain tasks are said to be \textit{eligible}: these are the tasks for which all predecessors are finished. The start times for the finished tasks are temporary as well, in the sense that these can be delayed at a later time.

The current partial schedule is then extended by selecting a task from the set of eligible tasks. In some cases, there is a unique task, or a unique combination of tasks, which can be scheduled directly. If this is not the case, a \textit{resource conflict} is at the root of this: some combinations of the eligible tasks and tasks in progress result in resource over-subscription. This leads to branching in the tree of solutions: a set of tasks is picked to be delayed, by adding precedence constraints between the last scheduled task and this set. This procedure is repeated until all the tasks have been added to the schedule.

In some cases however, not the whole schedule has to be constructed: the algorithm keeps track of a lower bound for the makespan, and if it can be proven that the current branch will not improve on this bound, it can be abandoned. The tree is then traversed upwards, to find a decision point at which alternative choices are still available. The choice made in the abandoned branch is undone, and an alternative choice is made. If we manage to find a complete solution, we check if the upper bound is equal to the lower solution: if this is the case, we have found the optimal solution.

Another method to effectively prune parts of the search space is the application of dominance rules. These rules allow us to prove that, if they apply, the current solution is dominated by another solution, and will never lead to an optimal solution.

Demeulemeester and Herroelen (1997) introduces a set of lower bounds and dominance rules that improve the performance markedly, creating a very competitive method for finding optimal solutions. Still, branch and bound methods suffer from the general tractability problems associated with solving an \textit{NP}-complete problem to optimality as well, and due to the size of the NedTrain problem instances, these methods are again impractical to use in this case.

For these reasons, we will next turn to heuristic methods to solve the \textit{RCPSP}. Such methods do not purport to find an optimal solution—often, they cannot even guarantee that a solution will be found. Still, in many common cases, these methods are able to find a solution of reasonable quality in a fraction of the time an exact method would need.
2.4 Schedule Generation Schemes

The first heuristic methods we will discuss are the serial and parallel schedule generation schemes (SGS, Hartmann (1999); Kolisch (1996)). After this, we will turn to more complex heuristic methods.

2.4.1 Serial SGS

The serial SGS is a greedy algorithm, which incrementally constructs a solution σ in a single pass, by considering a single activity at a time and adding it to the current solution. An outline of the SGS is shown in Algorithm 2.1. The algorithm works by considering the set $D$ of all the eligible tasks (line 1), defined as

\[ D = \{ t_i : \sigma(t_i) \text{ is undefined} \land \forall t_j < t_i \in C[\sigma(t_j) \text{ is defined}] \}, \]

(2.5)

the set of tasks for which all predecessors have been scheduled. Using a priority heuristic, the eligible tasks are ranked, and the task with the highest value is chosen for execution (line 2). The algorithm assigns the earliest time point to this task at which all predecessors have completed execution (line 9) and all resources required to execute this task are available (line 12, where $S_\tau = \{ t_i : \sigma(t_i) \leq \tau < \sigma(t_i) + l_i \}$ is the set of tasks scheduled to be executing at time $\tau$).

This procedure is repeated, until there are no more eligible tasks in $D$. At this point, all tasks are assigned a start time and the solution is complete.

The serial SGS can be implemented to work in $O(n^2m)$ time, where $n$ is the number of tasks and $m$ the number of resources: we do not need to consider all time points, it is enough to merely consider those points at which a task starts or ends.

2.4.2 Parallel SGS

The parallel SGS also constructs a schedule $\sigma$ greedily, but instead of considering a single task at a time, it considers as many tasks as possible at a time point, before advancing it. The set of eligible tasks for the parallel SGS is defined as

\[ D_\tau = \{ t_i : \sigma(t_i) \text{ undefined} \land \forall t_j < t_i \in C[\sigma(t_j) \text{ is defined} \land \sigma(t_j) + l_j < \tau] \}, \]

(2.6)

and contains all tasks for which the predecessors have been scheduled, and finished, at time point $\tau$.

The outline is shown in Algorithm 2.2. The parallel nature is evident in line 4: every task in the eligible set is checked for feasibility, in the order specified by the heuristic, and all tasks that are feasible at time $\tau$ are added (line 15). Again, since we only need to consider the starting and ending time points of the tasks, the algorithm has a time complexity of $O(n^2m)$.

2.4.3 Priority rules

The performance of both SGS approaches depends highly on the priority rule used to pick the task to schedule next. Many different rules have been proposed in literature; a few of the more important ones are discussed below.
Algorithm 2.1: Serial schedule generation

\begin{algorithm}
\begin{algorithmic}[1]
\While{$D \neq \emptyset$}
\State $t_i = h(D)$
\State $\tau = -1$
\Repeat
\State $\tau \leftarrow \tau + 1$
\State prec-feasible $\leftarrow$ TRUE
\State res-feasible $\leftarrow$ FALSE
\ForAll{$t_j : t_j < t_i$}
\If{$\sigma(t_j) + l_j \geq \tau$} prec-feasible $\leftarrow$ FALSE \EndIf
\EndFor
\For{$\tau \leq \tau' \leq \tau + l_i$}
\If{$\exists r_k[(\sum_{t_j \in S_{\tau'}} \text{req}(t_j, r_k)) + \text{req}(t_i, r_k) > \text{cap}(r_k)]$} res-feasible $\leftarrow$ FALSE \EndIf
\EndFor
\Until{prec-feasible = TRUE $\land$ res-feasible = TRUE}
\State $\sigma(t_i) \leftarrow \tau$
\EndWhile
\end{algorithmic}
\end{algorithm}

Algorithm 2.2: Parallel schedule generation

\begin{algorithm}
\begin{algorithmic}[1]
\State $\tau = 0$
\While{$\exists t_i[\sigma(t_i) \text{ is undefined}]$}
\State $D'_\tau \leftarrow D_\tau$
\While{$D'_\tau \neq \emptyset$}
\State $t_i = h(D'_\tau)$
\State $D'_\tau \leftarrow D'_\tau \setminus \{t_i\}$
\State prec-feasible $\leftarrow$ TRUE
\State res-feasible $\leftarrow$ FALSE
\ForAll{$t_j : t_j < t_i$}
\If{$\sigma(t_j) + l_j \geq \tau$} prec-feasible $\leftarrow$ FALSE \EndIf
\EndFor
\For{$\tau \leq \tau' \leq \tau + l_i$}
\If{$\exists r_k[(\sum_{t_j \in S_{\tau'}} \text{req}(t_j, r_k)) + \text{req}(t_i, r_k) > \text{cap}(r_k)]$} res-feasible $\leftarrow$ FALSE \EndIf
\EndFor
\If{prec-feasible $\land$ res-feasible} $\sigma(t_i) \leftarrow \tau$ \EndIf
\EndWhile
\State $\tau \leftarrow \tau + 1$
\EndWhile
\end{algorithmic}
\end{algorithm}
random  Tasks are selected at random, each eligible task has an equal chance of being picked. This strategy is often used for comparison purposes, to see how much is gained by using a certain heuristic.

latest start time  Alvarez-Valdes and Tamarit (1989) propose to calculate \( \text{lst}(t) \) for each task, which is the latest starting time for \( t \), given the temporal constraints, such that \( t \) and all its successors can still finish in time, assuming unlimited resource capacity. If the heuristic needs to select a task to schedule, it selects task \( t \) such that \( \text{lst}(t) \) is minimized. The rationale is that this task needs to be executed most urgently, to be able to finish in time.

most total successors  In the same paper, a different rule is proposed as well, which selects the task with the largest number of total successors to be executed first. A slightly different intuition of urgency holds for this task: since this task has many successors, many tasks need this task to be finished to become eligible.

Randomization is often employed in combination with the priority rules described above. Tie breaking is one reason: often, multiple tasks may have the same priority, and a choice needs to be made. More importantly, randomization may be employed to improve performance significantly for certain pathological cases, in which the rule persistently chooses exactly the wrong task.

It is clear from the complexity of the basic algorithms that both schemes scale well, compared to the exact solution methods discussed earlier. It is expected that problem sizes common for the NedTrain case are easily solvable with these methods. This seems to solve the questions raised by the first problem statement at the end of the first chapter. One of the sgs schemes can be used to quickly generate a set of solutions to a scheduling problem, reducing the labor-intensive character of the scheduling process. The solution itself however is still in the form of fixed starting times assigned to each task. In light of the second problem statement, this is undesirable: such a fixed schedule would be invalidated quickly in the highly dynamic environment of a maintenance workshop. We will therefore discuss a method yielding a more flexible solution form next.

2.5 Precedence constraint posting

As discussed above, it is desirable to have a method to solve the RCPSP that is based on heuristics, and that results in a solution in a flexible form. One such family of methods is based on precedence constraint posting (PCP). These methods can be seen as a transformation of the RCPSP into a much simpler problem, from which in turn we can efficiently derive multiple fixed-time solutions. This is exactly in line with the idea of the second problem statement: it is a flexible way to prescribe task execution. Within the bounds of the simplified problem, it is easy to generate different fixed-time schedules, which can be used to adapt to changing circumstances, without completely generating a new solution.
The basic idea underlying PCP is to make a distinction between resource constraints and temporal constraints. As it turns out, a problem having only (non-disjunctive) temporal constraints is relatively easy to solve: finding a fixed-time schedule for such a problem can be done in polynomial time (Dechter et al., 1991; Planken, 2013). The method of constraint posting therefore tries to add a set of precedence constraints that replace the resource constraints: if these precedence constraints are satisfied, the resource constraints will be satisfied as well. The problem is thus transformed from one with resource and temporal constraints to a problem with only temporal constraints. From the latter, we can easily extract multiple fixed-time schedules, as will be discussed later. But, the transformation is not symmetrical, owing to the NP-complete nature of the original problem. While any solution to the transformed problem is also a solution to the original problem, the reverse does not hold.

2.5.1 Finding resource conflicts

The first step in the algorithm is finding resource conflicts. We will start with a high-level discussion of profile-based approaches (Cesta et al., 1998, 2000). These methods focus on the resource profile of the problem: the resource use over time by the tasks in the problem is computed, and based on the capacity of each of the resources peaks are identified that violate the resource constraints. Precedence constraints are then added to the problem based on these peaks, in an attempt to flatten them such that the resource capacity is no longer exceeded.

Profile-based methods

For a fixed time solution, computing the resource profile is conceptually very simple: we know the exact start time for each task, and its length, so we can determine which tasks are executing at each point in time. Combining this information with the resource usage per task gives the total resource usage at a certain time point. But in this case we do not have a fixed time schedule available, as the whole point is to use a more flexible representation.

To still be able to construct a resource profile we make a distinction between two tasks necessarily or possibly overlap. In the first case, the two tasks always execute at the same time, for every possible fixed time schedule we can derive. In the second case, there is at least one fixed time schedule where the two tasks overlap. Based on necessarily and possibly overlapping tasks, we can now compute a lower and upper bound for the resource profile at each time point. The lower bound is based on all the tasks that necessarily execute at a time point—the resource usage can never be lower than this. Correspondingly, the upper bound is computed based on all the tasks that possibly overlap (Cesta et al., 1998).

Using the resource profile, we can identify peaks: time points at which a certain set of tasks together consume more resources than available. It is at these points that we need to add precedence constraints between tasks: by doing this, we spread the execution of the tasks, lowering the contention for a resource. A peak is said to be unresolvable if the lower bound resource profile of a resource is higher than the available capacity of that resource. In this case, there are no fixed
time solutions we can generate that satisfy the resource capacity constraints at this point in time. This means that no solution can be found, and we can terminate the search. Note that a solution might still exist: we are using a heuristic to arrive at a solution, and the heuristic does not always make the right choices.

If the lower bound is equal to, or lower than the resource capacity, we know that we can derive at least one feasible fixed-time solution. If the upper bound exceeds the resource capacity, some solutions will still exceed the resource capacity. In the ideal situation, the upper bound resource profile never exceeds the allowed capacity: this means that all solutions we can generate are resource-feasible.

Cesta et al. (1998) uses a pairwise method to compute the resource profiles, which is very quick but over-estimates the resource consumption; this results in an over-constrained solution.

Example 2.3 Consider three tasks \( t_a, t_b \) and \( t_c \), all making use of a shared resource with capacity two. If \( t_a \) and \( t_b \) can execute concurrently, and \( t_a \) and \( t_c \) as well, a pair-wise method might conclude that the upper bound on the resource use equals three, leading to the need for a temporal constraint. This might not be needed, since there might be a constraint such that \( t_b < t_c \).

**Minimal critical sets**

To overcome this over-estimation, a different strategy, based on minimal critical sets can be used. The idea behind this method is that we want to find a critical set, that is, a set of tasks (executing at a certain time point) that together exceeds the resource capacity. If, in addition, this set is minimal, it means that by removing just one task from this set, this set is no longer critical, meaning that the resource conflict caused by these tasks is solved. Removing a task is done in this case by adding a precedence constraint such that this task no longer executes at the chosen time point. If no (minimal) critical sets are present, all the fixed time schedules we can generate are guaranteed to be resource-feasible.

Finding minimal critical sets however is a difficult problem, as the number of minimal critical sets is in general exponential in the size of the problem. Nonetheless, several good approximation algorithms have been proposed (Lombardi and Milano, 2009b,a; Laborie and Ghallab, 1995).

### 2.5.2 Solving resource conflicts

Above, methods have been described allowing us to find the tasks involved in a resource conflict. The next step in the algorithm, from which its name is derived, is solving that conflict by adding (posting) a precedence constraint—hence the name, precedence constraint posting. In some of the possible variants we first need to select two tasks between which we are going to post the constraint. For example, in a profile-based method we select a contention peak containing multiple (more than two) tasks. To flatten this peak, two tasks need to be selected. Cheng and Smith (1996) proposes to look at the available flexibility, and to select the pair of tasks for which the flexibility is lowest: the intuition is that we have to
make a choice when it is still possible—if choices are made at other points in the schedule, the flexibility for this choice might decrease even further, limiting our options.

If a pair of tasks has been selected, the next decision is the direction of the precedence constraint to add. In some cases, our decision is forced: one of the directions might be temporally infeasible. In other cases, both options are still open. In this case, a similar reasoning is used as above (Cheng and Smith, 1996; Cesta et al., 1998): in order to preserve as many future choices as possible, we select the ordering that removes as little flexibility from the schedule as possible.

2.5.3 Esta and chaining

An interesting variation on the profile-based method is proposed by Cesta et al. (1998): instead of computing resource profile bounds to represent all possible fixed-time schedules we can generate, the resource profile for a single fixed-time assignment is computed, namely that of the earliest start-time assignment (hence the name, esta). The advantage is that the resource profile computation is very quick and straightforward, and as such we quickly arrive at a feasible solution. The downside is that only a single fixed-time schedule is guaranteed to be resource-feasible.

Cesta et al. (1998) proposes, and Policella et al. (2007) extends chaining, a method to convert such a schedule into a flexible one, in which we can again generate multiple fixed-time schedules from a single solution. The basic idea of chaining is that a single fixed-time solution is taken, and all precedence constraints are removed, and different precedence constraints are added. The key is in adding the right precedence constraints: chaining looks at the resource flow between the tasks and adds precedence constraints that encode the transfer of a resource between one task and the next. For each resource, a chain of tasks is constructed in this way, ensuring that the capacity of the resource can never be exceeded.

2.6 Simple temporal networks

The previous section introduced precedence constraint posting as a method to transform the complex RCSPSP into a simpler problem without resource constraints, representing a subset of the solution space, from which several fixed-time solutions can be derived efficiently. This section will discuss the formalism most commonly used, the Simple Temporal Network (STN). Note that many authors also refer to the Simple Temporal Problem (STP)—however, as Planken (2013) points out there is no single Simple Temporal Problem: instead, several different temporal queries can be formulated.

Recall that the idea of PCP is to remove all resource contention in an RCSPSP instance, resulting in an STN, which is a purely temporal representation, in which resources do not play a role. Such an STN can be seen as a flexible solution to the original RCSPSP instance: from it, various solution assignments $\sigma$ can be retrieved...
very efficiently. This intermediate step avoids re-solving the RCPSP instance every time an incident occurs during execution of the computed solution.

### 2.6.1 The STN formalism

The Simple Temporal Network (STN) was introduced by Dechter et al. (1991), and has become a commonly used temporal problem representation, with the advantage that solutions can be obtained efficiently. It consists of a pair $S = (\mathcal{X}, \mathcal{C})$, where $\mathcal{X} = X_0, X_1, \ldots, X_n$ is a set of time point variables, and $\mathcal{C}$ is a set of constraints. Each constraint $C_{ij}$ may only contain a single interval, of the form $X_j - X_i \leq c_{ij}$. Like in the RCPSP, the goal is to find a schedule for the events in $\mathcal{X}$, that is, a function $\sigma : \mathcal{X} \rightarrow \mathbb{R}$ that assigns a non-negative value to every time point in $\mathcal{X}$, such that all constraints in $\mathcal{C}$ are satisfied. Unlike the RCPSP, resource usage is not considered, i.e., there are no concurrency constraints other than those specified in $\mathcal{C}$. If a schedule does not exist, the STN is said to be inconsistent, and consistent otherwise. Note that there might be many different schedules satisfying the STN, as such an STN can be seen to encode a family of schedules. This is a desirable property in the case of NedTrain, since this can be used to adapt to changing circumstances during execution. In order to be able to express absolute time constraints, the time-point $X_0$, also denoted by $z$, is used. It represents a fixed reference point on the timeline, usually having the value 0.

#### Example 2.4

As an example, consider the three tasks introduced in Example 2.1. The start time of each task is modeled using time point variables $X_1, X_2, X_3$ for $t_1, t_2, t_3$. The release and due times are modeled using constraints relative to $X_0$. For the release time, we add $X_0 - X_i \leq 0$ for $i \in \{1, 2, 3\}$. For the due time, the task lengths have to be considered as well. Recall that we have $l_1 = 60$, $l_2 = 45$ and $l_3 = 30$, and due times of $d_i = 240$ for $i \in \{1, 2, 3\}$. This yields the constraints $X_1 - X_0 \leq 180$, $X_2 - X_0 \leq 195$ and $X_3 - X_0 \leq 210$.

Next, we encode the precedence constraints from the problem. In Example 2.1, the constraint $t_2 < t_3$ is formulated explicitly. Additionally, in Example 2.2, we have that “the other two tasks can then be started after $t_2$ has finished”, so this implies $t_2 < t_1$ as well. The first constraint is encoded as $X_3 - X_2 \leq 45$, and the second constraint as $X_1 - X_2 \leq 45$, again taking into account the task length $l_2 = 45$.

#### 2.6.2 Graph representation

A STN can be represented as a directed weighted constraint graph, this is the origin of the “network” in STN. Each node in this graph represents a time point variable, and an edge $X_i \rightarrow X_j$ with weight $c_{ij}$ represents a constraint $X_j - X_i \leq c_{ij}$. For simplicity, the edges $X_i \rightarrow X_j$ and $X_j \rightarrow X_i$ are often combined into a single edge, labeled with an interval, to represent the constraint $X_j - X_i \in [-c_{ji}, c_{ij}]$. 
2.6. Simple temporal networks

Example 2.5  Continuing the previous example, if the stn instance described is transformed into its equivalent graph representation, we obtain the following graph:

\[ \begin{align*}
X_3 & \\
X_2 & \quad (\infty, 45) \\
X_1 & \quad (\infty, 45) \\
X_0 & \quad [0, 210]
\end{align*} \]

As there is no constraint limiting \( X_2 - X_3 \) and \( X_2 - X_1 \), these weights have been set to \( -\infty \).

The graph representation is very helpful in finding constraints implied by the set \( \mathcal{C} \) of constraints, interpreting the weights \( c_{ij} \) on the arcs as lengths of the path from \( X_i \) to \( X_j \). If \( \mathcal{C} \) contains the constraints \( X_j - X_i \leq c_{ij} \) and \( X_k - X_j \leq c_{jk} \), then there exists a path from \( X_i \) via \( X_j \) to \( X_k \) having a length \( c_{ij} + c_{jk} \). This means \( X_k - X_i \leq c_{ij} + c_{jk} \) is an implied constraint, even though \( \mathcal{C} \) may not have contained a constraint limiting \( X_k - X_i \).

Example 2.6  We return to the example described above, to demonstrate implied constraints. The distance \( X_2 - X_3 \) is not limited by any constraint, however, we have

\[ X_2 - X_0 \leq 195 \land X_0 - X_3 \leq 0, \]

which implies

\[ (X_2 - X_0) + (X_0 - X_3) \leq 195 \iff X_2 - X_3 \leq 195. \]

In similar fashion, other constraints can be derived for the example. This procedure is also called constraint tightening. If all constraints are tightened as much as possible, we obtain the so-called minimal stn:
2.6.3 Algorithms

Exploiting the parallel between path lengths and constraints, an obvious procedure to find the tightest constraints, and thus the minimal stN, is to use an algorithm that computes the shortest paths in a graph, such as Floyd-Warshall (Floyd (1962), see Algorithm 2.3). This algorithm runs in $\Theta(n^3)$, and the tightest constraints it finds are represented as the elements of the $(n+1) \times (n+1)$ distance matrix $D_S$, containing for every pair of time-point variables $X_i$ and $X_j$ the length of the shortest path in the distance graph between $X_i$ and $X_j$. Using this minimal network, a solution can be extracted using any time point variable as starting point.

Other algorithms exist, which are more efficient (but more complex). One such algorithm is Snowball (Planken et al., 2011), which runs in $O(n^2w_d)$, where $w_d$ is the graph width induced by a vertex ordering $d$.

**Theorem 2.1** (Dechter et al., 1991) An stN is consistent if, and only if, its associated graph representation does not contain any negative cycles.

*Proof.* Assume the stN contains a negative cycle $C = X_1, \ldots, X_k = X_1$. The sum of the inequalities represented by the edges in $C$ yields $X_1 - X_1 < 0$, which can never be satisfied.

If there is no negative cycle, the shortest path between any pair of nodes is well-defined. For any pair of nodes $X_i$ and $X_j$, the shortest paths satisfy the equation

$$d_{0j} \leq d_{0i} + c_{ij},$$

(2.7)
Algorithm 2.3: Floyd-Warshall’s all-pairs shortest-path algorithm

Input: An STP $S = (X, C)$
Output: A distance matrix $D_S$

1. Initialize all entries in $(n + 1) \times (n + 1)$ matrix $D_S$ to $\infty$
2. for $i := 0$ to $n$ do
3. \quad $D_S[i, i] = 0$
4. end
5. foreach $X_j - X_i \leq c_{ij} \in C$ do
6. \quad $D_S[i, j] = c_{ij}$
7. end
8. for $k := 0$ to $n$ do
9. \quad for $i := 0$ to $n$ do
10. \quad \quad for $j := 0$ to $n$ do
11. \quad \quad \quad $D_S[i, j] := \min(D_S[i, j], D_S[i, k] + D_S[k, j])$
12. \quad \quad end
13. \quad end
14. end

which can be rewritten as

$$d_{0j} - d_{0i} \leq c_{ij}. \quad (2.8)$$

From this, it can be seen that assigning $X_j = d_{0j}$ and $X_i = d_{0i}$ satisfies the constraint $X_j - X_i \leq c_{ij}$. Note that these are the entries $D_S[0, i]$ and $D_S[0, j]$ in the distance matrix $D_S$. Therefore, the tuple

$$X = (d_{01}, \ldots, d_{0n}) \quad (2.9)$$

is a solution to the entire STN. Since $X_i - X_0 \leq d_{0j}$, this tuple contains the latest starting times $\text{lst}(X_i)$ for all time point variables $X_i \in X$ (assuming $X_0 = 0$). Similarly, it can be shown that the tuple

$$X = (-d_{10}, \ldots, -d_{n0}) \quad (2.10)$$

is a solution containing the earliest starting times $\text{est}(X_i)$ for all time point variables.

If we are only interested in the paths from a single source, for example to check the consistency of an STN instance, a more efficient procedure is to use the Bellman-Ford algorithm (see Algorithm 2.4), which computes the shortest paths to all nodes from a single source. This algorithm runs in $\Theta(nm)$ time (with $n = |X|$ and $m = |C|$) and can be used to obtain an earliest start time solution by first running this algorithm, and then using Equation 2.9. In the loop starting at line 5, the distance matrix is updated at most $|X|$ times, since a path can never be
Algorithm 2.4: Bellman-Ford's single-source shortest-path algorithm

**Input:** An STN $S = (X, C)$
**Output:** A distance list $D$, relative to $X_0$, or inconsistent

1. for $i := 0$ to $n$ do
2. \hspace{1em} $D[i] := \infty$
3. end
4. $D[X_0] := 0$
5. repeat $|X|$ times
6. \hspace{1em} foreach $X_j - X_i \leq c_{ij} \in C$ do
7. \hspace{2em} $D[X_j] = \min(D[X_j], D[X_i] + c_{ij})$
8. \hspace{1em} end
9. end
10. foreach $X_j - X_i \leq c_{ij} \in C$ do
11. \hspace{1em} if $D[X_j] > D[X_i] + c_{ij}$ then return inconsistent
12. end
13. return $D$

longer than the total number of nodes in the graph. In line 11, an inconsistency can be detected. If, after $|X|$ updates, a shorter distance is still possible, this can only be caused by a negative cycle.

The following theorem plays an important role in the dispatching of an STN: it describes how we can construct one of the many fixed-time schedules represented by the STN, by picking a start time for one time point variable satisfying the shortest path constraints, and then extending this partial solution variable by variable, until all time point variables have been assigned a value.

**Theorem 2.2** (Dechter et al., 1991) Let $S = (X, C)$ be a consistent STN. Any instantiation of a subset $X_k \subset X$ of $k$ variables ($1 \leq k < n$) that satisfies all the shortest path constraints applicable to $X_k$ is extensible to any other variable.

**Proof.** Our proof will be by induction on $|X_k| = k$.

For $k = 1$, $X_1$ consists of a single variable $X_i$, instantiated to $v_i$. We need to show that we can find an assignment $X_j = v_j$ for any other variable $X_j$ that satisfies the shortest path constraint between $X_i$ and $X_j$. The value $v_j$ must satisfy

$$-d_{ji} \leq v_j - v_i \leq d_{ij}. \tag{2.11}$$

We know that $d_{ji} + d_{ij} \geq 0$, since a consistent STN does not have any negative cycles, hence we can conclude that there exists a value $v_j$ that satisfies Equation 2.11.

Now, assume the theorem holds for $X_k$. We need to show that it also holds for $X_{k+1}$. Without loss of generality, let $X_k = \{X_1, \ldots, X_k\}$, and let $\{X_i = v_i : 1 \leq i \leq k\}$ be an assignment that satisfies the shortest path constraints among the variables in $X_k$. Let $X_{k+1} \notin X_k$. Now we need to find a value $X_{k+1} = v_{k+1}$
that satisfies the shortest path constraints between \( X_{k+1} \) and all variables in \( X_k \):

\[
-d_{k+1,i} \leq v_{k+1} - v_i \leq d_{i,k+1}
\]

for \( i = 1, \ldots, k \), or,

\[
 v_{k+1} \leq \min \{ v_i + d_{i,k+1} : 1 \leq i \leq k \},
\]

\[
 v_{k+1} \geq \max \{ v_i - d_{k+1,i} : 1 \leq i \leq k \}.
\]

Now, suppose that the minimum over the set above is attained at \( i_0 \), and the maximum is attained at \( j_0 \). Then, we know that \( v_{k+1} \) must satisfy

\[
v_{j_0} - d_{k+1,i_0} \leq v_{k+1} \leq v_{i_0} + d_{i_0,k+1}.
\]

Since, again, we know that \( v_{i_0} \) and \( v_{j_0} \) satisfy the constraint between \( X_{i_0} \) and \( X_{j_0} \), we have

\[
v_{j_0} - v_{i_0} \leq d_{i_0,j_0}.
\]

Since \( d \) represents a shortest path constraint, we also have

\[
d_{i_0,j_0} \leq d_{i_0,k+1} + d_{k+1,j_0}.
\]

Combining Equations 2.16 and 2.17, we get

\[
v_{j_0} - d_{k+1,i_0} \leq v_{i_0} + d_{i_0,k+1}.
\]

Therefore, we can conclude that there exists at least one value \( v_{k+1} \) that satisfies Equation 2.15.

\[
\square
\]

The following corollary can be directly derived from this theorem, and contains two important properties that hold for STNs and their schedules:

**Corollary 2.1** Let \( S = (X, C) \) be an STN and \( D_S \) its distance matrix. For \( i = 1, \ldots, n \), let \( \text{lst}(X_i) = D_S[0, i] \) and \( \text{est}(X_i) = -D_S[i, 0] \). Then, for every schedule \( \sigma \) for \( S \), and every \( X \in X \), it holds that \( \sigma(X) \in [\text{est}(X), \text{lst}(X)] \). Moreover, given any \( X \in X \) and \( v \in [\text{est}(X), \text{lst}(X)] \), there exists a schedule \( \sigma \) for \( S \) such that \( \sigma(X) = v \).

This shows the flexible nature of an STN: regardless of which value \( v \in [\text{est}(X), \text{lst}(X)] \) we pick for \( X \), we are guaranteed to be able to find a schedule in which \( X \) starts at the selected time. So, in this sense, the STN formalism fits well with the second problem statement: we can quickly derive fixed-time solutions from an STN, making it a flexible method to prescribe task execution. Combined with precedence constraint posting, we are able to flexibly prescribe the execution of an RCPSF instance.
2.7 Multi-agent scheduling

Proposition 2.1 shows that multiple fixed-time schedules can be constructed for a single STN, but the process by which these schedules are constructed is centralized. This is the source of the third problem statement: often, and especially in the case of NedTrain, a schedule has to be executed by multiple teams. This does not combine well with the need for central control over the schedule execution: we would rather have a form of decentralized execution.

To map the model of the STN to the idea of decentralized execution, we can distribute the tasks in \( \mathcal{X} \) over a set of \( m \) agents \( A_1, \ldots, A_m \). Each agent \( A_i \) is responsible for the execution of a subset \( \mathcal{X}_i \subseteq \mathcal{X} \) of events, where these subsets are mutually disjoint, i.e., \( \mathcal{X}_i \cap \mathcal{X}_j = \emptyset \) when \( i \neq j \). If the schedule we want to execute is fixed, i.e., a function \( \sigma : \mathcal{X} \rightarrow \mathbb{R} \), this poses no particular problem. But in the case of NedTrain, it is desirable for the teams executing their schedule to have some form of flexibility: due to the uncertainty present in the tasks it is desirable if a team has at least partial control over the exact start times of the tasks it needs to execute. This is a more difficult problem, because here, we want agent \( A_i \) to determine \( \sigma_i : \mathcal{X}_i \rightarrow \mathbb{R} \) for the STN \(^3\) \( S_i = (\mathcal{X}_i \cup \{ z \}, \mathcal{C}_i) \), derived from the original STN \( S = (\mathcal{X}, \mathcal{C}) \) solving the RCPSP instance, where \( \mathcal{X}_i \) is the subset of time point variables assigned to \( A_i \), and \( \mathcal{C}_i = \mathcal{X}_i \times \mathcal{C} \) is the set of constraints from \( \mathcal{C} \) restricted to the time point variables in \( \mathcal{X}_i \). This problem is known as the Temporal Decoupling Problem (Hunsberger, 2002).

**Definition 2.3** Let \( S = (\mathcal{X}, \mathcal{C}) \) be a consistent STN. Suppose that \( \mathcal{X} - \{ z \} = \{ \mathcal{X}_i \}_{i=1}^m \) is partitioned in \( m \) subsets \( \mathcal{X}_i \). Then the **temporal decoupling problem** is to find \( m \) STNs \( S_i = (\mathcal{X}_i \cup \{ z \}, \mathcal{C}_i) \) such that, whenever \( \sigma_1, \ldots, \sigma_m \) are independently constructed schedules for the individual STNs \( S_1, \ldots, S_m \), respectively, their merge \( \sigma = \bigcup_{i=1}^m \sigma_i \) is a schedule for the original STN \( S \).

