Beyond Fixed-Time Scheduling

Kiriakos Simon Mountakis

October 30, 2017

Contents

List of Figures

List of Tables

List of Algorithms

Preface

A preface generally covers the story of how the book came into being, or how the idea for the book was developed; this is often followed by thanks and acknowledgments to people who were helpful to the author during the time of writing.

Chapter 1

Introduction

The subject of this thesis is inspired by a real scheduling problem as experienced by the NedTrain company.[1] After a general introduction to the NedTrain company, we identify two main problems in scheduling for maintenance engineering which according to our view are not only problems this particular company is facing, but also are of general interest for the research field of scheduling. After a description of these research problems, in subsequent chapters we derive some scientific research questions from these research problems and analyse them in full detail.

1.1 NS Group and NedTrain

NedTrain is a subsidiary company of Nederlandse Spoorwegen (NS) Group, the principal Dutch railway operator, the origins of which can be traced back to the begining of the 19-th century. NS, which in 2016 made close to 90% of their revenue from passenger trasport, serves approximately 1 million passengers per day mainly in the Netherlands but also in Germany, the United Kingdom (UK) and other parts of Europe [66]. Figure 1.1 shows the group of companies composing NS: NS Reizigers (or NSR, with approximately 11,000 staff), NedTrain (3,000), NS Stations (5,000, including retail) and Abellio (13,000)

NS Reizigers and NS International operate the majority of trains on the Dutch rail network. They handle domestic and international passenger transportation, respectively, with a fleet of approximately 3000 rolling-stock units. Abelio handles passenger transportation in the UK and in Germany. NS Stations handles the development and operation of 410 large and small train stations, in cooperation with ProRail which is the company managing the Dutch rail network (not part of NS Group).

NedTrain is responsible for maintaining a high availability rate for NS Reizigers and NS International trains, providing the following types of maintenace:

[1] www.nedtrain.nl

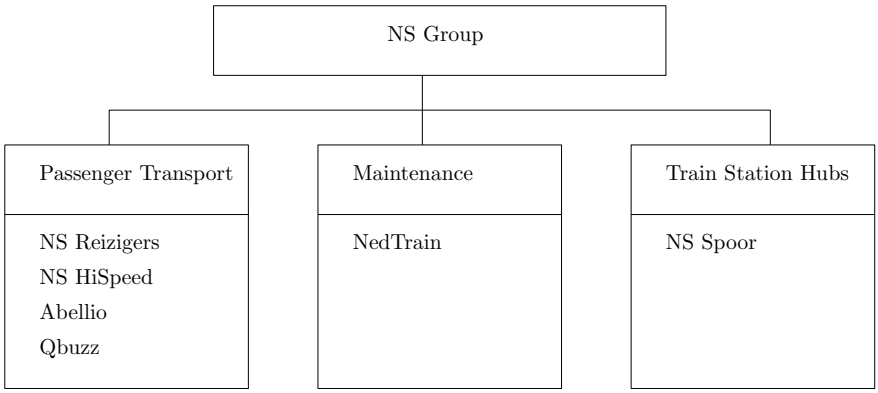


Figure 1.1: NS Group and subsidiary companies (adapted from [66]).

First-line service First-line service involves a daily cleaning and fixing of small technical problems of each train at least once a day at one of thirty 'servicebedrijf' (SB) facilities throughout the country.

Technical maintenance Once every three months or after a critical part has reached a certain mileage each train goes to one of four NedTrain workshop, or 'onderhoudsbedrijf' (OB) facilities for technical maintenance.

Refurbishment Once or twice in its lifetime, a train might have to be refurbished to meet modern standards, and refurbishment also takes place in OB facilities.

The work presented in this dissertation deals with scheduling problems related to technical maintenance operations in a NedTrain workshop. NedTrain has four workshops for technical maintenance in The Netherlands: in Amsterdam, Leidschendam, Onnen and Maastricht, and each location specializes in specific train types (Figure 1.2,1.3). Each workshop operates non-stop, 24 hours a day, covered by three 8-hour personnel shifts. For each workshop there is a week-long "abstract" schedule that mostly stays the same throughout the year. This schedule specifies when and which types of trains are expected to arrive at the workshop and also when they must be returned for circulation in the rail network. This week-long abstract schedule is designed in harmony with the week-long time-tables used by NS Reizigers to carry passengers and such that NedTrain handles an evenly distributed workload over the year.

Even though the abstract schedule stays more-or-less fixed throughout the year, the scheduling problem to be dealt with in a workshop changes on a weekly basis. This is because the list of necessary maintenance tasks for each specific train-fleet unit depends on its past visits at the workshop and on its current condition. That is, each week the workshop operates based on an instantiation of the abstract schedule, depending on the particular train units that will arrive for maintenance. In other words, the abstract schedule simply specifies arrival

and due-dates for types of trains, since from a passenger service point-of-view, it is only important that some train unit of a specific type is available when needed.

About two weeks prior to an upcomming week, knowledge about which specific trains will arrive at the depot becomes available. Each train is expected to arrive on a respective *release-date* and it must be delivered for circulation before a respective *due-date*. If train constitutes a maintenance project then multiple projects will typically execute in parallel at the workshop. Depending on weather conditions and other factors, the number of trains under technical maintenance in NedTrain workshops at the same time might peek to 300, i.e. to about 10% of the entire fleet. With about 50 tasks per train on average, up to 1500 tasks with uncertain durations might be taken into consideration when creating a weekly workshop schedule.

1.2 NedTrain's Research & Development program

NedTrain has the ambition to be a first class European rolling stock maintenance company. To support their ambition, NedTrain intiated the 'Rolling Stock Life-Cycle Logistics' (RSLCL) applied research and development program, in cooperation with several Dutch universities. The main question to be addressed by the RSLCL program can be formulated as follows:

> How to obtain and maintain the best combination of rolling stock, maintenance operations and supply chain, within the context of railway operations, to enable our customers to deliver competitive high quality services to their passengers?

The search for an answer to this question is mostly directed by the following Key Performance Indicators (KPIs) concerning the train-fleet of NS:

1. total Cost of Ownership (or Life-Cycle Cost) of the fleet;
2. availability of sufficient transport capacity;
3. reliability of transport;
4. quality of transport.

The scope of the RSLCL program is too wide to be undertaken as a single study. As such, the program has been divided into three different levels: the strategic, tactical and operational level, each associated with a different time-scale of planning and control. In what follows we give a summary for each of the three levels of the program.

Strategic Since rolling stock has several decades of life-time, decisions regarding the acquisition of rolling stock constitute long-lasting investments that tie

up resources for years. Moreover, buying rolling stock amounts to less than 40% of the overall life-cycle cost. That is, most money is spent on operation and maintenance during the years rolling stock is in service. Since this amount of money is mostly allocated during the acquisition process, the main objective at the strategic level is the development of methods for ranking acquisition options from the perspective of *supportability* and the design optimal logistics support.

Tactical The tactical level concerns the allocation and planning of spare parts, human resources and maintenance tasks and the management of physical flows, given that decisions about the acquisition of rolling stock and the maintenance infrastructure have been made and will remain fixed on the intermediate timescale. Moreover, the tactical level considers the impact of changing the supply chain in order to outsource some of the maintenance operations to the manufacturer of rolling stock.

Operational The operational level concerns the effective scheduling of maintenance operations in a NedTrain workshop, in order to ensure timely delivery of rolling stock for circulation in the rail network. Maintenance tasks having uncertain durations (partly because of the conditional nature of repairs, i.e. not knowing which repairs must be performed until after the arrival and inspection of a train in the workshop) makes the scheduling of technical maintenance a rather challenging feat. Due to the possibility of maintenance running late, ensuring a certain level of fleet availability relies on buffer stock. The main objective at the operational level is to reduce the total cost of fleet ownership by improving the planning and scheduling process at the workshop, which in turn would allow the level of buffer stock to decrease.

The study associated with each of the three branches of the research program is undertaken as a separate PhD project at a corresponding university. Research concerning the strategic level has been conducted by Parada Puig et al. [80] at the University of Twente and research concerning the tactical level has been conducted by Arts et al. [5] at the Technical University of Eindhoven. Research concerning the operational level is underaken by Wilson et al. [96] at the Technical University of Delft. As a follow-up to the work of Wilson et al., this dissertation also focuses on the operational level, i.e. the scheduling of maintenance tasks in a NedTrain workshop, in the face of uncertainty. The overall research effort concerning all three levels of the program is monitored and directed at regular Steering Group meetings that take place (up to) four times a year, involving a representative for each of the stakeholders.

In what follows we focus on the current scheduling process at NedTrain and highlight the basic issues caused by the presence of uncertainty. Moreover, we propose two potential approaches for better dealing with uncertainty and formulate corresponding research problems that we will be addressing in our work.

Figure 1.2: Inside the Leidschendam workshop.

Figure 1.3: Shunting yard of the workshop in Onnen.

1.3 Basic issues with maintenance scheduling

As with most (if not all) real-life applications of scheduling, the main complication in NedTrain's case is that tasks have uncertain durations. In a maintenance workshop, dismounting parts is a very common operation. In contrast with assembling a part (as in production), the time needed for a dismount tends to vary according to uncontrollable factors. Another source of uncertainty is due to the conditional nature of repairs. That is, some tasks involve inspecting a certain part and then repairing it, if necessary. The duration of such tasks is by definition highly uncertain since whether any time will be spent on repairing the part is unknown in advance.

A schedule, on the other hand, is created subject to temporal and resource constraints and allocates a fixed amount of time per task. If the amount of time needed (i.e. the outcome duration) exceeds the amount of time allocated for a task (i.e. the predicted duration), then some other tasks might have to be rescheduled in order to comply with temporal constraints and/or resource constraints. As a result, scheduling in an uncertain environment is not so much about finding a good schedule as it is about keeping the schedule up-to-date with outcome durations without deteriorating its quality in the long run.

Currently, the scheduling process at NedTrain is semi-automated. A baseline schedule is created and adapted during task execution as necessary by a team of planners who rely on their domain expertise and the help of software like ProPlan or Microsoft Excel. To cope with uncertainty, a significant amount of

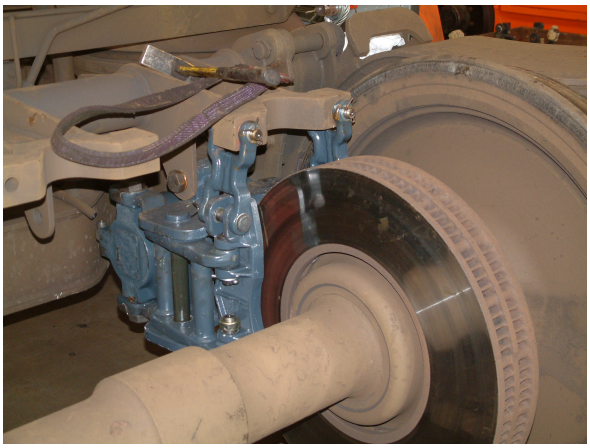


Figure 1.4: Replacement of a faulty part.

slack is inserted in the schedule. Moreover, as a last resort option, some tasks might be skipped and postponed for the next workshop visit in order to deliver a train on time.

Adding too much slack stabilizes dispatching times (meaning they will probably not change much during task execution) but at the same time it compromises efficiency or punctuality, because the schedule becomes long. Adding too little slack, on the other hand, means we will have to update the schedule often during execution. Aggressive and frequent rescheduling of human resources is highly undesirable as it creates confusion, or nervousness, which in the end also hinders performance.

In the NedTrain workshop we are dealing with large scheduling problems with several hundreds of tasks and constraints and a great deal of uncertainty. The demand to keep the schedule up-to-date with a changing environment without introducing confusion, or "shop-floor nervousness" while achieving a high degree of timeliness, might easily outstrip the capacity of human planners. The current approach works but there is clearly room for improvement, as we see more in detail in the next section. The mission of NedTrain's R&D, at the operational level, is to modernize the schedulng process by finding state-of-the-art scheduling techniques that can be adapted and/or extended for the particular requirements of the NedTrain workshop. Based on eventual research findings, the purpose is to develop custom-tailored scheduling tools that can help operational planners to enhance punctuality and eliminate nervousness at the workshop, without sacrificing throughput.

1.4 Research Problems

Since human resources are very much against the idea of being rescheduled frequently, management has to choose between two options:

(I) Adding sufficient slack to avoid the need for continuous rescheduling (nervous process), or

(II) give human resources the autonomy to take the decisions to (re)schedule themselves.

In other words, unless management is able to provide a schedule that remains stable (i.e. relatively unchanged) during execution, people would rather have the freedom to (re)schedule themselves and stay in control.

Pursuing option two (letting people reschedule themselves and be in control) involves transitioning from regular schedules, padded with slack, to *flexible schedules*. Instead of specific dispatching times, a flexible schedule prescribes a space of potential schedules, from which people can pick suitable dispatching times on-the-fly.

The main difficulty with letting people choose their own dispatching times is ensuring that scheduling constraints will be satisfied, i.e. the realization of a feasible schedule. Human resources in the NedTrain workshop are organized in groups, or work-teams. Each team is led by a corresponding foreman and is responsible for the completion of a certain subset of tasks. Moreover, each team would like to be able to plan-ahead and operate as an independent unit. Tasks belonging to different teams are usually interrelated, however, because of having to share workshop resources and satisfy certain temporal constraints between tasks.

Ensuring the generation of a schedule satisfying workshop constraints should not rely on synchronous communication between teams; it is important that teams retain their independence. That is, teams should not be expected to negotiate with other teams over the dispatching of hundreds of tasks. This would certainly compromise performance and create confusion. The main challenge in developing such a flexible scheduling technique, then, is ensuring constraint satisfaction by dispatching decisions taken in isolation. To make this possible, a flexible schedule should provide appropriate boundaries within which individual teams can make decisions, while offering as much flexibility as possible. In pursuit of such scheduling techniques, the first part of this thesis is devoted to addressing the following problem:

Research Problem I.
How to compute flexible schedules for independent work-teams that can be easily adapted to changes in the environment?

Having discussed the potential of pursuing option two, we will now also consider the potential to pursue the more traditional first option, i.e. using regular schedules with sufficient slack to avoid continuous rescheduling. In line with existing literature, we shall base our discussion on the concepts of stability and robustness. Stability refers to the quality to preserve dispatching times relatively unchanged during task execution. Robustness, on the other hand, guarantees good performance or timeliness with respect to the given due-dates (and does not imply stability).

Determining at which points in the schedule and at what quantities should slack be inserted in order to strike a good balance between stability and robustness can be difficult, because of the intricate manner in which uncertainty accumulates in a schedule. For example, more slack is needed near the end of a schedule since tasks that start later are susceptible to the accumulated effects of uncertainty, in contrast with tasks that started earlier. Moreover, more slack is necessary in order to protect the dispatching time of a task with many temporal dependencies, and so on. In pursuit of a sophisticated and fully automated method for inserting slack, we are interested in dealing with the following problem:

Research Problem II.
How to compute robust and stable schedules for work-teams in order to deal with uncertainty in the duration of maintenance tasks?

We then considered two potential methods for dealing with uncertainty in task durations and formulated two corresponding Research Problems. Addressing these problems, which we feel might be of interest to other organizations besides NedTrain, is precisely the purpose of our research. The next section outlines the structure of the dissertation, which consists of Part I and Part II, each devoted to the corresponding Research Problem.

1.5 Organization of the thesis

Our research findings concerning the two approaches for dealing with uncertainty discussed earlier are presented, respectively, in Part I and Part II. Despite their differences, Part I and Part II have an underlying theme in common: beyond finding a fixed schedule for a given scheduling problem, we are interested in finding a strategy for adjusting, or generating the schedule, in a continuous scheduling process.

Part I (Chapters 2,3,4) focuses on the approach of letting people (re)schedule themselves in the workshop. To enable this approach from a technical standpoint, we turn to the research area concerning Simple Temporal Porblem (STP) constraints. Such constraints restrict the minimum and maximum temporal distance between the dispatching of pairs of events. It is assumed that each event is associated with a respective actor that will choose when to dispatch his event from a respective time interval, in a non-deterministic manner. We are interested in finding a *flexible* strategy for initializing these time intervals and keeping them up-to-date as choices are being made, striving to maximize the freedom (or flexibility) with which actors can make choices. At the same time, we are also guaranteeing the formation of a feasible schedule regardless of the non-determinism with which actors choose within those intervals. Chapter 2 summarizes important concepts from the research area of STPs. Emphasis is put on the work of Wilson et al. [96], in which the use of STPs for the NedTrain workshop was originally examined, and whose findings we extend in this thesis. Following the summary of important concepts, we view Research Problem I through the prism of STP-related scheduling frameworks and break it down into

more specific Research Questions I.1, I.2, and I.3. Chapters 3 and 4 present our research findings for answering these questions, effectively addressing Resarch Problem I at a technical level.

Part II (Chapters 5,6,7) focuses on generating a schedule with sufficient slack to absorb the effects of uncertainty, prescribing a stable dispatching time for each task. To enable this approach from a technical standpoint we turn to the research area of Stochastic Task Networks, i.e. networks of precedence constraints between pairs of tasks with random durations. This second approach involves utilizing information from previous maintenance sessions in order to model the uncertain duration of each task as a random variable with a known probability distribution. We are interested in finding a dispatching strategy that enables us to predict with some confidence the outcome start-times of the tasks, even though the outcome task durations are unpredictable. More in particular, we are interested in finding a strategy that optimizes the trade-off between two conflicting qualities: dispatching tasks efficiently and dispatching tasks predictably. Chapter 3 summarizes important concepts and problems from the area of stochastic scheduling; mostly focusing on precedence constraint networks between tasks with random durations. Through a stochastic scheduling point-of-view, we break-down Research Problem II into specific Research Questions II.1 and II.2. Following, Chapters 6 and 7 present our research findings when attempting to answer those questions.

Part III concludes the dissertation. In Chapter 8 we revisit the research questions formulated in Chapter 2 and Chapter 5 and assess whether we managed to find adequate answers. Then we do the same from a more high-level standpoint for the research problems formulated earlier in this chapter. Finally, we present a collection of interesting problems that could be addressed in future work.

Part I

Simple Temporal Problems

Chapter 2

Background

As discussed in Chapter 1, in the NedTrain workshop we would like to allow human resources (or actors) to decide when to dispatch (or start) their tasks "just in time" as needed, instead of asking them to comit to a fixed schedule from the start. The scheduling system in place must guarantee the formation of a feasible schedule by dispatching decisions taken in a non-deterministic manner and without asking different teams in the workshop to coordinate their decisions. Our approach for enabling such a flexible scheduling process relies on representing constraints in the workshop as a Simple Temporal Problem (STP) [31]. The framework of STP-based techniques was developed with exactly this type of flexible scheduling process in mind. This chapter prepares the reader for Chapters 3 and 4, where our results are presented. In this chapter we offer a summary of important STP-related concepts, with an emphasis on earlier work by Wilson et al. who originally considered STPs as modelling devices for the NedTrain scheduling process [99, 98, 96]. Through the prism of an STP-centered approach, we break-down Research Problem I into fine-grained Research Questions, answers to which we seek for in Chapters 3 and 4.

2.1 Simple Temporal Problems

A STP is a type of Temporal Constraint Satisfaction Problem (TCSP) [31, 35] that restricts the possible temporal distances between a set of n instantaneous events. Each event is associated with a respective *time variable* and each constraint bounds, from above and below, the distance between a pair of time variables. A solution to a STP is a *schedule*: an assignment of dispatching times to the time variables satisfying all pair-wise constraints.

Mathematically, a STP S over n events is specified as a tuple $S = (T, C)$. Here, $T = \{t_0, t_1, \ldots, t_n\}$ is a set of temporal variables (events) and C is a finite set of binary difference constraints $t_j - t_i \leq c_{ij}$, for some real number c_{ij}. A solution (or schedule) is a sequence $(s_0, s_1, s_2, \ldots, s_n)$ of values such that, if each $t_i \in T$ takes the value s_i, then all constraints in C are satisfied. If such a

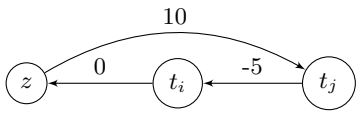


Figure 2.1: Simple STP with three variables.

solution exists, we say that the STP is *consistent*. In order to express absolute time constraints, variable $t_0 \in T$, often also denoted by z, is used. It represents a fixed reference point on the timeline, and is always assigned the value 0 (i.e. we always assume $s_0 = 0$).