The problem is that, while we want the agents to be able to independently pick \( \sigma_i \), we also want the merge \( \sigma = \bigcup_{i=1}^m \sigma_i \) of the individual schedules to be a valid total schedule for the original \( S \). If we take an inter-agent constraint, i.e., a constraint \( X_j - X_i \leq c_{ij} \) for which \( X_i \in \mathcal{X}_a \) and \( X_j \in \mathcal{X}_b \) with \( a \neq b \), this constraint is considered by neither \( A_a \) nor \( A_b \) in determining \( \sigma_a \) and \( \sigma_b \), it might very well be that \( \sigma_a(X_i) - \sigma_b(X_j) > c_{ij} \). Hunsberger (2002) proposes an iterative procedure, using the distance matrix \( D_S \) of the original problem \( S \). In this procedure, each inter-agent constraint \( X_j - X_i \leq D_S[i, j] \) is made obsolete by tightening the intra-agent constraints \( X_j - z \leq D_S[i, j] \) and \( z - X_i \leq D_S[i, 0] \). These tightenings consist in selecting values \( \delta_{i,1} \) and \( \delta_{i,2} \) such that

- \( X_j - z \leq \delta_{i,1} \leq D_S[0, j] \),
- \( z - X_i \leq \delta_{i,2} \leq D_S[i, 0] \), and

\(^3\)Note that we consider the execution of a flexible solution here. Hence, it suffices to only consider the STN decoupling problem.
• \( \delta_{i,1} + \delta_{i,2} \leq D_S[i, j] \).

If we add the new constraints \( X_j - z \leq \delta_{i,1} \leq D_S[0, j] \) and \( z - X_i \leq \delta_{i,2} \leq D_S[i, 0] \) to \( S \) while making sure that the resulting system remains consistent, the effect of adding these constraints is that an inter-agent constraint \( X_j - X_i = (X_j - z) + (z - X_i) \leq \delta_{i,1} + \delta_{i,2} \leq D_S[i, j] \) is now implied by the new, tighter, intra-agent constraints. Hence, the inter-agent constraint can be removed from the updated STN. Applying this procedure for every non-implied inter-agent constraint ensures that all inter-agent constraints are implied by intra-agent constraints and the resulting system is a temporal decoupling.

2.8 Conclusions and research questions

In this chapter, the exact scheduling problem as faced by NedTrain has been formalized, and a few of the many approaches proposed during the years have been discussed. It is clear from the nature of the problem that exact solution methods are not feasible. This is reflected in the sizes of the problems solved by such methods: they are much smaller than the problems NedTrain are faced with. We are thus forced to turn to heuristic methods to solve this problem. Classical methods such as the serial and parallel schedule generation schemes are able to solve problems of the scale NedTrain requires, but they fail to offer the flexibility required during the execution of the schedule, since they give fixed execution times for each task.

Simple temporal networks offer an elegant way to represent this flexibility, while avoiding the complexity of the original RCPSP. Constraint posting methods form the bridge between these two worlds: they transform the RCPSP problem into an STN, which can then be used to execute the tasks in a flexible way. The STN offers a quick way to adapt the execution, since new solutions can be computed quickly.

An important consideration is the amount of flexibility present in such an STN. Intuition suggests that this flexibility plays an important role in the ability of a schedule to adapt to changing events, but how we can compare different STNs, and how we can quantify this ability, remains unclear. This gives rise to our first research question:

RQ1 What criteria encapsulate our intuitive notion of the concept of flexibility, as applied to solutions of scheduling problems, and can we devise a method to measure flexibility, conforming to these criteria?

Having defined what flexibility is, the next question that arises is how to actually make use of said flexibility. It is natural to assume that, given a certain solution, there are multiple ways of distributing the available flexibility. Given the goal of attaining a robust schedule, i.e., a schedule in which a disturbance in one task has low impact on the rest of the schedule, we can ask which of the possible distributions yields the highest robustness:
**RQ2** Given a schedule with a certain potential flexibility, in what ways can this flexibility be used best to achieve a schedule with high robustness?

So far, our discussion on flexibility has focused on the temporal aspect of schedules. If we include the sequential aspect of schedules, it might be possible to increase the flexibility, leading to higher levels of robustness:

**RQ3** In what way can the concept of temporal flexibility of a schedule be extended to include some sequential flexibility?

Another important aspect of the scheduling process at NedTrain is that maintenance is performed on multiple trains in parallel, by independent teams. This can be abstractly represented as a multi-agent scheduling problem. An important concern here is the independence of the teams—ensuring their independence can be mapped to decoupling in multi-agent systems. The natural concern here is keeping teams independent in the face of uncertain events:

**RQ4** How can our earlier work on ensuring robustness of entire schedules be applied to the decoupling of multi-agent schedules, in which we strive to decouple in such a way that disturbances in the schedule of one agent minimally affect the schedules of the other agents?
The first problem statement from Chapter 1 describes the need for a flexible automated process that solves the scheduling problem at NedTrain. In Chapter 2 it is shown that the rCPSp formalism matches well with the scheduling problem at NedTrain, and precedence constraint posting (Cesta et al., 1998; Lombardi and Milano, 2009b) is a method matching the requirement of a flexible process: it solves the rCPSp heuristically, meaning that it is quick enough to be used interactively, and the result has the form of a Simple Temporal Network (STN). From this STN, multiple fixed-time schedules can be derived very efficiently, as described in the previous chapter—this shows that the requirement of having a process resulting in a flexible schedule is satisfied.

It is intuitively clear that such an STN can be called flexible, since it can be used to encode a range of possible schedules. But how flexible is a given STN exactly? And how can two STNs be compared with each other in terms of flexibility? These questions are summarized in the first research question, which will be answered in this chapter by exploring what criteria encapsulate this intuitive notion of flexibility, and by determining the flexibility of an STN, as representation of both intermediate and final schedules in the precedence constraint posting process. One strategy, which seems very straightforward, is to compare the range of possible schedules. But this only serves to replace our question by a different one, namely how exactly we can determine this range. Ideally, we would want to have a metric with which we can measure the flexibility of an STN. Such a metric can then be used to evaluate different STN solutions produced by precedence constraint posting algorithms, or even as a heuristic, to guide the process of precedence constraint posting.

There are several existing metrics attempting to do this, but we will show that they all overestimate the flexibility contained in a certain schedule, by not fully taking into account dependencies among tasks. A new method is presented here, which does take dependencies into account more completely, yielding a flexibility metric that better conforms to the intuitive properties such a metric should have.
3.1 The need for flexibility

The primary reason for wanting to have a flexible schedule is the limited preservation of a fixed-time schedule. Start times of tasks are determined off-line, before execution starts, which means that unforeseen disturbances during the execution can prevent certain tasks from starting at their predetermined time. Delaying the start of these tasks is not always the best option, since this will cause the delay to spread to other parts of the schedule, which can ultimately lead to a violation of the due time of certain tasks. A single disturbance could therefore be a reason to modify or completely recompute the schedule. However, there might not be enough time to compute a new schedule each time a disturbance occurs. Additionally, commitments might have been made already on the basis of the old schedule, which might have to be broken if the new schedule takes effect.

Some techniques that try to incorporate flexibility in their schedule already exist (see Policella et al., 2004, 2005). They ensure flexibility in the off-line phase of schedule construction, but a transformation is still needed to ensure efficient execution (Smith, 2003). To avoid such a time-critical transformation, we propose the construction of an interval schedule, an assignment of time intervals to tasks, such that we are free to choose a specific starting time out of the interval for a task, without running the risk of violating any of the imposed scheduling constraints. If this is possible, this interval corresponds to the intuitive concept of flexibility we have with respect to scheduling this task.

Using time intervals in lieu of fixed time assignments has been proposed before: if a scheduling formalism is decomposable, we can attain flexibility by assigning an interval of time points to a task (see Dechter et al., 1991; Dechter, 2003; Pollack and Tsamardinos, 2005; Policella et al., 2004). A scheduling problem is decomposable if, for any partial schedule satisfying the constraints, there exists a complete schedule extending that partial schedule, satisfying all constraints. Note that this property is useful in the case of working with interval schedules: If there is an interval \( I_t \) from which all values are valid starting times for a task \( t \), satisfying the constraints, we are sure that we can extend this partial schedule to a complete one. As we can start from any task \( t \), it has often been proposed that these intervals are representative of the flexibility of the system. While intuitively appealing, this idea is not sufficient to construct flexible schedules, since decomposability does not imply that these intervals are independent. Decomposability allows us to determine the flexibility of two tasks in isolation, but it does not allow us to determine the flexibility of the combination of these two tasks.

A different reason to investigate alternatives to fixed-time schedules is the presence of events in the problem that are not under control of the agent executing the schedule (see Vidal and Fargier, 1999; Pollack and Tsamardinos, 2005). For such events, we might only know the interval in which they will take place, not the exact time. Even having an exact schedule can only be a partial assignment of time points, in such a way that the schedule does not violate the imposed constraints, for any realization of the events not under control of the executing agents. This can be seen as flexibility required by an external party, and
in this case it also seems like a natural solution to use time intervals to represent the events not under our control.

3.2 Flexibility of an stn

To start our discussion on flexibility, we begin by examining what our intuitive notion of this concept might translate to, in the context of the stn. Intuitively, the flexibility of an stn $S = (\mathcal{X}, \mathcal{C})$ refers to the amount of freedom we have in assigning values to the time point variables in $\mathcal{X}$. Earlier, we defined a fixed schedule $\sigma$ for an stn as an assignment $\sigma : \mathcal{X} \rightarrow \mathbb{R}$ satisfying all the constraints in $\mathcal{C}$. Such a schedule, however, is just an arbitrary fixed assignment among a large set of possible alternative assignments also satisfying the constraints. Therefore a single fixed schedule does not offer any indication of the flexibility we have in assigning values to the variables of the stn it has been derived from. We are thus interested in other types of solutions, which do give an indication of the flexibility.

3.2.1 Using earliest and latest starting times

If we want flexibility to indicate our freedom of choice, then one option for defining the flexibility for scheduling event $X$ is to use the difference $\text{lst}(X) - \text{est}(X) = \text{flex}_N(X)$. The total flexibility of an stn $S$ then could be defined as

$$\text{flex}_N(S) = \sum_{X \in \mathcal{X}} \text{flex}_N(X).$$

(3.1)

Obviously, $\text{flex}_N(S)$ can be computed in $O(n^3)$ time. We also know that $\text{flex}_N(X)$ gives an exact measure of the flexibility of event $X$, since for every choice $v \in [\text{est}(X), \text{lst}(X)]$, there exists a schedule $\sigma$ for $S$ such that $\sigma(X) = v$ (see Proposition 2.1). However, although for every individual event, $\text{flex}_N(X)$ offers a precise measure of its flexibility, using the sum $\text{flex}_N(S)$ of these flexibilities has a serious disadvantage, due to dependencies that might exist between starting times of events. To see this, consider the following example:

Example 3.1 Consider the stns $S_1$ and $S_2$ shown on the next page, where $S_1$ specifies the concurrent execution of $k = 3$ events within 5 time units, while $S_2$ specifies the sequential execution of $k = 3$ consecutive events, i.e., $X_2$ should not occur before $X_1$, while $X_3$ should not occur before $X_2$ and all should occur within 5 time units.

Note that both in $S_1$ and in $S_2$, for each $i \in \{1, 2, 3\}$, $\text{est}(X_i) = 0$ and $\text{lst}(X_i) = 5$. Hence,

$$\text{flex}_N(S_1) = 3 \times 5 = 15 = \text{flex}_N(S_2).$$

(3.2)

So, according to the measure $\text{flex}_N$, $S_1$ and $S_2$ have the same flexibility, which, of course, is counterintuitive: For each event $X_i$ in $S_1$ we can determine its
starting time between 0 and 5 independently from the other events. So, the total flexibility, expressed as the amount of choice we have in selecting a starting time for an event equals \(5 \times 3 = 15\).

The choice for the starting time of an event \(X_i\) in \(S_2\), however, may influence the choice of the starting time of the next event \(X_{i+1}\). More precisely, if \(v_1\), \(v_2\) and \(v_3\) are the latest starting times for the events \(X_1\), \(X_2\) and \(X_3\), respectively, in \(S_2\), it is clear that

\[
0 \leq v_1 \leq v_2 \leq v_3 = 5. \tag{3.3}
\]

Hence, the flexibility of \(X_1\) is \(v_1 - 0\), the flexibility of \(X_2\) is \(v_2 - v_1\) and the flexibility of \(X_3\) is \(5 - v_2\). So, the total flexibility of \(S_2\) equals

\[
\text{flex}(S_2) = v_1 + (v_2 - v_1) + (5 - v_2) = 5. \tag{3.4}
\]

Therefore, the flexibility of \(S_2\) should come out as only \(\frac{1}{5}\) of the flexibility of \(S_1\). Generalizing \(S_1\) and \(S_2\) to systems containing \(k\) concurrent and sequential events, respectively, we see that the freedom to schedule in \(S_1\) equals \(\text{flex}(S_1) = 5 \times k\), while the flexibility of \(S_2\) is independent of \(k\) and remains \(\text{flex}(S_2) = 5\). Hence, if \(k\) goes to infinity, the ratio of the real flexibilities of \(S_1\) and \(S_2\) goes to infinity:

\[
\lim_{k \to \infty} \frac{\text{flex}(S_1)}{\text{flex}(S_2)} = \lim_{k \to \infty} \frac{5 \times k}{5} = \infty. \tag{3.5}
\]

while their ratio according to \(\text{flex}_N\) is exactly 1:

\[
\lim_{k \to \infty} \frac{\text{flex}_N(S_1)}{\text{flex}_N(S_2)} = \frac{5 \times k}{5 \times k} = 1. \tag{3.6}
\]

We conclude that the \(\text{flex}_N\) metric seriously overestimates the flexibility of STNS similar to \(S_2\).
Using upper and lower bounds for every pair of events
One reason \( \text{flex}_N \) fails is that it is not able to deal with dependencies between timed events. To overcome this disadvantage, we have to incorporate information about the dependencies between two events. This was proposed in the definition of flexibility by Hunsberger (2002) and others (Policella et al., 2005, 2007). Hunsberger defined his flexibility metric, which we will denote by \( \text{flex}_H(S) \), by taking into account not only the apparent flexibility \( \text{lst}(X) - \text{est}(X) \) of every event \( X \), but also the flexibility between pairs of events \( X_i \) and \( X_j \). Since the distance matrix \( D_S \) contains the entries \( D_S[i, j] \) and \( D_S[j, i] \) such that
\[
-D_S[j, i] \leq X_j - X_i \leq D_S[i, j]
\] (3.7)
are the strongest constraints implied by \( C \) w.r.t. the temporal difference \( X_j - X_i \), we can define the flexibility associated with this difference as the length of the corresponding interval:
\[
\text{flex}_H(X_i, X_j) = D_S[i, j] + D_S[j, i].
\] (3.8)
Now the flexibility of the system \( S \) can be defined by taking \( \text{flex}_N(S) \) and adding to it the sum of all these flexibilities for every distinct pair of time-points,\(^2\) i.e.,
\[
\text{flex}_H(S) = \text{flex}_N(S) + \sum_{i=1}^{n} \sum_{j>i}^{n} \text{flex}_H(X_i, X_j).
\] (3.9)

If there is a dependency between two events \( X_i \) and \( X_j \), this will result in a lower value of \( D_S[i, j] + D_S[j, i] \). Hence, dependencies among events will have an influence on \( \text{flex}_H \), allowing us to account for its impact on flexibility. It turns out this metric is indeed an improvement, compared to the naive metric \( \text{flex}_N \). But, the next example shows that \( \text{flex}_H \) also has serious shortcomings with respect to dependencies.

**Example 3.2** Consider again the stns \( S_1 \) and \( S_2 \), as described in the previous example. According to \( \text{flex}_H \), the flexibility of \( S_1 \) equals
\[
\text{flex}(S_1) = 3 \times 5 + 3 \times 10 = 45,
\] (3.10)
while the flexibility of \( S_2 \) equals
\[
\text{flex}(S_2) = 3 \times 5 + 3 \times 5 = 30.
\] (3.11)
So, according to the Hunsberger metric \( \text{flex}_H \), \( S_2 \) has a flexibility equal to \( \frac{2}{3} \) of the flexibility of \( S_1 \). This seems to be an improvement with respect to

\(^1\)Although Hunsberger defined the inverse of flexibility, i.e., rigidity of an stn, that is not relevant for our discussion at hand, since we assume a finite time horizon for an stn: this avoids the problem of infinitely sized intervals.

\(^2\)Actually, Hunsberger took the square root of the sum of the squares of these flexibility measures per pair, divided by \( (n \times (n - 1)/2) \), the total number of different pairs of time-points.
the previous flexibility metric $\text{flex}_N$ assigning the same flexibility to $S_1$ and $S_2$. However, generalizing $S_1$ and $S_2$ to systems containing $k$ concurrent and sequential events, respectively, we observe that

$$\text{flex}_H(S_1) = 5 \times k + \binom{k}{2} \times 10 = 5 \times k^2,$$  \hspace{1cm} (3.12)

while

$$\text{flex}_H(S_2) = 5 \times k + \binom{k}{2} \times 5 = 5 \times \frac{k^2 - k}{2}.$$  \hspace{1cm} (3.13)

This means that for larger values of $k$, the ratio between $\text{flex}_H(S_1)$ and $\text{flex}_H(S_2)$ converges to 2:

$$\lim_{k \to \infty} \frac{\text{flex}_H(S_1)}{\text{flex}_H(S_2)} = \lim_{k \to \infty} \frac{5 \times k^2}{5 \times \frac{k^2 - k}{2}} = 2.$$  \hspace{1cm} (3.14)

The ratio of the real flexibilities of $S_1$ and $S_2$ (see Equation 3.5 in the previous example) goes to infinity if $k$ goes to infinity. We conclude that this metric is also not able to capture the dependencies between the flexibilities of events in a satisfactory way.

### 3.2.2 Rationality postulates: towards a suitable flexibility metric

Having demonstrated the shortcomings of $\text{flex}_N$ and $\text{flex}_H$, we will now analyze these in more detail. We will do this by stating some simple rationality postulates (Alchourrón et al., 1985), characterizing these metrics. Using this characterization, an additional “independence” postulate is introduced, which represents the idea that the flexibility of one event should be independent of all other events. Having characterized the properties a metric should have to conform to the intuitive notion of flexibility, a suitable flexibility metric $\text{flex}$ is derived, and we show that it can be computed in an efficient way.

**Rationality postulates for $\text{flex}_N$**

We take $\text{flex}_N$ as main representative of the two metrics discussed above. The metric is defined as the sum of intervals, where each such interval $[a_X, b_X]$ indicates the flexibility of an event $X$. Flexibility is interpreted as the ability to pick a starting time for the event, which means that for every choice $v \in [a_X, b_X]$ as starting time for $X$, there is a schedule $\sigma$ for $S$ such that $\sigma(X) = v$. Obviously, there are many such intervals $[a_t, b_t]$ we can choose, including trivial choices having $a_X = b_X$. Since our aim is to determine the maximum available flexibility for a certain $\text{STN}$, we want the sum of these interval sizes to be maximal.

The postulates below capture the essence of a naive flexibility metric such as $\text{flex}_N$ for an $\text{STN} S = (\mathcal{X}, C)$. We go one step further, and claim that for all metrics $\text{flex}_n$ that satisfy these postulates, it holds that $\text{flex}_n = \text{flex}_N$, such that we can say that $\text{flex}_N$ is completely characterized by these postulates.
If we let \( S = (\mathcal{X}, C) \) be an stn, a naive flexibility metric \( \text{flex}_n \) is characterized by the following postulates:

**F1** The flexibility \( \text{flex}_n(X_0) \) of the temporal reference event \( X_0 \in \mathcal{X} \) is equal to zero.

*Comment* The temporal reference point should be fixed.

**F2** The flexibility \( \text{flex}_n(X) \) of an event \( X \in \mathcal{X} - \{X_0\} \) is a nonempty interval \([a, b]\) such that any \( v \in [a, b] \) can be chosen as a starting time \( \sigma(X) = v \) in some schedule \( \sigma \) for \( S \).

*Comment* It must be ensured that the intervals are not too large: all points must be valid start times.

**F3** Let \([a, b]\) denote the size \( b - a \) of the interval \([a, b]\). Then,

\[
\text{flex}_n(S) = \sum_{X \in \mathcal{X}} |\text{flex}_n(X)|
\]

(3.15)

is the maximal value that can be obtained for all functions \( \text{flex} : \mathcal{X} \to \{[m, n] : -\infty < m \leq n < \infty\} \)

(3.16)

satisfying postulates F1–F2.

*Comment* This postulate ensures that we take a solution yielding maximal interval sizes, since we want to determine the maximal available flexibility. Trivial solutions, like fixed time assignments, also satisfy the first two postulates, and these must be avoided if there are better choices.

Next, we will show that these postulates uniquely characterize \( \text{flex}_N \), meaning that for all possible metrics \( \text{flex}_n \) satisfying F1–F3, we have \( \text{flex}_n = \text{flex}_N \):

**Proposition 3.1** Let \( S = (\mathcal{X}, C) \) an stn, and let \( \text{flex}_n \) be a flexibility metric satisfying F1–F3. Then, \( \text{flex}_n = \text{flex}_N \).

*Proof.* It is clear that \( \text{flex}_N \) satisfies F1–F2. Let \( \text{flex}_n \) be any metric satisfying F1–F3. In order to satisfy F2, observe that for any \( X \in \mathcal{X} \), with \( \text{flex}_n(X) = [a, b] \), by Proposition 2.1 we must have

\[
est(X) \leq a \leq b \leq \text{lst}(X).
\]

(3.17)

But that implies, by F1 and F3,

\[
\text{flex}_n(S) = \sum_{X \in \mathcal{X}} \text{flex}_n(X) = \sum_{X \in \mathcal{X}} (\text{lst}(X) - \text{est}(X)) = \text{flex}_N(S).
\]

(3.18)
Rationality postulates for flex
As has been shown using the examples in the previous sections, the results given by flex$_N$ are counter-intuitive in the presence of dependencies between events, which results in an over-estimation of the flexibility. The underlying reason is the formulation of F2: If we have a single flexibility interval $[a_i, b_i]$ for $X_i \in \mathcal{X}$, we are indeed guaranteed to be able to find a schedule $\sigma$ for any $v_i \in [a_i, b_i]$, having $\sigma(X_i) = v_i$. This does however no longer hold if we want to make $k > 1$ simultaneous choices $v_1, \ldots, v_k$ in the intervals $[a_1, b_1], \ldots, [a_k, b_k]^3$.

A simple postulate F2$'$ can be added to the existing set of postulates F1–F3 to remedy this deficiency. We will prove that any flexibility metric satisfying this, and the previous set of postulates, satisfies the (missing) requirement that for any series $(v_1, \ldots, v_n)$ of choices $v_i \in [a_i, b_i] = \text{flex}(X_i)$ we are guaranteed to be able to find a schedule $\sigma$ for $S$ such that $\sigma(X_i) = v_i$.

The complete set of postulates for a suitable flexibility metric for an STN $S = (\mathcal{X}, \mathcal{C})$, conforming to our intuitive notions of the concept, is thus:

**F1** The flexibility $\text{flex}_n(X_0)$ of the temporal reference event $X_0 \in \mathcal{X}$ is equal to zero.

**F2** The flexibility $\text{flex}_n(X)$ of an event $X \in \mathcal{X} - \{X_0\}$ is a nonempty interval $[a, b]$ such that any $v \in [a, b]$ can be chosen as a starting time $\sigma(X) = v$ in some schedule $\sigma$ for $S$.

**F2$'$** For some $X_i, X_j \in \mathcal{X}$ with $\text{flex}(X_i) = [a_i, b_i]$ and $\text{flex}(X_j) = [a_j, b_j]$, there exists a schedule $\sigma$ for $S$ such that $\sigma(X_i) = v_i$ and $\sigma(X_j) = v_j$, for every $v_i \in [a_i, b_i]$ and every $v_j \in [a_j, b_j]$.

**Comment** This implies independence between the flexibility intervals for $X_i$ and $X_j$: a choice for $\sigma(X_i)$ has no impact on the available choices for $\sigma(X_j)$.

**F3** Let $|[a, b]|$ denote the size $b - a$ of the interval $[a, b]$. Then,

$$\text{flex}_n(S) = \sum_{X \in \mathcal{X}} |\text{flex}_n(X)|$$

(3.19)

is the maximal value that can be obtained for all functions

$$\text{flex} : \mathcal{X} \to \{[m, n] : -\infty < m \leq n < \infty\}$$

(3.20)

satisfying postulates F1–F2$'$.

The new postulate F2$'$ ensures independence for any pair of flexibility intervals. We will now show that this is sufficient to ensure independence of all flexibility intervals, in case there are more than two:

---

1We assume, without loss of generality, that $v_0 = 0$ is picked as reference time point.
Proposition 3.2 Let \( S = (\mathcal{X}, C) \) be an stn. For any metric \( \text{flex}' \) satisfying \( \text{F1} - \text{F2}' \) it holds that for every series \((v_0, \ldots, v_n)\) of choices, with \( v_i \in \text{flex}'(X_i) \) for \( i \in [1, n] \), there exists a schedule \( \sigma \) for \( S \) such that \( \sigma(X_i) = v_i \).

Proof. Let \( \text{flex}' \) be a metric satisfying \( \text{F1} - \text{F2}' \), and let \((v_0, \ldots, v_n)\) be a series of arbitrary choices, with \( v_i \in \text{flex}'(X_i) \) for \( i \in [1, n] \). Define a function \( \sigma: \mathcal{X} \to \mathbb{R} \) by \( \sigma(X_i) = v_i \forall X_i \in \mathcal{X} \). Suppose, on the contrary, that \( \sigma \) is not a valid schedule for \( S \). Then, there must exist a constraint \( X_j - X_i \leq c_{ij} \) such that

\[
\sigma(X_j) - \sigma(X_i) = v_j - v_i > c_{ij}.
\]

By Postulate \( \text{F2}' \) however, there exists a schedule \( \sigma' \) for \( S \) such that \( \sigma'(X_i) = v_i \) and \( \sigma'(X_j) = v_j \). Therefore,

\[
\sigma'(X_j) - \sigma'(X_i) = v_j - v_i \leq c_{ij},
\]

and we have derived a contradiction. Therefore, \( \sigma \) must be a schedule for \( S \). \( \square \)

This shows that the flexibility property, missing in \( \text{flex}_N \) and \( \text{flex}_l \), can be repaired easily, by adding a postulate ensuring independence of the flexibility intervals. Having characterized the essential properties of our new flexibility metric \( \text{flex} \), we will now show how to actually compute the flexibility \( \text{flex}(S) \) of an stn \( S \).

3.2.3 Computing the flexibility of an stn

As remarked already, we wish to depart from the idea of having fixed time assignments in a schedule \( \sigma \) for an stn \( S \). Instead, we would like to have an interval schedule for \( S \), having the property that we can freely choose a start time value \( v_i \) from within the interval for every event \( X_i \), such that the combination of all those values comprises a schedule \( \sigma' \) for \( S \). As a start, we will define this notion of an interval schedule formally:

Definition 3.1 Given an stn \( S = (\mathcal{X}, C) \), a set \( I_S = \{I_i = [l_i, u_i] : X_i \in \mathcal{X}\} \) of (non-empty) intervals for the time point variables \( X_i \in \mathcal{X} \) is an interval schedule for \( S \) iff for every \( X_i \in \mathcal{X} \) and every \( v_i \in [l_i, u_i] \), the assignment \( \sigma \) defined by \( \sigma(X_i) = v_i \) is a schedule for \( S \).

Any flexibility metric satisfying the postulates \( \text{F1} - \text{F2}' \) defines an interval schedule \( I_S = \{I_i : X_i \in \mathcal{X}\} \) for an stn \( S \) by defining the correspondence \( I_S = \{\text{flex}(X_i) : X_i \in \mathcal{X}\} \):

Proposition 3.3 Let \( S = (\mathcal{X}, C) \) be an stn. The function \( \text{flex}: \mathcal{X} \to \{[m, n] : -\infty < m \leq n < \infty\} \) satisfies the postulates \( \text{F1} - \text{F2}' \) iff the set of intervals \( I_S = \{\text{flex}(X_i) : X_i \in \mathcal{X}\} \) is an interval schedule for \( S \).
3. Measuring flexibility

Proof (if). Suppose \( I_S = \{ \text{flex}(X_i) : X_i \in \mathcal{X} \} \) is an interval schedule for \( S \). Then any series of choices \((v_0, \ldots, v_n)\) from the given intervals \( I_i \) in the interval schedule implies that the assignment \( \sigma(X_i) = v_i \) is a schedule. In particular, this means that \( \sigma(X_0) = 0 \). Hence, we have \( I_0 = [0, 0] \), satisfying \( F1 \). For all \( i \geq 1 \), this property also immediately implies that \( F2 \) and \( F2' \) are satisfied.

(only-if). Suppose \( \text{flex} : \mathcal{X} \to \{ [m, n] : -\infty < m \leq n < \infty \} \) satisfies the postulates \( F1-F2' \). By Proposition 2.1 and Definition 3.1, it follows that \( I_S = \{ \text{flex}(X_i) : X_i \in \mathcal{X} \} \) is an interval schedule for \( S \).

From this proposition it follows that if we want to compute a flexibility metric for an \( \text{stn} \) \( S = (\mathcal{X}, \mathcal{C}) \), satisfying \( F1-F3 \), we must be able to

1. find an interval schedule \( I_S = \{ [l_i, u_i] : X_i \in \mathcal{X} \} \) for \( S \), such that

2. the sum \( \sum_{X_i \in \mathcal{X}} (u_i - l_i) \) is maximal.

The first problem, that of computing an interval schedule, is solved by a transformation of the given \( \text{stn} \) \( S \), to another \( \text{stn} \) \( S' \). We will show that any solution to \( S' \) can be used to construct an interval schedule for \( S \).

The second problem is solved by realizing that any \( \text{stn} \), such as \( S' \), can be expressed as a set of linear expressions. A linear programming approach can then be used to solve the problem, and if we set the maximization of the sum of the interval sizes as objective, we are able to efficiently find an interval schedule that is maximal.

Computing interval schedules for an \( \text{stn} \)

As a start in computing an interval schedule for an \( \text{stn} \) \( S = (\mathcal{X}, \mathcal{C}) \), we will state a useful relationship between the bounds of an interval in a schedule \( I_S \) for \( S \), and the constraints occurring in \( \mathcal{C} \). Intuitively, as a consequence of requiring independent intervals, if there exists a constraint \( X_j - X_i \leq c_{ij} \in \mathcal{C} \), then the intervals \( I_i \) and \( I_j \) should be such that independent choices for \( v_i \in I_i \) and \( v_j \in I_j \) will not lead to a violation of this constraint. The following proposition states that the independence property is preserved if this constraint is satisfied whenever \( v_j \) takes its maximal value in \( I_j \), while \( v_i \) takes its minimal value in \( I_i \):

**Proposition 3.4** Let \( S = (\mathcal{X}, \mathcal{C}) \) be an \( \text{stn} \). A set \( I_S = \{ [l_i, u_i] : X_i \in \mathcal{X} \} \) of intervals for the variables in \( \mathcal{X} \) is an interval schedule for \( S \) if, for all \( (X_i, X_j) \in \mathcal{X}^2 \), \( X_j - X_i \leq c_{ij} \in \mathcal{C} \) implies \( u_j - l_i \leq c_{ij} \).

**Proof (if).** Suppose the implication holds for all \( (X_i, X_j) \in \mathcal{X}^2 \). To show that \( I_S \) is an interval schedule for \( S \), we choose an arbitrary value \( v_i \in [l_i, u_i] \) for every \( X_i \in \mathcal{X} \), and we let \( \sigma \) be defined by \( \sigma(X_i) = v_i \) for all \( X_i \in \mathcal{X} \).

We show that \( \sigma \) is a schedule for \( S \). Suppose, on the contrary, that it is not. Then \( \sigma \) violates some constraint in \( \mathcal{C} \), meaning that there exists a constraint
3.2. Flexibility of an STN

Let \( S = (\mathcal{X}, C) \) be an STN. Consider the double STN \( S' = (\mathcal{X}', C') \) derived from \( S \) as follows:

\[
\mathcal{X}' = \{ X_i^+ : X_i \in \mathcal{X} \} \cup \{ X_0' \}
\]

\[
C' = \{ X_i^+ - X_i^- \leq c_{ij} : X_j - X_i \leq c_{ij} \in C \} \cup \\
\{ X_i^- - X_i^+ \leq 0 : X_i \in \mathcal{X} \} \cup \\
\{ X_0' - X_0^- \leq 0, X_0^+ - X_0' \leq 0 \}
\]

Then, for every solution \( \sigma \) of \( S' \), the set \( I_S = \{ [\sigma(X_i^-), \sigma(X_i^+)] : X_i \in \mathcal{X} \} \) is an interval schedule for \( S \).
Computing an interval schedule $I_S$ for $S$ by using a schedule $\sigma'$ for a double STN $S'$.

$$S = (\mathcal{X}, \mathcal{C}) \quad \rightarrow \quad S' = (\mathcal{X}', \mathcal{C}')$$

$$I_S = \{[\sigma'(X_i^-), \sigma'(X_i^+)] : X_i \in \mathcal{X}\}$$

**Proof.** First, remember that we always assume a given STN $S$ to be consistent. We will start with showing that $S'$ is consistent, too. If we take an arbitrary schedule $\sigma$ for $S$ and construct an assignment $\sigma'$ for $S'$, by letting $\sigma'(X_i^+) = \sigma'(X_i^-) = \sigma(X_i)$ for every $X_i \in \mathcal{X}$, it is obvious that $\sigma'$ will satisfy all constraints in $\mathcal{C}'$. Hence, $S'$ is consistent.

Since $S'$ is consistent, we can assume some arbitrary schedule $\sigma'$ for $S$. We show that $\{[\sigma'(X_i^-), \sigma'(X_i^+)] : X_i \in \mathcal{X}\}$ is an interval schedule for $S$. If we take some arbitrary constraint $X_j - X_i \leq c_{ij} \in \mathcal{C}$, by Proposition 3.4, it is sufficient to show that $\sigma'(X_j^+) - \sigma'(X_i^-) \leq c_{ij}$. But this is immediate, since by construction of $\mathcal{C}'$, we know that $X_j^+ - X_i^- \leq c_{ij}$ is a constraint in $\mathcal{C}$, and $\sigma'$ is a schedule for $S'$.

In Figure 3.1, a commutative diagram is shown, depicting the suggested route to compute an interval schedule for an STN $S$, using the construction of the double STN.

**Example 3.3** Consider the STN $S_2$ presented in Example 3.1. Let STN $S_2' = (\mathcal{X}', \mathcal{C}')$ be the double STN derived from $S_2$ where

$$\mathcal{X}' = \{X_0', X_0^-, X_1^-, X_2^-, X_2^+, X_1^+, X_3^+, X_3^+\} \quad (3.27)$$

$$\mathcal{C}' = \{X_j^+ - X_j^- \leq 0 : 1 \leq i < j \leq 3\} \cup
\{X_j^+ - X_j^- \leq 5 : 1 \leq i < j \leq 3\} \cup
\{X_0^+ - X_i^- \leq 0 : i = 1, 2, 3\} \cup
\{X_i^+ - X_0^- \leq 0 : i = 1, 2, 3\} \cup
\{X_i^- - X_i^+ \leq 0 : X_i \in \mathcal{X}\} \cup
\{X_0^+ - X_0^- \leq 0, X_0^- - X_0^- \leq 0\} \quad (3.28)$$

One possible solution is

$$\sigma'(X_0^0) = 0 \quad \sigma'(X_0^+) = 0$$
$$\sigma'(X_1^-) = 0 \quad \sigma'(X_1^+) = 1$$
$$\sigma'(X_2^-) = 1 \quad \sigma'(X_2^+) = 3$$
$$\sigma'(X_3^-) = 3 \quad \sigma'(X_3^+) = 4$$
This schedule can be used to construct the following interval schedule for $S$:

$$I_S = \{ [0, 0], [0, 1], [1, 3], [3, 4] \}. \quad (3.29)$$

**Computing a maximal interval schedule for an STN**

In the previous section we introduced the concept of a double STN $S'$, with which it is possible to efficiently find interval schedules for the original STN $S$. Finding an arbitrary interval schedule is however not sufficient to determine the flexibility of an STN: the sum of the sizes of the intervals found is not necessarily maximal, as required by $F3$. We have thus to devise a method to find those solutions $\sigma'$ to the double STN $S'$ that maximizes the sum of the interval sizes.

It is easy to see that the following LP-formulation will solve this maximization problem, and thus also the problem of computing $\text{flex}(S)$:

**Theorem 3.1** Let $S = (\mathcal{X}, \mathcal{C})$ be an STN. Then $\text{flex}(S)$ can be computed by solving the following linear program:

$$\begin{align*}
\text{max} & \quad \sum_{X_i \in \mathcal{X}} (X_i^+ - X_i^-) \\
\text{subject to} & \quad X_i^- \leq X_i^+ \quad \forall X_i \in \mathcal{X} \\
& \quad X_j^+ - X_i^- \leq c_{ij} \quad \forall (X_j - X_i \leq c_{ij}) \in \mathcal{C} \\
& \quad X_0^+ = 0 \\
& \quad X_0^- = 0
\end{align*} \quad (3.31)$$

**Proof.** Let $S = (\mathcal{X}, \mathcal{C})$ be an STN. The constraints of the LP correspond to the constraints of the double STN $S'$. Hence, any solution $\sigma'$ of this set of constraints can be used to create an interval set for $S$. So, according to Proposition 3.3, the function $\text{flex}$ defined as

$$\text{flex}(X_i) = [\sigma'(X_i^-), \sigma'(X_i^+)] \quad \forall X_i \in \mathcal{X} \quad (3.32)$$

satisfies $F1$–$F2'$. Maximizing the sum of the sizes of the intervals $\sum_{X_i \in \mathcal{X}} (X_i^+ - X_i^-)$ thus obtained guarantees that this function also satisfies the last postulate, $F3$. \hfill \square

**Example 3.4** Applying the flexibility metric $\text{flex}$ to the STNs $S_1$ and $S_2$ from our earlier example results in the following flex functions:

1. For $S_1$, we have $\text{flex}(X_i) = [0, 5]$ for $i = 1, 2, 3$ and $\text{flex}(S_1) = 3 \times 5 = 15$. 


2. For \( S_2 \), we have \( \text{flex}(S_2) = 5 \); one possible flex function is \( \text{flex}(t_1) = [0, 5] \), \( \text{flex}(t_2) = \text{flex}(t_3) = [5, 5] \).

So as intended, this flexibility function will assign \( 3 \times 5 \) flexibility units to \( S_1 \), while assigning 5 units to \( S_2 \).

### 3.3 Applications

In this section we will discuss two applications of the flexibility metric introduced in this chapter. They are both focused on some of the problems encountered in dynamic schedule execution environments, such as found at NedTrain.

First, we apply the results of our flexibility metric to the temporal decoupling problem, in distributed scheduling, and we show that a globally optimal decoupling of an \( \text{STN} \) can be obtained. Next to being optimal, we also show that the flexibility of the subsystems is not affected, refuting a widespread belief in this research area.

A second application is in the field of the Simple Temporal Network with Uncertainty (\( \text{STNU} \)). Here, we show how the double \( \text{STN} \) construction can be applied to yield an elegant solution to the problem of strong controllability and its generalizations, in \( \text{STNUS} \).

#### 3.3.1 Decoupling without loss of flexibility

When events have to be scheduled in a distributed environment, \( \text{STNs} \) are often used as well (see Hunsberger, 2002; Boerkoel and Durfee, 2011, 2012; Brambilla, 2010). An \( \text{STN} S = (\mathcal{X}, C) \) is called distributed if the events in \( \mathcal{X} \) are distributed over \( k \) agents \( A_1, A_2, \ldots, A_k \), such that each individual agent \( A_i \) is responsible for scheduling a (non-empty and disjoint) subset \( \mathcal{X}_i \subseteq \mathcal{X} - \{X_0\} \) of events. Together, we assume that the set \( A = \{A_1, A_2, \ldots, A_k\} \) of agents induces a partition \( \{\mathcal{X}_i\}_{i=1}^k \) of \( \mathcal{X} \), i.e., the union of all sets \( \mathcal{X}_i \) equals \( \mathcal{X} - \{X\} \), meaning that all time point variables are controlled by an agent.

The main reason for modeling the scheduling of the events \( \mathcal{X} \) as a distributed environment is the assumption that the agents \( A_i \) act as autonomous schedulers—each agent \( A_i \) wants to determine its own schedule \( \sigma_i : \mathcal{X}_i \rightarrow \mathbb{R} \) for the set \( \mathcal{X}_i \) of tasks assigned to it, independent of the other agents. The problem with the requirement of autonomously scheduling agents however is that we still have to ensure, no matter what choices the agents make in constructing their individual schedule \( \sigma_i \), that the composition of all these schedules satisfies all constraints \( C \) in the global \( \text{STN} S = (\mathcal{X}, C) \). This means that, for every agent \( A_i \), we have to find an \( \text{STN} S_i = (\mathcal{X}_i, \mathcal{C}_i) \) such that the merge \( \sigma = \bigcup_{i=1}^k \sigma_i \) of these individual schedules always constitutes a valid schedule for the global \( \text{STN} S \).

---

\( ^4 \)Note that the reference time point \( X_0 \) is excluded: it is fixed, and hence no agent controls it.
3.3. Applications

This problem, of ensuring that the merge $\sigma$ of individual schedules $\sigma_i$ for the stns $S_i$ controlled by the agents, is known as the Temporal Decoupling Problem Hunsberger (2002).

**Definition 3.2** Let $S = (\mathcal{X}, \mathcal{C})$ be an stn. Suppose that $\mathcal{X} - \{z\} = \{X_i\}_{i=1}^k$ is partitioned in $k$ subsets $\mathcal{X}_i$. Then the temporal decoupling problem is to find $k$ stns $S_i = (\mathcal{X}_i \cup \{X_0\}, \mathcal{C}_i)$ such that, whenever $\sigma_1, \ldots, \sigma_k$ are independently constructed schedules for the individual stns $S_1, \ldots, S_k$, respectively, their merge $\sigma = \bigcup_{i=1}^k \sigma_i$ is a schedule for the original stn $S$.

The heart of the problem are constraints involving events $X_i$ and $X_j$ that belong to different agents $A_f$ and $A_h$. These agents may choose schedules $\sigma_f$ and $\sigma_h$, respectively, that satisfy their own private constraints $C_f$ and $C_h$, respectively, but violate an inter-agent constraint $X_j - X_i \leq c_{ij}$ by setting $\sigma_f(X_j) - \sigma_h(X_i) > c_{ij}$.

**Example 3.5** Suppose that in the stns $S_1$ and $S_2$ of our earlier examples, we have three agents $A_1, A_2$ and $A_3$, where $A_i$ is responsible for scheduling event $X_i$, with $i = 1, 2, 3$. Suppose that in $S_i = (\mathcal{X}, \mathcal{C}_i)$ we simply assign the constraints relating to $X_i$ to $A_i$. This means that each agent $A_i$ only needs to take care of the subset $\mathcal{C}_i = \{0 \leq X_i - X_0 \leq 5, 0 \leq X_0 - X_i \leq 5\}$. Hence, agent $A_1$ might choose $\sigma_1(X_1) = 4$, $A_2$ might choose $\sigma_2(X_2) = 2$, and $A_3$ might choose $\sigma_3(X_3) = 1$. Each of these assignment satisfies the local constraints, but together they violate the inter-agent constraints: for example, while $X_1 - X_2 \leq 0 \in \mathcal{C}$, we have $\sigma_1(X_1) - \sigma_2(X_2) > 0$.

Hunsberger (2002) gave an elegant solution to this problem. Assuming a minimal stn has been computed for the stn to be decomposed (i.e., the distance matrix $D_S$ is known), he proposed an iterative procedure that basically makes each inter-agent constraint $X_i - X_j \leq D_S[j, i]$ obsolete by tightening the associated intra-agent constraints $X_i - X_0 \leq D_S[0, i]$ and $X_0 - X_i \leq D_S[j, 0]$.

These tightenings consist in selecting values $\delta_{i,1}$ and $\delta_{i,2}$ such that

1. $X_i - X_0 \leq \delta_{i,1} \leq D_S[0, i]$,
2. $X_0 - X_j \leq \delta_{i,2} \leq D_S[j, 0]$, and
3. $\delta_{i,1} + \delta_{i,2} \leq D_S[i, j]$.

Now, if we add the new constraints

$$X_i - X_0 \leq \delta_{i,1} \leq D_S[0, i] \tag{3.33}$$

and

$$X_0 - X_j \leq \delta_{i,2} \leq D_S[j, 0] \tag{3.34}$$
to \( S \), while making sure that the resulting system remains consistent, the effect of adding these constraints is that an inter-agent constraint \( X_i - X_j \leq D_S[i, j] \) is implied by the additional, tighter, inter-agent constraints, since we have

\[
(X_i - X_j) = (X_i - X_0) + (X_0 - X_j) \leq \delta_{i,1} + \delta_{i,2} \leq D_S[i, j].
\]

(3.35)

The result is that such an inter-agent constraint can be removed from the updated \( \text{stn} \). Applying this procedure for every non-implied inter-agent constraint ensures that all inter-agent constraints are implied by intra-agent constraints and the resulting system is a temporal decoupling. All the inter-agent constraints can be safely removed from the \( \text{stn} \), and the resulting system is a temporal decoupling of \( S \).

**Example 3.6** Again referring to the two \( \text{stn} \) instances \( S_1 \) and \( S_2 \), note that in \( S_1 \), every inter-agent constraint \(-5 \leq X_i - X_j \leq 5, 1 \leq i \neq j \leq 3, \) is already implied by the intra-agent constraints \( 0 \leq X_i - X_0 \leq 5 \) and \( 0 \leq X_j - X_0 \leq 5 \), since

\[
X_i - X_j = (X_i - X_0) + (X_0 - X_j) \leq 5 + 0 = 5.
\]

Therefore, \( S_1 \) is in fact already a decoupled \( \text{stn} \).

In \( S_2 \), however, we can achieve a decoupling by tightening the intra-agent constraints to (for example):

\[
0 \leq X_1 - X_0 \leq 2 \quad 3 \leq X_2 - X_0 \leq 4 \quad 4 \leq X_3 - X_0 \leq 5.
\]

The reader might verify that by these tightenings indeed every inter-agent constraint is now implied.

An arbitrary decoupling, however, is not always what we want, since the added constraints may limit the resulting flexibilities of the subsystems \( S_i \). As a result, the sum \( \sum_{i=1}^{k} \text{flex}(S_i) \) of the flexibilities of the subsystems \( S_i \) could be considerably less than the flexibility \( \text{flex}(S) \) of the original system.

**Example 3.7** To illustrate this flexibility loss due to decoupling, consider the decoupling of the \( \text{stn} \) \( S_2 \) as discussed in Example 3.6. We know that for the original \( \text{stn} \), we have \( \text{flex}_N(S_2) = 15 \), \( \text{flex}_H(S_2) = 30 \) and \( \text{flex}(S_2) = 5 \). If we compute the sum of the flexibilities of all the subsystems \( S_{2,i} \), it is evident that the given decoupling affects the flexibility of the system, regardless of the metric considered:

\[
\sum_{i=1}^{3} \text{flex}_N(S_{2,i}) = 2 + 1 + 1 = 4 < 15
\]

\[
\sum_{i=1}^{3} \text{flex}_H(S_{2,i}) = 2 + 1 + 1 + 3 + 2 + 3 = 12 < 30
\]

\[
\sum_{i=1}^{3} \text{flex}(S_{2,i}) = 2 + 1 + 1 = 4 < 5
\]
It is a popular belief in the research community that an optimal decoupling
i.e., a decoupling that would maximize the flexibilities of the subsystems, would
still suffer from some flexibility loss. In the literature we are aware of (e.g.,
Hunsberger, 2002; Boerkoel and Durfee, 2011, 2012) several experiments have
been performed to measure this loss of flexibility. However, the metrics used all
have been based on the $\text{flex}_N$ and $\text{flex}_H$ metrics discussed above.

Instead of using this metric we try to find a decoupling that maximizes
the sum $\sum_{i=1}^{k} \text{flex}(S_i)$ of the flexibilities of the induced subsystems $S_i$, and we
consider it an optimal decoupling if the ratio $(\sum_{i=1}^{k} \text{flex}(S_i))/\text{flex}(S)$ is maximal.

Contrary to the popular beliefs in the research community we will show now
that

1. an optimal decoupling can be obtained in $O(k)$-time using the flexibility
metric we discussed before, and
2. this optimal decoupling need not affect the flexibility of the system, i.e., we
have $\sum_{i=1}^{k} \text{flex}(S_i) = \text{flex}(S)$.

**Proposition 3.6** Let $\{S_i\}_{i=1}^{k}$ be an optimal decoupling of an STN $S$. Then
$\sum_{i=1}^{k} \text{flex}(S_i) = \text{flex}(S)$.

**Proof.** Consider the set of intervals $\{(l_i, u_i) : X_i \in \mathcal{X}\}$ occurring as solutions
of the LP from Theorem 3.1. Given the set $A = \{A_i\}_{i=1}^{k}$ of $k$ agents, let $C_{\text{inter}} \subseteq \mathcal{C}$
be the set of all inter-agent constraints, and for $i = 1, \ldots, k$, let $C_i \subseteq \mathcal{C}$ be the set
of constraints restricted to $\mathcal{X}_i$. For every inter-agent constraint $X_j - X_i \leq c_{ij}$,
where $X_i$ occurs in $S_i$ and $X_j$ occurs in $S_j$, with $i \neq j$, add the constraint
$X_j - X_0 \leq u_j$ to $C_i$ and add $X_0 - X_i \leq -l_i$ to $C_j$. We show the following.

**Claim 1** The resulting systems $\{S_i\}_{i=1}^{k}$ constitute a decoupling of $S$, and

**Claim 2** The sum of the flexibilities $\text{flex}(S_i)$ of the systems $S_i$ in the decou-
pling $\{S_i\}_{i=1}^{k}$ equals $\text{flex}(S)$.

These claims, together, obviously imply the proposition.

**Proof of Claim 1** Suppose, on the contrary, that the merge $\sigma$ of individual
schedules $\sigma_i$ violates a constraint in $\mathcal{C}$. This must be an inter-agent con-
straint $X_j - X_i \leq c_{ij}$ where $X_i$ occurs in $S_i$ and $X_j$ occurs in $S_j$ ($i \neq j$). So
we must have

$$\sigma_j(X_j) - \sigma_i(X_i) > c_{ij},$$
(3.36)

but since $\sigma_i$ satisfies all constraints in $\mathcal{C}_i$, it follows by construction that
$\sigma_i(X_i) \geq l_i$, and similarly, $\sigma_j(X_j) \leq u_j$. Therefore,

$$\sigma_j(X_j) - \sigma_i(X_i) \leq u_j - l_i.$$  (3.37)
Observe that, since \( X_j - X_i \leq c_{ij} \), the constraint \( X_j^+ - X_i^- \leq c_{ij} \) occurs in the LP formulation. Since \( X_j^+ = u_j \) and \( X_i^- = l_i \) occur in the solution of this LP, we also know that \( u_j - l_i \leq c_{ij} \). Hence, by (3.37),

\[
\sigma_j(X_j) - \sigma_i(X_i) \leq c_{ij}, \tag{3.38}
\]

contradicting (3.36).

**Proof of Claim 2** Let \( S' = (\mathcal{X}', \mathcal{C}') \), where \( \mathcal{X}' = \bigcup_{i=1}^k \mathcal{X}_i \) and \( \mathcal{C}' = \bigcup_{i=1}^k \mathcal{C}_i \cup \mathcal{C} \). Note that \( \text{flex}(S') = \sum_{i=1}^k \text{flex}(S_i) \). Since \( \mathcal{X}_i = \mathcal{X} \) and \( \mathcal{C} \subseteq \mathcal{C}' \), it is easy to see that

\[
\text{flex}(S') = \sum_{i=1}^k \text{flex}(S_i) \leq \text{flex}(S). \tag{3.39}
\]

Let \( I_S = \{ [l_i, u_i] : X_i \in \mathcal{X} \} \) be an interval schedule for \( S = (\mathcal{X}, \mathcal{C}) \) realizing maximum flexibility. For every subsystem \( S_i = (\mathcal{X}_i, \mathcal{C}_i) \), consider the set

\[
I_{S_i} = \{ [l_i, u_i] \in I_S : X_i \in \mathcal{X}_i \}, \tag{3.40}
\]

being the restriction of \( I_S \) to \( \mathcal{X}_i \). We show that for each \( i = 1, \ldots, k \), \( I_{S_i} \) is an interval schedule for \( S_i \). First, let \( \sigma_i \) be such that \( l_j \leq \sigma_i(X_j) \leq u_j \) for every \( X_j \in \mathcal{X}_i \). We show that \( \sigma_i \) is a valid schedule for \( S_i \). So consider an arbitrary constraint \( X_j - X_i \leq c_{ij} \in \mathcal{C}_i \). We consider the following cases:

- \( X_j = X_0 \) In this case, \( X_0 - X_i \leq c_{i0} \in \mathcal{C}_i \). If this constraint occurs in \( \mathcal{C} \), too, we know that \( 0 - l_i \leq c_{i0} \) since \( X_0 - X_i^- \leq c_{i0} \) occurs in the LP. Since \( \sigma_i(X_i) \geq l_i \), it follows that \( 0 - \sigma_i(X_i) \leq c_{i0} \) and \( \sigma_i \) satisfies the constraint.

  If this constraint does not occur in \( \mathcal{C} \), then it is added in the decomposition. Therefore, \( c_{i0} = -l_i \). Since \( \sigma_i(X_i) \geq l_i \), it follows that \( 0 - \sigma_i(X_i) \leq -l_i = c_{i0} \) and \( \sigma_i \) satisfies the constraint in this case as well.

- \( X_i = X_0 \) In this case, \( X_j - X_0 \leq c_{0j} \). Again, we have two sub-cases. If \( X_j - X_0 \leq c_{0j} \) occurs in \( \mathcal{C} \), we know that \( u_j - 0 \leq c_{0j} \) since \( X_j^+ - X_0 \leq c \) occurs in the LP. Since \( \sigma_i(X_j) \leq u_j \), it follows that \( \sigma_i(X_j) \leq c_{0j} \) and the constraint is satisfied.

  If this constraint does not occur in \( \mathcal{C} \), again, it is added to achieve the decomposition. Then, \( c = u_j \), and since \( \sigma_i(X_j) \leq u_j \) we have \( \sigma(X_j) \leq c \), and the constraint is satisfied.

- \( X_i \neq X_0, X_j \neq X_0 \) In this case \( X_j - X_i \leq c_{ij} \) must occur in \( \mathcal{C} \). Hence \( X_j^+ - X_i^- \leq c_{ij} \) occurs in one of the conditions in the LP and \( u_j - l_i \leq c_{ij} \).

  Since \( \sigma_i(X_j) \leq u_j \) and \( \sigma_j(X_i) \geq l_i \), we have \( \sigma_i(X_j) - \sigma_i(X_i) \leq c_{ij} \) and \( \sigma_i \) satisfies the constraint.
This shows that $\sigma_i$ is a valid schedule for $S_i$, so, for each $i = 1, \ldots, k$, $I_S$ is an interval schedule for $S_i$. But, then it follows that for each $i = 1, \ldots, k$, we have

$$\sum_{I_j \in I_{S_i}} |I_j| \leq \sum_{X_j \in X_i} \text{flex}(X_j) = \text{flex}(S_i)$$  \hspace{1cm} (3.41)

Summing over $i = 1, \ldots, k$, we derive that

$$\text{flex}(S) = \sum_{i=1}^k \sum_{I_j \in I_{S_i}} |I_j| \leq \sum_{i=1}^k \sum_{X_j \in X_i} \text{flex}(X_j) = \sum_{i=1}^k \text{flex}(S_i)$$  \hspace{1cm} (3.42)

Now Claim 2 follows from Equations 3.39 and 3.42.

\[ \square \]

**Example 3.8** Consider Example 3.6 again. A decoupling for $S_3$ realizing maximum flexibility can be created by adding the following decoupling constraints:

$$X_2 - X_0 \leq 5 \quad X_0 - X_2 \leq -5 \quad X_3 - X_0 \leq 5 \quad X_0 - X_3 \leq -5.$$

The sum of flexibilities of the subsystems equals the flexibility of the original system.

### 3.3.2 Strong controllability in the STNU

In the Simple Temporal Problem with Uncertainty (STNU) the STN is extended with an explicit distinction between executable and contingent events. This is an explicit way of modeling uncertainty: The set $\mathcal{X}^E$ contains the executable events, which are controlled by an agent executing the schedule, and the set $\mathcal{X}^C$ contains the contingent events, which are controlled by an external entity beyond our control, such as nature itself.

Given this distinction between $\mathcal{X}^C$ and $\mathcal{X}^E$, we can define the strong controllability problem:

**Given:** An STNU $S = (\mathcal{X}^C \cup \mathcal{X}^E, C)$, where $|\mathcal{X}^C| = m$ and $|\mathcal{X}^E| = n$.

**Question:** Is there a choice of (scheduled) time point values $v_0, \ldots, v_n$ for the events $X_i \in \mathcal{X}^E$ such that for every choice of time points $w_1, \ldots, w_m$ for events $X_j \in \mathcal{X}^C$, the function $\sigma$ defined by $\sigma(X_i) = v_i$ for $X_i \in \mathcal{X}^E$ and $\sigma(X_j) = w_j$ for $X_j \in \mathcal{X}^C$ is a schedule for $S$?

In other words, is it possible to construct an assignment a priori for the executable events in $\mathcal{X}^E$, which will result in a valid schedule for $S$ for any possible execution of the events in $\mathcal{X}^C$ that are beyond our control?
While it has been shown (see Vidal and Fargier, 1999) that this problem is tractable, we would like to show how our approach to flexibility can handle this problem in a very natural way, even extending the concept of strong controllability to a more general one. While the actual distinction in an \textsc{stn} is between contingent and requirement links between events, for our purposes here we will focus on the events themselves instead, distinguishing between \textit{required} flexibility of an event, and \textit{computed} flexibility. The former refers to the contingent events, controlled by nature, and the latter to the events under control of the executing agent. For each contingent event $X_i$ we only know a lower bound $l_i$ and an upper bound $u_i$ on the time of its occurrence. It is easy to see that we can re-cast this in terms of an interval schedule, such as introduced above, by stating that for each such event $X_i \in \mathcal{X}^C$, its required flexibility equals $I_i^C = [l_i, u_i]$. The next step then follows quite naturally, as instead of asking for the existence of a fixed schedule for the executable events in $\mathcal{X}^E$, we ask for a maximal interval schedule for these events:

\begin{center}
\begin{tabular}{|l|}
\hline
\textbf{Given:} & An \textsc{stn} $S = (\mathcal{X}^C \cup \mathcal{X}^E, C)$, where $|\mathcal{X}^C| = m$ and $|\mathcal{X}^E| = n$. \\
\textbf{Question:} & Is there a set $I^E$ of intervals $I^E_i$ for the events $X_i \in \mathcal{X}^E$ such that for the set $I^C$ of required flexibility intervals $I^C_j$, $X_j \in \mathcal{X}^C$, the set $I^E \cup I^C$ is a maximal interval schedule for $S$ extending $I^C$? \\
\hline
\end{tabular}
\end{center}

Note that in the case that $\mathcal{X}^C = \emptyset$, this problem reduces to that of finding a maximal interval schedule for an \textsc{stn} to obtain $\text{flex}(S)$, as discussed in the previous section.

A simple solution to this problem can be given by a slight modification to our original method, which obtains a maximal interval schedule for $S$ by using the double \textsc{stn} method, and solves an \textsc{lp} with maximization of the flexibility as the objective function. In order to represent the given (partial) interval schedule $I^C$ in our method,

1. we should add required values for the lower bound variable $X_i^-$ and the upper bound variable $X_i^+$ for every $X_i \in \mathcal{X}^C$ in the construction of the double \textsc{stn}, to ensure that the given flexibility interval $I_i^C$ has its required width, and

2. we should restrict the objective function to the sum of the differences $(X_i^+ - X_i^-)$ where $X_i \in \mathcal{X}^E$, as the differences $(X_i^+ - X_i^-)$ for $X_i \in \mathcal{X}^C$ are fixed.

Adapting the formulation from Theorem 3.1, it is easy to see that the following \textsc{lp} formulation will solve this problem:

\begin{center}
\textbf{Proposition 3.7} Let $S = (\mathcal{X}^E \cup \mathcal{X}^C, C)$ be an \textsc{stn}, where for each $X_i \in \mathcal{X}^C$ the uncertainty of $X_i$ is given by the interval $I_i^C = [l_i, u_i]$. Then there exists a solution to the strong controllability problem for $S$ if the following linear
program has a solution:

\[
\begin{align*}
\text{max} & \quad \sum_{X_i \in \mathcal{X}^E} (X_i^+ - X_i^-) \\
\text{subject to} & \quad X_i^- \leq X_i^+ \quad \forall X_i \in \mathcal{X} \\
& \quad X_j^+ - X_i^- \leq c_{ij} \quad \forall (X_j - X_i \leq c_{ij}) \in \mathcal{C} \\
& \quad X_i^- = l_i \quad \forall X_i \in \mathcal{X}^C \\
& \quad X_i^+ = u_i \quad \forall X_i \in \mathcal{X}^C \\
& \quad X_0^+ = 0 \\
& \quad X_0^- = 0
\end{align*}
\]

Moreover, if there is a solution, we are guaranteed to have a maximum flexible schedule for \( S \) extending \( I^C \).

**Proof.** Let \( S = (\mathcal{X}^E \cup \mathcal{X}^C, \mathcal{C}) \) be an stnu. The constraints of the LP are as least as tight as the constraints of the double stp \( S' \), such as in Theorem 3.1, so any solution \( \sigma' \) of this set of constraints can be used to create an interval set \( I = I^E \cup I^C \) satisfying the constraints of \( S' \), which guarantees that at least one solution exists.

We know that this interval set will have \( I_i^C = [l_i, u_i] \) for every \( X_i \in \mathcal{X}^C \), guaranteeing us that any assignment within these bounds satisfies the constraints in \( S' \), therefore, we can conclude that the solution that has been proven to exist is a solution to the strong controllability problem.

Lastly, since the intervals given by \( I_i^C \) are fixed, and the size of the intervals for \( I_i^E \) is maximized, we are guaranteed that the solution we have found has maximal flexibility. \( \square \)

This approach to solving the strong controllability problem can be generalized further. Suppose we have an instance of the strong controllability problem for which no schedule \( \sigma \) exists, solving it. Instead of stopping here, we might ask if it is possible to find a set of maximal subintervals \( I_i'^C \subseteq I_i^C \) for the contingent events \( X_i \in \mathcal{X}^C \) such that a schedule does exist. The reason for this generalization is obvious: if we manage to find a subset of these intervals that covers a large portion of the original intervals, there is a high probability that a chosen schedule for the executable events does not violate the constraints.

This problem can be simply solved by another modification to the LP formulation. In this case, we want to find a solution minimizing the total sum of differences between the lower and upper bounds obtained, and the lower and upper bounds required by the contingent events. The following linear program
achieves this:

\[
\min_{X_i \in \mathcal{X}} \sum_{X_i \in \mathcal{X}} \left( (u_i - X_i^+) - (X_i^- - l_i) \right)
\]

subject to

\[X_i^- \leq X_i^+ \quad \forall X_i \in \mathcal{X}\]

\[X_j^+ - X_i^- \leq c_{ij} \quad \forall (X_j - X_i \leq c_{ij}) \in \mathcal{C}\]

\[X_i^+ \geq l_i \quad \forall X_i \in \mathcal{X}^C\]

\[X_i^+ \leq u_i \quad \forall X_i \in \mathcal{X}^C\]

\[X_0^+ = 0 \quad \forall X_i \in \mathcal{X}\]

\[X_0^- = 0\]

### 3.4 Conclusion and discussion

In this chapter, the first research question was investigated, asking which criteria encapsulate our intuitive notion of flexibility, and how to devise a method measuring that flexibility. For the first part of this question, the rationality postulates \(F_1\) through \(F_3\) introduced in this chapter provide a rigorously defined answer to this question, based on the concept of an interval schedule, in which each time point variable has an associated interval from which a value can be picked. The flexibility of the schedule is then defined as the sum of all the interval sizes. The most significant contribution here is postulate \(F_2'\), describing the independence requirement. As Proposition 3.2 shows, satisfying this postulate indicates that all the choices made in assigning time point variable values from such intervals are independent from each other. Assigning a value to a time point variable has no impact on the range of choices for any of the other intervals. Hence, we can be assured that the sum of the sizes of these intervals is representative of the flexibility we can make use of while executing the schedule.

The second major contribution is Theorem 3.1, which shows a method to actually calculate the sum of these interval sizes, hereby answering the second part of the research question. A side result of the method is a valid interval schedule for the original STN, which can be used to efficiently execute the schedule under dynamic circumstances.