The term Simple Temporal Network (STN) is often used as a synonym for STP, since algorithms for manipulating STPs typically use a network representation of the constraints, known as the *distance graph*. In the distance graph, each time variable is represented as a node and each minimum/maximum pair-wise distance constraint as a weighted arc.

Example 2.1. *Figure 2.1 illustrates the distance graph of an example* $S = (T, C)$ *with* $T = \{z, t_i, t_j\}$ *and* $C = \{z - t_i \leq 0, t_i - t_j \leq -5, t_j - z \leq 10\}$. *Each constraint of the form* $t_j - t_i \leq c_{ij}$ *is an arc from* t_i *to* t_j *with a weight of* c_{ij}. *Assuming* $z = 0$, *the first constraint in* C *asks that* t_i *is non-negative while the second constraint asks* t_j *to be at least 5 units larger than* t_i. *As such, the constraint* $t_j \geq 5$ *is implied by the first two constraints. Finally, the third constraint in* C *asks* t_j *to be at most 10 units greater than* z. *This, in turn, implies that* $t_i \leq 5$.

As shown in the seminal paper by Dechter et al. [31], a given STN implies, for each pair of variables t_i and t_j, the constraint $t_j - t_i \leq d_{ij}$ where d_{ij} is the shortest path from node t_i to node t_j in the distance graph. Moreover, this is the tightest upper bound for the distance $t_j - t_i$ implied by the network. As such, we can obtain the tightest constraints implied by a given STP by computing the all-pairs-shortest-path (APSP) matrix, typically denoted by $D = [d_{ij}]_{n \times n}$. Note that D can be obtained in $O(n^3)$ with an algorithm such as Floyd-Warshall [33].

Example 2.2. *Referring to the earlier example, the resulting APSP matrix is shown below:*

	z	t_i	t_j
z	*0*	*5*	*10*
t_i	*0*	*0*	*10*
t_j	*-5*	*-5*	*0*

Note that implied constraints $t_j \geq 5$ *and* $t_i \leq 5$ *can be easily obtained by* D. *More specifically, the shortest path from* t_j *to* z *is -5, meaning that* $z - t_j \leq -5 \Rightarrow t_j \geq 5$ *is implied. Similarly,* $t_i - z \leq 5 \Rightarrow t_i \leq 5$ *can be obtained by observing the shortest path from* z *to* t_i.

We have insofar considered STNs as modelling devices for instantaneous events subject to temporal distance constraints. STPs, however, can easily model

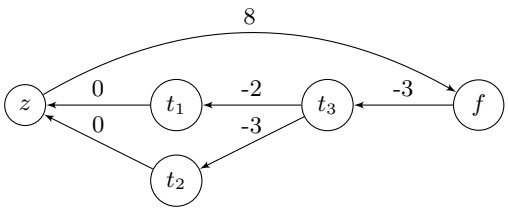


Figure 2.2: STP with three tasks and a due-date.

temporal constraints between non-instantaneous events, e.g. tasks with known durations. As demonstrated in the following example, a time variable for the dispatching time (or start time) of each task is needed and task durations are represented by means of pair-wise temporal distance constraints. Moreover, release-/due-date constraints on groups of tasks (e.g. when groups of tasks refer to the maintance work associated with a train) are also modelled easily. [1]

Example 2.3. *Figure 2.2 presents an example STP for three tasks with known durations that must be dispatched subject to precedence constraints and a due-date constraint. The dispatching times of tasks 1,2 and 3 are associated with time variables* t_1, t_2 *and* t_3*. Again, variable* z *is fixed to zero and denotes the beginning of time, e.g. the beginning of the week on Monday at 8:00 am. Constraints of the form* $z - t_i \leq 0$ *ask that no task may be dispatched before the beginning of the week. We assume that time, in this example, is measured in hours. Task 1 has a duration of 2 hours, while tasks 2 and 3 have a duration of 3 hours. In addition, precedence constraints ask that task 3 cannot be dispatched unless tasks 1 and 2 have finished. This is reflected by constraints* $t_1 - t_3 \leq -2$ *and* $t_2 - t_3 \leq -3$*. Effectively these imply* $t_3 \geq \max\{t_1 + 2, t_2 + 3\}$*, meaning that task 3 may only start after both tasks 1 and 2 have finished (since the finish time of a task equals its dispatching time plus its duration). Variable* f *and constraint* $f - z \leq 10$ *enable us to model a due-date constraint by asking that the finish time of task 3 (3 hours past its start) cannot be later than 8 hours since the beginning. Effectively, then, all tasks must have finished by Monday 4:00 pm.*

2.1.1 Forming a feasible solution

As we see now, shortest paths from and to special-purpose variable z are of particular interest. The first row and column of D hold the so-called *earliest start time* and the *latest start time* for each variable, which constitute tight lower and upper bounds, respectively. More in particular, assuming $z = 0$ and using d_{0i} and d_{i0} to denote the shortest path from z to a variable t_i and vice-versa, it has been shown in [31] that a solution is not feasible if the condition $t_i \in [-d_{0i}, d_{i0}]$

[1] In fact, as discussed further down the line, there are STP-related techniques allowing us to handle resource constraints by transforming them in a preprocessing phase into temporal constraints. By doing so, we only having to "worry" about temporal constraints during dispatching.

is not satisfied for some variable t_i. In addition, there always exists a feasible schedule such that $t_i = v$ for every value $v \in [-d_{0i}, d_{i0}]$ and every variable t_i.

That is, the first column and row of D provide time-windows from which we can pick suitable times for dispatching our variables. Moreover, the so-called *earliest start schedule* can be formed by setting each variable to its earliest start time (i.e. letting $t_i = -d_{0i}$) and this schedule is always feasible. Similarly, the so-called *latest start schedule* can be formed by letting $t_i = d_{i0}$ and is also always feasible. In effect, then, two feasible solutions become immediately available by computing matrix D.

Example 2.4. *Shortest path calculations to and from z in the network of Figure 2.2 reveal suitable dispatching time-windows for our tasks. For instance, the shortest path length from z to t_3 equals 5 while the shortest path from t_3 to z equals -3. This yields a time-window of $[3, 5]$ within which task 3 must be dispatched. More specifically, if task 3 starts later than 5 hours after Monday 8:00 am, then there is no way to meet the due-date. In addition, task 3 cannot start earlier than 3 hours past Monday 8:00 am without violating a precedence constraint. According to further shortest path calculations, violating any of the following conditions will prevent us from forming a feasible schedule:*

$$\begin{aligned} t_1 &\in [0, 3] \\ t_2 &\in [0, 2] \\ t_3 &\in [3, 5] \\ f &\in [6, 8] \end{aligned}$$

We note that condition $f \in [6, 8]$ means there exists at least one feasible solution in which all tasks finish 2 hours before the due-date, i.e. in which $f = 6$. In fact, such a solution can be formed by taking the earliest start schedule, i.e. by letting $t_1 = 0, t_2 = 0, t_3 = 3, f = 6$.

Apart from the marginal cases of the earliest and latest start schedule, however, picking an arbitrary combination of values from the time-windows given by the first column and row of matrix D is generally not guaranteed to result in a feasible solution. After fixing a particular time-variable to a specific value, one must generally recalculate matrix D (to obtain new time-windows) for a modified version of the STP with constraints that fix the variable to the chosen value. That is, say variable t_i is fixed to value $v \in [-d_{0i}, d_{i0}]$. The STP must now be updated such that t_i is now connected to z with constraints $z - t_i \leq -v$ and $t_i - z \leq v$, implying $t_i = v$. A new matrix D' can now be computed and a value $v' \in [-d'_{0j}, d'_{j0}]$ can now be chosen for another variable t_j.

In effect, then, dispatching the events of a given STP at feasible temporal distances amounts to issuing a sequence of queries of the following form: Given that some events have been dispatched (i.e. given a feasible partial schedule), within which time-window shall we dispatch each of the remaining events in order to guarantee we will not run out of feasible options later?

2.2 Flexibility metrics

Effectively, the STP/STN machinery enables us to maintain a compact encoding of all potentially realizable schedules and decide on-the-fly how to extend the partial schedule. It is this freedom during dispatching, as opposed to commiting to a fixed schedule, that makes STPs attractive for scheduling in a dynamic environment such as the NedTrain workshop. Intuitively, the larger the solution-space (i.e. the number of feasible schedules), the greater the amount of freedom in choosing preferable yet feasible dispatching timepoints. For this reason, an important concept often encountered in the literature is the *flexibility* of a given STP, which is intuitively proportional to the size of its solution-space.

The main challenge in defining a flexibility metric for STPs is ensuring it is efficient to compute yet accurate. The most widely accepted flexibility metric, namely the *naive flexibility metric*, essentially measures the perimeter of the bounding box of the solution-space (thus providing a sort of outer approximation). An attractive property of this metric is that it can be computed efficiently. More in particular, the amount of naive flexibility in a given STP equals the total width of the time-windows $[-d_{0i}, d_{i0}]$ defined by the earliest and latest start times in APSP matrix D.

Example 2.5. *Adding the widths of time-windows* $[0,0], [0,5], [5,10]$ *we get a measurement of* 10 *for the naive flexibility of the STP in Figure 2.1. Similarly, we get a measurement of 9 for the naive flexibility of the STP in Figure 2.1.*

However, as pointed out by Wilson et al. in [98], a main downside of this naive metric is that it can give counter-intuitive results by seriously over-estimating the actual amount of freedom in dispatching the events. In response to this disadvantage, in [98] Wilson et al. managed to define a more accurate metric, namely *concurrent flexibility*. The concept of concurrent flexibility lies at the center of the work presented in later Chapters 3 and 4. For this reason, and without getting into too many technical details, we shall hereby attempt to explain the basic idea behind it.

The main idea behind concurrent flexibility is that of inscribing a n-dimensional box within the solution-space of a given STP. For all practical purposes, we may only consider STPs the variables of which have bounded latest start times. As such, we may assume the solution-space of a STP to be a polytope, i.e. a finite region of n-dimensional space enclosed by a finite number of hyperplanes [24].

Consider a collection of n (non-empty) time-windows $[l_i, u_i]$, one per time variable t_i, chosen such that every schedule within

$$[l_1, u_1] \times [l_2, u_2] \times \ldots \times [l_n, u_n]$$

is a feasible solution. Clearly, the product of those time-windows constitutes such an inscribed n-dimensional hyperrectangle, or box. Wilson et al. defined concurrent flexibility in terms of the following problem: Find a collection of time-windows the product of which defines a *maximum perimeter* axis-aligned box, inscribed within the solution-space. Such a collection of time-windows is an

interval schedule and the amount of concurrent flexibility it provides equals the perimeter of the box it defines, i.e. the sum of the widths of the time-windows.

Example 2.6. *An interval schedule of maximum concurrent flexibility for the STP in Figure 2.2 is given below:*

$$t_1 \in [0, 3]$$
$$t_2 \in [0, 2]$$
$$t_3 \in [5, 5]$$
$$f \in [8, 8]$$

Adding the widths of those time-windows, we get a measurement of 5 *for the amount of concurrent flexibility. First, we note that each interval schedule time-window* $[l_i, u_i]$ *for variable* t_i *fits within the corresponding earliest/latest start time interval* $[-d_{0i}, d_{i0}]$*. Moreover, in contrast with naive flexibility, the concurrent metric does not account for infeasible schedules; every combination of values within the product of the time-windows above constitutes a feasible schedule.*

Unfortunately, however, this metric has been defined as the solution to a linear program (LP) with $2n$ variables and $|C|$ constraints, where C is the set of pair-wise temporal distance constraints. Note that solving a LP with modern interior-point methods has a worst-case complexity of $O(N^3L)$ with N the number of variables and L the bit-wise length of the problem description [79]. In computing concurrent flexibility, $N = 2n$ and L is bounded from above by $|C|$, in turn bounded from above by n^2. As such, computing concurrent flexibility with the LP-based method of Wilson et al. has a worst-case complexity of $O(n^5)$, which is substantially higher than the cost of shortest path computations required for naive flexibility.

2.3 Temporal decoupling

In several domains like the NedTrain workshop, events are associated with actors that control their dispatching. If dispatching times are determined during the dispatching process, instead of commiting to a fixed schedule from the start, then clearly, dispatching two events controlled by different actors at a feasible distance requires some form of communication between the actors. Such coordination based on synchronous communication, however, is often is undesired or even impossible. Communication during dispatching limits actor autonomy, it can make the overall process highly inefficient, and/or might violate privacy concerns as actors will have to share information about their plans. Focusing back on the NedTrain workshop, recall that actors are organized into teams, each responsible for dispatching a certain subset of events. Each team would like to have the freedom to choose a schedule for their part of the problem, i.e. only involving time variables controlled by the team, without having to negotiate with or be affected by the choices of other teams.

To facilitate dispatching without actors having to negotiating over the dispatching times of their events, a certain body literature focuses on the problem of *temporal decoupling*. This problem asks to *decouple* a given STP by partitioning it into smaller subproblems the partial schedules for which can always be merged into a feasible schedule for the whole STP. Finding a decoupling effectively amounts to tightening some constraints such that constraints between variables controlled by different actors become redundant, i.e. are already implied by other constraints. In effect, then, the solution-space of the resulting decoupled STP is only a part of the original STP's solution-space. And since flexibility is highly valuable for dispatching, the objective in finding a decoupling is to retain as much of the original flexibility as possible.

In [44] Hunsberger first formally defined the decoupling problem and proposed heuristics for the problem of maximizing the retained naive flexibility. In [72] Planken et al. showed that finding an optimal temporal decoupling with respect to a general objective is NP-hard, but a decoupling that optimizes a linear function of the time variables (unlike the objective used by Hunsberger) can be formulated as a LP. In contrast with both these centralized approaches, Boerkoel and Durfee [14] proposed a distributed algorithm for finding a decoupling. Their approach deals with possible privacy concerns among different actors since it does not require a central process with access to the whole STP.

The methods discussed above aim at maximizing the amount of retained naive flexibility. Wilson et al., on the other hand, focused on decoupling while retaining concurrent flexibility [98]. Interestingly, they found that their LP-based method for computing the concurrent flexibility of a STP actually gives, as a by-product, an optimal *total decoupling*, retaining the maximum amount of concurrent flexibility. In contrast with a decoupling in the broad sense, a total decoupling enables dispatching each time variable in isolation from every other variable. It can be used in the edge case with as many agents as there are events, each agent controlling a single event. A total decoupling is specified as a so-called *interval schedule*: a time-window per event within which the controlling agent can dispatch it without having to coordinate with other agents. An interval schedule allows for a particularly efficient dispatching process, since dispatching an event amounts to just picking a suitable time from a time-window, in constant time. Moreover, it facilitates an optimally flexible dispatching process, as far as concurrent flexibility is concerned.

Example 2.7. *Consider again the STP shown in Figure 2.2 and assume time variables are distributed over two parties such that one party is responsible for time variables t_1 and t_2 and the other party is responsible for t_3 and f. The interval schedule computed in the earlier example can be used to derive a decoupled STP, as shown in Figure 2.3, in which variables of the same color belong to the same party. With appropriate constraints between z (fixed to zero) and every other variable, in this STP we simply enforce that each variable takes its value from within the time window specified by the interval schedule. Doing so ensures that every solution for the resulting STP is also feasible for the original STP of Figure 2.2. Note that inter-party constraints, i.e. between variables belonging*

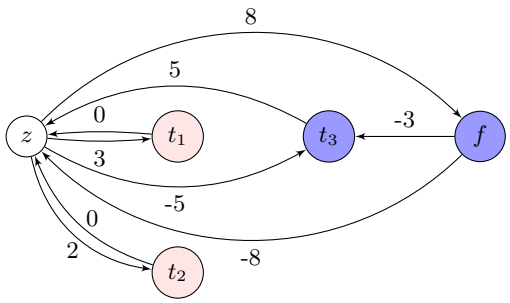


Figure 2.3: STP with three tasks and a due-date, partitioned into two independent sub-problems.

to different parties, are no longer necessary. In fact intra-party constraints are also unnecessary, but included in our example for illustrative purposes, i.e. to illustrate a partitioning of the original STP in two. In effect, given such a decoupled STP, only the two constraints between a variable and special-purpose variable z need to be taken into account for picking a feasible dispatching time for that variable.

2.4 Resource constraints

So far in our discussion we have ignored that in many applications of scheduling, tasks occupy certain amounts of one or more resources during their execution. Such is also the case with NedTrain's maintenance workshop, where resources correspond to people and equipment. Depending on their outcome dispatching times and durations, two or more tasks which are not precedence-related may overlap during a certain time interval. A combination of precedence-unrelated tasks the total resource demands of which would exceed resource capacities in case they overlapped is often known as a *forbidden set*.

Example 2.8. *We shall use the example STP shown in Figure 2.4 to clarify the concept of a forbidden set. In this example, tasks 1, 2 and 3 (with durations 2, 2 and 5) can start in parallel but task 4 (with duration 2) has to wait for the completion of task 2. Note that due to the precedence constraint, task 4 will never overlap with task 2. However, it might overlap with tasks 1 and 3 depending on the chosen dispatching times. Assume that tasks 1, 3 and 4 involve the participation of 3,3 and 2 engineers respectively. In addition, assume there are only 6 engineers available in the workshop at any point in time. Due to resource limitations, then, tasks 1, 3 and 4 constitute a forbidden set, as they cannot all execute in parallel at a point in time.*

The resource demands of certain tasks and the resource capacities of the environment are often known together as *resource constraints*.[2] Given a STP

[2]Resource demands and capacities constitute a compact encoding of the collection of

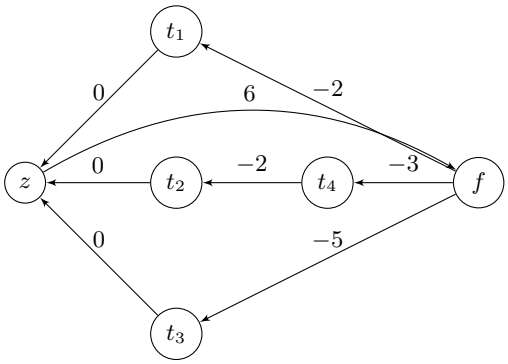


Figure 2.4: STP with four tasks and a due-date. Tasks 1, 3 and 4 form a forbidden set.

along with resource constraints, an emerging problem not addressed by the dispatching strategies discussed so far is that of ensuring a realized schedule satisfying both temporal and resource constraints.

Existing literature related to this problem can be divided in two categories. On one hand, there are techniques for finding a fixed schedule satisfying both STP and resource constraints.[3]

The problem of finding a feasible schedule with the smallest makespan subject to STP and resource constraints is known as RCPSP/max [82, 67], Effectively, it is an extension of the simpler resource-constraint project scheduling problem (RCPSP) [39], which also asks to find a feasible schedule of minimum makespan. In contrast with the RCPSP/max, however, RCPSP assumes a simple form of STP constraints. Specifically, only precedence constraints are allowed between pairs of tasks and and no maximum distance constraints between pairs of tasks are allowed; hence the suffix "/max" since RCPSP/max does allow for such maximum distance constraints. Depending on the complexity of temporal and resource constraints one may encounter several other variants. For instance, as with the RCPSP, the so-called cumulative scheduling problem (CSP) [60] also only allows precedence constraints but also assumes a simpler form of resource constraints, with only a single type of resource available. One of the most well-known optimization problems, the so-called Job Shop Scheduling Problem (JSSP) [1], assumes even simpler temporal and resource constraints than the CSP. Beyond only allowing a single resource, the JSSP assumes that each task (or job) may wait (with a single precedence constraint) for (at most) one other task and requires exactly one unit of the available resource. Existing literature offers a variety of solution approaches for the rich ecosystem of different problem

forbidden sets. Enumerating forbidden sets directly instead would lead to an inefficient problem formulation since there can by as many forbidden sets as there are members in the power-set of the given set of tasks.