Besides schedule execution, two other applications are discussed as well. The first application is in the optimal decoupling of multi-agent schedules, partially answering the fourth research question, asking how to decouple schedules in such a way that disturbances in the schedule in one agent have low impact on the other agents. We disprove the wide-spread belief in the research community that the temporal decoupling of an STN leads to a reduction of the available flexibility, based on the flexibility definition using our rationality postulates: we show that our earlier result on the creation of maximally flexible interval schedules can be applied to create a temporal decoupling for an STN.
The second application is in solving the strong controllability problem for the STNU, the Simple Temporal Network with Uncertainty. Here, uncertain events are modeled explicitly, as contingent events over which the executing agent has no control. Our method to find interval schedules can be adapted to this problem, by explicitly assigning the required flexibility to these contingent events, and then finding a maximally flexible interval schedule for the controllable events. Additionally, we generalize the notion of finding a solution to the strong controllability problem. If a solution cannot be found, our method can, with another adaptation, find a solution that maximizes the probability of a consistent schedule, by finding new interval bounds for the contingent events, encapsulating the largest possible portion of the original intervals still resulting in a solution.

3.4.1 Discussion

As can be seen from Example 3.4, presenting an interval schedule with maximal flexibility, and Example 3.8, presenting a decoupling with maximal flexibility, interval schedules and decouplings can be obtained in many, sometimes unfair, ways. In both these examples, one event, and one agent, respectively, gets all the available flexibility, leaving none for the other events or agents. This indicates that there are multiple ways to make use of the available flexibility in a schedule, which shows the need for the second research question, asking how flexibility can be used best to achieve a schedule with high robustness. In the next chapter we will therefore investigate modifications to the LP formulation presented in Theorem 3.1, with the aim of influencing the flexibility distribution to improve fairness and robustness.

Another extension of the work in this chapter, of which the careful reader might have already realized the possibility, lies in the computation of the flexibility for general linear systems of the form $\mathbf{A}\mathbf{x} \leq \mathbf{b}$, as has already been done (see Witteveen et al., 2014). This offers the possibility to compute sets of solutions for this system, guaranteeing that any choice from within this set preserves the maximum value of the objective function. Moreover, the extension from schedule flexibility to schedule decoupling can also be made in this case, giving efficient solutions for decoupling problems, such as analyzed by Brodsky et al. (2005).
In the previous chapter, a method for constructing a maximally flexible interval schedule for an STN was introduced, answering the first research question. But as already mentioned in the discussion section, there is more to the flexibility of a schedule than simply maximizing it: often, the distribution of available flexibility is very lopsided. We therefore continue exploring the concept of the flexibility of a schedule in this chapter, turning to the second research question: we analyze the effect of different ways of distributing the available flexibility over the schedule, with the goal of achieving high robustness. Flexibility and robustness are two interrelated concepts. Flexibility, as discussed in the previous chapter, is the ability of a schedule to adapt to unexpected events without changes, for example by utilizing the intervals assigned to each task to adapt to the new situation.

It is clear that some events have such a disruptive nature that this is no longer possible. In this case the schedule has to be changed to adapt, usually by delaying tasks and starting them at a later time than intended. We define the robustness of a schedule as the ability of a schedule to withstand such disruptions—the fewer the number of tasks that have to be delayed outside their planned execution time to accommodate disruptions the more robust the schedule is. Often, a schedule is not only used to plan the execution of the tasks itself, but also for the reservation of resources to be used by the tasks, or to make commitments with external parties involved in the execution of tasks. If tasks have to be delayed to accommodate unexpected events, such commitments are put at risk, which is undesirable. If the distribution of flexibility is lopsided, parts of the schedule have very low flexibility, and disturbances will be much more disruptive. In this chapter we will examine ways to arrive at a distribution of flexibility in which this problem is reduced, in an attempt to increase the robustness.

This chapter starts with a motivation, which shows using examples why there is a need to pay attention to how the available flexibility in a schedule is distributed over the individual tasks. In the next section, methods are proposed to actually achieve certain distributions of flexibility, with the aim of improving the robustness of the schedule during execution. These methods are then tested using a simulated execution of the schedule, and the results are discussed and explained.
4.1 Motivation

In the previous chapter, we introduced a new method for measuring the flexibility of an interval schedule, which represents a set of solutions to a resource-constrained project scheduling problem instance: the original RCPSP is transformed into a STN, by using constraint posting techniques that add temporal constraints to eliminate resource constraints; for this STN we construct a solution in the form of an interval schedule, which then also serves as a solution for the original RCPSP instance. Many different interval schedules can be found for a single STN, however. To calculate the flexibility of the STN, an interval schedule is found that maximizes the sum of the sizes of the intervals, and hence the flexibility. But even using this objective, many different interval schedules are still possible. Looking back to the original problems as discussed in Chapter 1, one of the motivations for our work is the need for robust schedules, i.e., schedules that are resistant, to a certain extent, to external disturbances. Next to their use in dispatching tasks for execution, schedules are often also used to reserve resources needed for the execution of a task at some time point in the future, or to make commitments to external parties needed to execute a task. This is where robustness comes into play: when disturbances are present, it is important to still be able to rely on the predictions of a schedule, such that commitments that have been made based on the schedule can be honored. Robustness is therefore operationalized as the ability of a schedule to dispatch the tasks as planned, even when disturbances are present. It is in this light that this chapter will evaluate the different interval schedules that can be constructed for a given STN transformation of the original planning problem. To motivate this more clearly, let us look at the following example.

Example 4.1 Consider an RCPSP instance containing five tasks $t_1, \ldots, t_5$, where for all $i$, $s_i = 0$, $d_i = 10$ and $l_i = 1$, i.e., all tasks have length 1, and they have to execute within the time interval $[0, 10]$. Resource constraints dictate that none of the tasks can execute in parallel, such that one STN transformation for this problem would be $S_{\text{seq}} = (\mathcal{X}, \mathcal{C})$, with

$$\mathcal{X} = \{X_0, X_1, \ldots, X_5\}$$

representing the reference time point $X_0$ and the start times for the tasks, and

$$\mathcal{C} = \{X_0 - X_i \leq 0 : i = 1, \ldots, 5\} \cup \\ \{X_i - X_0 \leq 10 - l_i = 9 : i = 1, \ldots, 5\} \cup \\ \{X_i - X_{i+1} \leq -l_i = -1 : i = 1, \ldots, 4\}$$

representing, respectively, the release time constraints, due time constraints, and precedence constraints enforcing one particular sequential execution of the tasks. Below, the temporal network for this problem is shown (to reduce clutter, the constraints between a point and $X_0$ are shown as a label on that time point).
As the reader might verify, we have $\text{flex}(S_{\text{seq}}) = 5$ for this simple example. Two possible interval schedules achieving this flexibility score are

$$I_r = \{[0, 0], [1, 1], [2, 2], [3, 3], [4, 9]\},$$

a schedule placing all available flexibility at the “rightmost” time point $X_5$, and

$$I_l = \{[0, 5], [6, 6], [7, 7], [8, 8], [9, 9]\},$$

which places all flexibility at the “leftmost” time point $X_1$ (for both, $X_0 \in [0, 0]$). Both schedules have the same amount of total flexibility, but $I_l$ is much more vulnerable if task processing times are increased due to unexpected delays: for every task except $X_1$ such a delay has a direct effect on the makespan of the schedule. In contrast, $I_r$ is much less vulnerable to delay with regard to the makespan of the schedule, since all flexibility is placed at the end of the schedule.

As the example shows clearly, simply maximizing the flexibility is not enough if we are interested in robustness of the schedule: Both $I_r$ and $I_l$ satisfy the objective of maximal flexibility, but one of them is much better at guaranteeing we can satisfy the imposed deadline than the other. Besides satisfying a due date, or guaranteeing a certain makespan, however, there might be other aspects in which we could want to ensure robustness, as motivated by the following example.

**Example 4.2** Suppose that in the case of the scheduling problem outlined in the example above, task $t_4$ requires some sort of special tool, or an outside contractor, which has to be reserved at some time point well before executing the task itself. For both the interval schedules $I_l$ and $I_r$, as presented in the previous example, there is a risk that tasks preceding $t_4$ have an impact on the starting time of $t_4$, causing it to be started outside of the interval $I_4$ assigned to it. While this might have no impact on the deadline of the entire schedule, any previous commitment made to reserve the aforementioned special resource will be violated.

A schedule tailored to this specific situation is

$$I_4 = \{[0, 0], [1, 1], [2, 2], [3, 8], [9, 9]\},$$

where all available flexibility is given to $t_4$, which gives a high certainty that, whatever the delays occurring, it will be possible to execute it in the planned interval.
The above assumes a priori knowledge of which task(s) have external commitments that are to be honored. We can generalize this idea for situations where this is not known in advance, and try to improve the likelihood that the actual execution of a task \( t_i \) takes place in its assigned interval \( I_i \), such that any (prior) commitments related to the execution of \( t_i \) will be honored. If, in our example, the delays to be expected are small relative to the task length (i.e., the delay is rarely larger than the length of the actual task), a more equalized distribution of flexibility over the schedule, such as in a schedule like

\[
I_e = \{[0, 1], [2, 3], [4, 5], [6, 7], [8, 9]\},
\]

has a much lower risk of tasks execution outside their assigned interval due to delays. Note that we still have \( \text{flex}(I_e) = 5 \), meaning that the flexibility is still maximal.

We can even go a step further, and state that sacrificing some of the available flexibility, as seen over the total schedule, can be beneficial, if it improves the distribution of flexibility over the rest of the schedule. The following example provides a concrete motivation for this claim.

**Example 4.3** Consider a variation on the r\(\text{CP}\)\(\text{SP} \) instance presented in Example 4.1, in which \( t_2 \) is split into three sub-tasks that can be executed in parallel, given the available resources. The temporal network for one possible stn transformation \( S_{\text{par}} = (\mathcal{X}, C) \) is shown below.

Maximizing \( \text{flex}(S_{\text{par}}) \) yields, in this case, the unique interval schedule

\[
I_{\text{par}} = \{[0, 0], [1, 6], [1, 6], [7, 7], [8, 8], [9, 9]\}.
\]

For this schedule, we have \( \text{flex}(I_{\text{par}}) = 15 \), but note that all flexibility is concentrated in the parallel part of the schedule, i.e., in tasks \( t_2 \), \( t_3 \) and \( t_4 \). The sequential part of the schedule receives no flexibility at all. Recalling the argument made in Example 4.2, there might be valid reasons for wanting to have a more equalized distribution of flexibility over the tasks. A schedule
that achieves this for this example is

\[ I'_{\text{par}} = \{[0,1], [2,3], [2,3], [2,3], [4,5], [6,7], [8,9]\}, \]

in which every task has an equal amount of flexibility. Note, however, that \( \text{flex}(I'_{\text{par}}) = 7 \): in global terms, this schedule offers lower flexibility. However, if honoring overall commitments made prior to execution is a concern, it is clear that \( I'_{\text{par}} \) is to be preferred over \( I_{\text{par}} \): flexibility is distributed more evenly, such that the sequential parts of the schedule have higher robustness than when using \( I_{\text{par}} \).

From these examples, it is clear that there is more to it than maximizing the flexibility of a schedule if our goal is to achieve some form of robustness. Moreover, we observe that, even if we keep our aim of having maximal global flexibility, there may be multiple schedules satisfying that goal, with differing properties regarding robustness during execution. Finally, sometimes there is a clear case to be made for sacrificing a certain amount of (global) flexibility, to be able to improve the distribution of flexibility over all individual tasks in a schedule. In the following section, we will formalize this notion of flexibility distribution.

### 4.2 Methods for distributing flexibility

As a first step in formalizing the distribution of flexibility over interval methods, we note that, while our aim is to distribute flexibility more equally, it is undesirable to enforce a strictly uniform distribution of flexibility—this could easily lead to schedules with unreasonably low or even zero flexibility: If there is a single task with low or zero flexibility, this would enforce this level of flexibility to all other tasks, needlessly reducing flexibility.

**Equalizing flexibility** To make this idea more explicit, we start by examining the intervals \( I_a \) and \( I_b \) assigned to two time point variables \( X_a \) and \( X_b \) in some stn \( S = (\mathcal{X}, \mathcal{C}) \) over which we need to distribute flexibility. Our goal is to discriminate between two situations, in which the total flexibility is equal, but in which the distribution differs. In this case, we would like to differentiate between \( |I_a| = |I_b| \), and \( |I_a| - \varepsilon < |I_b| + \varepsilon \). While \( |I_a| + |I_b| = |I_a'| + |I_b'| \), the first assignment is to be preferred, since the distribution of the flexibility is more equal. Minimizing this deviation \( \varepsilon \) bears resemblance to the idea of least squares minimization, in which the squared residual error of a curve as compared to a set of data points is minimized. Therefore, we propose to use the objective function

\[
\min \sum_{X_i \in \mathcal{X}} ((\text{lst}(X_i) - \text{est}(X_i)) - (X_i^+ - X_i^-))^2,
\]

which is used to minimize the squared difference between the maximally available flexibility for each time point variable \( X_i \), and the actual flexibility received
in the interval schedule. This will result in equalization of flexibility, since distribution of a certain amount of flexibility over multiple time point variables will yield a lower score than if all flexibility were concentrated on a single variable. This equation replaces Equation 3.30 in Theorem 3.1.

The above assumes that all tasks in a scheduling problem are equal, in terms of flexibility requirements. In practice, this is not always the case. If it is known in advance that some tasks are more susceptible to disturbances, this should be reflected in the flexibility distribution of the schedule, by favoring these tasks. Similarly, the structure of the dependency network connecting the tasks in the schedule can also have influence on the consequences of a disturbance. As an example, if a certain task $t_i$ has a large number of predecessors, the chances that at least of them will suffer a delay is higher. Assigning more flexibility to the associated time point $X_i$ as a prevention can increase the robustness of the schedule.

**Influencing the flexibility distribution** Reflecting this desire to more directly influence the flexibility distribution, we introduce the objective function

$$\min \sum_{X_i \in \mathcal{X}} w(X_i) \cdot ((\text{lst}(X_i) - \text{est}(X_i)) - (X_i^+ - X_i^-))^2,$$

(4.2)
a weighted objective function, where $w : \mathcal{X} \rightarrow \mathbb{R}$ is a weight function influencing the amount of flexibility $X_i$ should have. The higher $w(X_i)$, the more flexibility is given to $X_i$, at the expense of tasks with a weight lower than $w(X_i)$.

An obvious choice for $w$ would be some function representing a priori knowledge of delay probabilities for the tasks in the schedule. If it is known in advance that a certain task is highly probable to be delayed, it makes sense to give it a higher weight, to ensure that additional flexibility is available for it. However, we will assume that no such information is available beforehand, and that we will have to use other properties to influence the weight per time point variable. One evident property that we could use is of course the structure of the network of tasks surrounding a time point variable $X_i$, in particular the number of predecessors and successors.

To start, we consider the number of predecessors $X_i$ has. Our rationale is that a time point variable with more predecessors has a higher risk of being impacted by a delay, since any of those predecessors could incur a delay that might propagate to $X_i$. A basic weight function $w(t_i)$ based on the number of predecessors of $X_i$ can be defined as

$$w(X_i) = |\{X_j : X_j < X_i\}|.$$

(4.3)

This function simply counts the number of predecessors $X_i$ has, and assigns that value to $w(X_i)$.

\(^1\) Instead of a linear programming problem, we are now confronted with a quadratic programming problem. But as it turns out, the functions to be optimized are convex, making the use of the ellipsoid method possible, which solves this problem in polynomial time.
A disadvantage might be that this function heavily biases the “tail end” of the schedule: time point variables at the end of the schedule invariably have more predecessors than those at the beginning. Furthermore, we can pose that a delay at the very beginning has a low chance of reaching the very end of a schedule. Time points in between will usually also have some flexibility, which absorbs the delay, especially if the delay is small. In the most extreme case, we could propose to use the function

\[ w(X_i) = |\{ X_j : X_j \ll X_i \}|, \quad (4.4) \]

where \( \ll \) is the immediate predecessor relation, i.e., \( X_j \ll X_i \) if, and only if, \( X_j < X_i \) and there is no \( X_k \) such that \( X_j < X_k < X_i \), with \( i \neq j \neq k \). This function hence only counts the direct predecessors of a time point variable \( X_i \) and uses that value to assign a value to \( w(X_i) \).

It might be desirable to have a weight distribution between these two extremes, i.e., we want to account for some of the non-direct predecessors, but not for all of them, in how we assign weight to time point variables. To this end, we can generalize our weight function to

\[ w(X_i) = \sum_{X_j \in \text{pre}(X_i)} \max\left(1 - \frac{\text{distance}(X_j, X_i) - 1}{\phi}, 0\right), \quad (4.5) \]

where \( \text{pre}(X_i) = \{ X_j : X_j < X_i \} \) contains the set of predecessors of \( X_i \), distance\((X_j, X_i)\) is the length of the shortest path between \( X_j \) and \( X_i \) (i.e., the number of precedence constraints connecting them), and \( \phi \) is a discounting factor. For \( \phi = 1 \), this function reduces to that of Equation 4.4, counting only the direct predecessors. In general, for \( \phi = j \), predecessors up to \( j \) steps removed from \( X_i \) are considered, where their value is discounted linearly, i.e., a predecessor removed \( j \) steps contributes a weight of \( 1/j \) to the total value of \( w(X_i) \).

Next to arguing that more predecessors cause more delay, and we should have more flexibility for time point variables that have more predecessors, we could as well reason more pro-actively: time point variables that have a high number of successors are at risk of propagating a delay to many other points in a network, hence we should give more weight to such variables.

Therefore, we can define

\[ w(X_i) = |\{ X_j : X_i \ll X_j \}|, \quad (4.6) \]

a weight function that assigns to \( w(X_i) \) a value equal to the total number of successors of \( X_i \). Similar objections can be raised here as for Equation 4.3, which counts all predecessors, so here we can also define

\[ w(X_i) = |\{ X_j : X_i \ll X_j \}|, \quad (4.7) \]

a function counting only the direct successors, in similar vein to Equation 4.4, and of course

\[ w(X_i) = \sum_{X_j \in \text{succ}(X_i)} \max\left(1 - \frac{\text{distance}(X_i, X_j) - 1}{\phi}, 0\right), \quad (4.8) \]
where \( \text{succ}(X_i) = \{ X_j : X_i < X_j \} \) is the set of successors of \( X_i \).

### 4.3 Experiments

In Examples 4.1 and 4.3 the main motivation behind the ideas in this chapter was sketched: maximizing the flexibility of a schedule need not be the same as maximizing the robustness of that schedule. One reason, demonstrated in the first example, is that multiple such schedules can be constructed, which will have different properties during execution. An even more compelling reason is outlined in Example 4.3: maximizing can lead to very skewed distributions of flexibility, which in turn will have a negative impact on the robustness. While these examples prove the existence of such effects, more general conclusions cannot be drawn from them. For this reason, this section presents hypotheses and results from simulation experiments designed to see if these hypotheses indeed hold.

A first hypothesis is that the problem outlined by the examples in the motivation is not a contrived example, but that sacrificing flexibility to improve the flexibility distribution has a positive effect on the robustness of a schedule in general. Several new distributions have been proposed in this chapter, the next question is therefore which of these distributions works best. The hypothesis is that network structure plays a role in the propagation of disturbances, and that adapting the distribution to that structure has a positive effect on the performance. It is unclear from the theory how to adapt the distribution: one hypothesis is that tasks with many predecessors benefit from receiving more flexibility, another hypothesis is that tasks with many successors benefit instead.

To test these hypotheses, we will be performing simulation experiments, in which an interval schedule constructed using one of the proposed flexibility distribution methods will be executed repeatedly. For each execution, a set of tasks will be picked that will be artificially delayed by increasing their execution time. To evaluate the robustness of an interval schedule, we will be focussing on how well the simulated execution adheres to the original intervals. If a task has to start outside its assigned interval, this is seen as a violation. The more violations an interval schedule has, the lower its robustness. The rationale behind this idea is that it is our aim to construct a robust schedule, which can be used as a basis to make commitments. If a task starts outside its assigned interval, due to an unforeseen delay, this violates such a commitment.

We start by examining the overall performance, averaged over all experiments performed, then we will zoom in and analyze the effect of different types of delay. In particular, we will vary both the number of tasks affected by delay, and the severity of the delay itself. It is expected that there will be a clear correlation between increasing both of these parameters and the performance of the interval schedules. It is unclear however which of these will have the biggest impact.

Lastly, we will perform a more extended analysis of the relationships between sacrificing flexibility and increasing robustness, and between the trade-off of constraint violations and schedule tardiness. For the first relationship, two
4.3. Experiments

opposing forces are at work: On one side, we expect that sacrificing flexibility will improve performance, as this is the idea behind the equalized flexibility distribution. On the other side, if too much flexibility is lost, schedule execution performance will eventually drop. For the second relationship, we refer back to Example 4.1: From these examples it is clear that if flexibility is placed at the end, we can expect lower tardiness. But, this also means that the other tasks have lower flexibility, which leads us to expect that more constraints will be violated.

Note that we are not able to compare the results optimizing our flexibility metric to the metrics discussed in Section 3.2, since those metrics do not provide us with an interval schedule.

4.3.1 Setup of Experiments

Task scheduling problem instances are obtained from the PSPLIB instances (Kolisch and Sprecher, 1996; Kolisch et al., 1998), whereby only the network part of the instance is used—all resource constraints are ignored. Four sizes are available (30, 60, 90 and 120 tasks), we will concentrate on the set with 120 tasks. Since the instances contain no due date, a due date is imposed by taking the earliest start time schedule of the instance, and increasing the end time of this schedule by 10%.

The unexpected delays that we simulate are controlled by two parameters. The first parameter controls the number of tasks that are delayed. For each set of simulations, a fixed number of tasks are delayed. We delay between 10% and 50% of the tasks in the schedule, in steps of 10%, for a total of 5 different settings. The second parameter controls the amount of delay, relative to the length of the selected task. The delay is implemented by increasing the processing time of the selected task, without updating the schedule to take this increased time into account. This way, an accurate simulation can be performed of an unexpected delay, of which no a priori information is known. The processing time is increased by a factor between 10% and 100% in steps of 10%, and between 100% and 200% in steps of 25%, for a total of 14 different settings.

Having created an interval schedule for a problem instance, and having selected the tasks to be delayed, simulation is performed by dispatching the tasks of the instance in their assigned intervals. We choose to dispatch a task as soon as possible, within the constraints of its assigned interval. This accurately simulates, for instance, most production environments, in which personnel executing tasks is expected not to sit around idling when tasks are available to be executed. Since the tasks to be delayed are selected randomly, the interval schedule resulting from each combination of parameters is simulated 150 times for each setting of the delay parameters.

The performance of a simulated schedule execution is measured by determining the number of schedule violations occurring during execution. A violation occurs as soon as a task has to be started outside its execution interval. The size of the violation is equal to the distance of the upper bound of the execution interval, and the realized start time of the task.
### Table 4.1
Flexibility loss of the different flexibility distributions

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Flexibility</th>
<th>Flexibility loss</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>average</td>
<td>std.dev</td>
</tr>
<tr>
<td></td>
<td>average</td>
<td>std.dev</td>
</tr>
<tr>
<td>Maximal (3.30)</td>
<td>1347.8</td>
<td>334.2</td>
</tr>
<tr>
<td>Equalized (4.1)</td>
<td>1281.2</td>
<td>305.9</td>
</tr>
<tr>
<td>Weighted, direct predecessors (4.4)</td>
<td>1192.1</td>
<td>277.3</td>
</tr>
<tr>
<td>Weighted, discounted (φ = 5, 4.5)</td>
<td>1091.2</td>
<td>262.5</td>
</tr>
<tr>
<td>Weighted, all predecessors (4.3)</td>
<td>1045.1</td>
<td>251.2</td>
</tr>
<tr>
<td>Weighted, direct successors (4.7)</td>
<td>1221.8</td>
<td>279.7</td>
</tr>
<tr>
<td>Weighted, all successors (4.6)</td>
<td>1134.0</td>
<td>255.7</td>
</tr>
</tbody>
</table>

### Example 4.4
Consider for example two tasks $X_1$ and $X_2$, with $l_1 = 2$, $l_2 = 2$ and $X_1 \ll X_2$. If $\sigma_i(X_1) = [0, 1]$ and $\sigma_i(X_2) = [3, 4]$, we will start $X_1$ at time point 0. If $X_1$ is now delayed, such that $l'_1 = 5$, $X_2$ cannot be started in $\sigma_i(X_2)$. We start $X_2$ at the earliest possible time, which is directly after $X_1$ ends at time point 5. This is counted as a violation, of size $5 - 4 = 1$, since the upper bound of $\sigma(X_2)$ is 4.

### 4.3.2 Comparing Flexibility Distributions

We start with an overall comparison of the performance of the different flexibility distributions proposed, averaged over the full range of delay parameters on all instances, as discussed in the previous section. The different flexibility distributions proposed in Section 4.2 are compared with the one resulting from the flexibility metric introduced in Theorem 3.1, having maximal flexibility. Since there may be many different distributions maximizing the flexibility for a certain STN, we pick one which also maximizes Equation 4.1. This means that the maximally flexible distribution we use is as equalized as possible, such that we do not give unfair disadvantage to this distribution.

In Table 4.1, the resulting average flexibility is shown for each of the flexibility distributions, as well as the loss, when comparing the flexibility of each distribution to the maximal flexibility. We see that the price we have to pay in terms of flexibility loss to equalize flexibility is not very high: 95% of the available flexibility is retained. If we modify the equal distribution by applying a weight function $w$ however, we see that the more $w$ deviates from an equal distribution, the more flexibility we lose.

In Table 4.2, the average performance over all delay parameters is summarized for the different flexibility distributions, in terms of the number of violations, and the tardiness relative to the (undelayed) makespan. Here, we see that equalizing the flexibility gives an advantage in terms of the number of violations, even at the price of some flexibility loss. From this we can thus
4.3. Experiments

Table 4.2
Average performance for the different flexibility distributions

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Violations</th>
<th>Tardiness, %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>average</td>
<td>std.dev</td>
</tr>
<tr>
<td>Maximal (3.30)</td>
<td>42.9</td>
<td>16.9</td>
</tr>
<tr>
<td>Equalized (4.1)</td>
<td>37.0</td>
<td>(−13.8%)</td>
</tr>
<tr>
<td>Weighted, direct predecessors (4.4)</td>
<td>42.9</td>
<td>(0.0%)</td>
</tr>
<tr>
<td>Weighted, discounted ($\varphi = 5, 4.5$)</td>
<td>50.1</td>
<td>(+16.8%)</td>
</tr>
<tr>
<td>Weighted, all predecessors (4.3)</td>
<td>53.6</td>
<td>(+24.9%)</td>
</tr>
<tr>
<td>Weighted, direct successors (4.7)</td>
<td>35.3</td>
<td>(−17.7%)</td>
</tr>
<tr>
<td>Weighted, all successors (4.6)</td>
<td>33.9</td>
<td>(−21.0%)</td>
</tr>
</tbody>
</table>

conclude that sacrificing some flexibility can indeed improve the performance of schedule execution.

For the weighted flexibility distributions, this is not the case everywhere, contrary to our expectations. Weight functions distributing more flexibility to tasks with more predecessors (Equations 4.3, 4.4 and 4.5) lead to a higher number of violations, on average, but to a slight decrease in the average tardiness (i.e., the finish time of the latest task, as compared to the undelayed schedule). It is easy to understand why this happens: all these distributions, to a greater or lesser extent, concentrate flexibility in tasks with many predecessors. Tasks at the end of the schedule of course have more predecessors than tasks at the beginning, so this leads to a concentration of flexibility at the end of the schedule. A delay occurring anywhere in the schedule can thus be more easily absorbed by the tasks at the end of the schedule, but this comes at the price of more violations, for the tasks between the delayed task and the end of the schedule.

For the weight functions distributing more flexibility to tasks with more successors (Equations 4.6 and 4.7) we see the reverse, for the exact same reason. Tasks near or at the end of the schedule will have lower flexibility, explaining the slightly higher average tardiness. The number of violations is lower however, and from this we can conclude that it is better to distribute flexibility in such a way that delays can be compensated for immediately, before they spread to other tasks.

These numbers are all averages however, taken over all delay parameter combinations (i.e., number of tasks delayed, amount of delay per task). In the next section we will more closely analyze the influence these parameters have for the various distributions.

4.3.3 Influence of Delay Parameters on Performance

So far, we have discussed the performance of interval schedules averaged over all delay parameters used. We will now examine the dependency of the performance of the various flexibility distributions, in terms of the number of violations and
Figure 4.1
Influence of delay length on violations, for a delay in 10% of the tasks.

the tardiness, on both the number of tasks delayed and the amount of delay inserted.

In Figure 4.1 the effect of the length of the delay (relative to the task affected) is shown, for a constant number (10%, relative to the total number of tasks) of delayed tasks. The same picture emerges as from the summarized data: equally distributed flexibility consistently outperforms interval schedules having maximal flexibility, and interval schedules where flexibility is concentrated on tasks with many predecessors (right part of the figure). It can also be seen that maximally flexible schedules are influenced more for smaller delay lengths, relative to the other distributions. A possible explanation is that these schedules have more tasks with no flexibility at all: a delay has to propagate through all such tasks, until a task with enough flexibility is reached to absorb it. For the distributions concentrating the flexibility on tasks with many successors (left part of the figure), we see that they perform almost identical to the equalized distribution. This suggests that the slight performance increase we observed in Table 4.2 occurs for a higher number of delayed tasks only.

In Figure 4.2 the effect of the number of delayed tasks (relative to the instance size) is shown, for a fixed amount of delay (10%, relative to the task affected). The effects are largely similar to the effect of the length of the delay. Again, maximally flexible schedules are less affected by larger percentages of tasks delayed; the possible explanation is the same, as a large percentage of small delays give similar effects as a small percentage of large delays. The number of violations increases more slowly for larger percentages of delayed tasks; a possible explanation is that, due to re-convergence in the task graph, the effects of additional delays have less impact on the number of violations: if one delay already causes a series of violations, an additional delay cannot cause more violations in that path. Interestingly enough, here, we also see (right figure) that distributions concentrating flexibility on tasks with many successors perform virtually identical to distributions with maximal flexibility. We must therefore
4.3. Experiments

**Figure 4.2**
Influence of the number of delays on violations, for a delay length of 10%.

**Figure 4.3**
Influence of delay length on tardiness, for a delay in 10% of the tasks.

**Figure 4.4**
Influence of the number of delays on tardiness, for a delay length of 10%.
conclude that the advantage for these distributions we see in Table 4.2 occurs when both the number of tasks delayed and the size of the delay is large.

Next to the number of violations during execution, the tardiness is also of importance. In Figures 4.3 and 4.4, the dependence of the tardiness on the different delay parameters is shown. The results closely follow the summarized data in Table 4.2, and the relative performance of the distributions is not dependent on the delay parameters.

4.3.4 Influence of Flexibility on Performance

The discussion in the previous section centers on the relationship between the chosen distribution and the performance, in terms of violations and tardiness. It already shows that, sometimes, sacrificing some flexibility compared to the maximum achievable can give performance improvements for these criteria. Now, we will analyze this in some more detail by comparing the relative flexibility of certain instances to the relative performance on that instance, in terms of the number of violations. To be exact, 60 instances are selected at random, and the flexibility loss using the various distribution methods is computed, compared to the maximum achievable using Equation 3.30. The flexibility loss is plotted on the horizontal axis, and on the vertical axis the performance is shown, also relative to the performance, for a representative setting of the delay parameters (30% of the tasks are delayed by 30%, other settings yield comparable results). The data is separated across multiple graphs, to improve legibility.

In Figure 4.5 the data for equalized flexibility is shown, as well as the data for distributions based on the number of predecessors. For both the equalized distribution as well as the distribution based on the number of direct predecessors, the graph shows a correlation between the loss of flexibility and the performance. For the distribution based on the total number of predecessors, this correlation is almost indiscernible.

At first glance, this correlation appears to be in the wrong direction: the less flexibility remains, the higher the performance improvement, as compared to the maximally flexible schedule for the same instance. If we examine the basic idea behind the distribution of flexibility, this result does make sense after all: if an instance has a lot of parallel sections, the maximally flexible schedule will have more concentrations of flexibility, as demonstrated in Example 4.3. These concentrations are distributed when using the equalized objective function, and this will result in a relatively large difference in flexibility, but also in improved performance. In other words: the more a schedule can be changed using our equalized flexibility distribution, compared to a maximally flexible schedule, the higher the performance increase will be.

We do see a decrease in performance for the distributions based on the number of predecessors, compared to equalized flexibility distribution; this was also reflected in the previous section. The distributions based on the number of successors are shown in Figure 4.6, and here we also see the slight increase in performance, compared to the equalized distribution. For both distributions, the correlation between flexibility loss and performance holds. Here, we also see
4.3. Experiments

Figure 4.5
Relative number of violations versus loss of flexibility, for 30% delay in 30% of the tasks, for equalized flexibility and distributions based on predecessors.

Figure 4.6
Relative number of violations versus loss of flexibility, for 30% delay in 30% of the tasks, for equalized flexibility and distributions based on successors.
Figure 4.7
Violations versus tardiness, for 30% delay in 30% of the tasks, for equalized and maximal flexibility, and for distributions based on predecessors.

Figure 4.8
Violations versus tardiness, for 30% delay in 30% of the tasks, for equalized and maximal flexibility, and for distributions based on successors.
that deviating too far from the equalized distribution results in larger flexibility losses, which eventually begin to impact the performance, as seen in the results for the distribution based on the total number of successors.

### 4.3.5 Violations Versus Tardiness

As can already be seen from the data in Table 4.2, there is a relationship between the number of violations and the tardiness of a schedule, which can be noted especially in the distributions based on the number of predecessors. If flexibility is concentrated at the tasks at the end of the schedule, there is more slack available in this portion to compensate delays that would otherwise lead to tardiness. But, this also leads to more violations in the first part of the schedule. Conversely, a more equalized distribution avoids these violations, at the cost of tardiness.

In Figure 4.7 this trade-off is visualised for 60 instances simulated using 30% delay in 30% of the tasks. The difference between the schedules with equalized flexibility and the two weighted distributions based on the number of predecessors is very clear: an equalized schedule optimizes towards a low number of violations, a schedule weighted using the total number of predecessors optimizes towards low tardiness, and a schedule weighted using the number of direct predecessors compromises between the two. This graph shows again that maximizing the flexibility offers the worst performance of the differing flexibility distribution: it neither scores good on tardiness nor on the number of violations.

Next, we look at the two distributions based on the number of successors, these are also compared with the equalized flexibility distribution and the distribution with maximal flexibility, in Figure 4.8. Irregardless of the fact that, using a distribution based on the number of successors, less flexibility is concentrated at the end of the schedule, the tardiness does not increase a lot. The distribution based on the number of direct successors offers a slight increase in terms of the number of violations, with (almost) equal tardiness. Only when we deviate even more, by basing the distribution on the total number of successors, we indeed see a slight increase in the tardiness. This effect is visible even more in Table 4.2, this is caused by the fact that delay parameters causing more extreme delays are represented here as well.

The likely cause for the difference between Figures 4.7 and 4.8 lies in the different focus of the two distribution types. If we concentrate flexibility at the end of the schedule, we manage to lower tardiness, but the consequence is that more violations occur before delays are absorbed by the available flexibility. The other distributions, focussing on the number of successors, manage to decrease the number of violations slightly. While there might be less flexibility near the end of the schedule, the higher risk of tardiness is compensated by the lower propagation of delay.

### 4.3.6 Flexibility guarantees

A downside to the approach discussed so far is that all methods based on the objective function from Equation 4.1 focus on minimizing flexibility loss differences,
without taking into account the absolute value of the resulting flexibilities.

**Example 4.5** Consider two time point variables $X_a$ and $X_b$, with $\text{lst}(X_a) - \text{est}(X_a) = 4$ and $\text{lst}(X_b) - \text{est}(X_b) = 2$. Imagine that, due to the surrounding network structure, applying the linear programming method presented in the previous chapter combined with the objective function from Equation 4.1, both $X_a$ and $X_b$ lose two units of flexibility. Both time points are treated equally in terms of flexibility loss, but $X_a$ has two units of flexibility remaining, whereas $X_b$ has no flexibility at all.

As can be seen from this example, the method based on Equation 4.1 might result in tasks receiving no flexibility at all. A method to avoid this problem is to add constraints of the form

$$X_i^+ - X_i^- \geq f$$

(4.9)
to all time point variables, which forces the allocation of at least $f$ units of flexibility to each of them. Note however that large values of $f$ might make the problem infeasible. A simple solution is to adapt our linear program, again, using a new objective function that is able to find the largest value of $f$ we can use:

$$\max \min_{X_i \in X} |(X_i^+ - X_i^-)|.$$  \hspace{2cm} (4.10)

This objective function finds an interval schedule in which the smallest interval is as large as possible. It is easy to see that the problem remains feasible if we use this value for $f$, forcing all intervals to be at least this size: the solution found when using the objective function in Equation 4.10 is one such solution for which these constraints all hold. There might however still be many different solutions for this new problem. To find the most suitable one, we can use the original objective function from Equation 4.2 combined with any of the weight functions, to distribute the remaining flexibility over the time point variables: first, we use a linear program to find the highest lower bound on the flexibility per task, and then this bound is enforced using additional constraints. The rest of the flexibility can then be distributed using any of the other methods. In this case, we choose to equalize the flexibility, and we compare the results with and without the lower bound enforcing a minimal amount of flexibility.

In Figure 4.9 we see that this approach works extremely well if the amount of delay is small. This is to be expected: small delays can be absorbed immediately, if the minimal amount of flexibility is high enough. The right part of the figure shows that, for a small amount of delay (10%, in this case), the number of delayed tasks has almost no influence on the performance at all. If delays are larger, there is a small performance penalty if a minimal amount of flexibility is enforced per task—the likely cause is a smaller total flexibility, over the entire instance.

The same trend is shown for the effect on the tardiness, in Figure 4.10. It is less apparent, since the tardiness is relatively low already, for small delay lengths (left figure), but on the right it is clear that there is a large influence on the tardiness if delays are small enough, independent of the amount of tasks delayed.

4.4 Conclusion and discussion

This chapter investigated the second research question, which asks how available flexibility can be best distributed over the tasks in a schedule, to achieve high robustness. The need for this is first illustrated using examples, and different distributions are proposed and analyzed in an experimental setting. These experiments show that some, but not all our intuitions behind these distributions are valid.

Our idea of equalizing the flexibility distribution over the tasks in the schedule is shown to have positive influence on both the number of violations, i.e., the number of tasks executing outside their assigned interval, as well as on the tardiness of the schedule, i.e., how much the deadline of the original schedule
was exceeded due to delay. Our idea to further improve the robustness by allocating more flexibility to certain tasks, based on either the number of predecessors or the number of successors, showed that concentrating flexibility based on predecessors improves the performance with regard to the tardiness, at the cost of a larger number of violations. Concentrating flexibility based on the number of successors however is shown to decrease the number of violations with little effect on the tardiness. The likely explanation for the first effect is that flexibility is concentrated at the end of the schedule, and as such any delays can be absorbed before the final task of the schedule is executed, at the cost of more propagation through the schedule. The likely explanation for the fact that tardiness does not suffer if flexibility is concentrated based on the number of successors is that it reduces violations, and thus the spread of delay over the schedule, before the final task is executed.

If the delays are known to be small, the best performance is obtained however by guaranteeing that a certain minimal amount of flexibility is allocated to every task in the schedule. We propose a method to maximize this enforced minimal amount of flexibility, and the experiments show that this distribution performs very good, with almost no violations at all, if the delays are reasonably small.

4.4.1 Discussion

The idea of robustness discussed in this chapter is important in the context of schedule execution at NedTrain, since here, we have highly uncertain task durations, combined with the use of shared resources by independently operating teams. If the schedules used are not robust in the face of the uncertain execution times, subsequent tasks in the schedule will be delayed. Resources can be the cause of such a delay propagation, as these might not be freed in time to start the next task. They can however also be a reason to want to avoid delays: if some type of (external) commitment has been made for a resource needed for a certain task, like the hiring of special machinery or extra personnel, delays can lead to penalties. The independently operating teams at NedTrain are another reason to strive to avoid the spread of delay: if unforeseen circumstances lead to extra work for one team, it is desirable if this has no effect on the schedule for the other teams, since this would lessen their independence.

This chapter, as well as the previous one, focused exclusively on the level of the STN and the properties of, and effects on execution performance of the different interval schedules that can be constructed for those STNs. The original problem that we are trying to solve, however, is the more complex RCPSP, which is reduced using precedence constraint posting to an STN as a convenient and flexible solution representation. Important to note is that a single RCPSP instance can be reduced to many different STNs—this is an important direction for further research: instead of maximizing the flexibility of a single STN solution to an RCPSP instance, it might be a much better strategy to modify the choices that are made in the precedence constraint posting process such that a different STN is obtained that admits an interval schedule with more flexibility.
In solving the RCPSP in a flexible way, the previous two chapters explored the temporal flexibility of simple temporal networks, which are used to represent a set of solutions to an RCPSP instance. Chapter 3 explored the intuition behind the concept of temporal flexibility, in an effort to answer research question 1. We have discovered that existing methods do not conform to these expectations and we introduced a new method to measure the flexibility of a simple temporal network. The second research question represents our aim to use the flexibility present in a solution to create robust solutions for the scheduling problem at NedTrain, and this question is explored in Chapter 4, where various ways to distribute flexibility over a simple temporal problem are tested for their performance in this regard.

A shared feature of the previous chapters (and research questions 1 and 2) is that they did not focus on the RCPSP directly: both are based on a flexibility measure for the STN. An RCPSP instance was converted to an STN, and using our flexibility metric, this in turn was converted into an interval schedule, yielding an efficient and flexible method of executing a portfolio of temporal solutions for the original RCPSP instance. This chapter will focus on the third research question, and explore how we can extend the concept of temporal flexibility to include a form of sequential flexibility. For this, we need to move a level up, to the conversion of an RCPSP instance to an STN.

We start by noting that the flexibility of the solution of an RCPSP instance, in the form of an STN, is determined by the temporal constraints in that STN, and if this STN is the result of a reduction from an RCPSP instance obtained using constraint posting techniques, we can make a distinction between two types of temporal constraints: one part of the constraints stems from temporal constraints in the original RCPSP instance, such as due times and precedence constraints between tasks, and the other part of the constraints is added by the reduction procedure to eliminate resource contention. The second type of constraint usually originates from heuristic solution procedures. Different types of heuristics can be employed, some even with different parameters. This means that there are many different choices, and hence many different sets of
constraints, which will all result in an _STN_ that contains solutions\(^1\) for the same associated _RCPSP_ instance. Some of these choices will be more arbitrary than others, based on the information the heuristic is able to provide. To briefly recap, constraint posting is an iterative process, consisting of the following steps:

- Compute the resource usage profile of the current _STN_, and identify resource contention peaks, i.e., places in the current solution where the combined resource usage of the executing tasks exceeds the capacity of that resource.

- Select a pair of tasks contributing to said resource contention peak, select an ordering for these two tasks, and add (post) a precedence constraint to the _STN_ enforcing this ordering.

These steps are repeated until the resource profile as computed in the first step no longer contains any resource contention peaks.

One obvious research avenue is to tune or develop heuristics that are able to make these decisions in such a fashion that the flexibility of the resulting _STN_ is as high as possible. However, in this chapter a different approach is proposed: instead of making these choices and arriving at an _STN_ where these constraints are _fixed_, we present an approach in which these choices are _postponed_—this results in a schedule in which these choices can be made during _execution time_, giving the agent executing the schedule the flexibility to re-order the tasks before dispatching them. To distinguish this from the flexibility discussed in the previous chapters, we propose to call the earlier form of flexibility _temporal_ flexibility, as this form is concerned with the exact time at which the tasks can start, and to call this form of flexibility _sequential_ flexibility, to indicate that the flexibility lies in the possibility to re-order the sequence of tasks.

Being able to (partially) determine the order of tasks at the time of execution fits well within the goal of wanting to have a schedule that is flexible in the face of uncertainty. Imagine that a maintenance engineer at NedTrain has to start with a certain task, which he has to perform together with a colleague with certain special skills. If this colleague is unavailable, because he is still working on a prior, delayed, task, it is very natural for the maintenance engineer to decide to start with another task first, for which the second engineer is not needed. The effects of swapping the execution order of these tasks will however propagate through the schedule. In this chapter we will formalize this concept of flexibility in such a way that it can be encoded in a solution to the _RCPSP_, to be used during execution of the schedule to combat the spread of delays due to uncertainty. The method we use, in short, consists of creating _grouped_ tasks at certain points in the constraint posting procedure, instead of adding precedence constraints. Such a grouped task is a placeholder, representing any ordering of the tasks contained therein. At execution time, the final order can be determined.

---

\(^1\)Note that we are only interested in _STNs_ in which _all_ solutions are also solutions to the original _RCPSP_. There are also many _STNs_ in which only part of the solutions are solutions to the original _RCPSP_, other solutions are resource-infeasible.
5.1 Motivation

After this introduction, this chapter will continue with a slightly more detailed motivation of why and how we want to achieve sequential flexibility. Having motivated the why of sequential flexibility, we first describe the existing method of constraint posting, on which our idea is based. Then, task grouping itself is described, the actual method we propose to achieve sequential flexibility, and we follow this formal description by a section containing experiments to gain further insight into how task grouping achieves the desired flexibility. The chapter is ended with a conclusion and discussion.

5.1 Motivation

Just as in the previous chapters, we continue to aim at building schedules that are prepared for possible disruptions to occur. Instead of having a fixed start time assignment \( \sigma : T \to \mathbb{R} \), we want to have a set \( S \) of schedules with which we are able to react to unforeseen circumstances during execution—if a delay in a task would cause the current schedule \( \sigma \in S \) to violate some of the constraints, we are then able to find a different \( \sigma' \in S \) that performs better under the new circumstances. In the previous chapters we introduced the concept of interval schedules, which can be seen as a restricted form of such a set of schedules \( S \) for a certain \textsc{rcpsp} problem: they only allow schedules \( \sigma \) in which the order of tasks is an extension of the partial ordering as defined by the \textsc{stn} reduction from the original \textsc{rcpsp} problem, i.e., only those schedules adhering to the extra precedence constraints added by the constraint posting procedure are admitted.

To lift this restriction, this chapter examines the method by which the \textsc{rcpsp} is converted into an \textsc{stn}. We will base our method on the class of precedence constraint posting methods, as described in Chapter 2. These methods are well-suited in the context of our problem: since they are based on heuristics, the are able to quickly provide a solution, and the solution is in the form of an \textsc{stn}, which is a flexible representation of multiple, fixed-time solutions.

A central component of constraint posting methods is the resolution of resource conflicts between two tasks. If a resource conflict is detected between two tasks \( t \) and \( t' \), constraint posting tries to remove this conflict by adding (posting) an additional precedence constraint between \( t \) and \( t' \), adding either \( t < t' \) or \( t' < t \). Selection of the tasks \( t \) and \( t' \) between which a constraint is posted, as well as the direction of this constraint, is based on heuristics. Sometimes, however, the heuristic to decide on the direction of the constraint does not indicate a very strong preference for one direction or the other, and it is not clear which choice is the best one to make. After suitable pairs of tasks \( t \) and \( t' \) are selected, our approach is therefore to group these tasks together without specifying the exact order (\( t \) before or after \( t' \)). In this way, the resource conflict is (partially) solved, since the tasks are no longer allowed to execute concurrently, but the order is not specified, since the group is constructed in such a way that either order is still possible. Note that this principle can be applied recursively: one, or both, of the tasks \( t \) and \( t' \) being grouped can be a grouped task itself. Note that the original grouping is not kept in such a case, i.e., groups with more than
two tasks do not contain sub-groups or any other form of hierarchy. This way, the final result can contain groups with an arbitrary number of tasks. At execution time we can pick any ordering we like for the tasks within these groups—in this way, the solution can be seen to represent a large set of possible schedules for the original rcpsp instance.

To summarize, an agent executing a classical non-grouped schedule only has two available options to handle delayed tasks: adhering to the schedule and delaying successive tasks as well, or generating a new, different schedule. Our approach construct a schedule with grouped tasks, which allows a third option: re-ordering the tasks in a task group such that execution of this group can commence with a different task without changing the global characteristics of the schedule. In this chapter we will explore the possibilities of this grouping approach to absorb disruptions that might happen during execution of the schedule.

5.2 Constraint posting

Before discussing task grouping, we will briefly recap the resource-constrained project scheduling problem (rcpssp), which is the problem we are solving, and how precedence constraint posting works.

The rcpssp consists of the following components:

- A set $T$ with $n$ tasks, where task $t_i \in T$ has a length $l_i \in \mathbb{R}$.
- A precedence relation $<$ inducing a partial order on $T$; the intuitive interpretation of $t_i < t_j$ is that $t_i$ has to be finished before $t_j$ can start.
- A set $R$ of $m$ renewable resources, where each resource $r_k \in R$ has integer capacity $\text{cap}(r_k) \geq 1$.
- For each $r_k \in R$ and $t_i \in T$, $\text{req}(t_i, r_k) \in \mathbb{N}$ specifies the number of units task $t_i$ needs of resource $r_k$ in order to be executed.

In addition to these constraints, we also have a release time $s_i \in \mathbb{R}$, and a due time $d_i \in \mathbb{R}$, which are in this case used to model when a train enters and leaves the workshop. The classical goal for the rcpssp is to find a start time assignment (schedule) $\sigma : T \rightarrow \mathbb{R}$ such that both the precedence relations and the resource constraints are satisfied. In our case this is also the final goal, but we would like there to be an intermediate step, in the form of a set of schedules $S$ from which we can (efficiently) derive a multitude of final solutions $\sigma$.

Example 5.1 As a running example throughout this chapter, we will use a simple rcpssp instance containing five tasks using two different resources, $r_a$ and $r_b$. The capacities of the resources are $\text{cap}(r_a) = 2$ and $\text{cap}(r_b) = 1$,
respectively. The length and resource usage of the tasks are shown in the following table:

<table>
<thead>
<tr>
<th>task</th>
<th>$t_1$</th>
<th>$t_2$</th>
<th>$t_3$</th>
<th>$t_4$</th>
<th>$t_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>23</td>
<td>25</td>
<td>23</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>use of $r_a$</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>use of $r_b$</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

The five tasks share a common release time of 0, and a due time of 75. In addition, there are three precedence constraints between the tasks: $t_1 < t_3$, $t_2 < t_4$ and $t_2 < t_5$. The temporal information from this example is combined in the following diagram:

In this diagram, the earliest starting times of the tasks are calculated based on the given precedence constraints. For example, since we have $t_1 < t_3$, and $l_1 = 23$, we can derive $est(t_3) = 23$, the earliest starting time of $t_3$. Instead of the latest starting time, the diagram displays the latest ending time of a task, since this makes the diagram easier to read.

Note that the location of the tasks along the lines indicating their earliest start time and latest end time is but one of many possible locations, and any combination of locations together represents a schedule $\sigma$. These schedules, however, do not necessarily present a resource-feasible solution. While some of the start times satisfying the temporal constraints will yield a resource-feasible solutions, many others do not. In fact, the shown assignment is resource-infeasible, since $t_3, t_4$ and $t_5$ execute concurrently, and $req(t_3, r_a) + req(t_4, r_a) + req(t_5, r_a) = 3 > 2 = cap(r_a)$.

To find schedules that satisfy both the temporal and the resource constraints, some techniques have already been discussed in Chapter 2. The technique most suited to our purposes is precedence constraint posting, such as described by Cesta et al. (1998): due to its heuristic nature, it is able to solve large problems very quickly, and it is able to represent solutions in a flexible way, because it is essentially a reduction from the original RCPSP to an STN. This reduction works by adding precedence constraints to the original problem specification.
such that any start time assignment respecting the combined set of precedence constraints results in a resource-feasible schedule. This is a reduction to an STN in the sense that any solution to the STN is also a solution to the original RCPSP instance. However, the reduction is one-way: not all solutions to the RCPSP instance necessarily have a counterpart in the STN constructed by constraint posting.

In this section we will describe the existing constraint posting techniques upon which our task grouping method is based. The constraint posting technique starts with constructing a provisional schedule for the problem instance, taking into account only the temporal constraints. These constraints are represented using an STN. In such a representation, for each task \( t_i \) a time point variable \( X_i \) is introduced. Additionally, the time point variable \( X_0 \) is introduced as reference point (usually, \( X_0 = 0 \)). To represent the release and due time constraints, we add the constraints \( X_0 - X_i \leq -s_i \) and \( X_i - X_0 \leq d_i - l_i \), respectively. These constraints ensure that the start time of \( t_i \) is at or after the release time, and that the end time of \( t_i \) (i.e., the start time represented by \( X_i \), plus the length \( l_i \) of the task) is at or before the due time, respectively. To represent a precedence constraint \( t_i < t_j \), we add the constraint \( X_i - X_j \leq -l_i \), to ensure that the start time of \( t_j \) is at or after the end time of \( t_i \).

**Example 5.2** Applying this transformation to our running example, the release time and due time constraints are represented by

\[
\begin{align*}
X_1 - X_0 & \leq 75 - 23 = 52 \\
X_2 - X_0 & \leq 75 - 25 = 50 \\
X_3 - X_0 & \leq 75 - 23 = 52 \\
X_4 - X_0 & \leq 75 - 20 = 55 \\
X_5 - X_0 & \leq 75 - 20 = 55 \\
X_0 - X_1 & \leq 0 \\
X_0 - X_2 & \leq 0 \\
X_0 - X_3 & \leq 0 \\
X_0 - X_4 & \leq 0 \\
X_0 - X_5 & \leq 0,
\end{align*}
\]

and the precedence constraints in the problem are represented by

\[
\begin{align*}
X_1 - X_3 & \leq -23 \\
X_2 - X_4 & \leq -25 \\
X_2 - X_5 & \leq -25.
\end{align*}
\]

A solution to such an STN can be produced efficiently, as outlined in Chapter 2. In particular, like is done in Cesta et al. (1998), one can obtain the earliest start time solution that satisfies all the temporal and precedence constraints of the RCPSP instance in polynomial time. Using such an initial schedule \( \sigma \), the constraint posting technique aims to satisfy the resource constraints as well, by
posting additional precedence constraints. To do so, for every resource \( r_k \) its resource usage profile with respect to the current schedule \( \sigma \) is computed. Such a profile indicates for every time point the total amount of resource \( r_k \) required, if each task \( t_i \) is started at time \( \sigma(X_i) \).

More exactly, given a resource \( r_k \) and the set \( T_k = \{ t_j \in T \mid \text{req}(t_j, r_k) > 0 \} \) of tasks using this resource, the demand of this resource \( r_k \) at time \( \tau \) is defined as

\[
D_k(\tau) = \sum_{t_j \in T_k} \delta_j(\tau) \cdot \text{req}(t_j, r_k),
\]

with

\[
\delta_j(\tau) = \begin{cases} 
1 & \text{when } \sigma(X_j) \leq \tau < \sigma(X_j) + l_j \\
0 & \text{otherwise.}
\end{cases}
\]

This equation sums the resource usage for \( r_k \) for all tasks executing at time point \( \tau \). Usually, \( D_k \) is only computed for the time points \( \tau \) corresponding to the start times of the tasks \( t_i \in T \) of the current solution. This is more efficient, because we want to compute the resource profile to know at which time points a resource becomes oversubscribed—this can only happen at the time points at which a task is started: resource use is constant over the duration of a task, meaning that the start time points are the only time points at which resource oversubscription can occur.

**Example 5.3** Applying this technique to the earliest start time solution of our example (this is the solution represented in the diagram in Example 5.1), we arrive at the following resource profiles, for \( r_a \) and \( r_b \):

![Diagram showing resource profiles for \( r_a \) and \( r_b \).](image)

The capacity for each of the resources is indicated by the dashed lines, ---- for \( r_a \), and --- for \( r_b \). Note that, for clarity, the resource profile is shown as a line, whereas in fact it is only computed at the start point of the tasks using the resources. It can be seen that the resource capacity of \( r_a \) is exceeded using this schedule.

Now the constraint posting method focuses on the peaks in this profile. Such a peak is a point in time where the use of a resource \( r_k \) increases, and is above its available capacity \( \text{cap}(r_k) \). Formally, the set of tasks forming a peak on \( r_k \) at time point \( \tau \) is defined as

\[
C_{k,\tau} = \{ t_j \in T_k \mid \delta_j(\tau) = 1 \},
\]
where
\[ D_k(\tau) > \text{cap}(r_k). \]  \hspace{1cm} (5.4)

The peak \( C_{k,\tau} \) represents a resource constraint violation, or a conflict, on resource \( r_k \) at time point \( \tau \), and contains all tasks contributing to this resource constraint violation.

**Example 5.4** Continuing the previous example, there are two peaks using the current schedule: what looks like one violation of \( r_a \) are actually two violations, first by \( t_3 \) combined with \( t_2 \) starting at time 23, \( C_{r_a,23} = \{t_2, t_3\} \), and then by \( t_3, t_4 \) and \( t_5 \) starting at time 25, \( C_{r_a,25} = \{t_5, t_4, t_5\} \).

A resource constraint violation can be lowered by posting a precedence constraint between a pair of tasks \( t_i, t_j \in C_{k,\tau} \). As a consequence of posting this constraint, we ensure that the resource peak causing the violation is lowered and spread. Note that the peak is not necessarily solved, only reduced. Such a set of tasks \( \{t_i, t_j\} \) is called a (pairwise) conflict. Now, we define

\[ d(t_i, t_j) = \text{lst}(t_j) - (\text{est}(t_i) + l_i) \]  \hspace{1cm} (5.5)

to represent the maximally allowed temporal distance between two tasks \( t_i \) and \( t_j \) (also known as the slack) in the current problem specification. In other words, \( d(t_i, t_j) \) is the temporal distance between \( t_i \) and \( t_j \) if we start \( t_i \) as early as possible, and \( t_j \) as late as possible. Remark that, usually, \( d(t_i, t_j) \neq d(t_j, t_i) \). For a given set \( \{t_i, t_j\} \), we can define four possible conflict cases (Oddi and Smith, 1997):

1. \( d(t_i, t_j) < 0 \land d(t_j, t_i) < 0 \),
2. \( d(t_i, t_j) < 0 \land d(t_j, t_i) \geq 0 \),
3. \( d(t_j, t_i) < 0 \land d(t_i, t_j) \geq 0 \), and
4. \( d(t_i, t_j) \geq 0 \land d(t_j, t_i) \geq 0 \).

The first case is **pairwise unresolvable**: \( t_i \) cannot finish before \( t_j \) must start, and \( t_j \) cannot finish before \( t_i \) must start; this means that this conflict cannot be resolved. The second and third conditions are **pairwise uniquely resolvable**: there is only one feasible ordering, the decision is unconditional. The last condition is **pairwise resolvable**: both orderings are possible, and a choice needs to be made.

**Example 5.5** To flatten the peaks listed in Example 5.4, we pick two tasks from both sets, and post a constraint between them. In this case we will choose to add the constraints \( t_2 < t_3 \) and \( t_4 < t_5 \). Note that both pairs are of the fourth type, both orderings are possible. This will result in a new temporal network, yielding the following solution diagram (in the format of Example 5.1):
In this solution, no peaks exceeding the resource capacity are present, so additional constraints do not have to be posted. Note that this is caused by the simplicity of this example, usually a peak needs more than a single constraint to be eliminated completely.

Summarizing, the structure of a constraint posting method generally has the following form:

1. Using a provisional schedule $\sigma$, compute the resource usage profile over time and select a resource peak violating the resource capacity.

2. Select two (partially) concurrent tasks $t_i, t_j$ contributing to the selected peak.

3. Decide on adding a precedence constraint between these tasks: $t_i < t_j$ or $t_j < t_i$.

4. Add this precedence constraint, and solve the underlying $\mathcal{S\mathcal{T}}\mathcal{N}$ to generate a new schedule.

### 5.2.1 Heuristics in constraint posting

At three different points in the outline above, choices must be made:

1. a peak must be chosen,

2. a pair of (partially) concurrent tasks within this peak must be selected, and

3. a direction for the precedence constraint between these tasks must be decided.

To make these choices, heuristics are employed.

To select a peak, we look at the overcommitment each peak represents. The peak that exceeds the capacity of a resource by the largest amount is chosen first. Intuitively, this points the algorithm towards the areas where resource contention
is largest. In many algorithms (e.g., Cesta et al. (1998)), this step is omitted, and
the algorithm selects a task pair from all peaks in the current solution. We opted
to limit task pair selection to a single peak for two reasons. First, this causes
the algorithm to give priority to the largest overcommitment on a resource. The
intuition behind this is that such a peak usually requires a lot of constraints to
be entirely flattened—the more constraints are added to the problem the less
flexibility remains, and the higher the chance that the problem cannot be solved
anymore. A second reason is that the number of task pairs grows rapidly with
the number of tasks from which the algorithm can choose, so limiting ourselves
to the tasks in a single peak gives a performance increase.

Within a peak, a conflict \( \{ t_i, t_j \} \), which is a pair of (partially) concurrent
activities, must be chosen. Recall the four different conflict types described
before. If all conflicts in this peak are of the first type, we can immediately
conclude that the instance is no longer solvable without backtracking. If there
are conflicts of the second or third type (i.e., conflicts that can only be resolved
in one way), we select the conflict for which

\[
\omega_{\text{res}}(t_i, t_j) = \min(d(t_i, t_j), d(t_j, t_i))
\]

is lowest (Cesta et al., 1998). The lower this value, the closer the two tasks are
to being forced into a resource-consistent state. In other words, the temporal
flexibility we lose by posting a constraint between these two tasks is the lowest.

For conflicts of the fourth type (i.e., conflicts that can be resolved both ways),
we cannot completely rely on this measure to select a conflict to resolve.

**Example 5.6** Consider two conflicts \( \{ t_a, t_b \} \) and \( \{ t_c, t_d \} \), for which the tem-
poral distance \( d(t_a, t_b) = d(t_b, t_a) = 4 \), but \( d(t_c, t_d) = 3 \) and \( d(t_d, t_c) = 100 \). Here, \( \omega_{\text{res}}(t_a, t_b) = 4 \) and \( \omega_{\text{res}}(t_c, t_d) = 3 \). However, it is clear that
\( \{ t_a, t_b \} \) has less temporal flexibility in which it can be resolved than \( \{ t_c, t_d \} \):
\( d(t_d, t_c) = 100 \) indicates that posting \( t_d < t_c \) leaves a very large amount of
flexibility in the schedule.

Therefore, \( \omega_{\text{res}} \) is adapted to be biased towards conflicts that have similar
slack in both directions (Cheng and Smith, 1996):

\[
\omega'_{\text{res}}(t_i, t_j) = \frac{\min(d(t_i, t_j), d(t_j, t_i))}{\sqrt{S}}, \quad (5.6)
\]

with

\[
S = \begin{cases} 
\min(d(t_i, t_j), d(t_j, t_i)) & \text{for } \max(d(t_i, t_j), d(t_j, t_i)) \neq 0 \\
1 & \text{otherwise} 
\end{cases} \quad (5.7)
\]

**Example 5.7** For \( \{ t_a, t_b \} \), we have \( S = 1 \), resulting in \( \omega'_{\text{res}}(t_a, t_b) = 4 \). But,
for \( \{ t_c, t_d \} \), we have \( S = 3/100 \), resulting in \( \omega'_{\text{res}}(t_c, t_d) = 3/\sqrt{3/100} \approx 17.3 \).
This results in \( \{ t_a, t_b \} \) being selected for resolution before \( \{ t_c, t_d \} \).
As shown by the example, constraints that are close to being unresolvable in both directions are now selected for resolution first. If other constraints are posted first instead, we risk reducing the flexibility of this constraint even further, turning it into an unresolvable conflict. If this happens, the only way to arrive at a solution is to backtrack, which is highly undesirable. Our algorithm, like many other constraint posting algorithms, does not perform backtracking due to reasons of efficiency, and will simply give up if an unresolvable conflict is encountered.

The last step is to determine a direction for the constraint to be posted between the selected tasks. For \( \{ t_i, t_j \} \), we post \( t_i < t_j \) if \( d(t_i, t_j) > d(t_j, t_i) \), and \( t_j < t_i \) otherwise. The intuition for this decision is that we want to preserve the largest amount of temporal flexibility.

To summarize, the three decisions are made using the following criteria:

1. Select the peak with the largest amount of resource overcommitment.
2. Within this peak, select the conflict with the least amount of scheduling flexibility left.
3. Post a constraint preserving the largest amount of flexibility.