[3]It should be noted that answering whether such a schedule exists is a hard problem, in contrast with doing so only subject to STP constraints.

models, including mathematical optimization approaches [51], branch-and-bound procedures [21], constraint programming [82], (meta-)heuristics [49] and so on.

Instead of finding a single feasible schedule, the other category of techniques deal with resource constraints as follows. The given STP is transformed into a so-called *partial order schedule* (also a STP) by addding (or "posting") additional precedence constraints that eliminate forbidden sets. As such, every feasible schedule for the resulting partial order schedule (or POS) will always satisfy the given resource constraints. In effect, then, a whole space of feasible schedules is produced by the solution process, in the form of a STP. An illustration of this technique is given in the following example.

Example 2.9. *Referring to the previous example, we had identified* $\{1, 2, 3\}$ *as a forbidden set. To eliminate it, it suffices to post a precedence constraint between any pair of its members. This would produce a POS the solution space of which has no schedules with tasks 1 2 and 3 executing in parallel. Using notation* (i, j) *to signify a precedence constraint* $t_j \geq t_i + d_i$ *meaning task* j *cannot start unless task* i *has finished, following is the list of precedence constraints any of which can be added to the given STP in order to produce a POS:*

1. *(1,2), yielding a makespan of 7*
2. *(2,1), yielding a makespan of 5*
3. *(1,3), yielding a makespan of 7*
4. *(3,1), yielding a makespan of 7*
5. *(2,3), yielding a makespan of 7*
6. *(3,2), yielding a makespan of 10*

Along with each constraint, we list here the resulting makespan, i.e. the earliest achievable completion time for all tasks, in a feasible schedule for the resulting POS. As constraint $f - z \leq 6$ *imposes a maximum completion time for all tasks within 6 time units, all other options besides option 2 would produce a POS which is inconsistent, i.e. has no feasible schedule, or in other words, an empty solution space.*

Note that even for a small example, the number of different ways to eliminate forbidden sets and produce a POS can be quite high. In fact, as mentioned before, the number of forbidden sets may grow exponentially with the number of tasks. In turn, the number of different combinations of precedence constraints that resolve forbidden sets, also grows exponentially with the number of tasks. Different efficient methods for enumerating forbidden sets (also known as critical sets) have been investigated [86, 57]. Without having to explicitly enumerate all forbidden sets in a given problem, existing work focuses either on finding a POS of minimum makespan [67] or on finding a POS that preserves as much of the original STP's flexibility as possible [75, 73]. It should be noted that no work, to our knowledge, focuses on preserving *concurrent* flexibility while finding a POS.

Since we have rejected the idea of scheduling according to a fixed schedule, techniques falling in the first category are not within the scope of our approach. Methods for finding partial order schedules, on the other hand, are perfectly suitable. They enable us to deal with resource constraints in an offline, preprocessing phase and then dispatch the partial order schedule as a regular STP, with a focus on *decoupling* and *flexibility* as exlained earlier.

2.5 Research Questions

After summarizing important concepts from the literature, in this section we examine to which extent existing approaches allow us to address Research Problem I, originally stated in Chapter 1. Any gaps in existing literature that prevent us from adequately addressing this problem will be mapped to corresponding Research Questions that will be addressed in subsequent chapters.

According to Research Problem I, we would like to compute flexible schedules for independent work-teams that can be easily adapted to changes in the environment. A flexible schedule should provide, during dispatching, access to a range of potential dispatching times, per task, from which we can quickly choose on-the-fly. In addition, a flexible schedule should enable each team to consider alternative dispatching times for their tasks without worrying about conflicts with the decisions of other teams.

In the tradition of earlier work by Wilson et al., we also propose the use of STPs for modelling temporal constraints within the NedTrain workshop. The framework of techniques on concurrent flexibility and interval schedules developed by Wilson et al. enables us to deal, in part, with Research Problem I. This is because an interval schedule is functionally equivalent to a flexible schedule. More in particular, an interval schedule allows us to maintain, for each task, a range of potential dispatching times from which we can choose in constant time. Moreover, an interval schedule represents a total decoupling of the STP, enabling us to dispatch a task within its prescribed interval regardless of the dispatching times chosen for other tasks, always guaranteeing feasibility. In effect, then, interval schedules offer both team independence and the freedom to consider suitable dispatching times on-the-fly.

The problem with the approach of Wilson et al., however, is its relatively high computational cost. The LP used for computing an interval schedule of maximum concurrent flexibility can have as many as n^2 constraints and the cost of solving it is bounded by $O(n^5)$, as seen earlier. STP instances of the NedTrain workshop can contain several hundreds or even thousands of variables. As such, the existing LP approach is expected to hit performance barriers in practical applications. This problem is exaggerated by considering that resource constraints must also be addressed in the NedTrain workshop. As we saw in Section 2.4, resource constraints can in fact be translated into temporal constraints, in a pre-processing phase. More in particular, a search procedure enumerates candidate partial order schedules (POSs), each corresponding to a potential transformation of resource constraints into additional temporal constraints that eliminate forbidden sets.

In the end, that candidate POS with the highest amount of concurrent flexibility will be chosen. As dealing with resource constraints involves computing the maximum achievable amount of concurrent flexibility in each enumerated POS, a highly efficient metric for measuring the amount of concurrent flexibility in a given STP is needed. Note that computing an actual interval schedule is not necessary, i.e. only computing the maximum achievable amount flexibility would suffice for enumerating candidate POSs.

Research Question I.1. *How to efficiently compute concurrent flexibility in a given STP, in low-order polynomial time?*

Another challenge in addressing Research Problem I relates to the dynamic nature of the dispatching process. According to the existing concurrent flexibility framework, an interval schedule is computed once and remains fixed until all tasks are completed. In practice, however, the number of dispatching times that remain undecided gradually diminishes as task execution unfolds. Referring back to Research Problem I, we would like to be able to use any new information about already dispatched tasks, in order to potentially enhance flexibility. Indeed, there is an opportunity to continuously improve the flexibility of the dispatching process by adapting the interval schedule to new information. Consider that an actor commits to dispatching a certain task at a certain time-point in the (near) future. From that point on, the flexibility available for determining the dispatching time of that event is no longer needed. We can therefore update the interval schedule by narrowing the time-window of that task to a point and redistributing unused flexibility over the time-windows of tasks with undecided dispatching times. By doing so, we expect to observe the available flexibility per yet-undispatched event to continuously increase as task execution unfolds.

In other words, we would like to extend the existing "static" flexibility framework into a "dynamic" framework that allows us to keep track of new information regarding already dispatched tasks and update the interval schedule accordingly. Note that simply recomputing an interval schedule from scratch when new information becomes available is not an adequate approach, as care should be taken to ensure that continuously adapting the interval schedule during dispatching does not cause disruptions. If, for example, time-windows are radically re-arranged every time the interval schedule is updated, we would compromise the predictability, or the visibility into the future, offered to actors by the dispatching process. Updating the current interval schedule should be done incrementally, i.e. without invalidating any planning-ahead permitted by the current interval schedule. As such, we are interested in answering the following question:

Research Question I.2. *How to incrementally recompute a concurrent flexibility interval schedule during dispatching?*

Moreover, updating the interval schedule should not be computationally expensive, or we might end up amplifying the effects of uncertainty by introducing disruptive delays. Recomputing an interval schedule from scratch with the LP

method of Wilson et al. can have a prohibitive computational cost as a real-time rescheduling operation in the NedTrain workshop. As such, we would also like to address the following question:

Research Question I.3. *How to redistribute concurrent flexibility as fast as possible (using heuristic methods if necessary)?*

Research question I.1 is answered in Chapter 3, where we manage to lower the complexity of computing the amount of concurrent flexibility in a given STP from $O(n^5)$ to $O(n^3)$. Effectively, then, we show that concurrent flexibility can be computed with the same cost as naive flexibility. Research question I.2 is answered in the first part of Chapter 4, where we extend the approach of Chapter 3 to enable the computation of interval schedules of maximum concurrent flexibility as well, with the same complexity. Research question I.3 is answered in the second part of Chapter 4, where we propose an efficient heuristic that allows us to update the interval schedule by redistributing unused flexibility in near-linear time and with almost no loss of optimality.

Chapter 3

Flexible static decoupling

3.1 Introduction

Simple Temporal Networks (STNs) [30, 29], offer a convenient framework for modelling temporal aspects of scheduling problems, distinguishing between a set T of temporal variables and a set C of linear difference constraints between them. STNs have been used quite extensively in a number of different domains where automated planning and scheduling are key issues, such as manufacturing, maintenance, unmanned spacecraft and robotics. One of the attractive properties of STNs is that various important search and decision problems can be solved quite efficiently. For example, deciding whether an STN admits a schedule, finding such a schedule, as well as finding a complete schedule extending a partial one are all problems that can be solved in low-polynomial time.

Constructing one single schedule, however, for an STN is often not sufficient. In quite some applications one has to react immediately to possible disturbances without having the time to recompute a new schedule. In such a case, instead of one single schedule, we would like to have a *set of schedules* available from which we can select in constant time a suitable alternative. The availability of such a set of alternative schedules implies that the scheduler could offer some amount of *flexibility* in scheduling: instead of determining a fixed value for a temporal variable, a flexible schedule offers for each temporal variable a set of values to choose from. Clearly, such a flexibility is determined by properties of the set C of constraints of an STN. Therefore, one would like to specify a *flexibility metric* for STNs based on the properties of this set C. One of the first proposals [74, 77] for such a flexibility metric has been based on considering a (non-empty) interval $[est(t), lst(t)]$ for each temporal variable $t \in T$ and defining the flexibility of an STN S as $flex(S) = \Sigma_{t \in T}(lst(t) - est(t))$. Here, $est(t)$ refers to the earliest time t can be executed in any feasible schedule for S and $lst(t)$ to its latest time. We will discuss this idea in more detail below.

As has been pointed out recently, this so-called *naive flexibility metric flex* has some serious shortcomings. Its main disadvantage is that, in general, it is

far too optimistic with respect to the available flexibility in an STN. The reason is that the flexibility intervals $[est(t), lst(t)]$ assigned to temporal variables t, in general, are not independent: The flexibility of two temporal variables together might be substantially smaller than the sum of their individual flexibilities [97, 98]. As an alternative, these authors proposed to assign flexibility intervals to temporal variables such that these intervals are *independent*: for every $t \in T$, its flexibility interval $[t^-, t^+]$ enables one to make a choice for a time point in the interval without any consequence for the validity of the choices for other temporal variables. In their paper they use a Linear Programming (LP) based approach to compute this so-called *concurrent flexibility metric flex** to show that it can be computed in polynomial time. They do not, however, mention exact upper bounds for the time complexity of computing this metric.

In this paper, we start by presenting a geometric interpretation of the concurrent and the naive flexibility metric. Then, by using duality theory, we show that there exists a nice correspondence between (a slightly modified) minimal distance matrix D_S associated with an STN S and both flexibility metrics. If this matrix D_S is conceived as the specification of weights on the edges of a (complete) weighted bipartite graph G_S, then the value of the concurrent flexibility metric equals the costs of a minimum matching in G_S, while the naive flexibility metric corresponds to the costs of a maximum matching in G_S. Using this correspondence, we are able to deduce an efficient $O(n^3)$ algorithm for determining both the concurrent and naive flexibility of an STN.

3.2 Preliminaries

An STN is a pair $S = (T, C)$, where $T = \{t_0, t_1, \ldots, t_{n-1}\}$ is a set of temporal variables and C is a finite set of binary constraints on T, each constraint having the form[1] $t_j - t_i \leq c_{ij}$, for some real number c_{ij}. A solution to S is a *schedule* for S, that is, a function $\sigma : T \to \mathbb{R}$, assigning a real value (time-point) to each temporal variable in T such that all constraints in C are satisfied. If such a schedule exists, we say that the STN is *consistent*.[2] The time-point $t_0 \in T$, also denoted by z, is used to be able to express absolute time constraints. It represents a fixed reference point on the timeline, and is assigned the value 0.

Example 3.1. *We consider two STNs* $S_1 = (T_1, C_1)$ *and* $S_2 = (T_2, C_2)$ *where* $T_1 = T_2 = \{z, t_1, t_2, t_3\}$ *and*

$$\begin{aligned} C_1 = &\quad \{0 \leq t_i - z \leq 50 \mid i = 1, 2, 3\}, \\ C_2 = &\quad C_1 \cup \{0 \leq t_i - t_j \leq \infty \mid 1 \leq j < i \leq 3\}. \end{aligned}$$

Every assignment $\sigma_1(t_i) = v_i \in [0, 50]$ *is a schedule for* S_1*. On the other hand, a schedule* σ_2 *for* S_2 *has to satisfy* $0 \leq \sigma_2(t_1) \leq \sigma_2(t_2) \leq \sigma_2(t_3) \leq 50$*.*

[1] If both $t_j - t_i \leq c_{ij}$ and $t_i - t_j \leq c_{ji}$ are specified, we will also use the more compact notation $-c_{ji} \leq t_j - t_i \leq c_{ij}$.

[2] Without loss of generality, in the remainder of the paper, we assume that an STN to be consistent. Note that consistency of an STN can be determined in low-order polynomial time [30].

Using a shortest path interpretation of an STN $S = (T, C)$ [29], there is an efficient method to find all tightest constraints implied by C, by searching for all shortest paths between the time points in T using e.g. Floyd and Warshall's all-pairs shortest paths algorithm [33]. The $n \times n$ *minimum distance matrix* D_S contains for every pair of time-point variables t_i and t_j the length $D_S[i, j]$ of the *shortest path* from t_i to t_j in the distance graph. In particular, the latest starting time $lst(t_i)$ and earliest starting time $est(t_i)$ can be obtained directly from the first row and first column of D_S: $lst(t_i) = D[0, i]$ and $est(t_i) = -D[i, 0]$. Given an STN S, this matrix D_S can be computed in low-order polynomial ($O(n^3)$) time [29]. Here, n denotes the number of temporal variables.

The following proposition is a restatement of Theorem 3.3 in [30] and essentially states that every STN is decomposable via its distance graph D_S:

Proposition 3.1. *Let $S = (T, C)$ be an STN and D_S its distance matrix. For $i = 1, \ldots, n-1$, let $est(t_i) = -D_S[i, 0]$ and $lst(t_i) = D_S[0, i]$. Then, for every schedule σ for S, and every $t \in T$, it holds that $\sigma(t) \in [est(t), lst(t)]$. Moreover, given any $t \in T$ and $v \in [est(t), lst(t)]$, there exists a schedule σ for S such that $\sigma(t) = v$.*[3]

Finally, we assume that for each $t \in T$ there is a finite interval for scheduling it. In particular this means that for each STN $S = (T, C)$ we assume that for all $t \in T$, $t \geq z$ holds and there exists a finite constant (horizon) h_S such that for all $t \in T$ it holds that $t \leq h_S$. This avoids the use of unbounded time intervals such as $-\infty \leq t_i - t_j \leq \infty$.

3.3 The naive and the concurrent flexibility metric

Intuitively, a flexibility metric should indicate our freedom of choice in choosing values for temporal variables such that the constraints are satisfied. In the naive flexibility metric one has chosen, for each variable $t \in T$, the interval $[est(t), lst(t)]$ to indicate this freedom of choice.[4] The naive flexibility of an STN S then is defined as $\mathit{flex}(S) = \sum_{t \in T} (lst(t) - est(t))$. Obviously, since D_S is computable in $O(n^3)$ time, $\mathit{flex}(S)$ can be computed in $O(n^3)$ time.

As we already mentioned, using $\mathit{flex}(S)$ has a serious disadvantage, due to *dependencies* that might exist between temporal variables. We give a simple example to illustrate these dependencies and their consequences.

Example 3.2. *Consider the STNs presented in Example 4.1. For both STNs it holds that $est(t_i) = 0$ and $lst(t_i) = 50$, for $i = 1, 2, 3$. Hence, $\mathit{flex}(S_1) = \mathit{flex}(S_2) = 150$. Intuitively, however, due to the ordering constraints between t_1,*

[3] In fact it can be shown that there exists a simple backtrack-free polynomial-time algorithm to construct such a schedule σ for S.

[4] One of the reasons for this choice is that taking any value $v \in [est(t), lst(t)]$ for t allows us, by decomposability (Proposition 3.1), to extend the partial schedule $\sigma(t) = v$ to a total schedule σ for an STN S.

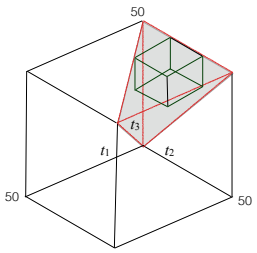


Figure 3.1: The polytope P_1 generated by S_1 is a cube with edges of size 50; the polytope P_2 generated by S_2 is the shaded subspace of P_1. The inner bounding box of P_2 is the small green cube in the shaded area. The inner bounding box of S_1 and also the outer bounding box of S_1 and S_2 equal P_1.

t_2 and t_3, the flexibility of S_2 should be much lower: giving a flexibility of f_1 to t_1 results in a flexibility of $f_2 = v_2 - f_1$ for t_2, where $50 \geq v_2 \geq f_1$, and a flexibility of $50-v_2$ for f_3. The sum of these flexibilities equals $f_1+(v_2-f_1)+(50-v_2) = 50$, one-third of the value of the naive flexibility for S_2.

One way to see how these dependencies are neglected in computing the naive flexibility $flex(S)$ is to give a geometric interpretation of this metric.

The solution space of a set of linear constraints is a polytope. In (our) case of a consistent STN it is a bounded polytope. The set of intervals $\{[est(t_i), lst(t_i)]\}_{i=1}^{n}$ can be thought of as determining the smallest (hyper)cube containing this polytope. Clearly, if the polytope itself is not a hypercube, there are points in the hypercube that do not belong to the solution space of the STN. Exactly in these cases $flex(S)$ will overestimate the amount of flexibility of S.

Example 3.3. *Consider again the STNs S_1 and S_2 as presented in Example 4.1. The polytope generated by C_1 is the large (hyper)cube depicted in Figure 3.1. This hypercube equals the smallest hypercube containing the polytope, hence $flex(S_1) = 50+50+50$ is the real flexibility of S_1. The polytope generated by C_2 is the shaded subspace depicted in Figure 3.1. The smallest hypercube containing it, however, equals the hypercube specified for S_1. Hence, $flex(S_2) = flex(S_1) = 150$: an overestimation of the flexibility.*

A new flexibility metric for STNs [97] has been introduced to overcome the disadvantages of $flex(S)$. Instead of determining the smallest *outer bounding box* (hypercube) containing the polyhedron generated by a set of constraints C, they determine the largest *inner bounding box* contained in this polyhedron (see below for a computation). An obvious advantage of such an inner bounding box is that every point in the box belongs to the polyhedron (i.e. solution space) containing it.

Example 3.4. *Consider again the STN S_2. A largest inner bounding box contained in the polytope determined by C_2 is the small box contained in the*

shaded area in Figure 3.1. All its points are contained in the solution space of S_2.

To construct a flexibility metric based on the largest inner bounding box we have to make sure that for every constraint $t_j - t_i \leq c_{i,j}$ the flexibility intervals $[t_i^-, t_i^+]$ and $[t_j^-, t_j^+]$ associated with t_i and t_j are independent, i.e., every value v_i and v_j for t_i and t_j, respectively, chosen in these intervals has to satisfy the constraint $v_j - v_i \leq c_{i,j}$. This is exactly the case if t_j assumes the maximum value and t_i assumes the minimum value in its interval, i.e., we have to ensure that $t_j^+ - t_i^- \leq c_{i,j}$. Therefore, the value *flex*$^*(S)$ for the concurrent flexibility for S can be found by solving the following LP (see [98]):

$$\begin{aligned} \text{maximize} \quad & \sum_{t_i \in T} (t_i^+ - t_i^-) && \text{(LP1)} \\ \text{subject to} \quad & t_i^- \leq t_i^+ \quad , && \forall\, t_i \in T \\ & t_j^+ - t_i^- \leq c_{i,j}\,, && \forall\, (t_j - t_i \leq c_{i,j}) \in C \end{aligned}$$

Here, *flex*$^*(S)$ indicates the maximum flexibility that can be obtained when all flexibility intervals of temporal variables are independent from each other and values in these intervals can be chosen concurrently.