This way, the algorithm focuses on the most critical part of the problem, where it tries to make decisions keeping the amount of flexibility remaining as high as possible.

5.3 Task grouping

In the preceding section, the technique of constraint posting has been described, with which the \( \text{rcp}\text{sp} \) can be solved easily, up to the scale of the problems that are present at NedTrain. However, flexibility still remains low: while the technique of interval schedules can be used, as described in the previous chapter, to give the schedule some amount of temporal flexibility, the added dependency constraints impose a certain rigidity on the ordering of the tasks. Ideally, we would like to have sequential flexibility as well, such that we are not forced to use a certain ordering, decided in advance, at execution time—conditions might have changed, making another ordering more attractive.

Therefore, this section introduces an addition to the precedence constraint posting method, one by which we can include multiple execution orderings in a solution to an \( \text{rcp}\text{sp} \) problem instance. The method operates on the basis of the idea of task grouping: instead of posting a constraint between two tasks and enforcing an ordering, the two tasks are grouped together in an encapsulating group task, which reserves resources such that both execution orders remain feasible at execution time. The concept of grouping is recursive, meaning that such an encapsulating group task can take part in another grouping operation,
resulting in a group task containing three tasks, which can be executed in any order at execution time.

This section starts by describing the exact procedure by which tasks are grouped together, followed by a description of the procedure by which such a schedule can be executed, i.e., how this sequential flexibility can actually be used.

### 5.3.1 The grouping operation

We propose to extend the constraint posting technique by adding the operation of grouping two tasks together, forming an encapsulating group task, as an alternative to posting a precedence constraint to the schedule: instead of always adding such a constraint we offer the possibility to group two tasks $t_i$ and $t_j$ together in order to postpone the decision on their ordering to the time of execution of the schedule. The grouping $\text{Group}(t_i, t_j)$ of $t_i$ and $t_j$ results in a composite task $t_{i,j}$ where $t_i$ and $t_j$ are executed sequentially without specifying the exact order: either $t_i$ is executed first, and $t_j$ second, or execution starts with $t_j$, and $t_i$ is second. $\text{Group}(t_i, t_j)$ consists of the following steps:

1. Both $t_i$ and $t_j$ are removed from $T$, and the composite task $t_{i,j}$ is added to $T$ instead.

2. Release time, due time and duration of this group task $t_{i,j}$ are computed as follows:

\[
\begin{align*}
    s_{i,j} & = \max(s_i, s_j), \\
    d_{i,j} & = \min(d_i, d_j), \\
    l_{i,j} & = l_i + l_j.
\end{align*}
\]

For the release time, the maximum of the release times of $t_i$ and $t_j$ is taken. This ensures that $t_{i,j}$ can start with either $t_i$ or $t_j$. If the minimum were taken, we would be forced to commence with the earliest task of the two, during execution. In similar vein, the minimum of the due times $d_i$ and $d_j$ is taken. For the task length of $t_{i,j}$, the sum of $l_i$ and $l_j$ is used, since the two tasks are to be executed sequentially.

3. A similar reasoning is used behind the resource usage of the grouped task. Since the ordering of the two tasks is not known until execution time, the grouped task is given a resource profile that ensures that either task can run at any time within the time limits of the grouped task. Hence, for each resource $r_k$, we compute the resource usage of the group task to be equal to the maximum of the resource usage of the individual tasks:

\[
\text{req}(t_{i,j}, r_k) = \max\{\text{req}(t_i, r_k), \text{req}(t_j, r_k)\}.
\]

Note that this creates a worst-case resource profile, which ensures that enough resources are available to execute either $t_i$ or $t_j$ at any time during $t_{i,j}$. 
Lastly, when forming a grouped task, existing precedence constraints must be updated. All constraints of the form $t_k < t_l$ with $l = i$ or $l = j$ are replaced by $t_k < t_{i,j}$, and all constraints of the form $t_l < t_k$ with $l = i$ or $l = j$ are replaced by $t_{i,j} < t_k$. This means that all tasks that were constrained to take place before (respectively, after) either $t_i$ or $t_j$ are now constrained to take place before (respectively, after) $t_{i,j}$ instead.

These steps are formalized in Algorithm 5.1.

Algorithm 5.1: Operation to group tasks together.


def GROUP(t_i, t_j):
    T ← $T - \{t_i, t_j\}$
    $l_{i,j} ← l_i + l_j$
    $s_{i,j} ← \max(s_i, s_j)$
    $d_{i,j} ← \min(d_i, d_j)$
    for all the $r_k \in R$ do
        req($t_{i,j}, r_k$) ← $\max\{req(t_i, r_k), req(t_j, r_k)\}$
    end
    for all the $t_k < t_l : l = i \lor l = j$ do
        $C ← C - \{t_k < t_l\}$
        $C ← C \cup \{t_k < t_{i,j}\}$
    end
    for all the $t_l < t_k : l = i \lor l = j$ do
        $C ← C - \{t_l < t_k\}$
        $C ← C \cup \{t_{i,j} < t_k\}$
    end
    $T ← T \cup \{t_{i,j}\}$
end

To show that the grouping operation has no effect on the validity of the solution space, we need the following proposition:

**Proposition 5.1** Any temporally and resource-feasible start time assignment $\sigma$ for the tasks in a group task is a temporally and resource-feasible start time assignment for the tasks in the original problem as well.

**Proof.** To start, we note that since $l_{i,j} = l_i + l_j$, the tasks contained in the group can be executed sequentially during the time the group is scheduled for execution. For any of the tasks $t_l$ with $l \in \{i, j\}$ in the task group, we have

$$\sigma(t_l) \geq s_{i,j} = \max(s_i, s_j) \Rightarrow \sigma(t_l) \geq s_i \land \sigma(t_l) \geq s_j,$$

(5.12)

showing that the release time constraint is satisfied.

Similarly, we have

$$\sigma(t_l) + l_l \leq d_{i,j} = \min(d_i, d_j) \Rightarrow \sigma(t_l) + l_l \leq d_i \land \sigma(t_l) + l_l \leq d_j,$$

(5.13)
showing that the due time constraint is satisfied.

For the resource constraints, we know that $\text{req}(t_{i,j}, r_k)$ of capacity is available for resource $r_k$ during the execution of $t_{i,j}$, since $\sigma$ is a resource-feasible assignment. Since

$$\text{req}(t_{i,j}, r_k) = \max(\text{req}(t_i, r_k), \text{req}(t_j, r_k)) \geq \text{req}(t_i, r_k), \quad (5.14)$$

we know that there must also be enough capacity available to execute $t_l$ ($l \in \{i, j\}$).

Lastly, for the precedence constraints, for any of the tasks $t_l$ for which there is a constraint $t_k < t_l$ in the original problem, we have $t_k < t_{i,j}$, such that

$$\sigma(t_k) + l_k \leq \sigma(t_{i,j}) \leq \sigma(t_l), \quad (5.15)$$

showing that the precedence constraint is satisfied. Similarly, if we have $t_l < t_k$ in the original problem, we have $t_{i,j} < t_k$, such that

$$\sigma(t_l) + l_l \leq \sigma(t_{i,j}) + l_{i,j} \leq \sigma(t_k). \quad (5.16)$$

Now that we have specified how to group two tasks together, we need to be able to select two suitable tasks in an RCPSP on which to perform this grouping. For this, we must find two tasks that should be executed sequentially, due to resource constraints, and for which the exact execution order is not very important. This idea bears some resemblance to the stochastic procedure used by Oddi and Smith (1997): here, for tasks that cannot be executed in parallel but for which the heuristic used has low discriminatory power, a random ordering is chosen. Instead, in this case our grouping method will simply replace such tasks with a grouped task.

More precisely, while we normally would post $t_i < t_j$ if $d(t_i, t_j) > d(t_j, t_i)$, and $t_j < t_i$ otherwise, we now employ a threshold parameter $\gamma$, enforcing a minimal difference before the algorithm turns to posting a constraint. If $|d(t_i, t_j) - d(t_j, t_i)| \leq \gamma$, we decide to group $t_i$ and $t_j$ into the composite task $t_{i,j}$ using the procedure outlined above.

In Algorithm 5.2, the structure of the resulting algorithm is shown. Note that task grouping and constraint posting are interleaved: if the chosen conflict falls below the threshold $\gamma$, the tasks are grouped, otherwise, a constraint is posted. This is possible because both constraint posting and task grouping are essentially problem transformations:

- constraint posting takes one RCPSP instance, and returns another RCPSP instance, with an additional precedence constraint.

- task grouping takes one RCPSP instance and returns another RCPSP instance, with two tasks replaced by one other task representing a task group.
5.3. Task grouping

**Algorithm 5.2:** Using task grouping and constraint posting to solve an RCPSP instance

```plaintext
while the RCPSP instance contains a peak do
    p ← largest peak
    if p is unresolvable then terminate
    \{t_i, t_j\} ← SelectConflict(p)
    if |d(t_i, t_j) - d(t_j, t_i)| ≤ γ then
        GROUP(t_i, t_j)
    else if d(t_i, t_j) > d(t_j, t_i) then
        add t_i < t_j
    else
        add t_j < t_i
end
```

**Example 5.8** Consider again the conflicts from Example 5.4. An alternative way of resolving them would be to first post the constraint \( t_2 < t_3 \). For the next peak, we can, among other options, choose to post \( t_4 < t_5 \) (as done before) or \( t_5 < t_4 \). From a temporal perspective, there is little difference between the two options. Therefore, we choose to apply GROUP\((t_4, t_5)\) to the problem, which results in the following transformation:

<table>
<thead>
<tr>
<th>before GROUP((t_4, t_5))</th>
<th>task</th>
<th>( t_1 )</th>
<th>( t_2 )</th>
<th>( t_3 )</th>
<th>( t_4 )</th>
<th>( t_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>23</td>
<td>25</td>
<td>23</td>
<td>20</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>use of ( r_a )</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>use of ( r_b )</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>after GROUP((t_4, t_5))</th>
<th>task</th>
<th>( t_1 )</th>
<th>( t_2 )</th>
<th>( t_3 )</th>
<th>( t_{4,5} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>23</td>
<td>25</td>
<td>23</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>use of ( r_a )</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>use of ( r_b )</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Note that the length of \( t_{4,5} \) is the sum of the lengths of \( t_4 \) and \( t_5 \), and that \( t_{4,5} \) uses \( r_b \) over its entire length—even though \( t_5 \) does not need this resource.

Graphically, this solution looks as follows:
Now, consider an execution where \( t_1 \) is delayed. Using the solution in Example 5.5, task \( t_3 \) will be delayed too, due to the precedence constraint. However, \( t_4 \) cannot start until \( t_1 \) is finished either, due to the use of \( r_b \), so this task, and \( t_5 \), will be delayed as well. Contrast this with the solution in this example: here, the agent executing the schedule can still choose to start with \( t_5 \) instead. This task does not need \( r_b \), and, since \( \text{cap}(r_a) = 2 \), \( t_5 \) can execute concurrently with \( t_1 \). Hence, if \( t_1 \) is not delayed past the end of \( t_5 \), the makespan of this instance is not influenced at all.

## 5.4 Execution of grouped schedules

Having defined how to build a schedule containing sequential flexibility, in the form of grouped tasks, this section describes how to execute such a schedule, and to make use of this flexibility. Execution of a schedule containing grouped tasks differs from that of a normal schedule, since at run time, an ordering needs to be determined for the tasks contained in a group. If a schedule is executed without any tasks being delayed, executing a group task is trivial: since the group task has ensured that resources needed by any task in that group are available for the duration of that group, any ordering picked is guaranteed to be feasible.

When tasks are delayed during their execution special precautions need to be taken, both for normal and for grouped tasks: the start of tasks succeeding the delayed task might need to be delayed, since the delayed task might now finish after the planned start time of these tasks. As concurrent execution is not possible, due to either resource constraint violations, or violations of precedence constraints present in the original problem, any task that can not start at its planned time is added to the end of a queue of pending tasks. Whenever a task finishes executing, this queue is examined first, in order, to see if any of these tasks are now eligible to be started. Notice that this causes delay propagation: resources freed by the finished task might be claimed by one of the pending tasks, resulting in tasks that would otherwise be able to start being added to the pending queue.

Grouped tasks can be executed using this mechanism as well: a grouped task is in essence a set of pending tasks, one of which can be executed at the start time of that grouped task. This means that, if a grouped task is encountered, all tasks in the group are added to the pending queue, and then the first eligible task in the queue is selected for execution. Recall that at any time a task finishes, the queue is examined again for eligible tasks. This ensures that all tasks from the group are executed eventually.

Note that the order in which the tasks from a group are added to the queue can be determined at execution time. This is the mechanism by which grouped tasks achieve sequential flexibility: the tasks can be added to the pending queue in any order, and if there are no delays, the construction of the schedule ensures that they can be executed in sequence. If there are delays, the mechanism of picking eligible tasks from the queue ensures that the group task is automatically reordered to adapt to the changed circumstances during execution.
Example 5.9 To show how delays are handled during the execution of a schedule, consider the (grouped) schedule shown in Example 5.8. Assume that \( t_1 \) is lengthened by 20%, such that its length will be \( l'_1 = 1.2 \cdot l_1 = 27.6 \). Execution proceeds normally, until time point 25, when \( t_3 \) and \( t_{4,5} \) are to be started. Task \( t_3 \) depends on \( t_1 \) to be finished, so we cannot execute this task now, and we add it to the queue of postponed tasks.

While \( t_{4,5} \) does not directly depend on \( t_1 \), it does make use of \( r_b \), which is still in use by \( t_1 \) at its start time. If \( t_{4,5} \) was a normal task, we would have to delay its execution, but in this case, we can make use of the option to reorder the execution of this group task: if we choose to execute \( t_5 \) before \( t_4 \), no delay is necessary, since \( t_5 \) does not make use of \( r_b \). For larger group tasks we check for all tasks in the group if all needed resources are available, and if all predecessors have finished. From the set of tasks for which this holds, we pick one (here, only \( t_5 \) is eligible), and the rest (in this case, \( t_4 \)) is added to the queue of postponed tasks.

At time point 27.6, \( t_1 \) is finished, and the queue of postponed tasks is examined. Task \( t_4 \) can not start, since \( t_5 \) is still executing, but \( t_3 \) can now be started, such that it finishes at time point 50.6. The last task in the queue is \( t_4 \), which can be started after \( t_5 \) has finished. Since the start time of \( t_5 \) was not delayed, \( t_4 \) finishes at the scheduled time.

5.5 Experiments

As shown in the preceding section, task grouping offers increased flexibility in executing a schedule. This is demonstrated by the increased number of topological orderings allowed by such a schedule, as well as by Example 5.8 in which execution-time reordering is demonstrated. Recall, however, that our reason for constructing flexible schedules was born out of a desire to improve robustness during execution, if delays were to be expected. Both these observations offer a basis for the hypothesis that schedules using task groups are more robust when delays are present. To test this hypothesis, this section presents experiments using simulated executions, with delays, of such schedules.

We start with a description of how the experiments are setup. After this, we begin with a general analysis of the structure of the schedules resulting from our experiments, which is followed by a series of experiments to determine if our method indeed leads to higher robustness, and under which circumstances and by which mechanism it does so.

5.5.1 Setup of experiments

To be able to simulate schedule executions and measure the effect of task grouping on the robustness, we must be able to construct these schedules, with and without task groups. For this, the well-known benchmark instances from PSPLIB are used (see Kolisch and Sprecher (1996); Kolisch et al. (1998) for an in-depth
Table 5.1
Summary of the PSPLIB benchmark instances.

<table>
<thead>
<tr>
<th>Name</th>
<th>Number of tasks</th>
<th>Number of instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>j30</td>
<td>30</td>
<td>480</td>
</tr>
<tr>
<td>j60</td>
<td>60</td>
<td>480</td>
</tr>
<tr>
<td>j90</td>
<td>90</td>
<td>480</td>
</tr>
<tr>
<td>j120</td>
<td>120</td>
<td>600</td>
</tr>
</tbody>
</table>

discussion of the design of these instances). From this set we use the single-mode instances only.\textsuperscript{2} Note that the PSPLIB benchmark instances were originally designed to explore the various levels of difficulty of RCPSP instances, therefore the instances in the benchmark have a high variety, as different parameters potentially affecting the difficulty of the instances are varied over the instances. For each parameter combination, ten instances are included in the benchmark set. Van Nijs and Klos (2014) analyze these instance complexity issues further, and improvements are suggested to generate instances that are more constant over some of the complexity parameters. However, for our purposes, having a varied set of instances is beneficial, since this will show the performance improvement independent of instance difficulty parameters. Table 5.1 presents a short summary of the properties of the instances contained in this benchmark.

5.5.2 Characteristics of grouped schedules

To convert these RCPSP instances into schedules, Algorithm 5.1 is used. If $\gamma = 0$ is used as threshold, the grouping operation is never applied. When $\gamma > 0$, the algorithm creates group tasks in an opportunistic fashion: tasks are grouped whenever the heuristic that decides on their ordering falls below the threshold $\gamma$. The number of group tasks in the solution is thus dependent on the combination of the structure of the problem, and the value of $\gamma$—there is no method to directly control the number or size of group tasks in the solution. Since the idea behind task grouping was to create groups to add flexibility, it is natural to start with an analysis examining the structure of the schedules created by the algorithm, in terms of the number of grouped tasks, and the size of these groups. The hypothesis here is clear: for higher values of $\gamma$, more and bigger groups are expected in the resulting schedule.

In Table 5.2 these data are summarized for $\gamma \in \{2, 4, 6, 8\}$. The threshold parameter has a clear effect on both the number of group tasks as well as on the number of tasks in a group: the higher the threshold the more grouping takes place, up to a certain level, confirming our hypothesis. The difference between $\gamma = 2$ and $\gamma = 4$ is much higher than the difference between $\gamma = 6$ and $\gamma = 8$, and since values of $\gamma > 8$ give little additional benefit in terms of number of group tasks and size of task groups, further experiments have been limited to schedules constructed using $\gamma \leq 8$.

\textsuperscript{2}Multi-mode instances have multiple versions of each task, with a different length and different resource usage pattern. These represent a different type of scheduling problem, outside of the scope of our research, and are thus not used in the experiments.
Table 5.2
Characteristics of grouped schedules for different parameters.

<table>
<thead>
<tr>
<th>Set</th>
<th>$y = 2$</th>
<th>$y = 4$</th>
<th>$y = 6$</th>
<th>$y = 8$</th>
<th>$y = 2$</th>
<th>$y = 4$</th>
<th>$y = 6$</th>
<th>$y = 8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>j30</td>
<td>1.79 (6.0%)</td>
<td>2.43 (8.1%)</td>
<td>2.69 (9.0%)</td>
<td>2.82 (9.4%)</td>
<td>2.26</td>
<td>2.46</td>
<td>2.62</td>
<td>2.77</td>
</tr>
<tr>
<td>j60</td>
<td>2.58 (4.3%)</td>
<td>3.39 (5.6%)</td>
<td>3.83 (6.4%)</td>
<td>4.08 (6.8%)</td>
<td>2.35</td>
<td>2.61</td>
<td>2.84</td>
<td>3.02</td>
</tr>
<tr>
<td>j90</td>
<td>3.21 (3.6%)</td>
<td>4.20 (4.7%)</td>
<td>4.64 (5.2%)</td>
<td>4.94 (5.5%)</td>
<td>2.39</td>
<td>2.69</td>
<td>2.94</td>
<td>3.13</td>
</tr>
<tr>
<td>j120</td>
<td>8.13 (6.8%)</td>
<td>9.85 (8.2%)</td>
<td>10.42 (8.7%)</td>
<td>10.62 (8.8%)</td>
<td>2.47</td>
<td>2.86</td>
<td>3.17</td>
<td>3.44</td>
</tr>
</tbody>
</table>

An interesting observation is that in general, the (relative) number of group tasks gets lower as instances get bigger, except for the instances in the j120 set. A likely cause for this difference is the different set of resource strength parameters used in generating the instances. The resource strength parameter $RS$ sets the capacities of the resources of an instance (Kolisch and Sprecher, 1996), as the ratio between the minimally needed capacity (i.e., the capacity needed to be able to sequentially execute each task) and the highest possible capacity (i.e., the capacity needed to be able to execute the schedule without any additional leveling constraints). The sets $j30$, $j60$ and $j90$ use $RS \in \{0.2, 0.5, 0.7, 1.0\}$, whereas $j120$ uses $RS \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$ (Kolisch et al., 1998) — this is also the reason that $j120$ has more instances. The (on average) lower resource strength for the $j120$ instances lead to more resource peaks, which in turn lead to more constraints being posted and tasks being grouped.

5.5.3 Robustness
The main purpose of task grouping, increasing the flexibility of the schedules to improve the robustness during execution, is investigated next. To analyze the robustness, a certain percentage of the tasks in the constructed schedule is selected, and their lengths are extended to simulate a delay. These schedules are then executed as described in Section 5.4. For every instance, a percentage of the tasks is delayed with a constant factor. Each instance is simulated 150 times, where the tasks to be delayed are selected at random. The robustness is measured by how well execution adheres to the schedule: tasks that complete after their projected finish time are counted as late, and the robustness of a schedule is measured as the average number of late tasks over all simulated executions. Note that this differs from the concept of violations as used in the previous chapter: in that chapter, we counted how often a task had to be executed outside of its assigned interval. As we do not use intervals here, this approach is not applicable, and we resort to counting the number of tasks that fail to finish at their scheduled time.

The delays inserted during execution are characterized by two parameters: the number of tasks that are delayed for a given execution, and the amount of delay given to each of these tasks. The first parameter, the number of delayed
Figure 5.1
The effect of delay characteristics on the robustness of executed schedules, for various settings of the grouping threshold parameter $\gamma$. Simulations are performed on all PSPLIB instances, shown is the average percentage of late tasks.

(a) Increasing the number of delayed tasks. The delay per task is kept constant, at 10%.

(b) Increasing the amount of delay of tasks. The number of delayed tasks is kept constant, at 20%.

tasks, is taken from the set \{10%, 20%, 30%, 40%, 50%\}, relative to the total number of tasks in the instance. The second parameter, the amount of delay in a task, is taken from the set \{5%, 10%, 25%, 50%\}, relative to the length of the task receiving the delay.

The effect of these parameters on the performance, in terms of robustness, is investigated separately. We start by fixing the delay per task at 10%, and varying the number of tasks with a delay. The results are shown in Figure 5.1(a). The number of tasks that complete in time generally drops rapidly when increasing the number of delayed tasks, which is in line with expectations. The performance of grouping increases for larger values of $\gamma$; the cause is the larger number of groups present in the solution. For a low number of tasks with a delay, $\gamma$ has less effect on the performance. A reason might be that all the delays that can be absorbed are already absorbed with a low number of groups—adding more groups does not give an increase in performance in this case.

Similarly, in Figure 5.1(b), the predictability for increasing the amount of delay in each task is shown, for a delay in 20% of the tasks. A similar, but more pronounced, effect is seen for simulations where only 5% delay is inserted in a delayed task: adding more groups, by increasing $\gamma$, has only very little effect.

We have so far analyzed the performance of the algorithm by looking at the delay parameters and the various values for $\gamma$. While the $\gamma$ parameter can be used to indirectly influence the formation of groups by the algorithm, as evidenced
5.5. Experiments

Figure 5.2
The relationship between the number of commitments and the robustness. Results are shown for the j120 set, with 20% of the tasks delayed by 10%. For each level of commitment, a box and whiskers plot is shown representing the distribution of the percentage of late tasks. If less than ten instances are available for a given level, the individual results are shown.

by Table 5.2, the effects we witness are not directly coupled to this parameter, but to its result: a higher participation of tasks in groups. This higher participation can be viewed as a reduction of the level of commitment present in the schedule: if we have a task group containing \( m \) tasks, this is a single commitment, whereas \( m \) separate tasks represent \( m \) separate commitments. Our hypothesis is thus that schedules with lower commitment levels will show an increase in performance in terms of robustness. Since the size of the problem also determines the level of commitment in the resulting schedule, we will restrict ourselves to the j120 set: as discussed before, these instances have, on average, lower resource strength, and contain more/bigger resource peaks, leading to more complex solutions. We also opt to fix the delay parameters at a delay of 10% per task, for 20% of the tasks.

In Figure 5.2 the lateness percentage is shown for the various levels of commitment present in all solutions for the j120 instances. Solutions with equal levels of commitment are grouped together, represented by a box plot. This means that all schedules constructed without task grouping are represented by the box plot on the far right. The schedules constructed using \( \gamma > 0 \) are represented by the other box plots, depending on the amount of grouping that took place.

There is indeed a clear correlation between the level of commitment and the lateness, with diminishing returns for lower levels of commitment. A more interesting observation is that the lateness distribution for the ungrouped solutions has a lower median than that of the first few boxplots of the solutions containing a low number of task groups. There is apparently a common factor
in the structure of these instances that makes them both less amenable to the creation of task groups and more susceptible to the effects of delay. A possible reason might be the existence of a larger number of precedence constraints in the original problem. These may lead to longer chains of tasks in the solution, which result in more tasks being affected by delay propagation. At the same time, instances with more precedence constraints have a lower number of tasks in parallel during the construction of the schedule, resulting in fewer opportunities to group tasks together.

5.5.4 Solution structure

An astute reader may have already observed that the preceding analysis based on the level of commitment neglects an important distinction in the structure of the solution. Schedules with a similar level of commitment can have a very different composition in terms of task groups: one solution might have only a few larger groups, whereas another solution might contain many smaller groups. From our earlier discussion of the effect of task groups on the flexibility, it would seem that increased group sizes would have more effect than an increased number of groups:

\[ \text{Example 5.10} \] Consider a task group with \( m > 2 \) tasks. At execution time, such a group permits \( m! \) possible orderings. If we now divide these tasks over two smaller, sequentially executed groups of size \( m/2 \) the number of possible orderings is reduced, to \( 2 \cdot (m/2)! \).

As the number of groups is also dependent on the number of tasks in the problem, we again restrict our analysis to the j120 set, with delay parameters fixed to the same values as earlier, a delay of 10% per task, for 20% of the tasks.

In Figure 5.3 the lateness is plotted relative to both the number of groups and the average size of the groups. These two solution structure parameters form a discrete, two-dimensional space. At each point in this space where solutions are present, a circle is plotted, where its size is scaled according to the average lateness for solutions at this point. As a visual aid, the circles are also given a shade of gray according to the quartile in which they belong.

Looking at the figure, it is clear that both the number of groups as well as the size of the groups contributes to decreasing the lateness percentage. What is surprising, however, is that our hypothesis that the increased flexibility offered by a smaller number of large groups as compared to a larger number of smaller groups should lead to higher robustness and thus lower lateness does not seem to hold. To validate this visual impression, the Spearman rank-order correlation was calculated between both average group size and number of groups, and the lateness. For the group size, a correlation of \( \rho = -0.567 \) was found, and for the number of groups, a correlation of \( \rho = -0.749 \) was found. This confirms the visual impression that having more groups has a more consistent effect on lateness than having large groups.

An explanation for why our hypothesis does not seem to hold lies in the difference between flexibility and robustness. While a schedule with few large
5.5. Experiments

Figure 5.3
Lateness versus both the number and average size of group tasks. Results are shown for the j120 set, with 20% of the tasks delayed by 10%.

group tasks might have high flexibility, this flexibility is necessarily concentrated in the areas containing these large groups. Our method of assessing robustness is based on introducing delays in a schedule during execution, and measuring how many tasks are affected by these delays. Since these delays can be inserted at any location in the tasks, a delay might impact many other tasks before a task group, which can absorb it by reordering as demonstrated by Example 5.8. If the schedule contains more, smaller task groups, less tasks will be affected, explaining the higher correlation between the number of tasks and lateness.

**Example 5.11** To illustrate this effect, consider the schedule depicted below (for clarity, the lengths of the tasks are not shown). Imagine that an unexpected event causes \( t_1 \) to be delayed. This delay propagates to both \( t_2 \) and \( t_3 \), and then further down the schedule. The background of affected tasks is colored. Tasks all the way up to, and including, \( t_{11} \) are affected. The single, large group task \( t_{6,7,8,9} \) is not in the “path” of the delay, and therefore cannot absorb it.
Contrast this with the schedule below, for the same network structure, in which two smaller group tasks are present. Here, only $t_2$ and $t_3$ are affected, and then group task $t_{4,5}$ can absorb the delay.

5.6 Conclusion and discussion

This chapter presents a novel way to create flexible schedules, addressing the third research question raised by this thesis. The method works by creating solutions to scheduling problems containing groups with multiple tasks in them. The order of the tasks in a group is not fixed in the solution, and can be freely determined at execution time. This serves a dual purpose: it answers the needs of the third research question, by giving a method with which it is possible to efficiently represent solutions to the RCPSP containing sequential flexibility, and it gives another way of constructing schedules that are robust in the presence of uncertainty.

The algorithm we present forms task groups based on a threshold parameter $\gamma$. Experiments confirm our expectation that higher values of $\gamma$ lead to both more and bigger groups. Independent control of these parameters is not possible with the current form of the algorithm. Simulating executions of schedules with delays demonstrates that schedules with group tasks are better able to combat the spread of those delays. Larger values of $\gamma$ lead to better performance in that regard: fewer tasks are started late as a result of the simulated delays in the execution of the schedule.

Since $\gamma$ offers no direct control of either the number or the size of groups, we next looked at the level of commitment versus the lateness. Each task, whether it is a grouped task or a normal one, is seen as one commitment, to represent the flexibility we still have in executing a grouped task. In this way, the more tasks take part in a grouping operation, either in the form of more groups, or bigger groups, the lower the total level of commitment in the schedule. There is indeed a clear correlation between commitment and lateness, but it also leads to the next question: what influences the performance more, the number of groups or the size of the groups?

This question leads to a rather surprising result. Our initial hypothesis was that bigger groups will give higher robustness, since they offer higher flexibility: if we split a big group in two smaller, sequential groups, the number of possible orderings is made smaller. The experiments show however that the reverse is
true: the number of groups has a bigger effect on the robustness than the size of the groups. An explanation for this effect is that the origin of a delay is random. In an instance with few large groups, such a delay might impact a large number of (ungrouped) tasks before a task group that can absorb it, by reordering, is reached.

5.6.1 Discussion

The concept of task grouping offers a lot of opportunities for future research. The experiments show that the number of groups in the schedule offers a good indication of the performance in terms of late tasks. Using the current method however, it is hard to directly control the number of groups. This could be very desirable in a situation in which we want to ensure that the resulting schedule has a certain amount of sequential flexibility, be it to combat uncertainty or to offer autonomy to an executing agent. The current method is opportunistic in performing the grouping operation: tasks are grouped only if the heuristic, using the threshold parameter $\gamma$, indicates so. For some situations it would be more practical to determine the number of required groups beforehand, and to have a method that would perform grouping operations until that level was reached.

A second important point that has so far not been addressed is the selection of the tasks to be grouped. In essence, two opposing objectives can be identified here. On the one hand, it is beneficial if tasks to be grouped have a similar resource usage. This way, the over-estimation of resource usage for the group task will be kept to a minimum, such that the makespan will be less affected. On the other hand, to be able to prevent delay propagation, tasks in a group should have as diverse a resource usage as possible. This way, if one resource is still in use due to a delay, the probability that a task can be found in a group that does not use this resource is higher. Different policies for selecting tasks to group should therefore be tested, to see if these effects are indeed present.
The research presented in this thesis was undertaken in the context of the applied research and development program at NedTrain, which performs the maintenance operations for the rolling stock of the Dutch national railway services. It focuses on the planning challenges encountered at the operational level, dealing with the scheduling of maintenance jobs in a single workplace.

The changing environment of the railway industry introduces new challenges to maintenance operations. Two developments are discussed that present new challenges to maintenance operations at the operational level. In the past, new rolling stock was developed in very close cooperation with national industries, leading to a large build-up of knowledge among engineers long before the new stock was taken into service. In modern railway operations, there is a preference to buy existing designs, tailored to the local specifications. Combined with higher complexity of on-board systems, this leads to maintenance operations being more difficult and less predictable. A second development is the increased need for punctuality due to the desire to have multiple smaller maintenance visits such that trains are only in workshops during off-peak hours. The inherent uncertainties in the maintenance jobs tend to average out if a visit is fairly long, but this effect is lost if the visits are made much shorter. For such visits it is therefore important to have a flexible method of planning, with flexible results.

In this last chapter, we want to re-examine the research goals as introduced in the first chapter, and evaluate if these goals were met. To do this, we work back by first re-examining the research questions together with the answers as presented in this thesis. Then, we turn to the research goals, and also re-examine them, to see if they have been met. Along the way, some insights on future research avenues will be presented.

### 6.1 Research questions

We will start by re-examining the research questions presented at the end of the second chapter, and we will compare them with the research presented in this thesis.
The nature of the scheduling problem at NedTrain is such that flexibility, to be used during execution of a schedule, is of high importance. Existing research shows that the Simple Temporal Network (STN) can be used as a flexible solution representation for an instance of the Resource-Constrained Project Scheduling Problem (RCPSP), which is able to capture the most important aspects of our scheduling problem. But from this existing literature it is not clear how to quantify the flexibility of a solution exactly, or how to compare two solutions to the same problem based on their flexibility. Therefore, we posed our first research question:

**RQ1** What criteria encapsulate our intuitive notion of the concept of flexibility, as applied to solutions of scheduling problems, and can we devise a method to measure flexibility, conforming to these criteria?

The rationality postulates $F_1, F_2, F_2'$ and $F_3$ introduced in Chapter 3 capture this notion of flexibility. Of special interest is our contribution of $F_2'$, which states that the flexibility of one task has to be independent of the chosen start time of any of the other tasks: picking a certain start time for one task should not reduce the flexibility of another task.

Applying these postulates to existing metrics, we discovered that these severely over-estimate the available flexibility, and we introduced a new measure that better takes into account dependencies between tasks, satisfying $F_2'$. This measure is constructive: as a result of calculating the flexibility of an STN we arrive at an assignment of intervals to tasks. Such an assignment has the property that the intervals are independent: a choice in one interval has no effect on the available choices in any of the other intervals.

Having determined how flexible a certain solution is, the next question is how to best make use of this flexibility. Our goal here is to arrive at a robust schedule—a schedule in which a disturbance in one task has low impact on the rest of the schedule.

**RQ2** Given a schedule with a certain potential flexibility, in what ways can this flexibility be used best to achieve a schedule with high robustness?

The experiments performed in Chapter 4 show that the best way to distribute flexibility depends on the exact aim. A more even distribution of flexibility lowers the number of violations (i.e., tasks executing outside their assigned interval because of a delay), and lowers the tardiness (i.e., how severely the deadline of the schedule is violated because of delays). If this distribution is skewed such that tasks having more predecessors receive more flexibility, the effect on tardiness is more pronounced, while the number of violations increases.

One result we encountered is contradictory at first glance: using a solution with lower flexibility can improve the robustness when executing the schedule. This seems to contradict our desire that our measure of flexibility corresponds to intuition, such as expressed in the first research question. The effect occurs because sometimes, a higher global level of flexibility can be achieved if multiple
tasks or regions of a schedule are ‘sacrificed’ in terms of available flexibility such that one (or a few) tasks can be allocated more flexibility. The result is a solution with high level of flexibility, concentrated in only a small part of the schedule. The apparent contradiction is caused because our operationalization of the intuitive idea of flexibility is applied at a global scale, looking at the complete schedule. The effects influencing robustness operate at a much more local scale: for a certain disturbance, only tasks that are successors of the delayed tasks are influenced.

So far, we have solely considered the temporal aspect of flexibility in a schedule. The third research question extends the scope of our research to sequential flexibility:

**RQ3** In what way can the concept of temporal flexibility of a schedule be extended to include some *sequential flexibility*?

In answering the previous research questions, we considered solutions at the level of the STN. For the third research question, we widen our scope to include the conversion of an RCPSP to an STN, and we propose a task grouping extension to the STN with which not only temporal flexibility can be encoded, but sequential flexibility as well. Constraint posting normally adds precedence constraints between tasks to convert resource constraints into temporal constraints. Our task grouping extension adds the option to *group* two tasks together, enforcing sequential execution (and thus avoiding a resource conflict), but allowing the order of the two tasks to be decided upon at execution time. Experiments show that this method can give improvements to robustness during execution as well.

The last research question focuses on the workplace structure of parallel maintenance by independent teams. Ensuring team independence can be seen as an instance of the *decoupling problem* in multi-agent scheduling. Here, again, our focus is on robustness: we want disturbances in the execution of the schedule of one team to have minimal effects on the other teams in the workplace.

**RQ4** How can our earlier work on ensuring robustness of entire schedules be applied to the *decoupling* of multi-agent schedules, in which we strive to decouple in such a way that disturbances in the schedule of one agent minimally affect the schedules of the other agents?

Our work in Chapter 3 in answering the first research question yielded a method to construct interval schedules, and as it turns out, the fact that these intervals are constructed to be *independent* enables such a schedule to serve as a *decoupling*, precisely because of this independence, thereby answering the last research question as well. Additionally, we show that our method to construct an interval schedule by definition leads to an *optimal* decoupling: an optimal decoupling has no loss of flexibility compared to the original schedule. As our decoupling is derived directly from the interval schedule constructed to achieve maximal flexibility it is intuitively evident that the decoupling is optimal.
6.2 Research goals and further research

Having discussed how the research questions posed at the end of Chapter 2 have been answered, we will now turn to the research goals formulated at the end of Chapter 1, and we will compare the answers to the research questions with the original goals.

The first research goal is to determine a way in which the scheduling process at NedTrain workshops can be automated.

**RG1** The scheduling process employed in the NedTrain workshops relies on a large amount of slack to keep unexpected events under control. In addition, it is labor-intensive and inflexible. In what way can this process be at least partially automated? Note that automation alone is not sufficient: the process of constructing and adapting a schedule has to be quick enough to make it part of an interactive process, and it has to be flexible, to be able to adapt to the incomplete and changing nature of the environment.

This goal is already partially met by existing research discussed in Chapter 2, since the scheduling problem at NedTrain can be represented as a Resource-Constrained Project Scheduling Problem, for which Precedence Constraint Posting is a flexible process resulting in a flexible solution.

The requirement of having a flexible process is solved by this method, but as stated in the second research goal, the requirement of a flexible and robust solution needs further research:

**RG2** Schedules, by their nature, prescribe a certain way to execute a set of tasks, giving rise to a form of rigidity. To be able to adapt to uncertain events it is desirable to reduce this rigidity.

a) As a primary goal, we want to have a schedule that prescribes task execution in a flexible way, such that we can adapt to changing events without needing to revise the schedule, and without having to resort to using large amounts of slack.

b) As a secondary goal, we want the resulting schedule to be robust: if the available flexibility is not enough to be able to adapt to some events, we want the unavoidable revisions we have to make to continue executing the needed tasks to be as minor as possible.

These goals are investigated with the first three questions. The answer to **RQ1** gives a method to prescribe task execution in a non-rigid way, by assigning intervals to each task. This method is further investigated in answering **RQ2**, when we determine how the flexibility in a schedule can be used most effectively. And finally, **RQ3** investigates a different method of achieving non-rigid prescription of task execution, by postponing some ordering decisions to execution time. Experiments performed using the proposed methods show that clear improvements
in robustness can be achieved when delays are present during the execution of a schedule.

The answer to in particular the second research question raises an important question for further research, as we show that there is a discrepancy between flexibility and robustness: our answer to the first research question gives an efficient method to determine the flexibility of a schedule a priori—but the experiments performed to determine the robustness show that high flexibility does not translate directly to high robustness. It would be advantageous to have a method by which the (expected) robustness of a solution could be measured directly, especially if such a metric could then be used as optimization target.

Our investigation of the third research question yielded the concept of task grouping, to also have a form of sequential flexibility. An interesting approach for future research would be to combine this concept with the construction of interval schedules, such as introduced in this thesis as well. This combination would give even more robustness during execution. Another limitation of the task grouping approach is that while a group task can contain an arbitrary number of tasks, they have to be executed sequentially. One could think of more advanced group task constructions, where such tasks can also contain parallel components. This would create more opportunities to use the grouping operation, at the expense of a more complex algorithm.

The last research goal describes the need for isolation between teams in the workplace that make use of shared resources.

RG3 The need to use shared resources to complete tasks, combined with uncertainties in the schedule negatively affects the independence of the teams in the workshop. Our goal is to construct schedules such that there is more isolation between the teams, in terms of disturbances in the execution of the schedule in one team not affecting the other teams.

This is translated to the decoupling problem, which is a well-known problem in scheduling research. The answer to rq1 shows that it is possible to decouple a schedule without loss of flexibility: the flexibility is calculated using an interval schedule that maximizes flexibility, and we prove that such an interval schedule is in itself a decoupling.

However, as already described in the preceding section, our decoupling strategy focuses on individual tasks instead of teams. This method works well if each executing agent is in control of a single task, but in real-life situations agents are often in control of a group of tasks. An adaptation of our method to only consider the constraints between such groups of tasks is fairly straightforward. The expected benefit of such an adaptation is that more flexibility can be allotted to these inter-agent constraints, leading to better isolation between the agents: a delay in one agent has a higher chance of being absorbed by inter-agent constraints if the available flexibility on these constraints is higher.
6.2.1 The greater context

When placing the results of our research in the greater context of the maintenance scheduling problem at NedTrain, some additional opportunities for future research can be identified.

Real-life problem sets

All experiments performed in this thesis use synthetic benchmark instances from the PSPLIB set of scheduling problems. This set of instances is constructed with variation in mind: it is constructed as a full factorial design using several instance parameters. The advantage is that this leads to varied instances, showing that our methods work in very different settings. The disadvantage is that there are relatively few instances per parameter combination. An important validation step would be the use of problem instances more closely resembling the actual scheduling problems present in the NedTrain workplaces.

Stochastic extensions

As has been noted at many points in this thesis, the uncertainty in the tasks to be executed is one of the major problems. The approach taken has been to hedge against the uncertainty by ensuring that the schedules contain enough flexibility to adapt to the uncertainty. If the amount of uncertainty is known beforehand, one of the obvious opportunities for extension is to make use of this information to better control schedule execution: the start time and duration of tasks can be extended with probabilistic information, enabling us to make informed predictions about schedule deadline violation probabilities. Mountakis (2015) presents a proof-of-concept of this idea, where techniques for timing analysis of complex digital integrated circuits are adapted to analyse schedules with tasks having probabilistic information.

Interactive constraint posting

Another important direction for future research in constraint posting methods is their use in interactive environments. Precedence constraint posting methods take a temporal and resource constrained network of tasks, and add temporal constraints to eliminate the resource constraints step by step—the end result is a purely temporal problem. The form of this temporal problem is identical to the temporal part of the problem used as input, which shows that constraint posting is well-suited for iterative or interactive environments: adding resource constraints or removing unsuited temporal constraints can be followed by another application of the algorithm resulting in a new solution. This thesis has not investigated the effects of such a scheme on the flexibility and robustness. Such interactive usage is important if our methods are to be used in a workshop—often, not all the constraints of the scheduling problem can be encoded in an RCPSP instance, and manual corrections might be necessary. Intuition suggests that having high flexibility would be of use in this case, but more research is needed to validate this claim.
Bibliography


Summary

This thesis presents research on scheduling in an uncertain environment, which forms a part of the rolling stock life cycle logistics applied research and development program funded by Dutch railway industry companies. The focus therefore lies on scheduling of maintenance operations on rolling stock in the railway industry.

The first chapter describes some of the history of the Dutch railways, focusing on the rolling stock used, and it introduces the context in which NedTrain, a major Dutch rolling stock maintenance company, operates. While maintenance of rolling stock is not a new problem, recent changes in the field are identified in this chapter which introduce new challenges: the first is the declining involvement of maintenance experts in the procurement of new rolling stock, the second is the on-going demand for increasing efficiency. Based on this discussion, the goal of this thesis is to devise a method with which schedules can be created based on a flexible process. The schedules resulting from this method are to be both flexible, meaning that they can be adapted easily, and robust, meaning that they are resilient to the effects of uncertain events. Lastly, our goal is to create the schedules in such a way that the effects of disturbances in one team have little or no effect on any other teams in the same workshop.

Chapter 2 examines existing research in the context of the formulated research goals. It is shown that the well-known Resource-Constrained Project Scheduling Problem (RCPSp) can be mapped to the problem of scheduling maintenance tasks on trains in the NedTrain workshops. The most important methods to solve the RCPSp are discussed, among which are many exact methods. Both the scale of the problem and the requirement of having a flexible schedule instead of a fixed time assignment for the tasks prohibit using these methods. Among the heuristic methods discussed, the Precedence Constraint Posting method turns out to be an especially good fit: it is able to solve large-sized problems very quickly, and it uses a Simple Temporal Network (STN) as a way of representing many different solutions to the original problem, offering a lot of flexibility. Based on the research discussed, research questions are formulated to answer the research goals.

- The first question asks how to define criteria encapsulating the notion of
flexibility as posed by our research goals, and how to measure the flexibility of a schedule.

- The second question asks how we can make the best use of available flexibility to attain robustness.

- The third question asks if there is also a way to extend the concept of temporal flexibility to include a form of sequential flexibility.

- The last question asks how we can use our work on robustness to decouple schedules, ensuring that disturbances in the schedule for one agent have no impact on any other agents.

The third chapter investigates criteria corresponding to our intuitive notion of flexibility in scheduling. The concept of interval schedules serves as a basis for this investigation: the idea is that each task in the schedule is assigned an interval from which its start time can be picked, and the length of all these intervals serves as a representation for the flexibility of the schedule. We show that flexibility measures in existing research lead to an over-estimation of the total available flexibility, because dependencies between tasks are not taken into account properly: simply adding the interval sizes for two dependent tasks causes over-estimation, because the picked time for one task can reduce the interval size for another task. Our first major contribution is an independence requirement for the creation of these intervals: we state that intervals should be constructed in such a way that any choice in one interval has no impact on the available choices in any of the other intervals. Our second major contribution is a method to actually construct such intervals, which results in a valid interval schedule for an STN, allowing efficient schedule execution under dynamic circumstances. Additionally, we note that an interval schedule can also serve as a mechanism to decouple a schedule, giving a partial answer to the last research question.

In the fourth chapter, we investigate how to make use of the available flexibility to ensure that schedules are robust. We show that having a maximal total amount of flexibility does not imply that the schedule is as robust as possible: maximization might lead to a very skewed distribution of flexibility over the schedule. Different ways of distributing flexibility over a schedule are proposed and analyzed in an experimental setting. From the experiments it can be clearly concluded that sacrificing some of the total flexibility to improve the distribution of flexibility can have a positive influence on the robustness of the schedule. We also propose a different maximization strategy: one which maximizes the minimum of the interval sizes for the schedule. This strategy is shown to work especially well for small delays.

So far we only discussed strategies concerning temporal flexibility, i.e., those in which start times remain flexible instead of fixed. Chapter 5 introduces a novel representation of solutions for planning problems in which not all ordering decisions needed to avoid resource contention are made in advance. This is achieved by grouping tasks which need to be executed sequentially together, in
such a way that either ordering of the tasks remains feasible during execution time. An analysis using simulation experiments shows that this technique results in higher robustness, at the cost of somewhat lower resource utilization. The proposed algorithm to construct such schedules offers limited control over the size of the grouped tasks. The experiments show a slightly counter-intuitive results: schedules with larger groups offer lower robustness than those with smaller groups. A plausible explanation is the fact that delays can occur at any place in a schedule. Having multiple smaller groups instead of a few very large ones increases the chance that such a delay can be compensated for using a task group.
Samenvatting

Dit proefschrift bevat onderzoek naar planning in een onzekere omgeving, en maakt onderdeel uit van het “rolling stock life cycle logistics applied research and development program”, gefinancierd door bedrijven in de Nederlandse spoorsector. Het accent ligt derhalve op het plannen van de onderhoudswerkzaamheden aan rollend materieel in de spoorwegindustrie.

Het eerste hoofdstuk beschrijft een deel van de geschiedenis van de Nederlandse spoorwegen, gaat in op het gebruikte materieel en beschrijft de context waarbinnen NedTrain, één van de belangrijkste onderhoudsbedrijven van Nederlands spoorwegmaterieel, functioneert. Ondanks het feit dat het onderhoud van rollend materieel geen nieuw probleem is, worden in dit hoofdstuk recente veranderingen geïdentificeerd die leiden tot nieuwe uitdagingen: de eerste is de afnemende betrokkenheid van onderhoudsexperts bij de aankoop van nieuwe treinen, de tweede is de alsmashjoa toenemende wens naar het verhogen van de efficiëntie. Het doel van dit proefschrift is derhalve het verzinnen van een methode waarmee planningen kunnen worden gemaakt op basis van een flexibel proces.

De planningen die deze methode oplevert moeten zelf zowel flexibel als robuust zijn, wat betekent dat ze makkelijk kunnen worden aangepast, maar ook dat ze bestand zijn tegen de effecten van onvoorziene gebeurtenissen. Bovendien moeten deze planningen een zodanige structuur hebben dat de effecten van verstoringen in het ene team weinig of geen effect hebben op andere teams in de werkplaats.

Hoofdstuk 2 behandelt bestaand onderzoek in samenhang met de geformuleerde onderzoeksdoelen. Er wordt aangetoond dat het welbekende “Resource-Constrained Project Scheduling Problem” (rCPS, projectplannen met beperkt aanwezige hulpbronnen) goed correspondeert met het probleem van het plannen van onderhoudstaken op treinen binnen de werkplaatsen van NedTrain. De belangrijkste methoden voor het oplossen van het rCPS worden besproken. Een aantal van deze methoden zijn exact; echter, dergelijke methoden zijn niet inzetbaar vanwege de schaal van het probleem en de eis voor het hebben van een flexibele planning in plaats van vaste starttijden voor de diverse taken. Onder de methoden op basis van heuristieken blijkt de “Precedence Constraint Posting” methode (een methode op basis van het incrementeel toevoegen van ordeningsrelaties) bijzonder goed te passen bij deze eisen: zij is in staat proble-
men van grote afmeting zeer snel op te lossen, en zij maakt gebruik van een Simple Temporal Network (STN) als methode om een scala aan verschillende oplossingen voor het originele probleem te bevatten, wat een grote mate van flexibiliteit oplevert. Op basis van het besproken onderzoek worden vervolgens onderzoeksvragen geformuleerd om te beantwoorden aan de onderzoeksdoelen.

- De eerste vraag is hoe criteria kunnen worden gedefinieerd die de intuïtieve notie achter flexibiliteit omvatten, zoals gesteld door de onderzoeksdoelen, en hoe de flexibiliteit van een planning kan worden gemeten.

- De tweede vraag is hoe we het beste gebruik kunnen maken van bovenge-noemde flexibiliteit om tot een robuuste planning te komen.

- De derde vraag is op welke wijze het idee van flexibiliteit in de tijd kan worden uitgebreid om ook een vorm van flexibiliteit qua ordening te omvatten.

- De laatste vraag is hoe het resultaat van robuustheid kan worden ingezet voor het “ontkoppelen” van planningen, waarbij het doel is dat verstoringen in de planning voor de ene partij geen effect hebben op andere deelnemende partijen.

In het derde hoofdstuk wordt ingegaan op de criteria die corresponderen met de intuïtieve notie van flexibiliteit in planning. Het concept van een intervalplanning fungeert als basis voor het onderzoek: het idee is dat elke taak in de planning een interval krijgt toegekend waaruit de starttijd kan worden gekozen. De lengte van al deze intervallen geeft een maatstaf voor de flexibiliteit van de planning. Er wordt aangetoond dat flexibiliteitsmaten uit bestaand onderzoek leiden tot een overschatting van de totaal beschikbare flexibiliteit. De oorzaak is dat afhankelijkheden tussen taken onvoldoende in beschouwing worden geno- men: het simpelweg optellen van de intervallengtes van twee afhankelijke taken leidt tot een overschatting, aangezien de gekozen starttijd voor de ene taak het beschikbare interval van de andere taak kan beïnvloeden. De eerste bijdrage is een onafhankelijkheidseis voor het opstellen van deze intervallen: de intervallen moeten worden gekozen op zodanige wijze dat een keuze in één interval geen invloed heeft op de beschikbare keuzes in willekeurig welk ander interval. De tweede bijdrage is een methode om daadwerkelijk dergelijke intervallen op te stellen, hetgeen resulteert in een geldige intervalplanning voor een STN, waarmee efficiënte uitvoer van taken bewerkstelligd kan worden, ook in een dynamische omgeving. Ten laatste wordt opgemerkt dat een intervalplanning ook als mechanisme kan dienen om een planning te ontkoppelen, hetgeen ten dele de laatste onderzoeksvraag beantwoordt.

In het vierde hoofdstuk wordt onderzocht hoe gebruik te maken van de beschikbare flexibiliteit zodanig dat planningen robuust zijn. Er wordt getoond dat het hebben van een maximale hoeveelheid flexibiliteit niet impliceert dat een planning zo robuust als mogelijk is: maximaliseren kan leiden tot een zeer scheve verdeling van flexibiliteit over de planning. Verschillende manieren om
flexibiliteit te verdelen over een planning worden geïntroduceerd en geanalyseerd
door middel van experimenten. Uit de experimenten kan worden geconcludeerd
dat het opofferen van een deel van de totale flexibiliteit ter bevordering van de
verdeling een positieve invloed kan hebben op de robuustheid van een planning.
Ook wordt een nieuwe strategie van maximalisatie voorgesteld: één waarbij de
minimaal voorkomende intervalgrootte wordt gemaximaliseerd. Deze strategie
werkt met name goed bij het voorkomen van kleine vertragingen.

Tot dusver zijn slechts strategieën besproken die betrekking hebben op temp-
orele flexibiliteit, dat wil zeggen, strategieën waarbij de starttijden van taken
flexibel zijn in plaats van vast. Hoofdstuk 5 introduceert een nieuwe wijze om
oplossingen van planningsproblemen vast te leggen waarbij niet alle ordeningsbe-
slissingen ter voorkoming van overgebruik van hulpbronnen volledig vast liggen.
Dit wordt bereikt door het groeperen van taken die achtereenvolgend moeten
worden uitgevoerd, op zodanige manier dat beide volgordes nog mogelijk zijn tij-
dens de uitvoer van de planning. Een analyse van deze techniek met behulp van
simulatie toont aan dat op deze wijze een grotere robuustheid wordt verkregen,
ten koste van een wat lagere benutting van de hulpbronnen. Het voorgestelde
algemeen biedt beperkte controle op de grootte van de gegroepeerde taken. Als
experimenten geven een iets tegen-intuïtief resultaat: planningen met grotere
groepen resulteren in een lagere robuustheid dan planningen met kleinere groe-
pen. Een verklaring voor dit feit is dat verstoringen op elke willekeurige plaats
in de planning op kunnen treden. Het hebben van meerdere kleine groepen in
plaats van slechts enkele grotere vergroot de kans dat een dergelijke vertraging
opgevangen kan worden met de flexibiliteit in een groep.
Michel Wilson was born in Rozenburg on Saturday, April 18, 1981. From 1994 to 1997 he received secondary education at OSG ’De Ring van Putten’ in Spijkenisse, and from 1997 to 1999 at CSG ’Blaise Pascal’, also in Spijkenisse, where he obtained his VWO diploma. Starting in the fall of 1999, he attended six months of community college at Green River Community College, in Auburn, WA, USA.

In the fall of 2000, Michel started his education at Delft University of Technology, beginning with a bachelor in Technical Informatics, followed by a master in Computer Science, specializing in Software Technology. As a master student, he researched the application of spectrum-based fault localization techniques to hardware systems, supervised by prof.dr.ir. A.J.C. van Gemund, resulting in an MSc degree in January 2010.

Michel began his work as PhD researcher in February 2010, at the Algorithmics group of the faculty of Electrical Engineering, Mathematics and Computer Science, also at Delft University of Technology, under the supervision of prof.dr. C. Witteveen and dr. T.B. Klos. In this period, he published five peer-reviewed papers as principal author, one of which received an IJCAI Distinguished Paper Award. During his time as PhD researcher, Michel was also involved in the teaching of courses offered by the Algorithmics group, and he supervised three MSc students. In May 2014 he started work at West IT Solutions B.V. in Delft.

In his spare time, Michel likes to tinker around with electronics and small embedded devices. Starting as a child, when his parents gave him an old manual SLR camera, he has been interested in photography. As a child he was also already interested in trains and railroads. This interest was revived during his time as PhD researcher, in which he enjoyed the collaboration with NedTrain, and the opportunities to visit maintenance workshops.
SIKS dissertation series

1998
1. Johan van den Akker (CWI) DEGAS—An Active, Temporal Database of Autonomous Objects
2. Floris Wiesman (UM) Information Retrieval by Graphically Browsing Meta-Information
3. Ans Steuten (TUD) A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective
4. Dennis Breuker (UM) Memory versus Search in Games
5. E.W. Oskamp (RUL) Computerondersteuning bij Strafoometing

1999
1. Mark Sloof (VU) Physiology of Quality Change Modelling; Automated modelling of Quality Change of Agricultural Products
2. Rob Potharst (EUR) Classification using decision trees and neural nets
3. Don Beal (UM) The Nature of Minimax Search
4. Jacques Penders (UM) The practical Art of Moving Physical Objects
5. Aldo de Moor (KUB) Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems
6. Niek J.E. Wijngaards (VU) Re-design of compositional systems
7. David Spelt (UT) Verification support for object database design

2000
1. Frank Niessink (VU) Perspectives on Improving Software Maintenance
2. Koen Holtman (TUE) Prototyping of CMS Storage Management
4. Geert de Haan (VU) ETAG, A Formal Model of Competence Knowledge for User Interface Design
5. Ruud van der Pol (UM) Knowledge-based Query Formulation in Information Retrieval.
6. Rogier van Eijk (UU) Programming Languages for Agent Communication
8. Veerle Coupé (EUR) Sensitivity Analysis of Decision-Theoretic Networks
11. Jonas Karlsson (CWI) Scalable Distributed Data Structures for Database Management

2001
1. Silja Renooij (UU) Qualitative Approaches to Quantifying Probabilistic Networks
2. Koen Hindriks (UU) Agent Programming Languages: Programming with Mental Models
3. Maarten van Someren (UvA) Learning as problem solving

133
4 Evgueni Smirnov (UM) Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets
5 Jacco van Ossenbruggen (VU) Processing Structured Hypermedia: A Matter of Style
6 Martijn van Welie (VU) Task-based User Interface Design
7 Bastiaan Schönhage (VU) Diva: Architectural Perspectives on Information Visualization
8 Pascal van Eck (VU) A Compositional Semantic Structure for Multi-Agent Systems Dynamics.
9 Pieter Jan ’t Hoen (RUL) Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes
10 Maarten Sierhuis (UvA) Modeling and Simulating Work Practice BRAHMS: a multiagent modeling and simulation language for work practice analysis and design
11 Tom M. van Engers (VUA) Knowledge Management: The Role of Mental Models in Business Systems Design

2002
1 Nico Lassing (VU) Architecture-Level Modifiability Analysis
2 Roelof van Zwol (UT) Modelling and searching web-based document collections
3 Henk Ernst Blok (UT) Database Optimization Aspects for Information Retrieval
4 Juan Roberto Castelo Valdueza (UU) The Discrete Acyclic Digraph Markov Model in Data Mining
5 Radu Serban (VU) The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-concerned Agents
6 Laurens Mommers (UL) Applied legal epistemology; Building a knowledge-based ontology of the legal domain
7 Peter Boncz (CWI) Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications
8 Jaap Gordijn (VU) Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas
9 Willem-Jan van den Heuvel (KUB) Integrating Modern Business Applications with Objectified Legacy Systems
10 Brian Sheppard (UM) Towards Perfect Play of Scrabble
11 Wouter C.A. Wijngaards (VU) Agent Based Modelling of Dynamics: Biological and Organisational Applications
12 Albrecht Schmidt (UvA) Processing XML in Database Systems
13 Hongjing Wu (TUE) A Reference Architecture for Adaptive Hypermedia Applications
14 Wieke de Vries (UU) Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems
15 Rik Eshuis (UT) Semantics and Verification of UML Activity Diagrams for Workflow Modelling
17 Stefan Manegold (UVA) Understanding, Modeling, and Improving Main-Memory Database Performance

2003
1 Heiner Stuckenschmidt (VU) Ontology-Based Information Sharing in Weakly Structured Environments
2 Jan Broersen (VU) Modal Action Logics for Reasoning About Reactive Systems
3 Martijn Schuemie (TUD) Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy
4 Milan Petkovic (UT) Content-Based Video Retrieval Supported by Database Technology
5 Jos Lehmann (UVA) Causation in Artificial Intelligence and Law—A modelling approach
6 Boris van Schooten (UT) Development and specification of virtual environments
7 Machiel Jansen (UvA) Formal Explorations of Knowledge Intensive Tasks
8 Yongping Ran (UM) Repair Based Scheduling
9 Rens Kortmann (UM) The resolution of visually guided behaviour
10 Andreas Lincke (UvT) Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture
11 Simon Keizer (UT) Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks
12 Roeland Ordelman (UT) Dutch speech recognition in multimedia information retrieval
13 Jeroen Donkers (UM) Nosce Hostem—Searching with Opponent Models
<table>
<thead>
<tr>
<th>Name</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virginia Dignum (UU)</td>
<td>A Model for Organizational Interaction: Based on Agents, Founded in Logic</td>
</tr>
<tr>
<td>Lai Xu (UvT)</td>
<td>Monitoring Multi-party Contracts for E-business</td>
</tr>
<tr>
<td>Perry Groot (VU)</td>
<td>A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving</td>
</tr>
<tr>
<td>Chris van Aart (UVA)</td>
<td>Organizational Principles for Multi-Agent Architectures</td>
</tr>
<tr>
<td>Viara Popova (EUR)</td>
<td>Knowledge discovery and monotonicity</td>
</tr>
<tr>
<td>Bart-Jan Hommes (TUD)</td>
<td>The Evaluation of Business Process Modeling Techniques</td>
</tr>
<tr>
<td>Elise Bolties (UM)</td>
<td>Voorbeeld onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes</td>
</tr>
<tr>
<td>Joop Verbeek (UM)</td>
<td>Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politie gegevensuitwisseling en digitale expertise</td>
</tr>
<tr>
<td>Martin Caminada (VU)</td>
<td>For the Sake of the Argument; explorations into argument-based reasoning</td>
</tr>
<tr>
<td>Suzanne Kabel (UVA)</td>
<td>Knowledge-rich indexing of learning-objects</td>
</tr>
<tr>
<td>Michel Klein (VU)</td>
<td>Change Management for Distributed Ontologies</td>
</tr>
<tr>
<td>The Duy Bui (UT)</td>
<td>Creating emotions and facial expressions for embodied agents</td>
</tr>
<tr>
<td>Wojciech Jamroga (UT)</td>
<td>Using Multiple Models of Reality: On Agents who Know how to Play</td>
</tr>
<tr>
<td>Paul Harrenstein (UU)</td>
<td>Logic in Conflict. Logical Explorations in Strategic Equilibrium</td>
</tr>
<tr>
<td>Arno Knobbe (UU)</td>
<td>Multi-Relational Data Mining</td>
</tr>
<tr>
<td>Federico Divina (VU)</td>
<td>Hybrid Genetic Relational Search for Inductive Learning</td>
</tr>
<tr>
<td>Mark Winands (UM)</td>
<td>Informed Search in Complex Games</td>
</tr>
<tr>
<td>Vania Bessa Machado (UvA)</td>
<td>Supporting the Construction of Qualitative Knowledge Models</td>
</tr>
<tr>
<td>Thijs Westerveld (UT)</td>
<td>Using generative probabilistic models for multimedia retrieval</td>
</tr>
<tr>
<td>Madelon Evers (Nyenrode)</td>
<td>Learning from Design: facilitating multidisciplinary design teams</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Year</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005</td>
<td>Floor Verdenius (UVA) Methodological Aspects of Designing Induction-Based Applications</td>
</tr>
<tr>
<td></td>
<td>Erik van der Werf (UM) AI techniques for the game of Go</td>
</tr>
<tr>
<td></td>
<td>Franc Grootjen (RUN) A Pragmatic Approach to the Conceptualisation of Language</td>
</tr>
<tr>
<td></td>
<td>Nirvana Meratnia (UT) Towards Database Support for Moving Object data</td>
</tr>
<tr>
<td></td>
<td>Gabriel Infante-Lopez (UVA) Two-Level Probabilistic Grammars for Natural Language Parsing</td>
</tr>
<tr>
<td></td>
<td>Pieter Spronck (UM) Adaptive Game AI</td>
</tr>
<tr>
<td></td>
<td>Flavius Frasincar (TUE) Hypermedia Presentation Generation for Semantic Web Information Systems</td>
</tr>
<tr>
<td></td>
<td>Richard Vdovjak (TUE) A Model-driven Approach for Building Distributed Ontology-based Web Applications</td>
</tr>
<tr>
<td></td>
<td>Jeen Broekstra (VU) Storage, Querying and Inferencing for Semantic Web Languages</td>
</tr>
<tr>
<td></td>
<td>Anders Bouwer (UVA) Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments</td>
</tr>
<tr>
<td></td>
<td>Elth Ogston (VU) Agent Based Matchmaking and Clustering—A Decentralized Approach to Search</td>
</tr>
<tr>
<td></td>
<td>Csaba Boer (EUR) Distributed Simulation in Industry</td>
</tr>
<tr>
<td></td>
<td>Fred Hamburg (UL) Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen</td>
</tr>
<tr>
<td></td>
<td>Boris Omelayenko (VU) Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics</td>
</tr>
<tr>
<td></td>
<td>Tibor Bosse (VU) Analysis of the Dynamics of Cognitive Processes</td>
</tr>
<tr>
<td></td>
<td>Joris Graaumans (UU) Usability of XML Query Languages</td>
</tr>
<tr>
<td></td>
<td>Boris Shishkov (TUD) Software Specification Based on Re-usable Business Components</td>
</tr>
</tbody>
</table>
2006
2. Cristina Chisalita (VU) Contextual issues in the design and use of information technology in organizations
3. Noor Christoph (UVA) The role of metacognitive skills in learning to solve problems
4. Marta Sabou (VU) Building Web Service Ontologies
5. Cees Pierik (UU) Validation Techniques for Object-Oriented Proof Outlines
7. Marko Smiljanic (UT) XML schema matching—balancing efficiency and effectiveness by means of clustering
9. Mohamed Wahdan (UM) Automatic Formulation of the Auditor’s Opinion
10. Ronny Siebes (VU) Semantic Routing in Peer-to-Peer Systems
11. Joeri van Ruth (UT) Flattening Queries over Nested Data Types
12. Bert Bongers (VU) Interactivation—Towards an e-ology of people, our technological environment, and the arts
13. Henk-Jan Lebbink (UU) Dialogue and Decision Games for Information Exchanging Agents
15. Rainer Malik (UU) CONAN: Text Mining in the Biomedical Domain
16. Carsten Riggelsen (UU) Approximation Methods for Efficient Learning of Bayesian Networks
17. Stacey Nagata (UU) User Assistance for Multitasking with Interruptions on a Mobile Device
18. Valentijn Zhizhkin (UVA) Graph transformation for Natural Language Processing
20. Marina Velikova (UvT) Monotone models for prediction in data mining
22. Paul de Vrieze (RUN) Fundamentals of Adaptive Personalisation
23. Ion Juvina (UU) Development of Cognitive Model for Navigating on the Web
24. Laura Hollink (VU) Semantic Annotation for Retrieval of Visual Resources
25. Madalina Drugan (UU) Conditional log-likelihood MLE and Evolutionary MCMC
27. Stefano Bocconi (CWI) Vox Populi: generating video documentaries from semantically annotated media repositories
28. Borkur Sigurbjornsson (UVA) Focused Information Access using XML Element Retrieval