Example 3.5. LP1 *will return* *flex*$^*(S_1) = 150$*, but* *flex*$^*(S_2) = 50$*, showing that here the concurrent flexibility metric corresponds to our intuition.*

3.4 Concurrent flexibility by minimum matching

In this section we show that computing the maximum concurrent flexibility is identical to computing a perfect minimum weight matching of a weighted bipartite graph G_S completely specified by D_S. On the other hand, computing the naive flexibility corresponds to computing a maximum matching in this graph G_S.

To establish these results, first, we specify an LP equivalent to LP1 using the minimum distance matrix D_S instead of the original set C of constraints. Then we show that the entries of D_S can be used to compute a least upper bound on *flex*$^*(S)$. Finally, by using duality theory in Linear Programming and some adaptations, we show that this least upper bound is a maximum matching in a bipartite graph specified by D_S and in fact equals *flex*$^*(S)$.

Remember that the entries $d(i, j)$ of the minimum distance matrix D_S of an STN S specify the upper bound of the strongest difference constraint between the temporal variables t_j and t_i: $t_j - t_i \leq d(i, j)$. Hence, we can replace the specification LP1 by the following equivalent LP, since both LP's specify the

same solution space:

$$\begin{aligned} \text{maximize} \quad & \sum_{t_i \in T} (t_i^+ - t_i^-) && \text{(LP2)} \\ \text{subject to} \quad & t_i^- \leq t_i^+ \quad , \quad \forall\, t_i \in T \\ & t_j^+ - t_i^- \leq d(i,j) \quad , \quad \forall\, t_i \neq t_j \in T \end{aligned}$$

To remove the condition $\forall\ t_i \neq t_j \in T$, it is sufficient to realize that $t_i^+ - t_i^- = (t_i^+ - t_0^-) + (t_0^+ - t_i^-) \leq d(0,i) + d(i,0) = \mathit{lst}(t_i) - \mathit{est}(t_i)$, since $t_0^- = t_0^+ = 0$. Hence, setting $d(i,i) = d(0,i) + d(i,0)$ for all $t_i \in T$, we change LP2 to the following equivalent LP:

$$\begin{aligned} \text{maximize} \quad & \sum_{t_i \in T} (t_i^+ - t_i^-) && \text{(LP3)} \\ \text{subject to} \quad & t_i^- \leq t_i^+ \quad , \quad \forall\, t_i \in T \\ & t_j^+ - t_i^- \leq d(i,j) \quad , \quad \forall\, t_i, t_j \in T \end{aligned}$$

We will denote this modified distance matrix D_S, where $d(i,i) = \mathit{lst}(t_i) - \mathit{est}(t_i)$, by D_S^*.

Now consider the sum $\sum_{i=1}^n (t_i^+ - t_i^-)$ we want to maximize. This sum can be rewritten as $\sum_{i=1}^n (t_i^+ - t_i^-) = \sum_{i=1}^n (t_i^+ - t_{\pi(i)}^-)$ for an arbitrary permutation π of $(1, 2, \ldots, n)$. Since $t_i^+ - t_{\pi(i)}^- \leq d(\pi(i), i)$, we can derive an upper bound on this sum for every permutation π: $\sum_{i=1}^n (t_i^+ - t_{\pi(i)}^-) \leq \sum_{i=1}^n d(\pi(i), i)$. In particular, there are permutations π^* such that for all permutations π it holds that $\sum_{i=1}^n d(\pi^*(i), i) \leq \sum_{i=1}^n d(\pi(i), i)$. Such a (smallest) permutation π^* specifies a least upper bound on $\mathit{flex}^*(S) = \max \sum_{i=1}^n (t_i^+ - t_i^-)$.

There is an efficient way to compute such a smallest permutation π^* by obtaining a *minimum weighted matching* between $T^+ = \{t_1^+, t_2^+, \ldots, t_n^+\}$ and $T^- = \{t_1^-, t_2^-, \ldots, t_n^-\}$: Consider the weighted complete graph $G_S = (V, E, w)$ where $V = T^+ \cup T^-$, the edges $\{t_i^+, t_j^-\} \in E$ for $i, j = 1, \ldots n$, and for every edge $\{t_i^+, t_j^-\}$ its weight $w_{i,j} = d_{j,i}$. Note that G_S is completely specified by D_S^*. By Hall's theorem, this graph has a minimum weighted perfect matching, that is a set $M \subseteq E$ of n edges covering T^+ and T^- such that the sum of these edges is minimum. Clearly, such a minimum weighted matching determines a permutation π^* such that $\sum_{i=1}^n d(\pi^*(i), i)$ is minimum. Since it holds that $\sum_{t_i \in T} (t_i^+ - t_i^-) \leq \sum_{i=1}^n d(\pi^*(i), i)$, the cost $c(M)$ of such a minimum matching provides an upper bound for $\mathit{flex}^*(S)$.

Actually, by applying simple LP-duality theory, we can do better and show that there exists an *exact* correspondence between a minimum matching in D_S^* and the maximum concurrent flexibility $\mathit{flex}^*(S)$:

Proposition 3.2. *Let $S = (T, C)$ be a consistent STN having a minimum distance matrix D_S. Then the concurrent flexibility $\mathit{flex}^*(S)$ of S equals the cost $c(M)$ of a minimum matching M of the complete weighted graph G_S specified by D_S^*.*

Proof. The most elegant way to prove this result is making use of duality in linear programming. The dual of LP3 is the following LP:

$$\begin{aligned} \text{minimize} \quad & \sum_{1 \le i,j \le n} d(i,j)\, y_{j,i} && \text{(LP4)} \\ \text{subject to} \quad & \sum_{j=1}^{n} y_{i,j} = 1 + y_{0,i} && , i \in \{1,2,\dots n\} \\ & \sum_{i=1}^{n} y_{i,j} = 1 + y_{0,j} && , j \in \{1,2,\dots n\} \\ & y_{i,j} \ge 0 && , i,j \in \{0,1,\dots,n\} \end{aligned}$$

Since the original LP (LP3) has an optimal solution, by strong duality, the objective values of optimal solutions of both LP's coincide. The constraint matrix associated with both LP's is total unimodular, hence, LP4 has integral optimal solutions. This dual LP4 can be interpreted as follows: Let G_S be a complete bipartite graph having two sets of nodes $A = \{1,\dots,n\}$ and $B = \{1,\dots n\}$, representing the rows and columns of D_S^*. Let D_S^* specify the weights $d(j,i)$ of the edges $e = (i,j)$ of G_S. Then, in case $y_{0,i} = 0$ for all $i = 1,2,\dots n$), it is not difficult to see that a minimizer for LP4 specifies a minimum weight matching M in G_S. By strong duality, the cost $c(M)$ of M corresponds to $\mathit{flex}^*(S)$.

Therefore, the only part left to prove is to show that the assumption $y_{0,i} = 0$ for all $i = 1,2,\dots n$ does not affect the *value* of the solutions to LP4:

Claim Whenever LP4 has an optimal integral solution, there also exists an integral solution of LP4 with at most the same cost such that $y_{0,i} = 0$, for all $i = 1,2,\dots,n$.

Proof of the Claim Suppose there exists an integral solution $\underline{y}^* = (y_{i,j}^*)$ for LP4 where $y_{0,i}^* > 0$ for some $1 \le i \le n$. Then there exist $j,k \in \{1,\dots,n\}$, $j \ne i$ and $k \ne i$, such that $y_{i,j}^* = 1$ and $y_{k,i}^* = 1$.[5] This condition violates the matching conditions. But then, since $d(j,k) \le d(j,i) + d(i,k)$, there exists a solution $\mathbf{y} = (y_{i,j})$ such that $c(\mathbf{y}) \le c(\mathbf{y}^*)$ where $y_{i,j} = y_{k,i} = 0$ and $y_{k,j} = 1$, and $y_{0,i} = y_{0,i}^* - 2 + y_{k,j}^*$. Hence, there exists a solution $\mathbf{y}$ with at most the same cost as $\mathbf{y}^*$ such that $0 \le y_{0,i} < y_{0,i}^*$. Iterating this procedure, we conclude that there exists a solution such that $y_{0,i} = 0$ as well, with cost not exceeding the cost of the original solution. This procedure can be repeated until we have obtained an optimal solution where it holds that $y_{0,i} = 0$ for all i, i.e., a perfect minimum weight matching, thereby proving the claim. □

A maximum matching M for D_S^* can be determined immediately: Note that the value of any matching M' realised by a permutation π' satisfies $\sum_{i=1}^n d(i,\pi'(i)) \le \sum_{i=1}^n d(i,0)+d(0,\pi'(i)) = \sum_{i=1}^n d(i,0)+d(0,i) = \sum_{i=1}^n \mathit{lst}(t_i) - \mathit{est}(t_i) = \mathit{flex}(S)$. Since $d(i,i) = \mathit{lst}(t_i) - \mathit{est}(t_i)$, the matching $M = \{(i,i) : i = 1,2,\dots,n\}$ realized by $\pi(i) = i$, is a maximum matching for D_S^*:

[5] Wlog. we may assume $y_{i,j}^* \in \{0,1\}$, $1 \le i,j \le n$: If $y_{i,j}^* > 1$ then there exists a solution $\mathbf{y}'$ with $c(\mathbf{y}') \le c(\mathbf{y}^*)$ s.t. $y'_{i,j} = 1$ and $y'_{0,i} = y_{0,i}^* - (y_{i,j}^* - 1)$.

Proposition 3.3. *Let $S = (T, C)$ be a consistent STN having a minimum distance matrix D_S. Then the* naive flexibility *flex(S) of S equals the cost $c(M)$ of a* maximum matching *M of the weighted graph specified by D_S^*.*

3.5 From $O(n^5)$ to $O(n^3)$ to compute flexibility

The complexity of computing the concurrent flexibility of an STN by an LP method depends on the exact method used for the LP solver. Currently, the best (interior-point based) LP-solvers have a complexity of $O(n^3L)$ [78] where n is the number of unknowns to solve for (the dimension of the vector x) and L is the input complexity, i.e., the bit length of the input description. This means that given an STN $S = (T, C)$, the currently best LP-solvers could need $O(n^3m)$-time to find the concurrent flexibility *flex*$^*(S)$, where $n = |T|$ and $m = |C|$. Since $m = O(n^2)$, we could expect a worst-case running time $O(n^5)$ to compute *flex*$^*(S)$.

The correspondence (flexibility $\equiv$ matching) obtained in the previous section allows us to present a better upper bound of the running time needed to compute the maximum concurrent flexibility. For, we know that given an STN S its minimal distance matrix D_S can be computed in $O(n^3)$ time. A minimum matching algorithm based on the specification of D_S also requires $O(n^2 \log n + n \times n^2) = O(n^3)$-time [34]. Hence, the total computation time for determining a minimum matching is $O(n^3)$-time. Note that computing the naive flexibility also requires an $O(n^3)$ computation of earliest and latest times for the temporal variables, both of which can be obtained via D_S.

Hence, in conclusion, we obtain the following result:

Proposition 3.4. *The concurrent flexibility as well as the naive flexibility of an STN S can be computed in $O(n^3)$-time.*

3.6 Conclusion

We established a connection between the computation of two flexibility metrics and properties of the minimal distance matrix D_S of an STN S: computing the concurrent flexibility metric is identical to computing a minimum weight matching of a weighted bipartite graph completely specified by the minimum distance matrix D_S. On the other hand, computing the naive flexibility metric corresponds to computing a maximum matching in this graph.

Chapter 4

Flexible dynamic decoupling

4.1 Introduction

In quite a number of cases a joint task has to be performed by several actors, each controlling only a part of the set of task constraints. For example, consider making a joint appointment with a number of people, constructing a building that involves a number of (sub)contractors, or finding a suitable multimodal transportation plan involving different transportation companies. In all these cases, some task constraints are under the control of just one actor (agent), while others require more than one agent setting the right values to the variables to satisfy them. Let us call the first type of constraints *intra-agent* constraints and the second type *inter-agent* constraints.

If there is enough time, solving such multi-actor constraint problems might involve consultation, negotiation or other agreement technologies. Sometimes, however, we have to deal with problems where communication between agents during problem solving is not possible or unwanted. For example, if the agents are in competition, there are legal or privacy issues preventing communication, or there is simply no time for communication.

In this paper we use an approach to solve multi-actor constraint problems in the latter contexts: instead of using agreement technologies, we try to avoid them by providing *decoupling techniques*. Intuitively, a decoupling technique modifies a multi-actor constraint system such that each of the agents is able to select a solution for its own set of (intra-agent) constraints and a simple merging of all individual agent solutions always satisfies the total set of constraints. Usually, this is accomplished by tightening intra-agent constraints such that inter-agent constraints are implied. Examples of real-world applications of such decoupling techniques can be found in e.g. [19] and [93].

Quite some research focuses on finding suitable decoupling techniques (e.g., [45, 19, 15]) for Simple Temporal Problems (STPs) [29]. An STP $S = (T, C)$

is a constraint formalism where a set of temporal variables $T = \{t_0, t_1, \ldots, t_n\}$ are subject to binary difference constraints C and solutions can be found in low polynomial ($O(n^3)$) time. Since STPs deal with temporal variables, a decoupling technique applied to STPs is called *temporal decoupling*. The quality of a temporal decoupling technique depends on the degree to which it tightens intra-agent constraints: the more it restricts the flexibility of each individual agent in solving their own part of the constraints, the less it is preferred. Therefore, we need a flexibility metric to evaluate the quality of a temporal decoupling.

Originally, flexibility was measured by summing the differences between the highest possible (latest) and the lowest possible (earliest) values of all variables in the constraint system after decoupling ([45, 44]). This so-called *naive flexibility metric* has been criticized, however, because it does not take into account the dependencies between the variables and, in general, seriously overestimates the "real" amount of flexibility. An alternative metric, the *concurrent flexibility metric* has been proposed in [98]. This metric accounts for dependencies between the variables and is based on the notion of an *interval schedule* for an STP S: For each variable $t_i \in T$ an interval schedule defines an interval $[l_i, u_i]$, such that choosing a value within $[l_i, u_i]$ for each variable t_i, always constitutes a solution for S. The sum $\sum_{i=1}^{n}(u_i - l_i)$ of the widths of these intervals determines the flexibility of S. The concurrent flexibility of S then is defined as the maximum flexibility we can obtain for an interval schedule of S and can be computed in $O(n^3)$ (see [64]).

As shown in [98], the concurrent flexibility metric can also be used to obtain an optimal (i.e. maximum flexible) temporal decoupling of an STP. This decoupling is a *total decoupling*, that is, a decoupling where the n variables are evenly distributed over n independent agents and thus every agent controls exactly one variable. It has been shown that this total decoupling is optimal for every partitioning of the set of temporal variables if one considers the concurrent flexibility metric as the flexibility metric to be used. In this paper, we concentrate on such (optimal) total decouplings of an STP.

In all existing approaches, a single temporal decoupling is computed in advance and is not changed, even if later on some agents announce their commitment to a specific value (or range of values) for a variable they control. Intuitively, however, we can use such additional information for the benefit of all agents, by possibly increasing the flexibility of variables they are not yet committed to. Specifically, when an agent announces a commitment to a sub-range of values within the given interval schedule (that represents the current decoupling), we are interested in updating the decoupling such that the individual flexibility of no agent is affected negatively.

More precisely, the overall aim of this paper is to show that a decoupling update method with the following nice properties do exist: first of all, it never affects the current flexibility of the agents negatively, and, secondly, it never decreases (and possibly increases) the individual flexibility of the variables not yet committed to. We will also show that updating a temporal decoupling as the result of a commitment for a single variable can be done in almost linear (amortized) time ($O(n \log n)$), which compares favourably with the cost

of computing a new optimal temporal decoupling ($O(n^3)$).

Organisation In Sect. 4.2 we discuss existing relevant work on STPs and temporal decoupling (TD). Then, in Sect. 4.3, extending some existing work, we briefly show how a total TD can be computed in $O(n^3)$, using a minimum matching approach. In Sect. 4.4, we first provide an exact approach to update an existing decoupling after commitments to variables have been made. We conclude, however, that this adaptation is computationally quite costly. Therefore, in Sect. 4.5 we offer an alternative, heuristic method, that is capable to adapt a given temporal decoupling in almost linear time per variable commitment. To show the benefits of adapting the temporal decoupling, in Sect. 4.6 we present the results of some experiments using STP benchmarks sets with the heuristic decoupling update and compare the results with an exact, but computationally more intensive updating method. We end with stating some conclusions and suggestions for further work.

4.2 Preliminaries

Simple Temporal Problems

A Simple Temporal Problem (STP) (also known as a Simple Temporal Network (STN)) is a pair $S = (T, C)$, where $T = \{t_0, t_1, \ldots, t_n\}$ is a set of temporal variables (events) and C is a finite set of binary difference constraints $t_j - t_i \leq c_{ij}$, for some real number c_{ij}.[1]The problem is to find a solution, that is a sequence $(s_0, s_1, s_2, \ldots, s_n)$ of values such that, if each $t_i \in T$ takes the value s_i, all constraints in C are satisfied. If such a solution exists, we say that the STN is *consistent*. [2] In order to express absolute time constraints, the time point variable $t_0 \in T$, also denoted by z, is used. It represents a fixed reference point on the timeline, and is always assigned the value 0.

Example 4.1. *Consider two trains 1 and 2 arriving at a station. Train 1 has to arrive between 12:05 and 12:15, train 2 between 12:08 and 12:20. People traveling with these trains need to change trains at the station. Typically, one needs at least 3 minutes for changing trains. Train 1 will stay during 5 minutes at the station and train 2 stays 7 minutes before it departs. Let t_i denote the arrival time for train $i = 1, 2$. Let $t_0 = z = 0$ represent noon (12:00). To ensure that all passengers have an opportunity to change trains, we state the following STP $S = (T, C)$ where: $T = \{z, t_1, t_2\}$, and $C = \{5 \leq t_1 - z \leq 15,\ 8 \leq t_2 - z \leq 20,\ -2 \leq t_2 - t_1 \leq 4\}$. As the reader may verify, two possible solutions[3] for this STP are $s = (0, 10, 10)$ and $s' = (0, 15, 17)$. That is, there is a solution when both trains arrive at 12:10, and there is also a solution where train 1 arrives at 12:15, while train 2 arrives at 12:17.* ■

[1]If both $t_j - t_i \leq c_{ij}$ and $t_i - t_j \leq c_{ji}$ are specified, we sometimes use the more compact notations $-c_{ji} \leq t_j - t_i \leq c_{ij}$ or $t_j - t_i \in [-c_{ji}, c_{ij}]$.

[2]W.l.g., in the remainder of the paper we simply assume an STN to be consistent. Consistency of an STN can be determined in low-order polynomial time [30].

[3]Of course, there are many more.

An STP $S = (T, C)$ can also be specified as a directed weighted graph $G_S = (T_S, E_S, l_S)$ where the vertices T_S represent variables in T and for every constraint $t_j - t_i \leq c_{ij}$ in C, there is a directed edge $(t_i, t_j) \in E_S$ labeled by its weight $l_S(t_i, t_j) = c_{ij}$. The weight c_{ij} on the arc (t_i, t_j) can be interpreted as the length of the path from t_i to t_j. Using a shortest path interpretation of the STP $S = (T, C)$, there is an efficient method to find all shortest paths between pairs (t_i, t_j) using e.g. Floyd and Warshall's all-pairs shortest paths algorithm [33]. A shortest path between time variables t_i and t_j then corresponds to a *tightest constraint* between t_i and t_j. These tightest constraints can be collected in an $n \times n$ *minimum distance matrix* $D = [d_{ij}]$, containing for every pair of time-point variables t_i and t_j the length d_{ij} of the *shortest path* from t_i to t_j in the distance graph. In particular, the first row and the first column of the distance matrix D contain useful information about the possible schedules for S: The sequence $lst = (d_{00}, d_{01}, \ldots, d_{0n})$ specifies the latest starting time solution for each time point variable t_i. Analogously, $est = (-d_{00}, -d_{10}, \ldots, -d_{no})$ specifies the *earliest starting time* solution.