2007
1. Kees Leune (UvT) Access Control and Service-Oriented Architectures
2. Wouter Teepe (RUG) Reconciling Information Exchange and Confidentiality: A Formal Approach
3. Peter Mika (VU) Social Networks and the Semantic Web
6. Gilad Mishne (UVA) Applied Text Analytics for Blogs
7. Natasa Jovanovic (UT) To Whom It May Concern—Addressee Identification in Face-to-Face Meetings
8. Mark Hoogendoorn (VU) Modeling of Change in Multi-Agent Organizations
9. David Mobach (VU) Agent-Based Mediated Service Negotiation
10 Huib Aldewereld (UU) Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols
11 Natalia Stash (TUE) Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System
12 Marcel van Gerven (RUN) Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty
13 Rutger Rienks (UT) Meetings in Smart Environments; Implications of Progressing Technology
14 Niek Bergboer (UM) Context-Based Image Analysis
15 Joyca Lacroix (UM) NIM: a Situated Computational Memory Model
16 Davide Grossi (UU) Designing Invisible Handcuffs. Formal investigations in Institutions and Organizations for Multi-agent Systems
17 Theodore Charitos (UU) Reasoning with Dynamic Networks in Practice
18 Bart Orriens (UvT) On the development an management of adaptive business collaborations
19 David Levy (UM) Intimate relationships with artificial partners
20 Slinger Jansen (UU) Customer Configuration Updating in a Software Supply Network
21 Karianne Vermaas (UU) Fast diffusion and broadening use: A research on residential adoption and usage of broadband internet in the Netherlands between 2001 and 2005
22 Zlatko Zlatev (UT) Goal-oriented design of value and process models from patterns
23 Peter Barna (TUE) Specification of Application Logic in Web Information Systems
24 Georgina Ramírez Camps (CWI) Structural Features in XML Retrieval

2008
1 Katalin Boer-Sorbán (EUR) Agent-Based Simulation of Financial Markets: A modular, continuous-time approach
2 Alexei Sharpskykh (VU) On Computer-Aided Methods for Modeling and Analysis of Organizations
3 Vera Hollink (UVA) Optimizing hierarchical menus: a usage-based approach
4 Ander de Keijzer (UT) Management of Uncertain Data—towards unattended integration
5 Bela Mutschler (UT) Modeling and simulating causal dependencies on process-aware information systems from a cost perspective
6 Arjen Hommersom (RUN) On the Application of Formal Methods to Clinical Guidelines, an Artificial Intelligence Perspective
7 Peter van Rosmalen (OU) Supporting the tutor in the design and support of adaptive e-learning
8 Janneke Bolt (UU) Bayesian Networks: Aspects of Approximate Inference
9 Christof van Nimwegen (UU) The paradox of the guided user: assistance can be counter-effective
10 Wauter Bosma (UT) Discourse oriented summarization
11 Vera Kartseva (VU) Designing Controls for Network Organizations: A Value-Based Approach
12 Jozsef Farkas (RUN) A Semiotically Oriented Cognitive Model of Knowledge Representation
13 Caterina Carraciolo (UVA) Topic Driven Access to Scientific Handbooks
14 Arthur van Bunningen (UT) Context-Aware Querying: Better Answers with Less Effort
16 Henriette van Vugt (VU) Embodied agents from a user's perspective
17 Martin Op 't Land (TUD) Applying Architecture and Ontology to the Splitting and Alllying of Enterprises
18 Guido de Croon (UM) Adaptive Active Vision
19 Henning Rode (UT) From Document to Entity Retrieval: Improving Precision and Performance of Focused Text Search
20 Rex Arendsen (UVA) Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie van elektronisch berichtenverkeer met de overheid op de administratieve lasten van bedrijven.
21 Krisztian Balog (UVA) People Search in the Enterprise
22 Henk Koning (UU) Communication of IT-Architecture
23 Stefan Visscher (UU) Bayesian network models for the management of ventilator-associated pneumonia
24 Zharko Aleksovski (VU) Using background knowledge in ontology matching
Geert Jonker (UU) Efficient and Equitable Exchange in Air Traffic Management Plan Repair using Spender-signed Currency

Marjin Huijbregts (UT) Segmentation, Diarization and Speech Transcription: Surprised Data Unraveled

Hubert Vogten (OU) Design and Implementation Strategies for IMS Learning Design

Ildiko Flesch (RUN) On the Use of Independence Relations in Bayesian Networks

Dennis Reidsma (UT) Annotations and Subjective Machines—Of Annotators, Embodied Agents, Users, and Other Humans

Wouter van Atteveldt (VU) Semantic Network Analysis: Techniques for Extracting, Representing and Querying Media Content

Loes Braun (UM) Pro-Active Medical Information Retrieval

Trung H. Bui (UT) Toward Affective Dialogue Management using Partially Observable Markov Decision Processes

Frank Terpstra (UVA) Scientific Workflow Design; theoretical and practical issues

Sietse Overbeek (RUN) Bridging Supply and Demand for Knowledge Intensive Tasks—Based on Knowledge, Cognition, and Quality

Muhammad Subianto (UU) Understanding Categorization

Ronald Poppe (UT) Discriminative Vision-Based Recovery and Recognition of Human Motion

Volker Nannen (VU) Evolutionary Agent-Based Policy Analysis in Dynamic Environments

Benjamin Kanagwa (CWI) Designing Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets

Rinke Hoekstra (UVA) Ontology Representation—Design Patterns and Ontologies that Make Sense

Fritz Reul (UvT) New Architectures in Computer Chess

Laurens van der Maaten (UvT) Feature Extraction from Visual Data

Fabian Groffen (CWI) Armada, An Evolving Database System

Maksym Korotkiy (VU) From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)

Rasja Korotkiy (VU) Symmetric Causal Independence Models

Willem Robert van Hage (VU) Evaluating Ontology-Alignment Techniques

Josephine Nabakenya (RUN) Improving the Quality of Organisational Policy Making using Collaboration Engineering

Sietse Overbeek (RUN) Bridging Supply and Demand for Knowledge Intensive Tasks—Based on Knowledge, Cognition, and Quality

Muhammad Subianto (UU) Understanding Categorization

Ronald Poppe (UT) Discriminative Vision-Based Recovery and Recognition of Human Motion

Volker Nannen (VU) Evolutionary Agent-Based Policy Analysis in Dynamic Environments

Benjamin Kanagwa (CWI) Designing Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets

Rinke Hoekstra (UVA) Ontology Representation—Design Patterns and Ontologies that Make Sense

Fritz Reul (UvT) New Architectures in Computer Chess

Laurens van der Maaten (UvT) Feature Extraction from Visual Data

Fabian Groffen (CWI) Armada, An Evolving Database System

Maksym Korotkiy (VU) From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)
34 Inge van de Weerd (UU) Advancing in Software Product Management: An Incremental Method Engineering Approach
35 Wouter Koelwijn (UL) Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling
36 Marco Kalz (OUN) Placement Support for Learners in Learning Networks
37 Hendrik Drachslar (OUN) Navigation Support for Learners in Informal Learning Networks
38 Riina Vuorikari (OU) Tags and self-organisation: a metadata ecology for learning resources in a multilingual context
39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin) Service Substitution—A Behavioral Approach Based on Petri Nets
40 Stephan Raaijmakers (UvT) Multinomial Language Learning: Investigations into the Geometry of Language
41 Igor Berezhnyy (UvT) Digital Analysis of Paintings
42 Toine Bogers (UvT) Recommender Systems for Social Bookmarking
43 Virginia Nunes Leal Franqueira (UT) Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients
44 Roberto Santana Tapia (UT) Assessing Business-IT Alignment in Networked Organizations
45 Jilles Vreeken (UU) Making Pattern Mining Useful
46 Loredana Afanasiev (UvA) Querying XML: Benchmarks and Recursion

2010
1 Matthijs van Leeuwen (UU) Patterns that Matter
2 Ingo Wassink (UT) Work flows in Life Science
4 Olga Kulyk (UT) Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments
5 Claudia Hauff (UT) Predicting the Effectiveness of Queries and Retrieval Systems
6 Sander Bakkes (UvT) Rapid Adaptation of Video Game AI
7 Wim Fikkert (UT) Gesture interaction at a Distance
8 Krzysztof Siewicz (UL) Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments
9 Hugo Kielman (UL) A Politiegegevensverwerking en Privacy, Naar een effectieve waarborging
10 Rebecca Ong (UL) Mobile Communication and Protection of Children
11 Adriaan Ter Mors (TUD) The world according to MARP: Multi-Agent Route Planning
12 Susan van den Braak (UvA) Sensemaking software for crime analysis
13 Gianluigi Folino (RUN) High Performance Data Mining using Bio-inspired techniques
14 Sander van Splunteren (VU) Automated Web Service Reconfiguration
15 Lianne Bodenstaff (UT) Managing Dependency Relations in Inter-Organizational Models
16 Sicco Verwer (TUD) Efficient Identification of Timed Automata, theory and practice
17 Spyros Kotoulas (VU) Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications
18 Charlotte Gerritsen (VU) Caught in the Act: Investigating Crime by Agent-Based Simulation
19 Henriette Cramer (UvA) People's Responses to Autonomous and Adaptive Systems
20 Ivo Swartjes (UT) Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative
21 Harold van Heerde (UT) Privacy-aware data management by means of data degradation
22 Michiel Hildebrand (CWI) End-user Support for Access to Heterogeneous Linked Data
23 Bas Steunebrink (UU) The Logical Structure of Emotions
24 Dmytro Tykhonov Designing Generic and Efficient Negotiation Strategies
25 Zulfiqar Ali Memon (VU) Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective
26 Ying Zhang (CWI) XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines
27 Marten Voulon (UL) Automatisch contracteren
28 Arne Koopman (UU) Characteristic Relational Patterns
29 Stratos Ideros (CWI) Database Cracking: Towards Auto-tuning Database Kernels
30 Marieke van Erp (UvT) Accessing Natural History—Discoveries in data cleaning, structuring, and retrieval
31 Victor de Boer (UvA) Ontology Enrichment from Heterogeneous Sources on the Web
32 Marcel Hiel (UvT) An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems
33 Robin Aly (UT) Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval
34 Tedyu Dirghayat (UT) Interaction Design in Service Compositions
35 Dolf Trieschnigg (UT) Proof of Concept: Concept-based Biomedical Information Retrieval
36 Jose Janssen (OU) Paving the Way for Lifelong Learning: Facilitating competence development through a learning path specification
37 Niels Lohmann (TUE) Correctness of services and their composition
38 Dirk Fahland (TUE) From Scenarios to components
39 Ghazanfar Farooq Siddiqui (VU) Integrative modeling of emotions in virtual agents
40 Mark van Assem (VU) Converting and Integrating Vocabularies for the Semantic Web
41 Guillaume Chaslot (UM) Monte-Carlo Tree Search
42 Sybren de Kinderen (VU) Needs-driven service bundling in a multi-supplier setting—the computational e-service approach
43 Peter van Kranenburg (UU) A Computational Approach to Content-Based Retrieval of Folk Song Melodies
44 Pieter Bellekens (TUE) An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain
45 Vasilios Andrikopoulos (UvT) A theory and model for the evolution of software services
46 Vincent Pijpers (VU) eAlignment: Exploring Inter-Organizational Business-ICT Alignment
47 Chen Li (UT) Mining Process Model Variants: Challenges, Techniques, Examples
48 Withdrawn
49 Jahn-Takeshi Saito (UM) Solving difficult game positions
50 Bouke Huurnink (UVA) Search in Audiovisual Broadcast Archives
51 Alia Khairia Amin (CWI) Understanding and supporting information seeking tasks in multiple sources
52 Peter-Paul van Maanen (VU) Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention
53 Edgar Meij (UVA) Combining Concepts and Language Models for Information Access

2011

1 Botond Cseke (RUN) Variational Algorithms for Bayesian Inference in Latent Gaussian Models
2 Nick Tinnemeier (UU) Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language
3 Jan Martijn van der Werf (TUE) Compositional Design and Verification of Component-Based Information Systems
4 Hado van Hasselt (UU) Insights in Reinforcement Learning: Formal analysis and empirical evaluation of temporal-difference learning algorithms
5 Base van der Raadt (VU) Enterprise Architecture Coming of Age—Increasing the Performance of an Emerging Discipline.
6 Yiwen Wang (TUE) Semantically-Enhanced Recommendations in Cultural Heritage
7 Yujia Cao (UT) Multimodal Information Presentation for High Load Human Computer Interaction
8 Nieske Vergunst (UU) BDI-based Generation of Robust Task-Oriented Dialogues
9 Tim de Jong (OU) Contextualised Mobile Media for Learning
10 Bart Bogaert (UvT) Cloud Content Contention
11 Dhaval Vyas (UT) Designing for Awareness: An Experience-focused HCI Perspective
12 Carmen Bratosin (TUE) Grid Architecture for Distributed Process Mining
13 Xiaoyu Mao (UvT) Airport under Control. Multiagent Scheduling for Airport Ground Handling
14 Milan Lovric (EUR) Behavioral Finance and Agent-Based Artificial Markets
15 Marijn Koolen (UvA) The Meaning of Structure: the Value of Link Evidence for Information Retrieval
16 Maarten Schadd (UM) Selective Search in Games of Different Complexity
17 Jiyin He (UVA) Exploring Topic Structure: Coherence, Diversity and Relatedness
Mark Ponsen (UM) Strategic Decision-Making in complex games

Ellen Rusman (OU) The Mind’s Eye on Personal Profiles

Qing Gu (VU) Guiding service-oriented software engineering—A view-based approach

Linda Terlouw (TUD) Modularization and Specification of Service-Oriented Systems

Junte Zhang (UVA) System Evaluation of Archival Description and Access

Wouter Weerkamp (UVA) Finding People and their Utterances in Social Media

Herwin van Welbergen (UT) Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior

Syed Waqar ul Qounain Jaffry (VU) Analysis and Validation of Models for Trust Dynamics

Matthijs Aart Pontier (VU) Virtual Agents for Human Communication—Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots

Aniel Bhulai (VU) Dynamic website optimization through autonomous management of design patterns

Rianne Kaptein (UVA) Effective Focused Retrieval by Exploiting Query Context and Document Structure

Faisal Kamiran (TUE) Discrimination-aware Classification

Egon van den Broek (UT) Affective Signal Processing (ASP): Unraveling the mystery of emotions

Ludo Waltman (EUR) Computational and Game-Theoretic Approaches for Modeling Bounded Rationality

Nees-Jan van Eck (EUR) Methodological Advances in Bibliometric Mapping of Science

Tom van der Weide (UU) Arguing to Motivate Decisions

Paolo Turrini (UU) Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations

Maaike Harbers (UU) Explaining Agent Behavior in Virtual Training

Erik van der Spek (UU) Experiments in serious game design: a cognitive approach

Adriana Burlutiu (RUN) Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference

Nyree Lemmens (UM) Bee-inspired Distributed Optimization

Joost Westra (UU) Organizing Adaptation using Agents in Serious Games

Viktor Clerc (VU) Architectural Knowledge Management in Global Software Development

Luan Ibraimi (UT) Cryptographically Enforced Distributed Data Access Control

Michal Sindlar (UU) Explaining Behavior through Mental State Attribution

Henk van der Schuur (UU) Process Improvement through Software Operation Knowledge

Boris Reuderink (UT) Robust Brain-Computer Interfaces

Herman Stehouwer (UvT) Statistical Language Models for Alternative Sequence Selection

Beibei Hu (TUD) Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work

Aziz Bin Ab Aziz (VU) Exploring Computational Models for Intelligent Support of Persons with Depression

Mark Ter Maat (UT) Response Selection and Turn-taking for a Sensitive Artificial Listening Agent

Andreea Niculescu (UT) Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality

2012

Terry Kakeeto (UvT) Relationship Marketing for SMEs in Uganda

Muhammad Umair (VU) Adaptivity, emotion, and Rationality in Human and Ambient Agent Models

Adam Vanya (VU) Supporting Architecture Evolution by Mining Software Repositories

Jurriaan Souer (UU) Development of Content Management System-based Web Applications

Marijn Plomp (UU) Maturing Interorganisational Information Systems

Wolfgang Reinhardt (OU) Awareness Support for Knowledge Workers in Research Networks

Rianne van Lambalgen (UvT) When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions

Gerbende Vries (UVA) Kernel Methods for Vessel Trajectories

Ricardo Neisse (UT) Trust and Privacy Management Support for Context-Aware Service Platforms
10. David Smits (TUE) Towards a Generic Distributed Adaptive Hypermedia Environment
11. J.C.B. Rantham Prabhakara (TUE) Process Mining in the Large: Preprocessing, Discovery, and Diagnostics
12. Kees van der Sluijs (TUE) Model Driven Design and Data Integration in Semantic Web Information Systems
13. Suleman Shahid (UvT) Fun and Face: Exploring non-verbal expressions of emotion during playful interactions
16. Fiemke Both (VU) Helping people by understanding them—Ambient Agents supporting task execution and depression treatment
17. Amal Elgammal (UvT) Towards a Comprehensive Framework for Business Process Compliance
18. Eltjo Poort (VU) Improving Solution Architecting Practices
20. Ali Bahramisharif (RUN) Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing
21. Roberto Cornacchia (TUD) Querying Sparse Matrices for Information Retrieval
22. Thijs Vis (UvT) Intelligence, politie en veiligheidsdienst: verenigbare grootheden?
23. Christian Muehl (UT) Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction
24. Laurens van der Werff (UT) Evaluation of Noisy Transcripts for Spoken Document Retrieval
25. Silja Eckartz (UT) Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application
26. Emile de Maat (UVA) Making Sense of Legal Text
27. Hayrettin Gurkok (UT) Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games
29. Almer Tigelaar (UT) Peer-to-Peer Information Retrieval
30. Alina Pommeranz (TUD) Designing Human-Centered Systems for Reflective Decision Making
31. Emily Bagarukayo (RUN) A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure
32. Wietse Visser (TUD) Qualitative multi-criteria preference representation and reasoning
33. Rory Sie (OUN) Coalitions in Cooperation Networks (COCOON)
34. Pavol Jancura (RUN) Evolutionary analysis in PPI networks and applications
35. Evert Haasdijk (VU) Never Too Old To Learn—On-line Evolution of Controllers in Swarm-and Modular Robotics
36. Denis Seebugwawo (RUN) Analysis and Evaluation of Collaborative Modeling Processes
38. Selmar Smit (VU) Parameter Tuning and Scientific Testing in Evolutionary Algorithms
39. Hassan Fatemi (UT) Risk-aware design of value and coordination networks
40. Agus Gunawan (UvT) Information Access for SMEs in Indonesia
41. Sebastien Kelle (OU) Game Design Patterns for Learning
42. Dominique Verpoorten (OU) Reflection Amplifiers in self-regulated Learning
43. Withdrawn
44. Anna Tordai (VU) On Combining Alignment Techniques
45. Benedikt Kratz (UvT) A Model and Language for Business-aware Transactions
46. Simon Carter (UVA) Exploration and Exploitation of Multilingual Data for Statistical Machine Translation
47. Manos Tsagkias (UVA) Mining Social Media: Tracking Content and Predicting Behavior
48. Jorn Bakker (TUE) Handling Abrupt Changes in Evolving Time-series Data
49. Michael Kaisers (UM) Learning against Learning—Evolutionary dynamics of reinforcement learning algorithms in strategic interactions
50. Steven van Kervel (TUD) Ontology driven Enterprise Information Systems Engineering
51. Jeroen de Jong (TUD) Heuristics in Dynamic Sceduling: a practical framework with a case study in elevator dispatching
2013
1 Viorel Milea (EUR) News Analytics for Financial Decision Support
2 Errietta Liarou (CWI) MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing
3 Szymon Klaman (VU) Reasoning with Contexts in Description Logics
4 Chetan Yadati (TUD) Coordinating autonomous planning and scheduling
5 Dulce Pumareja (UT) Groupware Requirements Evolutions Patterns
6 Romulo Goncalves (CWI) The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience
7 Giel van Lankveld (UvT) Quantifying Individual Player Differences
8 Robbert-Jan Merk (VU) Making enemies: cognitive modeling for opponent agents in fighter pilot simulators
9 Fabio Gori (RUN) Metagenomic Data Analysis: Computational Methods and Applications
10 Jeewanie Jayasinghe Arachchige (UvT) A Unified Modeling Framework for Service Design
11 Evangelos Pournaras (TUD) Multi-level Reconfigurable Self-organization in Overlay Services
12 Marian Razavian (VU) Knowledge-driven Migration to Services
13 Mohammad Safiri (UT) Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly
14 Jafar Tanha (UVA) Ensemble Approaches to Semi-Supervised Learning Learning
15 Daniel Hennes (UM) Multitagant Learning—Dynamic Games and Applications
16 Eric Kok (UU) Exploring the practical benefits of argumentation in multi-agent deliberation
17 Koen Kok (VU) The PowerMatcher: Smart Coordination for the Smart Electricity Grid
18 Jeroen Janssens (UvT) Outlier Selection and One-Class Classification
19 Renze Steenhuizen (TUD) Coordinated Multi-Agent Planning and Scheduling
20 Katja Hofmann (UvA) Fast and Reliable Online Learning to Rank for Information Retrieval
21 Sander Wubben (UvT) Text-to-text generation by monolingual machine translation
22 Tom Claassen (RUN) Causal Discovery and Logic
23 Patricio de Alencar Silva (UvT) Value Activity Monitoring
24 Haitham Bou Ammar (UM) Automated Transfer in Reinforcement Learning
26 Alireza Zarghami (UT) Architectural Support for Dynamic Homecare Service Provisioning
27 Mohammad Huq (UT) Inference-based Framework Managing Data Provenance
28 Frans van der Sluis (UT) When Complexity becomes Interesting: An Inquiry into the Information Experience
29 Iwan de Kok (UT) Listening Heads
31 Dinh Khoa Nguyen (UvT) Blueprint Model and Language for Engineering Cloud Applications
32 Kamakshi Rajagopal (OUN) Networking For Learning: The role of Networking in a Lifelong Learner's Professional Development
33 Qi Gao (TUD) User Modeling and Personalization in the Microblogging Sphere
34 Kien Tjin-Kam-Jet (UT) Distributed Deep Web Search
35 Abdallah El Ali (UvA) Minimal Mobile Human Computer Interaction
36 Tham Lan Hoang (TUE) Pattern Mining in Data Streams
37 Dirk Börner (OUN) Ambient Learning Displays
38 Eelco den Heijer (VU) Autonomous Evolutionary Art
39 Joop de Jong (TUD) A Method for Enterprise Ontology based Design of Enterprise Information Systems
40 Pim Nijssen (UM) Monte-Carlo Tree Search for Multi-Player Games
41 Jochem Liem (UVA) Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning
42 Léon Planken (TUD) Algorithms for Simple Temporal Reasoning
43 Marc Bron (UVA) Exploration and Contextualization through Interaction and Concepts

2014
1 Nicola Barile (UU) Studies in Learning Monotone Models from Data
2 Fiona Tuliyano (RUN) Combining System Dynamics with a Domain Modeling Method
3 Sergio Raul Duarte Torres (UT) Information Retrieval for Children: Search Behavior and Solutions
4 Hanna Jochmann-Mannak (UT) Websites for children: search strategies and interface design—Three studies on children's search performance and evaluation
5 Jurriaan van Reijisen (UU) Knowledge Perspectives on Advancing Dynamic Capability
6 Damian Tamburri (VU) Supporting Networked Software Development
7 Arya Adriansyah (TUE) Aligning Observed and Modeled Behavior
8 Samur Araujo (TUD) Data Integration over Distributed and Heterogeneous Data Endpoints
9 Philip Jackson (UvT) Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language
10 Ivan Salvador Razo Zapata (VU) Service Value Networks
11 Janneke van der Zwaan (TUD) An Empathic Virtual Buddy for Social Support
12 Willem van Willigen (VU) Look Ma, No Hands: Aspects of Autonomous Vehicle Control
13 Arlette van Wissen (VU) Agent-Based Support for Behavior Change: Models and Applications in Health and Safety Domains
14 Yangyang Shi (TUD) Language Models With Meta-information
15 Natalya Mogles (VU) Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare
16 Krystyna Milian (VU) Supporting trial recruitment and design by automatically interpreting eligibility criteria
17 Kathrin Dentler (VU) Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability
18 Mattijs Ghijsten (VU) Methods and Models for the Design and Study of Dynamic Agent Organizations
19 Vinicius Ramos (TUE) Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support
20 Mena Habib (UT) Named Entity Extraction and Disambiguation for Informal Text: The Missing Link
21 Kassidy Clark (TUD) Negotiation and Monitoring in Open Environments
22 Marieke Peeters (UU) Personalized Educational Games—Developing agent-supported scenario-based training
23 Eleftherios Sidirourgos (UvA/CWI) Space Efficient Indexes for the Big Data Era
24 Davide Ceolin (VU) Trusting Semi-structured Web Data
25 Martijn Lappenschaar (RUN) New network models for the analysis of disease interaction
26 Tim Baarslag (TUD) What to Bid and When to Stop
28 Anna Chmielowiec (VU) Decentralized k-Clique Matching
29 Jaap Kabbedijk (UU) Variability in Multi-Tenant Enterprise Software
30 Peter de Cock (UvT) Anticipating Criminal Behaviour
31 Leo van Moergestel (UU) Agent Technology in Agile Multiparallel Manufacturing and Product Support
32 Naser Ayat (UvA) On Entity Resolution in Probabilistic Data
33 Tesfa Tegegne (RUN) Service Discovery in eHealth
34 Christina Manteli (VU) The Effect of Governance in Global Software Development: Analyzing Transaction Memory Systems.
36 Joos Buijs (TUE) Flexible Evolutionary Algorithms for Mining Structured Process Models
37 Maral Dadvar (UT) Experts and Machines United Against Cyberbullying
38 Danny Plass-Oude Bos (UT) Making brain-computer interfaces better: improving usability through post-processing.
39 Jasmina Maric (UvT) Web Communities, Immigration, and Social Capital
40 Walter Omona (RUN) A Framework for Knowledge Management Using ICT in Higher Education
41 Frederic Hogenboom (EUR) Automated Detection of Financial Events in News Text
42 Carsten Eijckhof (CWI/TUD) Contextual Multidimensional Relevance Models
43 Kevin Vlaanderen (UU) Supporting Process Improvement using Method Increments
Paulien Meesters (UvT) Intelligent Blauw. Met als ondertitel: Intelligence-gestuurde politiezorg in gebiedsgebonden eenheden.

Birgit Schmitz (OUN) Mobile Games for Learning: A Pattern-Based Approach

Ke Tao (TUD) Social Web Data Analytics: Relevance, Redundancy, Diversity

Shangsong Liang (UVA) Fusion and Diversification in Information Retrieval

2015

1. Niels Netten (UvA) Machine Learning for Relevance of Information in Crisis Response
2. Faiza Bukhsh (UvT) Smart auditing: Innovative Compliance Checking in Customs Controls
3. Twan van Laarhoven (RUN) Machine learning for network data
4. Howard Spoelstra (OUN) Collaborations in Open Learning Environments
5. Christoph Bösch (UT) Cryptographically Enforced Search Pattern Hiding
7. Maria-Hendrike Peetz (UvA) Time-Aware Online Reputation Analysis
8. Jie Jiang (TUD) Organizational Compliance: An agent-based model for designing and evaluating organizational interactions
9. Randy Klaassen (UT) HCI Perspectives on Behavior Change Support Systems
10. Henry Hermans (OUN) OpenU: design of an integrated system to support lifelong learning
11. Yongming Luo (TUE) Designing algorithms for big graph datasets: A study of computing bisimulation and joins
14. Bart van Straalen (UT) A cognitive approach to modeling bad news conversations
15. Klaas Andries de Graaf (VU) Ontology-based Software Architecture Documentation
16. Changyun Wei (UT) Cognitive Coordination for Cooperative Multi-Robot Teamwork
18. Holger Pirk (CWI) Waste Not, Want Not!—Managing Relational Data in Asymmetric Memories
19. Bernardo Tabuenca (OUN) Ubiquitous Technology for Lifelong Learners
20. Loïs Vanhée (UU) Using Culture and Values to Support Flexible Coordination
21. Sibren Fetter (OUN) Using Peer-Support to Expand and Stabilize Online Learning
22. Zhemin Zhu (UT) Co-occurrence Rate Networks
23. Luit Gazendam (VU) Cataloguer Support in Cultural Heritage
25. Steven Woudenberg (UU) Bayesian Tools for Early Disease Detection
27. Sándor Héman (CWI) Updating compressed column-stores
28. Janet Bagorogoza (TiU) Knowledge Management and High Performance; The Uganda Financial Institutions Model for HPO
29. Hendrik Baier (UM) Monte-Carlo Tree Search Enhancements for One-Player and Two-Player Domains
31. Yakup Koç (TUD) On Robustness of Power Grids
32. Jerome Gard (UL) Corporate Venture Management in SMEs
33. Frederik Schadd (UM) Ontology Mapping with Auxiliary Resources
34. Victor de Graaff (UT) Geosocial Recommender Systems

2016

1. Syed Sained Abbas (RUN) Recognition of Shapes by Humans and Machines
2. Michiel Christiaan Meulendijk (UU) Optimizing medication reviews through decision support: prescribing a better pill to swallow
4 Laurens Rietveld (VU) *Publishing and Consuming Linked Data*
5 Evgeny Sherkhonov (UVA) *Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers*
6 Michel Wilson (TUD) *Robust scheduling in an uncertain environment*