Example 4.2. *Continuing Example 4.1, the minimum distance matrix D of S equals*

$$D = \begin{bmatrix} 0 & 15 & 19 \\ -5 & 0 & 4 \\ -8 & 2 & 0 \end{bmatrix}$$

The earliest time solution therefore is $est = (0, 5, 8)$, and the latest time solution $lst = (0, 15, 19)$. ■

Given S, the matrix D can be computed in low-order polynomial ($O(n^3)$) time, where $n = |T|$, see [29].[4] Hence, using the STP-machinery we can find earliest and latest time solutions quite efficiently.

Temporal Decoupling

In order to find a solution for an STP $S = (T, C)$ all variables $t_i \in T$ should be assigned a suitable value. Sometimes these variables are controlled by different agents. That is, $T - \{z\} = \{t_1, \ldots, t_n\}$ is partitioned into k non-empty and non-overlapping subsets $T_1, \ldots, T_k$ of T, each T_j corresponding to the set of variables controlled by agent $a_j \in A = \{a_1, a_2, \ldots, a_k\}$. Such a partitioning of $T - \{z\}$ will be denoted by $[T_i]_{i=1}^k$. In such a case, the set of constraints C is split in a set $C_{intra} = \bigcup_{i=1}^k C_i$ of intra-agent constraints and a set $C_{inter} = C - C_{intra}$ of inter-agent constraints. Here, a constraint $t_i - t_j \leq c_{ji}$ is an intra-constraint occurring in C_h if there exists a subset T_h such that $t_i, t_j \in T_h$, else, it is an inter-agent constraint. Given the partitioning $[T_j]_{j=1}^k$, every agent a_i completely controls the (sub) STP $S_i = (T_i \cup \{z\}, C_i)$, where C_i is its set of intra-agent constraints, and determines a solution s_i for it, independently from the others. In general, however, it is not the case that, using these sub STPs, merging partial solutions s_i will always constitute a total solution to S:

[4] An improvement by Planken et al. [71] has shown that a schedule can be found in $O(n^2 w_d)$-time where w_d is the graph width induced by a vertex ordering.

Example 4.3. *Continuing Example 4.1, let train i be controlled by agent a_i for $i = 1, 2$ and assume that we have computed the set of tightest constraints based on C. Then $S_1 = (\{z, t_1\}, \{\ 5 \leq t_1 - z \leq 15\})$ and $S_2 = (\{z, t_2\}, \{\ 8 \leq t_2 - z \leq 19\})$. Agent a_1 is free to choose a time t_1 between 5 and 15 and suppose she chooses 10. Agent a_2 controls t_2 and, therefore, can select a value between 8 and 19. Suppose he chooses 16. Now clearly, both intra-agent constraints are satisfied, but the inter-agent constraint $t_2 - t_1 \leq 4$ is not, since $16 - 10 > 4$. Hence, the partial solutions provided by the agents are not conflict-free.* ■

The *temporal decoupling* problem for $S = (T, C)$, given the partitioning $[T_j]_{j=1}^k$, is to find a suitable set $C'_{intra} = \bigcup_{i=1}^k C'_i$ of (modified) intra-agent constraints such that if s'_i is an arbitrary solution to $S'_i = (T_i \cup \{z\}, C'_i)$, it always can be merged with other partial solutions to a total solution s' for S.[5]

We are interested, however, not in arbitrary decouplings, but an *optimal decoupling* of the k agents, allowing each agent to choose an assignment for its variables independently of others, while maintaining *maximum flexibility*. This optimal decoupling problem has been solved in [97] for the total decoupling case, that is the case where $k = n$ and each agent a_i controls a single time point variable t_i. In this case, the decoupling results in a specification of a lower bound l_i and an upper bound u_i for every time point variable $t_i \in T$, such that t_i can take any value $v_i \in [l_i, u_i]$ without violating any of the intra- or inter-agent constraints. This means that if agent a_i controls T_i then her set of intra-agent constraints is $C_i = \{l_j \leq t_j \leq u_j \mid t_j \in T_i\}$.

The total flexibility the agents have, due to this decoupling, is determined by the sum of the differences $(u_i - l_i)$. Therefore, the decoupling bounds $l = (l_i)_{i=1}^n$ and $u = (u_i)_{i=1}^n$ are chosen in such a way that the flexibility $\sum_i (u_i - l_i)$ is maximized. It can be shown (see [97]) that such a pair (l, u) can be obtained as a maximizer of the following LP:

$$\begin{aligned}
\max_{l,u} \quad & \sum_i (u_i - l_i) && && (\mathbf{TD}(D)) \\
\text{s.t.} \quad & l_0 = u_0 = 0 && && (4.1) \\
& l_i \leq u_i && \forall i \in T && (4.2) \\
& u_j - l_i \leq d_{ij} && \forall i \neq j \in T && (4.3)
\end{aligned}$$

where D is the minimum distance matrix associated with S.

Example 4.4. *Consider the matrix D in Example 4.2 and the scenario sketched in Example 4.3. Then the LP whose maximizers determine a maximum decoupling*

[5] In other words, the set $\bigcup_{i=1}^k C'_i$ logically implies the set C of original constraints.

is the following:

$$\begin{aligned} \max_{l,u} \quad & \sum_i (u_i - l_i) \\ s.t. \quad & u_0 = l_0 = 0, \\ & l_1 \leq u_1, \quad l_2 \leq u_2 \\ & u_1 - l_0 \leq 15, \quad u_2 - l_0 \leq 19 \\ & u_0 - l_1 \leq -5, \quad u_2 - l_1 \leq 4 \\ & u_0 - l_2 \leq -8, \quad u_1 - l_2 \leq 2 \end{aligned}$$

Solving this LP, we obtain $\sum_{i=0}^{2}(u_i - l_i) = 6$ *with maximizers* $l = (0, 15, 13)$ *and* $u = (0, 15, 19)$*. This implies that in this decoupling* (l, u) *agent* a_1 *is forced to arrive at 12:15, while agent* a_2 *can choose to arrive between 12:13 and 12:19.* ■

Remark Note that the total decoupling solution (l, u) for S also is a solution for a decoupling based on an arbitrary partitioning $[T_i]_{i=1}^{k}$ of S. Observe that (l, u) is a decoupling, if for any value v_i chosen by a_h and any value w_j chosen by $a_{h'}$ for every $1 \leq h \neq h' \leq k$, we have $v_i - w_j \leq d_{ji}$ and $w_j - v_i \leq d_{ij}$. Since (l, u) is a total decoupling, it satisfies the conditions of the LP $\mathbf{TD}(D)$. Hence, $u_i - l_j \leq d_{ij}$ and $u_j - l_i \leq d_{ji}$. Since $v_i \in [l_i, u_i]$ and $w_j \in [l_j, u_j]$, we then immediately have $v_i - w_j \leq u_i - l_j \leq d_{ji}$ and $w_j - v_i \leq u_j - l_i \leq d_{ji}$. Therefore, whatever choices are made by the individual agents satisfying their local constraints, these choices always will satisfy the original constraints, too. ■

Remark In [97] the (l, u) bounds found by solving the LP $\mathbf{TD}(D)$ are used to compute the concurrent flexibility $\mathit{flex}(S) = \sum_i^n (u_i - l_i)$ of an STP S. Taking the concurrent flexibility as our flexibility metric, the (l, u) bounds for decoupling are always optimal, whatever partitioning $[T_i]_{i=1}^{n}$ is used: first, observe that due to a decoupling, the flexibility of an STN can never increase. Secondly, if the (l, u) bounds for a total decoupling are used, by definition, the sum $\sum_{i=1}^{k} \mathit{flex}(S_i)$ of the (concurrent) flexibilities of the decoupled subsystems equals the flexibility $\mathit{flex}(S)$ of the original system. Hence, the total decoupling bounds (l, u) are optimal, whatever partitioning $[T_i]_{i=1}^{n}$ used. ■

In this paper, we will consider concurrent flexibility as our flexibility metric. Hence, a total decoupling is always an optimal decoupling for any partitioning of variables. Therefore, in the sequel, we concentrate on total decouplings.

4.3 Total decoupling by minimum matching

In the introduction we mentioned that an (optimal) total decoupling can be achieved in $O(n^3)$ time. In the previous section, we presented an LP to compute such a decoupling. If the STP has n variables, the LP to be solved has $2n$ variables and n^2 constraints. Modern interior-point methods solve LPs with n variables and m constraints in roughly $O(n^3 m)$ [78]. Thus, the complexity

of solving total decoupling as an LP might be as high as $O(n^5)$. In a previous paper ([64]), we have shown that computing the concurrent flexibility of an STP can be reduced to a *minimum matching* problem (see [69]) using the distance matrix D to construct a weighted cost matrix D^* for the latter problem.

This reduction, however, does not allow us to directly compute the corresponding flexibility maximizers (l, u). In this section we therefore show that there is a full $O(n^3)$ alternative method for the LP-based flexibility method to compute the concurrent flexibility and the corresponding maximizers (l, u), thereby showing that an optimal total decoupling can be computed in $O(n^3)$.

Flexibility and minimum matching

Given an STN $S = (T, C)$ with minimum distance matrix D, let $\mathit{flex}(S)$ be its concurrent flexibility, realised by the maximizers (l, u). Hence, $\mathit{flex}(S) = f(l, u) = \sum_{i=1}^{n}(u_i - l_i)$. Unfolding this sum –as was noticed in [64]– we obtain

$$f(l,u) = \sum_{i\in T}(u_i - l_i) = \sum_{i\in T}(u_{\pi_i} - l_i) \tag{4.4}$$

for every permutation π of T.[6] Since (l, u) is a total decoupling, we have

$$u_j - l_i \leq d_{ij} \qquad \forall i \neq j \in T \tag{4.5}$$

$$u_j - l_j = (u_j - z) + (z - l_j) \leq d_{0j} + d_{j0} \qquad \forall j \in T \tag{4.6}$$

Hence, defining the modified distance matrix (also called *weight matrix*) $D^* = [d^*_{ij}]_{n\times n}$ by

$$d^*_{ij} = \begin{cases} d_{ij}, & 1 \leq i \neq j \leq n \\ d_{0i} + d_{i0}, & 1 \leq i = j \leq n \end{cases}$$

we obtain the following inequality:

$$f(l,u) \leq \min\{\sum_{i\in T} d^*_{i\pi_i} : \pi \in \Pi(T)\} \tag{4.7}$$

where $\Pi(T)$ is the space of permutations of T. Equation (4.7) states that the maximum flexibility of an STN is upper bounded by the value of a minimum matching in a bipartite graph with weight matrix D^*. The solution of such a matching problem consists in finding for each row i in D^* a unique column π_i such that the sum $\sum_{i\in T} d^*_{i\pi_i}$ is minimized. As we showed in [64], by applying LP-duality theory, equation (4.7) can be replaced by an equality: $f(l, u) = \min_{\pi\in\Pi(T)} \sum_{i\in T} d^*_{i\pi_i}$, so the concurrent flexibility $\mathit{flex}(S) = f$ can be computed by a minimum matching algorithm as e.g. the Hungarian algorithm, running in $O(n^3)$ time.

[6] To avoid cumbersome notation, we will often use $i \in T$ as a shorthand for $t_i \in T$.

Finding a maximizer (l, u) using minimum matching

With the further help of LP-duality theory i.c., complementary slackness conditions ([69]), the following result is immediate:

Observation 4.1. *If π is a minimum matching with value m for the weight matrix D^*, then there exists a maximizer (l, u) for the concurrent flexibility flex(S) of S, such that flex$(S) = m$ and for all $i \in T$, $u_{\pi_i} - l_i = d^*_{i\pi_i}$.*

Now observe that the inequalities stated in the LP-specification $\mathbf{TD}(D)$ and the inequalities $u_{\pi_i} - l_i = d^*_{i\pi_i}$ in Observation 4.1 all are binary difference constraints. Hence, the STP $S' = (T', C')$, where

$$T' = L \cup U = \{l_i \mid i \in T\} \cup \{u_i \mid i \in T\},$$
$$C' = \{u_i - l_j \leq d^*_{ij} \mid i, j \in T\} \cup \{l_i - u_{\pi_i} \leq -d^*_{i\pi_i} \mid i \in T\} \cup \{l_i - u_i \leq 0 \mid i \in T\}$$

is an STP[7] and every solution $s' = (l_1, \ldots l_n, u_1, \ldots, u_n)$ of S' in fact is a maximizer (l, u) for the original STP S, since the flexibility associated with such a solution (l, u) satisfies $\mathit{flex}(S) \geq f(l, u) = \sum_{i \in T}(u_i - l_i) \geq \sum_{i \in T} d^*_{i\pi_i} = \mathit{flex}(S)$. Hence, this pair (l, u) is a maximizer realizing $\mathit{flex}(S)$.

In particular, the earliest and latest solutions of S' have this property. Hence, since (i) D^* can be obtained in $O(n^3)$ time, (ii) a minimum matching based on D^* can be computed in $O(n^3)$, and (iii) the STN S' together with a solution $s = (l, u)$ for it can be computed in $O(n^3)$, we obtain the following result:

Corollary 4.1. *Given an STN $S = (T, C)$ with distance matrix D, an optimal total decoupling (l, u) for S can be found in $O(n^3)$.*

4.4 Dynamic Decoupling by updating

A temporal decoupling allows agents to make independent choices or commitments to variables they control. As pointed out in the introduction, we want to adapt an existing (total) temporal decoupling (l, u) whenever an agent makes a new commitment to one or more temporal variables (s)he controls. To show that such a commitment could affect the flexibility other agents have in making their possible commitments, consider our leading example again:

Example 4.5. *In Example 4.3 we obtained a temporal decoupling $(l, u) = ((0, 15, 13), (0, 15, 19))$. Here, agent 1 was forced to arrive at 12:15, but agent 2 could choose to arrive between 12:13 and 12:19. Suppose agent 2 commits to arrive at 12:13. As a result, agent 1 is able to arrive at any time in the interval $[9, 15]$. Then, by adapting the decoupling to the updated decoupling $(l', u') = ((0, 9, 13), (0, 15, 13))$, the flexibility of agent 1 could increase from 0 to 6, taking into account the new commitment agent 1 has made. If the existing commitment (l, u), however, is not updated, agent 1 still has to choose 12:15 as its time of arrival.* ■

[7] We should note that this STP has two external reference variables $u_0 = l_0 = 0$.

In order to state this *dynamic decoupling* or *decouple updating* problem more precisely, we assume that at any moment in time the set T consists of a subset T_c of variables committed to and a set of not committed to, or *free*, variables T_f. The commitments for variables $t_i \in T_c$ are given as bounds $[l_i^c, u_i^c]$, specifying that for $i \in T_c$, t_i is committed[8] to choose a value in the interval $[l_i^c, u_i^c]$. The total set of commitments in T_c is given by the bounds (l^c, u^c). We assume that these committed bounds do not violate decoupling conditions.

Whenever (l, u) is a total decoupling for $S = (T, C)$, where $T = T_c \cup T_f$, we always assume that (l, u) respects the commitments, i.e. $[l_i, u_i] = [l_i^c, u_i^c]$ for every $t_i \in T_c$, and (l, u) is an optimum decoupling for the free variables in T_f, given these commitments. Suppose now an agent makes a new commitment for a variable $t_i \in T_f$. In that case, such a commitment $[v_i, w_i]$ should satisfy the existing decoupling, that is $l_i \leq v_i \leq w_i \leq u_i$, but as a result, the new decoupling where $[l_i, u_i] = [v_i, w_i]$, $T_c' = T_c \cup \{t_i\}$,and $T_f' = T_f - \{t_i\}$ might no longer be an optimal decoupling w.r.t. T_f' (e.g. see Example 4.5). In that case we need to update (l, u) and to find a better decoupling (l', u').

The decoupling updating problem then can be stated as follows:

> Given a (possibly non-optimal) total decoupling (l, u) for an STP $S = (T, C)$, with $T = T_c \cup T_f$, find a new total decoupling (l', u') for S such that
>
> 1. no individual flexibility of free variables is negatively affected, i.e., for all $j \in T_f$, $[l_j, u_j] \sqsubseteq [l_j', u_j']$[9];
> 2. all commitments are respected, that is, for all $j \in T_c$, $[l_j'^c, u_j'^c] = [l_j^c, u_j^c]$;
> 3. the flexibility realized by (l', u'), given the commitments (l'^c, u'^c), is maximum.

Based on the earlier shown total decoupling problem $\mathbf{TD}(D)$, this decoupling update problem can also be stated as the following LP:

[8] Note that this is slightly more general concept than a strict commitment of a variable t_i to one value v_i.

[9] that is, the interval $[l_j', u_j']$ contains $[l_j, u_j]$.

$$\begin{aligned}
\max \quad & \sum_j (u'_j - l'_j) && (\mathbf{DTD}(D, T_c, T_f, (l,u)) \\
\text{s.t.} \quad & u'_0 = l'_0 = 0 && && (4.8)\\
& u'_j - l'_i \leq d_{ij} && \forall i \neq j \in T && (4.9)\\
& u_j \leq u'_j,\ l'_j \leq l_j && \forall j \in T_f && (4.10)\\
& l'_j = l_j,\ u'_j = u_j && \forall j \in T_c && (4.11)
\end{aligned}$$

Here, (l, u) is the existing total decoupling.
In fact, by transforming **DTD**-instances into **TD**-instances, we can show that the dynamic decoupling (decoupling updating) problem can be reduced to a minimum-matching problem and can be solved in $O(n^3)$, too.[10]

4.5 A fast heuristic for updating

Although the dynamic total decoupling problem can be reduced to the static temporal decoupling problem, in practice, the computational complexity involved might be too high. While an initial decoupling might be computed off-line, an update of a decoupling requires an on-line adaptation process. Since the costs of solving such a dynamic matching problem are at least $O(n^2)$ per update, we would like to alleviate this computational investment. Therefore, in this section we discuss a fast heuristic for the decoupling updating problem. Using this heuristic, we show that an updated decoupling can be found in (amortized) $O(n \log n)$ per update step if $O(n)$ updates are assumed to take place.

The following proposition is a simple result we base our heuristic on:

Proposition 4.1. *If (l, u) is a total decoupling for S with associated weight matrix D^*, then, for all $j \in T$, $l_j = \max_{k \in T}(u_k - d^*_{kj})$ and $u_j = \min_{k \in T}(l_k + d^*_{jk})$.*

Proof. Since (l, u) is a maximizer of the LP $\mathbf{TD}(D)$, $u_i - l_j \leq d^*_{ij}$, for every $i, j \in T$. Hence, for each $i, j \in T$, we have $l_j \geq \max_{k \in T}(u_k - d^*_{kj})$ and $u_i \leq \min_{k \in T}(l_k + d^*_{ik})$. Now, on the contrary, assume that for some $i \in T$, $l_i > \min_{k \in T}(u_k - d^*_{ki})$. Then the bounds (l', u) where $l' = l$, except for $l'_i = \min_{k \in T}(u_k - d^*_{ki})$, would satisfy the constraints $u_i - l'_j \leq d^*_{ij}$ for every $i, j \in T$, as well as $l'_j \leq u_j$ for every $j \in T$. Hence, (l', u) is a decoupling as well. But then (l', u) satisfies the conditions of the LP $\mathbf{TD}(D)$ but $\sum_j (u_j - l'_j) > \sum_j (u_j - l_j)$, contradicting the fact that (l, u) is a maximizer of this LP. Hence, such an $i \in T$ cannot exist. The proof for $u_i < \min_{k \in T}(l_k + d_{ik})$ goes along the same line and the proposition follows. ■ □

The converse, however, of this proposition is not true: not every solution (l, u) satisfying the two equalities is a maximum decoupling. It can only be

[10] As has been observed by a reviewer, there exists an $O(n^2)$ algorithm for the dynamic variant of the minimum-matching problem. It is likely that our dynamic decoupling problem can be solved by such a dynamic minimum matching algorithm in $O(n^2)$ time, too.

shown that in such a case (l, u) is a *maximal* total decoupling. That means, if (l, u) satisfies the equations above, there does not exist a decoupling (l', u') containing (l, u) that has a higher flexibility.

It turns out that these *maximal total decouplings* and their updates can be computed very efficiently: given a (non-maximal) total decoupling (l, u), and a set $T = T_c \cup T_f$ of committed and free variables, there exists a very efficient algorithm to compute a new total decoupling $[l', u']$ such that

1. (l', u') preserves the existing commitments: for all $j \in T_c$, $[l'_j, u'_j] = [l_j, u_j]$;
2. $[l', u']$ respects the existing bounds of the free variables: for all $j \in T_f$, $[l_j, u_j] \sqsubseteq [l'_j, u'_j]$;
3. (l', u') satisfies the conditions stated in Proposition 4.1 above w.r.t. the free variables, i.e. (l', u') is a maximal decoupling with respect to variables in T_f.

The following surprisingly simple algorithm finds a maximal flexible total decoupling for a set $T_f \cup T_c$ of free and committed temporal variables that satisfies these conditions. The algorithm iteratively updates the existing (l, u)-decoupling bounds for the variables in T_f until all free variables satisfy the equations stated in Proposition 4.1.

Algorithm 1 Finding an update (l', u') of an existing total decoupling (l, u)

Require: (l, u) is a total decoupling for S; $D^* = [d^*_{ij}]$ is the weight matrix; $T = T_f \cup T_c$;

```
1: l' = l ; u' = u;
2: for i = 1 to |T_f| do
3:     min_i := max_{k∈T}(u'_k − d*_ik);
4:     max_i := min_{k∈T}(l'_k + d*_kj);
5:     if l'_i > min_i then
6:         l'_i ← min_i
7:     if u'_i < max_i then
8:         u'_i ← max_i
9: return (l', u');
```

It is easy to see that the algorithm preserves the existing commitments for variables in T_c, since only bounds of free variables in T_f are updated.

It is also easy to see that the existing bounds $[l_j, u_j]$ of free variables $j \in T_f$ are respected: For every j it holds that either $u_j < \max_j$ ($l'_j > \min_j$, respectively) or $u_j = \max_j$ ($l'_j > \min_j$, respectively). Hence, $u'_j \geq u_j$ and $l'_j \leq l_j$.

To show that the obtained decoupling (l', u') is maximal with respect to the free variables, we state two key observations: first of all, in every step i it holds that $l'_i \geq \min_i$ and $u'_i \leq \max_i$, because (l, u) is a decoupling and the $(\min_i, \max_i)$ bounds are not violated during updating. Secondly, once the bounds (l'_i, u'_i) for a variable $i \in T_f$ have been updated to $\min_i$ and $\max_i$, (l'_i, u'_i) will never need

to be updated again, since $\min_i$ depends on values u'_k that might increase or stay the same; hence $\min_i$ cannot decrease in subsequent steps. Likewise, $\max_i$ depends on values l'_k that might decrease or stay the same. Therefore, $\max_i$ cannot increase in later steps. Therefore, it is sufficient to update the bounds for the variables only once.

As a result, at the end all free variables have been updated and achieved their minimal/maximal bound. Then a maximal total decoupling has been obtained, since all variables in T_f will satisfy the equations stated in Proposition 4.1.

Example 4.6. *Continuing our example, notice that the weight matrix D^* obtained via the minimum distance matrix D (see Example 4.2) equals*

$$D^* = \begin{bmatrix} 0 & 15 & 19 \\ -5 & 10 & 4 \\ -8 & 2 & 6 \end{bmatrix}$$

Given the decoupling $(l, u) = ((0, 15, 13), (0, 15, 19))$ with $T_f = \{t_1, t_2\}$, let agent 2 commit to $t_2 = 13$. We would like to compute a new decoupling maximizing the flexibility for t_1. Note that $\min_1 = \max\{5, 5, 9\} = 9$ and $\max_1 = \min\{15, 25, 15\} = 15$. Hence, the heuristic finds an updated decoupling $(l', u') = ((0, 9, 13), (0, 15, 13))$.

Complexity of the heuristic

As the reader quickly observes, a naive implementation of Algorithm 1 would require $O(n^2)$-time to obtain an updated decoupling: To update a single variable $i \in T_f$, we have to compute $\min_i$ and $\max_i$. This costs $O(n)$ per variable and there may be n variables to update.

Fortunately, if there are $O(n)$ updating steps, the computational cost per step can be significantly reduced: First compute, for each $i \in T$, a priority queue $Q_{\min(i)}$ of the entries $u_k - d^*_{ik}$, $k \in T$, and a priority queue $Q_{\max(i)}$ of the entries $l_k + d^*_{ki}$, $k \in T$. The initialisation of these priority queues will cost $O(n \,.\, n \log n) = O(n^2 . \log n)$.

After a new commitment for a variable j, say $[v_j, w_j]$, we have to compute a decoupling update. It suffices to update the priority queues $Q_{\min(i)}$ and $Q_{\max(i)}$ for every $i \in T_f$. In this case, the element $u_j - d^*_{ij}$ in the queue $Q_{min(i)}$ has to be changed to $w_j - d^*_{ij}$ and the element l_j in queue $Q_{max(i)}$ has to be changed to $v_j + d^*_{ji}$. Maintaining the priority order in the priority queues will cost $O(\log n)$ per variable. Hence, the total cost for computing a new decoupling are $O(n \log n)$. If there are $O(n)$ updates, the total cost of performing these $O(n)$ updates are $O(n^2 . \log n) + O(n).O(n . \log n) = O(n^2 . \log n)$. Hence, the amortized time costs per update are $O(n . \log n)$.

In the next section we will verify the quality of such maximal flexible decouplings experimentally, comparing them with maximum flexible total decouplings and a static decoupling.

4.6 Experimental Evaluation

We discuss an experimental evaluation of the dynamic total decoupling method. These experiments have been designed (i) to verify whether updating a decoupling indeed significantly increases the decoupling flexibility compared to using a static decoupling and (ii) to verify whether the heuristic significantly reduces the computational investments in updating as expected.

Material We applied our updating method to a dataset of STP benchmark instances (see [70]). This dataset contains a series of sets of STP-instances with STPs of varying structure and number of variables. Table 4.1 contains the main characteristics of this benchmark set. The size of the STP-instances

Table 4.1: STP Benchmarksets used in the experiments.

benchmark set	**nr instances**	**min vars**	**av vars**	**max vars**
bdh	300	41	207	401
diamonds	130	111	857	2751
chordalgraphs	400	200	200	200
NeilYorkeExamples	180	108	1333	4082
sf-fixedNodes	112	150	150	150

in this dataset varies from instances with 41 variables with 614 constraints to instances with 4082 nodes and 14110 constraints. In total, these sets contain 1122 STP-instances. We used MATLAB (R2016b) on an iMac 3.2 Ghz, Intel Core i5, with 24Gb memory to perform the experiments.

Method For each instance in the benchmark set we first computed a maximum decoupling (l, u) with its flexibility $flex = \sum_{j=1}^{n}(u_j - l_j)$, using the minimum matching method. Then, according to an arbitrary ordering $< t_1, t_2, \ldots t_n >$ of the variables in T, each variable t_i is iteratively committed to a fixed value v_i, where v_i is randomly chosen in an interval (l_i, u_i) belonging to the current temporal decoupling (l, u). After each such a commitment, we compute a new updated decoupling (l^i, u^i) where $T_c = \{t_1, \ldots t_i\}$ and $T_f = \{t_{i+1}, \ldots t_n\}$. The total flexibility of the new decoupling (l^i, u^i) is now dependent upon the $n - i$ free variables in T_f and will be denoted by f_h^i. We initially set $f_h^0 = flex$. In order to account for the decreasing number of free variables after successive updates, we compute after each update the heuristic update flexibility per free variable in T_f: $f_h^i/(n - i)$. To compare these flexibility measures with the static case, for the latter we take the total flexibility of the free variables $\mathit{flex}^i = \sum_{j=i+1}^{n}(u_j - l_j)$ and then compute the flexibility per free time variable without updating: $\mathit{flex}^i/(n - i)$.

As a summary result for each benchmark instance k, we compute

1. the average over all updates of the static flexibility per free variable: $av\_stat = \sum_{i=1}^{n} \mathit{flex}^i/((n - i).n)$
2. the average over all updates of the heuristic update flexibility per free

variable:
$av\_h = \sum_{i=1}^{n} f_h^i/((n-i).n)$

Note that the ratio $rel\_flex_h = av\_h/av\_stat$ indicates the impact of the heuristic updating method for a particular instance: a value close to 1 indicates almost no added flexibility (per time variable) by updating, but a value of 2 indicates that the flexibility (per time variable) doubled by updating the decoupling.

Finally, we collected the rel_flex_h results per instance for each benchmark set and measured their minimum, mean and maximum values per benchmark set.

Results The rel_flex_h results are grouped by benchmark set and their minimum, mean, and maximum per benchmark set are listed in Table 4.2.

Table 4.2: Statistics of flexibility ratio's rel_flex_h of decoupling updates vss static decoupling per benchmark set.

benchmark set	**min**	**mean**	**max**
bdh-agent-problems	1.00	1.00	1.002
diamonds	1.34	1.95	2.39
chordalgraphs	1.08	1.31	1.65
NeilYorkeExamples	1.20	1.74	2.39
sf-fixedNodes	1.21	1.38	1.78

As can be seen, except for bdh-agent-problems, dynamic updating of a temporal decoupling increases the mean flexibility per variable rather significantly: For example, in the diamonds and NeilYorke benchmark sets, updating almost doubled the flexibility per free time variable.[11]

We conclude that, based on this set of benchmarks, one might expect a significant increase in flexibility if a decoupling is updated after changes in commitments have been detected, compared to the case where no updating is provided.

To verify whether the heuristic was able to reduce the computation time significantly, unfortunately, we can only present partial results. The reason is that for the more difficult instances in these benchmark sets, computing the updates with an exact minimum matching method was simply too time-consuming.[12] We therefore selected from each benchmark set the easy[13] instances and collected

[11] The reason that in the bdh-agent problems the updating did not increase, is probably due to the fact that in these instances, as we observed, the flexibility was concentrated in a very few time point variables. Eliminating the flexibilities by commitments of these variables did not affect the flexibility of the remaining variables that much.

[12] Some of the more difficult benchmark problems even did not finish within 36 hrs.

[13] A benchmark problem instance in [70] was considered to be ''easy" if the exact update method finished in 15 minutes. For the bdh-agent set we collected the instances until easybdh 8_10_50_350_49, for the diamonds set all instances up to diamonds-385.0, for the chordal graphs all instances until chordal-fixedNodes-150,5,1072707262, for the NeilYorkeExamples all instances until ny-419,10, and for the sf-fixedNodes the instances until sf-fixedNodes-150,5,1072707262.

them in the easy-<benchmark> variants of the original benchmark sets. We then measured the mean computation time per benchmark set for both the exact and heuristic updating method and also the mean performance ratio rel_flex/rel_flex_h of the exact versus the heuristic update method. The latter metric indicates how much the exact method outperforms the heuristic method in providing flexibility per time variable. The results are collected in Table 4.3.

Table 4.3: Comparing the time (sec.) and performance ratio of the exact and heuristic updating methods (easy variants of benchmark instances)

benchmark set	**cpu heuristic**	**cpu exact**	**perf. ratio**
easy-bdh-agent-problems	0.21	2.75	1.00
easy-diamonds	0.38	67.65	1.05
easy-chordalgraphs	0.78	12.6	1.06
easy-NeilYorkeExamples	0.61	135.62	1.01
easy-sf-fixedNodes	0.13	4.71	1.03

From these results we conclude that even for the easy cases, the heuristic clearly outperforms the exact update method, being more than 10 times and sometimes more than 200 times faster. We also observe that the heuristic method does not significantly lose on flexibility: The performance ratio's obtained are very close to 1.

4.7 Conclusions and Discussion

We have shown that adapting a decoupling after a new variable commitment has been made in most cases significantly increases the flexibility of the free, non-committed, variables. We also showed that this updating method did not induce any disadvantage to the actors involved: every commitment is preserved and the current decoupling bounds are never violated. We introduced a simple heuristic that replaces the rather costly computation of a decoupling with maximum flexibility by a computation of a decoupling with maximal flexibility. This heuristic reduces the computation time per update from $O(n^3)$ to $O(n.\log n)$ (amortized) time. As we showed experimentally, the computational investments for the heuristic are significantly smaller, while we observed almost no loss of flexibility compared to the exact method.

In the future, we want to extend this work to computing flexible schedules for STPs: Note that the update heuristic also can be used to find a maximal flexible schedule given a solution s of an STP. Such a solution is nothing more than a non-optimal decoupling (s, s) for which, by applying our heuristic, an $O(n \log n)$ procedure exists to find an optimal flexible schedule. Furthermore, we are planning to construct dynamic decoupling methods in a distributed way, like existing approaches to static decoupling methods have done.

Part II

Stochastic Task Networks

Chapter 5

Preliminaries

In Chapter 1 we identified two main sources of uncertainty affecting the execution of tasks, or activities, within the NedTrain maintenance workshop. In the first part of the thesis we focused on uncertainty originating in giving actors some room to decide, just-in-time, the exact dispatching time of their tasks, according to an unknown decision process. In the second part of this thesis we eliminate the source of uncertainty investigated in the first part by assuming actors behave deterministicaly and follow an earliest start dispatching strategy. That is, we assume actors will always choose to dispatch a task immediately when it becomes possible under certain constraints imposed by the dispatching process. At the same time, in the second part of this thesis we focus on the other major source of uncertainty, which we identified in Chapter 1 to stem from our inability to predict the actual time needed for the execution of a task. In other words, uncertainty now stems from tasks having unpredictable durations, while actors behave predictably.

In Part II we shall focus on problems related to dispatching stochastic task networks, also known as stochastic activity networks. Such a network is typically a directed acyclic graph, the nodes of which represent tasks (or activities) with *random durations*, the probability distributions of which are given. A directed edge between two tasks specifies a *precedence constraint*. Such constraints dictate that a task may only get dispatched after all its *predecessors* (i.e. tasks with an outgoing arc to it) have finished. Such networks have been used in various domains where there is a need to model a partial order of events with random durations. An early application of the concept lead to the so-called *PERT networks* [59], for modelling projects executed in an uncertain environment.

As in scheduling we usually seek to optimize efficiency, a very common objective is that of minimizing *makespan*–the overall completion time of a project, or collection of tasks. As such, task networks is typically associated with the *earliest-start dispatching* strategy, which amounts to simply starting a task immediately when its predecessors finish. Owing to the randomness of task durations, dispatching times and completion times of tasks are random variables. Clearly, then makespan, which equals the maximum of all completion

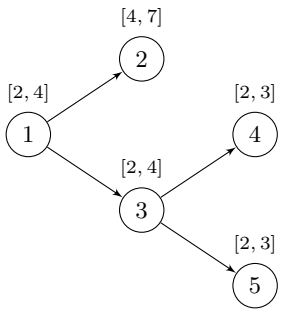


Figure 5.1: Task network consisting of precedence constraints over five tasks with uncertain durations. Each task is annotated with the interval within which its duration is expected to vary.

times, is also a random variable. Moreover, its probability distribution depends on exactly the following elements: *i*) the task duration distributions, *ii*) the network structure and *iii*) the dispatching strategy.

Mathematically, a stochastic task network can be described as a tuple $\langle G = (V, E), D\rangle$. Here, G is the directed graph of precedence constraints with $V = \{1, \ldots, n\}$ the indices of tasks 1 to n and $E \subseteq V \times V$ the set of directed arcs. Element $D = (D_1, \ldots, D_n)$ is a random vector, such that random variable D_i represents the duration of task i. We shall use $\mathbb{P}[D_i = d_i]$ to denote the probability of observing realization d_i of random variable D_i. The joint probability distribution $\mathbb{P}[D_1 = d_1, \ldots, D_n = d_n]$ of vector D, is given. For simplicity, it is usually assumed that variables D_i are independent. As such, the joint distribution of D typically evaluates to $\prod_i \mathbb{P}[D_i = d_i]$ and we are, in fact, given the individual distribution of each duration variable. If we use S_j to denote the (random) dispatching time of task j, then earliest start dispatching yields:

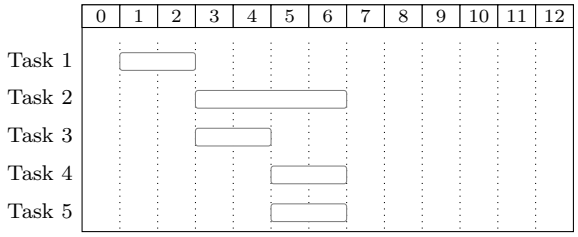
$$S_j := \max\{S_i + D_i : i \text{ is a predecessor of } j\} \tag{5.1}$$

That is, task j is dispatched immediately when all its predecessors finish (clearly, the finish time of task i is equal to $S_i + D_i$).

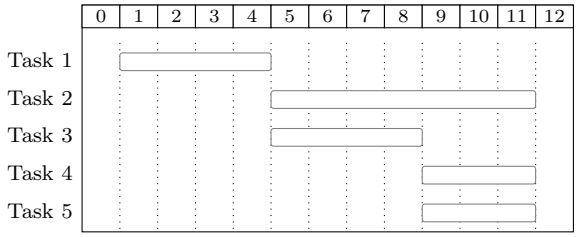
Example 5.1. *Consider the example task network displayed in Figure 5.1, illustrating a hypothetical situation with five maintenace operations to be performed in the NedTrain workshop, subject to precedence constraints. Tasks 2 and 3 cannot start before the completion of task 1 and similarly, tasks 4 and 5 cannot start before the completion of task 3. Assuming earliest start dispatching and that task 1 starts at time 0 (i.e. $S_1 = 0$), we derive the following equations*

$$S_2 = S_3 = S_1 + D_1$$
$$S_4 = S_5 = S_3 + D_3$$

The duration D_j of each task j is uncertain, but from data collected over past maintenance sessions, we are able to bound the duration of each task with a



(a) Resulting schedule with minimum task durations.



(b) Resulting schedule with maximum task durations.

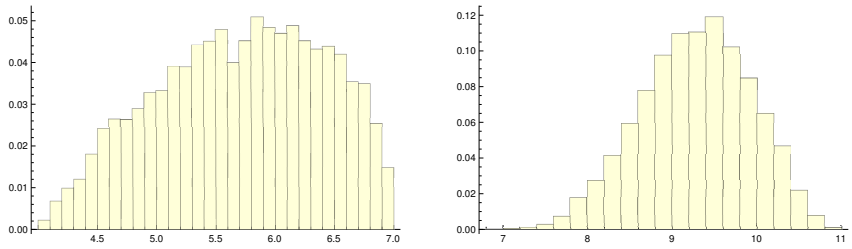
Figure 5.2: Earliest start dispatching in two marginal scenarios.

minimum and a maximum number of hours. This is indicated here by annotating each node with the interval within which its duration is expected to vary. That is, task 1 will take between 2 and 4 hours, task 2 between 4 and 7 hours, and so on.[1] *For now, we shall make no assumptions regarding the probability distribution of each task duration. But since durations are bounded by minimum and maximum values (which may not be the case in general), using earliest-start dispatching, the resulting makespan may vary between 6 and 11 hours, depending on the realized scenario. This is better illustrated in Figure 5.2 which presents two marginal cases of the realized schedule. Figure 5.2 examines two marginal scenarios: On the upper (lower) side, the outcome schedule is presented when earliest start dispatching is used and when each task duration takes its minimum (maximum) value.*

5.1 The makespan distribution problem

Always assuming earliest-start dispatching and since the appearance of PERT networks a few decades ago, research is still in progress concerning the problem of computing the makespan distribution. Makespan, as a variable, is usually denoted as C_{max} (representing the *max*imum *C*ompletion time of a task in the

[1] Note that annotating nodes with duration intervals as we do here is not standard practice in the literature.



(a) (b)

Figure 5.3: Sampling the distribution of task 2 (a) and the makespan distribution (b).

project) and is defined as:

$$C_{max} := \max\{S_i + D_i : i \in V\}$$

i.e. as the latest completion time of a task. Knowing the distribution $\mathbb{P}[C_{max} = t]$ is important for obvious reasons, since it amounts to knowing the expected project completion time, the chance of finishing the project by a certain due-date and so on. As shown by Hagstrom in [38], assuming duration distributions $\mathbb{P}[D_i = d]$ have finite and discrete support, computing the probability distribution of C_{max} or even its expectation $\mathbb{E}[C_{max}]$ is an intractable problem. As such, most approaches in the literature are heuristic. Quite often, approximating the makespan distribution involves Monte Carlo simulation based on a sample of task durations.

Example 5.2. *Let us now assume that duration of each task from our earlier example varies (within the given bounds) according to the Beta distribution with parameters $\alpha = 2, \beta = 1.5$. Figure 5.2a shows a histogram obtained by sampling the distribution of the duration of task 2, which ranges between 4 and 7 hours. Our choice of parameters α and β results in a forward skew which indicates a certain degree of pessimism, as we expect the duration of each task to be realized closer to its maximum value. Figure 5.2b shows a histogram obtained by sampling the distribution of makespan, which as we saw earlier will vary between 6 and 11 hours. Visual inspection of the histogram reveals that completing all the tasks in the network is unlikely to take less than 8 hours or more than 10 hours. Using such a makespan histogram, a due-date with arbitrarily high chances of being met can be readily determined.*

Obtaining such a histogram 'approximation' of the makespan distribution can be accomplished with the following straightforward process. First, a topological order of the task network is computed with an algorithm like [87]. Then, a sufficiently large number of realizations (or scenarios) of random vector $D = (D_1, \ldots, D_n)$ is generated, which can be trivial when D_i are independent random variables. Finally, the realized schedule for each generated scenario can be computed with Eq. 5.1 by visiting each task in topological order.

5.1.1 The makespan distribution problem in VLSI

In addition to project scheduling, computing the makespan distribution turns out to be important in the design of Very Large Scale Integration (VLSI) circuits. This problem emerges because a digital circuit can be modelled as a stochastic task network. Nodes correspond to gates (instead of tasks) and precedence constraints corresponding to wiring. The equivalent of a task duration is now the amount of time needed by a gate to prepare its output. Due to manufacturing imperfections, the latter varies randomly across the chip surface, and under certain conditions we may assume to know its probability distribution. Makespan corresponds to the length of the clock cycle and computing its distribution allows us to compute the *yield*–the probability that a certain percentage of manufactured chips satisfy certain timing criteria.

Computing the makespan distribution has received significant additional attention since the relatively recent discovery of its important role in the industry of circuit design [12]. The continuously growing area of so-called *Statistical Timing Analysis* (STA) includes a variety of efficient techniques for accurately estimating the distribution of networks with millions of nodes. STA techniques range from propagating analytical expressions of the distribution through the network [95] to sophisticated Monte Carlo sampling [94]. The use of STA techniques in large-scale project scheduling problems has been studied at least in [63].

5.2 Scheduling policies

Turning our attention back to scheduling, in some applications tasks occuppy certain amounts of one or more resources during their execution. The resource demands of tasks and the capacities of available resources are known together in scheduling as *resource constraints*. Note that one or more tasks which are not precedence-related may execute in parallel, in a schedule. A combination of precedence-unrelated tasks the total resource demands of which would exceed resource capacities in case they overlapped in time, is often known as a *forbidden set*. For a more comprehensive summary of resource constraints, the reader may refer to Section 2.4.

Example 5.3. *Consider the example task nework in Figure 5.1 and assume, for tasks 2,4 and 5, that each requires the use of a repair platform during its execution. If in addition we assume there are only two platforms available, then tasks 2, 4 and 5 form a forbidden set. Consider the two schedules formed by earliest start dispatching and shown in Figure 5.2. Neither schedule is feasible as both allow the three tasks to overlap within intervals* $[5, 6]$ *and* $[9, 11]$*, respectively, over-subscribing to the repair platform resource.*

Given a stochatic task network along with resource constraints, an emerging problem is to define a dispatching strategy that minimizes *expected makespan* while preventing a forbidden set from overlapping during execution, ensuring the

realized schedule satisfies both precedence and resource constraints. Note that earliest-start dispatching is generally not such a strategy, since it only accounts for precedence constraints. Möhring et al. have studied possible classes of such dispatching strategies, known as *scheduling policies* [61, 62].[2] A computational study examining the potential superiority of certain classes was conducted by Stork in [85], based on exact branch-and-bound procedures. Recently, a new exact method was proposed in [25] and other authors investigated (meta-)heuristics for minimizing expected makespan [6, 8].

We have already encountered a class of scheduling policies, back in Chapter 2. More specifically, in Section 2.4 we discussed how given a network of termporal constraints along with resource constraints, we can eliminate forbidden sets by adding precedence constraints to form a so-called partial order schedule (or POS for short). Partial order schedules are almost identical to so-called *earliest start policies*. The difference between an earliest start policy and a partial order schedule is subtle. Partial order schedules are discussed in the context of STP constraints between pairs of tasks, while earliest start policies are discussed in the context of simple precedence constraints. Just as a network of precedence constraints is a special case of a STP, it could be argued, then, that an earliest start policy is a special-case of a partial order schedule. We demonstrate the concept of an earliest start policy with the following example.

Example 5.4. *In our previous example, we saw that tasks 2, 4 and 5 form a forbidden set and as such, their execution should not overlap at any point in time in a feasible schedule. To prevent these tasks from overlapping, it suffices to add a precedence constraint between any two members of the forbidden set* $\{2, 4, 5\}$*. Recall that a precedence constraint is a directed arc* (i, j)*, meaning that* j *has to wait for the completion of* i*. Adding any of the constraints enumerated below would turn our example task network into an earliest start policy, the dispatching of which is guaranteed to produce a schedule satisfying both precedence and resource constraints, regardless of outcome task durations.*

1. $(2, 4)$*, yielding an expected makespan of 11.4 hours*
2. $(4, 2)$*, yielding an expected makespan of 14.5 hours*
3. $(2, 5)$*, yielding an expected makespan of 11.4 hours*
4. $(5, 2)$*, yielding an expected makespan of 14.5 hours*
5. $(4, 5)$*, yielding an expected makespan of 11.4 hours*
6. $(5, 4)$*, yielding an expected makespan of 11.4 hours*

Note that without adding any of the constraints listed above, we observe an expected makespan of 9.2 hours. Clearly, delaying the dispatching of task 2 (by adding either $(4, 2)$ *or* $(5, 2)$*) is not preferrable, as it results in the largest expected makespan increase.*

[2] From this point on, the terms scheduling policy and dispatching strategy shall be used interchangeably.

5.3 Stability and robustness

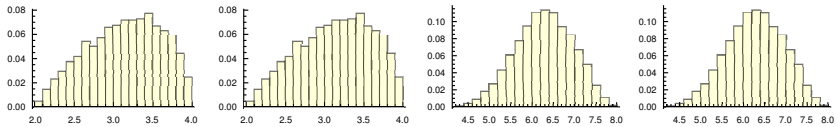
Dispatching a stochastic task network (with a scheduling policy) in order to minimize expected makespan is said to maximize *quality-robustness*, or simply robustness. All works cited so far focus on finding a policy of optimal quality-robustness. Optimizing quality-robustness alone, however, is not always a suitable objective. Next to quality robustness, in certain applications it is also important to have a *predictive schedule* from which the realized schedule is not expected to deviate too far. Without such a predictive schedule offering visibility into the future, human and other types of resources cannot plan ahead in order to be in the right place, at the right time. The quality of such a predictive schedule to remain close to the realized schedule is often known in scheduling literature as *solution-robustness* or *stability* [11, 92].

Example 5.5. *Figure 5.4a shows (a histogram of) the resulting dispatching times for tasks 2, 3, 4 and 5 of the network in Figure 5.1 when using earliest start dispatching and assuming durations follow the Beta distribution as described earlier. The dispatching time of task 1 is ommitted, as it has no predecessors to wait for and can thus always start at time 0. Clearly, the outcome dispatching times are rather unpredictable. Task 4, for instance, may start anywhere between time 4 and 8, with each of the possible start times having a chance of at most 10% of being observed. It is therefore impossible to plan ahead for the participation of human and other types of resources.*

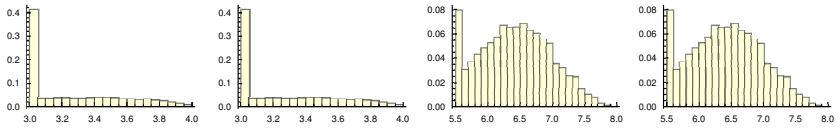
The unpredictability of future dispatching times in the workshop can, in fact, be controlled with an approach sometimes known as *railway scheduling* [91]. Given a task network and a scheduling strategy (if resource constraints are present), this approach amounts to defining a predictive dispatching time, per task, and modifying the dispatching process to force the outcome dispatching times to remain close to predictive ones. More in particular, each task j is associated with a respective predictive dispatching time t_j. Whatever dispatching strategy is in use must then observe this additional restriction: task j may not be dispatched before t_j, even if doing so would not violate any temporal or resource constraints.

As better illustrated in the following example, as those predictive (or minimum) start times are spread along the time-axis, unpredictability diminishes (or stability improves) and outcome dispatching times tend to stay close to predictive ones. In effect, then, vector $(t_1, t_2, \ldots, t_n)$ constitutes a reliable predictive schedule that can be used for planning ahead.

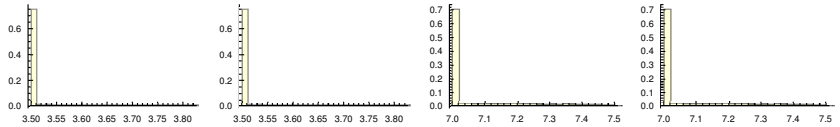
Example 5.6. *Figure 5.4b shows what happens when earliest start dispatching is used with the additional rule that tasks 2, 3, 4 and 5 cannot start before* $t_2 = 3.0, t_3 = 3.0, t_4 = 5.5$ *and* $t_5 = 5.5$*, respectively. Clearly, the start times of tasks 2 and 3 become more predictable, with a start time of* $t_2 = t_3 = 3.0$ *for tasks 2 and 3 having a relatively high chance of being observed, at 40%. Unfortunately, the start times of tasks 4 and 5 remain unpredictable. Figure 5.4c shows how stability improves when the predictive schedule is spread farther apart in time,*



(a) No predictive schedule.



(b) Using predictive schedule $t_1 = 0, t_2 = 3.0, t_3 = 3.0, t_4 = 5.5, t_5 = 5.5$.



(c) Using predictive schedule $t_1 = 0, t_2 = 3.5, t_3 = 3.5, t_4 = 7.0, t_5 = 7.0$.

Figure 5.4: Histogram representation of the realized dispatching times for tasks 2, 3, 4 and 5 (with task 1 always dispatched at time 0) for our example task network, when using earliest start dispatching with and without a predictive schedule. As predictive start times are spread farther apart, start time variability diminishes.

yielding, for each task, a chance of about 70% to start at its predictive start time. Moreover, note how even if a task does not start at its predictive time, it will most likely start somewhere near. Tasks 4 and 5, for instance, might start with a delay of at most 0.5 hours from their predictive time. Enhancing stability, however, has an impact on performance. When not using a predictive schedule at all (i.e. Figure 5.4a) the exepcted makespan is at 9.2 hours. Using the predictive schedule in Figure 5.4b, expected makespan increases only slightly at 9.4 hours. Ultimately, using the stable predictive schedule in Figure 5.4c yields an expected makespan of 9.9 hours.

As demonstrated in our previous example, scheduling tasks efficiently and scheduling tasks predictably (or optimizing quality-robustness and solution-robustness) are usually conflicting objectives [92]. A relatively small research area within the literature, known as *proactive-reactive project scheduling*, focuses on problems for which the solution is a pair formed by a scheduling policy and a predictive schedule. The objective is to find such a pair offering an optimal *trade-off* between quality-robustness and stability. The work presented in Chapters 6 and 7 of this dissertation focuses on the problem of trading robustness for stability.

5.4 Research Questions

In this Section we discuss how existing work, summarized in preceeding sections, can help us address Research Problem II, stated in Chapter 1. In doing so, we raise appropriate Research Questions that map to gaps in existing literature. The raised questions are eventually addressed in Chapters 6 and 7.

Referring back to Research Problem II, we would like to compute robust and stable schedules for work-teams, in order to deal with uncertainty in the duration of maintenance tasks. A robust schedule should guarantee good performance under a variety of possible outcome duration scenarios. Owing to the recurrent nature of maintenance operations, we are able to gather data over past maintenance sessions and build, for each task duration, a probability distribution according to which it varies over different executions. Given the description of each task duration as a random variable with a known distribution, a scheduling policy is functionally equivalent to a robust schedule, as it provides a set of rules for deriving appropriate dispatching times by reacting to observed task durations, such that feasibility and good performance are both guaranteed over a range of possible scenarios.

As explained in Section 5.3, however, robustness alone is not always sufficient. Next to ensuring good performance under a variety of potential scenarios and taking the human factor into account, we would also like to offer stability, or visibility of future dispatching times. That is, we would also like to force and/or predict that outcome dispatching times will fall within certain time-windows, or near certain predictive dispatching times. Such a stable or predictive schedule is necessary for planning-ahead the availability of human and other resources in the right place and at the right time in the workshop.

Modern literature on stochastic project scheduling addresses the problem of balancing robustness and stability with techniques for computing a scheduling policy along with a reliable predictive schedule. Existing approaches, however, suffer from certain limitations, as they tend to treat the scheduling policy and the predictive schedule separately. The prominent paper by Van de Vonder et al. [89], for instance, treats the optimization of a policy and predictive schedule separately, in a-two step fashion, while the exact approach of [52] follows a policy-agnostic approach, optimizing the predictive schedule without assuming anything about the structure of the policy. Considering, instead, the problem as a whole such that the structure of the policy is taken into account when choosing the predictive schedule, and vice-versa, gives us access to a solution-space of higher dimensionality. Since performance is determined in practice by the policy and the predictive schedule together, in pursuit of potentially better results we would like to address the following question:

Research Question II.1. *How to optimize a scheduling policy and a predictive schedule together as a pair?*

Moreover, as time progresses and outcome dispatching times and task durations are being observed, uncertainty gruadually diminishes in the workshop. As such, we are presented with an opportunity to adapt to new information

accordingly by rescheduling. In order to keep pace with execution and to avoid introducing further delays, the computational effort spent in rescheduling should be rather limited in practice. Therefore, during rescheduling we should avoid reconsidering the structure of the scheduling policy, in order to avoid the combinatorial explosion associated with reasoning about resource constraints. It might, however, be fruitful to react to new information by adapting the predictive schedule. As such, we are interested in addressing the following question:

Research Question II.2. *How to update the predictive schedule by reacting to outcome durations in low order polynomial time, keeping pace with execution?*

Research Question II.1 is answered in Chapter 6. In the first part of Chapter 6, we present a linear programming (LP) approach for finding a predictive schedule offering an optimal trade-off between makespan and stability, given a fixed earliest start policy. In the second part we develop a Mixed Integer LP extension of our approach which enables us to optimize an earliest start scheduling policy together with the predictive schedule. Research Question II.2 is answered in Chapter 7. There, we present a dynamic programming algorithm which can function as a much faster alternative to the LP approach developed in Chapter 6.

Chapter 6

Stable dispatching subject to resource constraints

6.1 Introduction

Project scheduling literature mostly concentrates on scheduling subject to temporal and resource constraints. The schedule sought for is an assignment of start-times to activities, facilitating the efficient use of limited resources in order to minimize a lateness measure such as the project makespan. Finding a schedule usually invovles solving the Resource-Constrained Project Scheduling Problem (RCPSP) (see [3, 39] for comprehensive surveys). This problem has been shown to be NP-Hard [13] and finds several industrial applications (e.g. [16, 9]). Associated literature includes numerous exact and (meta-)heuristic algorithms, able to find good schedules for large instances and diverse lateness measures (e.g. [82, 51, 49, 27]). Solving an RCPSP serves the purpose of preparing a feasible schedule, assuming a static deterministic project execution environment. In practice, however, this assumption is rarely valid. Activity durations used for preparing the schedule are mostly rough estimates, since most projects are subject to delays during execution and the final realized schedule is the result of subjecting the original schedule to modifications which make it consistent with the project constraints in the face of delays. Ad-hoc modifications lead to realized activity start-times that might differ from planned start-times, compromising project predictability and timeliness.

This paper addresses the issue of hedging against project uncertainty by preparing a schedule in combination with an execution strategy for coping with delays. In line with other works in *stochastic project scheduling* (see [42] for a comprehensive review) we assume activity durations are given as stochastic variables with known distributions and propose a new *proactive-reactive* scheduling method. First, we define the Proactive Stochastic (PS) RCPSP as an extension of RCPSP. The solution to PS-RCPSP is a *proactive schedule* and an *earliest-start (es-) policy* that together minimize the weighted

sum of the realized schedule's expected makespan and its expected deviation from the proactive schedule. PS-RCPSP is, in fact, a generalization of the Stochastic RCPSP (S-RCPSP), which asks to find a *stochastic scheduling policy* instead of a schedule and various *classes* of policies have been proposed in the literature [61, 62, 42]. In general, a policy defines a mapping between activity duration realizations to realized schedules. S-RCPSP asks to find a policy that minimizes the expected project makespan, with only few exact and heuristic approaches (mainly meta-heuristics) proposed over the last decade [85, 7, 8, 6].

Not preparing the project execution based on a schedule that can more or less be trusted (but rather, letting the realized schedule unfold during execution) has been recognized as a shortcoming of S-RCPSP [41]. This shortcoming motivates us and a number of other authors to propose a proactive-reactive approach, with [89, 28, 53] yielding the most promising computational results in existing literature. Different approaches pursue different optimization objectives; however, the common aim is to optimize some *tradeoff* between expected makespan and expected deviation from the proactive schedule. In line with other authors we represent activity duration distributions with a sample and propose a Linear Programming (LP)-based heuristic for PS-RCPSP, a Mixed-Integer LP (MILP) model enabling us to obtain exact solutions, and a MILP-based heuristic which asimilates *iterative flattening* [67]. The (MI)LP models for PS-RCPSP proposed in this paper are the result of adjusting the models proposed by Artigues et al. in [4] and [55]. We refer to exact solutions assuming stochastic duration distributions can be represented exactly by a sufficiently large sample.[1]

The contributions of this paper extend from section 6.3 and onwards. In order to base the paper on a consistent and self-contained framework of notation, section 6.2 summarizes existing concepts from deterministic, reactive, and proactive-reactive project scheduling. Section 6.3 introduces in a formal manner the problem studied here, that we name Proactive Stochastic RCPSP. Section 6.4 presents one main contribution of this paper: a LP-based heuristic for solving this problem. Section 6.5 presents another main contribution: a MILP model for this problem which enables us to obtain exact PS-RCPSP solutions. To our knowledge, no other exact solution methods have been proposed for poblems of similar type. Section 6.6 presents yet another contribution, a MILP-based heuristic for PS-RCPSP. Section 6.7 presents an experimental study in which we find that the LP-based heuristic performs favorably in comparison to the state-of-art, especially when the aim is to achieve near-zero instability. We also find the MILP-based heuristic to be more effective (albeit less efficient) when one is willing to accept medium levels of instability in order to minimize expected makespan. Section 6.8 concludes the paper.

[1]This notion of exactness is in line with Stork [85] who represent stochastic duration distributions with a sample when proposing exact search methods for the S-RCPSP.

6.2 Preliminaries

The purpose of this section is to introduce the research area of proactive-reactive (stochastic) project scheduling, which is where the contributions of this paper belong to. To establish autonomy and to facilitate discussion in further sections, we use convenient notation (which sometimes departs from standard notation) and begin with summarizing existing concepts from deterministic and (purely) reactive project scheduling. For a comprehensive survey of deterministic, reactive and proactive-reactive project scheduling, the reader may refer to [42].

6.2.1 Deterministic project scheduling

A project is usually represented as a directed acyclic graph $G(N, E)$, with nodes $N = \{1, \ldots, n\}$ corresponding to the set of n project activities. Each directed arc (i, j) in $E \subseteq \{(i, j) \in N^2\}$ defines a direct temporal constraint between activities i and j, meaning that j may not start unless activity i has finished. In effect, E defines a binary, irreflexive and transitive relation: if there is a path from activity i to activity j in $G(N, E)$ then j cannot start unless i has finished. Let us $T(E) \supseteq E$ denote the transitive closure of E, defined as

$$T(E) := \{(i, j) \in N^2 : \exists \text{ a path from } i \text{ to } j \text{ in } G(N, E)\}$$

We shall name a *temporally independent set* each subset of activities $X \subseteq N$ which are mutually independent with respect to temporal constraints. That is, if X is a temporally independent set, then $X^2 \cap T(E) = \emptyset$. Obviously, if only temporal constraints are taken into account, the activities of a temporally independent set may overlap in time in a schedule.

We assume as input a set $R := \{1, \ldots, m\}$ of m resources which must be shared among activities. Each resource $r \in R$ is associated with known capacity $b_r \in \mathbb{N}_0$. Furthermore, each activity $i \in N$ requires a known amount $q_{ir} \leq b_r$ of resource r while it executes. Vector $\boldsymbol{b} \in \mathbb{N}_0^m$ and matrix $\boldsymbol{q} \in \mathbb{N}_0^{n \times m}$ define the problem's resource constraints. Every independent set X for which $\sum_{i \in X} q_{ir} > b_r$ for some $r \in R$ is called a *forbidden* set. Even though it is allowed by the temporal constraints E, all activities in X may not overlap at some timepoint t because resource r will be used beyond its capacity, which is not possible.

Let $H \subseteq N^2$ denote a set of temporal constraints. Below we give the definition of a function Φ which returns the set of all forbidden sets w.r.t. temporal constraints H and the problem's resource constraints $(\boldsymbol{q}, \boldsymbol{b})$.

$$\Phi(H) := \{X \subseteq N : X^2 \cap T(H) = \emptyset, \sum_{i \in X} q_{ir} > b_r \text{ for some } r \in R\} \tag{6.1}$$

In addition to the parameters mentioned so far, we assume as input a vector $\boldsymbol{d} \in \mathbb{N}_0^n$ such that d_i defines the duration of activity i. Overall, a tuple $(N, R, E, \boldsymbol{d}, \boldsymbol{q}, \boldsymbol{b})$ specifies an instance of the RCPSP. A schedule $\boldsymbol{s} \in \mathbb{N}_0^n$ such

that s_i defines the start time of activity i, is a feasible solution when it satisfies the temporal and resource constraints, meaning that

$$s_j \geq s_i + d_i \qquad \forall (i,j) \in E \tag{6.2}$$

$$a(\boldsymbol{s}, t) \notin \Phi(E) \qquad \forall t \geq 0 \tag{6.3}$$

Here, $a(\boldsymbol{s}, t) := \{i \in N : t \in [s_i, s_i + d_i)\}$ is the set of activities executing at timepoint t according to $\boldsymbol{s}$ and Φ as defined earlier. Thus, (6.3) ensures there is no timepoint t at which the activities of a forbidden set overlap concurrently.

Project source-sink convention. RCPSP asks to find a feasible schedule of minimum makespan $C_{\max}(\boldsymbol{s}) := \max\{s_i + d_i : i \in N\}$. Most RCPSP-related works assume that the last activity, n, is a dummy activity with zero duration (i.e. $d_n = 0$) and that it must wait for the completion of every other activity (i.e. $(i, n) \in T(E)$ for every $i \in N - \{n\}$). This dummy activity is often known as the project "sink" and it holds that $C_{\max}(\boldsymbol{s}) = s_n$. Another convention of most RCPSP-related works is that the first activity, 1, often known as the project "source" must be waited on by every other activity (i.e. $(1, j) \in T(E)$ for every $j \in N - \{1\}$).

We shall hereafter assume activities 1 and n correspond to the project source and sink, respectively. The RCPSP can now be formally stated as:

$$\boldsymbol{s}^* := \arg\min\{s_n : (6.2), (6.3), \boldsymbol{s} \geq 0\} \tag{6.4}$$

6.2.2 Reactive project scheduling

In the research area of stochastic project scheduling, the activity durations vector $\boldsymbol{d}$ is replaced with a stochastic vector $\boldsymbol{D}$ such that D_i is the stochastic variable representing the uncertain duration of activity i, with a known probability distribution $\mathbb{P}[D_i = t]$. In line with recent works on S-RCPSP, we shall denote (elements of) stochastic vectors with a capital symbol.

S-RCPSP is a purely reactive extension of RCPSP. The solution sought for is no longer a schedule, but a reactive scheduling policy. A policy is a combinatorial object π which parameterizes the mapping from stochatic vector $\boldsymbol{D}$ to a corresponding realized schedule $\boldsymbol{S}(\pi, \boldsymbol{D})$. Note that $\boldsymbol{S}$ denotes a function which returns a vector of activity start times (of length n). Furthermore, if a realization $\boldsymbol{d}$ of $\boldsymbol{D}$ is passed as an argument, then $\boldsymbol{S}(\pi, \boldsymbol{d})$ denotes a deterministic vector. if $\boldsymbol{D}$ is passed as an argument, $\boldsymbol{S}(\pi, \boldsymbol{D})$ denotes a stochatic vector.

Different classes of policies have been proposed in the literature [61, 62, 85, 6]. One condition that all policy classes are expected to satisfy is that function $\boldsymbol{S}$ complies with the *non-anticipativity constraint*: the decision to start activity i at time $[\boldsymbol{S}(\pi, \boldsymbol{D})]_i$ cannot rely on information from the feature: the value of $[\boldsymbol{S}(\pi, \boldsymbol{D})]_i$ must be determined by time $t \leq [\boldsymbol{S}(\pi, \boldsymbol{D})]_i$. Other features such as the structure of π and the definition of function $\boldsymbol{S}$ depend on the class under study.

List-based policies. Two classes of policies which are prominent in the literature are *resource-based* (rb) policies and *activity-based* (ab) policies, also known collectively as *list-based policies*. A list-based policy is a priority vector $\boldsymbol{l} \in \mathbb{R}^n$ assigning priority l_i to activity i. Realized schedule $\boldsymbol{S}(\boldsymbol{l}, \boldsymbol{D})$ is computed by a variant of the well-known parallel schedule-generation-scheme (SGS) complying with the non-anticipativity constraint [8]; with the SGS definition being slightly different between rb-policies and ab-policies. As far as list-based policies are concerned, S-RCPSP asks to find a vector $\boldsymbol{l} \in \mathbb{R}^n$ that minimizes $\mathbb{E}[[\boldsymbol{S}(\boldsymbol{l}, \boldsymbol{D})]_n]$. Stork [85] proposes exact branch-and-bound algorithms for both rb and ab-policies. Ballestín [7] proposes an efficient genetic algorithm for ab-policies, providing the first computational experience on larger S-RCPSP instances. Ballestín and Leus [8] manage to obtain better results with a Greedy Randomized Adaptive Search Procedure (GRASP), again for the class of ab-policies. The best performance (w.r.t. expected makespan minimization) has so far been obtained with the more recent work of Ashtiani et al. [6] who propose a GRASP for a new class, namely *pre-processing* (pp) policies–a hybrid between rb-policies and es-policies.

Earliest-start policies. An es-policy is a set of temporal constraints $\mathcal{E} \subseteq N^2$ chosen such that

$$T(\mathcal{E}) \supseteq E, \tag{6.5}$$

$$\Phi(\mathcal{E}) = \emptyset, \tag{6.6}$$

$$G(N, \mathcal{E}) \text{ is acyclic} \tag{6.7}$$

Condition (6.5) ensures that a schedule $\boldsymbol{s}$ satisfying $s_j \geq s_i + d_i$ for each $(i, j) \in \mathcal{E}$ (here $\boldsymbol{d}$ can be any arbitrary choice of activity durations) is feasible with respect to the problem's precedence constraints E. Condition (6.6) ensures that $\boldsymbol{s}$ satisfying $\mathcal{E}$ implies that it also satisfies resource constraints prescribed by availabilities $\boldsymbol{b}$ and requirements $\boldsymbol{q}$ (as described earlier). Condition (6.7) ensures that the set of schedules satisfying $\mathcal{E}$ (for any arbitrary choice of activity durations $\boldsymbol{d}$) is non-empty.

When a project is executed according to an es-policy $\mathcal{E}$, the schedule that is realized, $\boldsymbol{S}(\mathcal{E}, \boldsymbol{D})$, is what is often known as the *earliest-start* schedule specified by $\mathcal{E}$. The earliest-start schedule of $\mathcal{E}$ can be defined as:

$$[\boldsymbol{S}(\mathcal{E}, \boldsymbol{D})]_j := \max\{[\boldsymbol{S}(\mathcal{E}, \boldsymbol{D})]_i + D_i : (i, j) \in \mathcal{E}\} \tag{6.8}$$

To put it simply, an activity j starts immediatelly when all its predecessors in $G(N, \mathcal{E})$ have finished. This time quantity (the latest finish time of j's predecessors) is often known as the length of the *critical path* from project source 1 to activity j. As far as es-policies are concerned, the S-RCPSP asks to find some $\mathcal{E}^*$ which minimizes $[\boldsymbol{S}(\mathcal{E}, \boldsymbol{D})]_n$ (the length of the critical path to the project sink) by expectation:

$$\mathcal{E}^* := \arg\min\{\mathbb{E}[[\boldsymbol{S}(\mathcal{E}, \boldsymbol{D})]_n] : (6.5 - 6.7), \mathcal{E} \in N^2\} \tag{6.9}$$

Constraints $(6.5 - 6.7)$ enforce the feasibility of any realized schedule with both the precedence and the resource constraints of the problem.

Stork [85] proposed an exact branch-and-bound search for problem (6.9). His algorithm considers each *minimal* forbidden set X (subset-minimal forbidden set) in some order and branches on each of $|\{(i,j) \in X^2\}|$ arcs which can be included in $\mathcal{E}$ in order to eliminate X from $\Phi(\mathcal{E})$. Without obtaining new computational results, in [54] Leus gives a formal treatment of es-policies as resource-flow networks (flow networks which can represent feasible RCPSP schedules) and proposes a refined version of the branch & bound algorithm of Stork. Exploiting the relation between resource-flows and es-policies, Artigues et al. [55] propose a robust optimization model for es-policies, for when a stochastic characterization of uncertainty is not available.

6.2.3 Proactive-reactive project scheduling

Reactive project scheduling allows one to pick activity start times dynamically during the project, under conditions of uncertainty. A main drawback of this approach (e.g. [41, 42, 18]) is that prior to (and during) project execution there is no schedule prescribing activity start times that can more or less be trusted. Such a "proactive" schedule can serve important organizational purposes; in fact, the deviation of the realized schedule from this proactive schedule is expected to induce organizational costs.

Attempts to overcome this drawback gave rise to the research area of proactive-reactive project scheduling, which is the research area that this paper belongs to. The main idea behind the proactive-reactive approach is to execute the project by using a proactive schedule together with a scheduling policy. Under uncertainty, some activities may not start at their proactive start times, because activities they have to wait for are not yet finished and/or resources they require are not yet released. In such cases, the scheduling policy determines which activities to start at their prescribed start times and which to postpone. It should be noted that most works assume *railway-mode scheduling*, meaning that an activity may not start earlier than its proactive start time, which strengthens the "stability" of the project execution. Clearly, the realized schedule is a function of the policy and the proactive schedule. Achieving low instability (deviation of the realized from the proactive schedule) requires "spreading" proactive activity start times far appart, in effect increasing the expected makespan. The general aim is to optimize some tradeoff between expected makespan and expected instability.

Van de Vonder et al. [92, 89, 88] propose and evaluate experimentally various proactive-reactive heuristics. The proposed heuristics assume as input an instance of S-RCPSP along with an initial schedule. The best performing heuristic is the so-called Starting Time Criticality (STC) heuristic. An es-policy is extracted from the structure of this initial schedule and used to iteratively transform the initial schedule into a proactive schedule by inserting time-buffers betwen activities. Deblaere et al. [28] propose an approach which integrates the proactive step (forming a proactive schedule) and the reactive step (forming the adjoining policy). Their approach is only possible to compare with ours

and others that assume railway-mode scheduling, by choosing sufficiently high penalties for earliness (w.r.t. the proactive schedule).[2] More recently, Vilches and Demeulemeester [53] propose a Chance-Constrained Programming model (CCP) for the RCPSP which asks to find a minimum makespan schedule subject to probabilistic temporal and resource constraints. They propose a Mixed Integer LP model, the solution to which is a proactive schedule that will most likely remain feasible under stochastic duration variability, without presumption on the policy that will be used during project execution.

6.3 Proactive Stochastic RCPSP

In deterministic and reactive project scheduling, the main problem under study (RCPSP and S-RCPSP, respectively) is stated clearly. A clearly stated problem model cannot be found in proactive-reactive project scheduling literature, perhaps because this research area is still in a burn-in phase.[3]Existing literature seems to pursue the general aim of optimizing some tradeoff between expected makespan and expected instability (deviation from the proactive schedule). This section presents the formal statement of a proactive-reactive scheduling problem for which (heuristic and exact) solution methods are proposed in subsequent sections.

The problem presented here, the Proactive Stochastic RCPSP (PS-RCPSP), asks to find a tuple $(\mathcal{E}, \boldsymbol{t})$ where $\mathcal{E}$ is an es-policy and $\boldsymbol{t}$ is a proactive schedule, minimizing the weighted sum of two performance criteria:

1. expected value of project makespan,
2. expected value of tardiness with respect to proactive release-times.

The first criterion measures lack of robustness and is relevant for obvious reasons. The second criterion measures instability and captures the expected deviation of the realized schedule from the proactive schedule. Note that $\boldsymbol{t}$ prescribes activity release-times (an activity i may not start earlier than t_i). Intuitively, instability represents the extent to which the proactive start-times can be trusted, when used for organizational purposes before and during project execution.

An instance of this problem is encoded by a tuple $(n, m, \boldsymbol{q}, \boldsymbol{b}, E, \boldsymbol{D}, \alpha)$. For clarity, we summarize the meaning of problem parameters. Positive integer n is the number of activities and m is the number of resources. Parameters $\boldsymbol{q} \in \mathbb{N}_0^{m \times n}$ and $\boldsymbol{b} \in \mathbb{N}_0^m$ define resource requirements and availabilities respectively. Set $E \subseteq \{1, \ldots, n\}^2$ defines pairwise precedence constraints. Stochastic vector $\boldsymbol{D}$ is of length n with each element D_i a stochatic variable (with given distribution $\mathbb{P}[D_i = t]$) which describes the uncertain duration of activity i. Finally, parameter $\alpha \in [0, 1]$ determines the desired tradeoff between robustness (i.e. minimization of expected makespan) and stability (i.e. minimization of expected instability). More emphasis can be put on either minimizining makespan (by choosing α closer to one) or minimizing instability (by choosing α closer to zero).

[2]We are grateful to one of our anonymous reviewers for this remark.

[3]Some works refer to [41] but this is a formal treatment of a proactive-reactive *machine* scheduling problem.

Formally, the problem can be stated as follows:

$$\min \quad f(\mathcal{E}, \boldsymbol{t}) := \alpha \cdot \mathbb{E}\left[[\boldsymbol{S}((\mathcal{E}, \boldsymbol{t}), \boldsymbol{D})]_n\right] + (1-\alpha) \cdot \mathbb{E}\left[\sum_{i=1}^{n}([\boldsymbol{S}((\mathcal{E}, \boldsymbol{t}), \boldsymbol{D})]_i - t_i)\right] \tag{6.10}$$

$$\text{s.t.} \quad \Phi(G(N, \mathcal{E})) = \emptyset \tag{6.11}$$

$$T(\mathcal{E}) \supseteq E \tag{6.12}$$

$$G(N, \mathcal{E}) \text{ is acyclic} \tag{6.13}$$

$$\mathcal{E} \in \{1, \ldots, n\}^2, \boldsymbol{t} \geq 0 \tag{6.14}$$

Conditions (6.12,6.13) ensure there is a non-empty set of schedules satisfying the problem's precedence constraints as prescribed in E. Condition (6.11) ensures that each such schedule also satisfies the problem's resource constraints prescribed by $\boldsymbol{q}$ and $\boldsymbol{b}$.

6.4 Heuristic LP-based approach

This section presents a polynomial-time heuristic for PS-RCPSP which consists of two steps:

1. using mean activity durations, the deterministic RCPSP $(n, m, \boldsymbol{q}, \boldsymbol{b}, E, \mathbb{E}[\boldsymbol{D}])$ is solved to obtain a good schedule $\boldsymbol{s}$ and a feasible es-policy $\mathcal{E}$ (i.e. satisfying (6.11),(6.12) and (6.13)) is derived from the structure of $\boldsymbol{s}$ in polynomial time (this procedure is described in [4]),
2. by solving a linear program presented below, we find a proactive schedule $\boldsymbol{t}$ that is optimally combined with $\mathcal{E}$ (which is kept fixed) so as to minimize an approximation of (6.10).

After $\mathcal{E}$ has been obtained in the first step, finding $\boldsymbol{t}$ which minimizes (an approximation of) the PS-RCPSP objective is achieved by solving the LP model presented below.

$$\min \quad \left[\alpha\left(\frac{1}{|\Gamma|}\sum_{\gamma\in\Gamma} s_n^\gamma\right) + (1-\alpha)\left(\frac{1}{|\Gamma|}\sum_{i=1}^{n}\sum_{\gamma\in\Gamma}(s_i^\gamma - t_i)\right)\right] \tag{6.15}$$

$$\text{s.t.} \quad s_j^\gamma \geq s_i^\gamma + d_i^\gamma \qquad \forall (i,j) \in \mathcal{E}, \gamma \in \Gamma \tag{6.16}$$

$$s_i^\gamma \geq t_i \qquad \forall i = 1, \ldots, n \tag{6.17}$$

$$\boldsymbol{t} \geq 0 \tag{6.18}$$

Here, (6.15) approximates the objective (6.10) based on $\Gamma \subseteq \mathbb{R}^n$: an adequately-sized sample of stochastic vector $\boldsymbol{D}$. The realization of activity durations under sample scenario $\gamma \in \Gamma$ is represented by vector $\boldsymbol{d}^\gamma = (d_1^\gamma, \ldots, d_n^\gamma)$. The corresponding realized schedule is *earliest-start*$((\mathcal{E}, \boldsymbol{t}), \boldsymbol{d}^\gamma) = (s_1^\gamma, \ldots, s_n^\gamma)$,

as computed by the model constraints. The solution is a proactive schedule $\boldsymbol{t} = (t_1, \ldots, t_n)$ that optimizes the tradeoff between expected makespan and instability for the given es-policy $\mathcal{E}$. This LP model has $n(|\Gamma|+1)$ linear variables (n variables t_i and $n|\Gamma|$ variables s_i^γ).

6.4.1 Related work

Van de Vonder et al. [89] propose several heuristics, of which the most competitive is the Starting Time Criticality (STC) heuristic and we shall therefore restrict our attention to it. Our LP-based heuristic bears similarities with STC. In fact, the first step of our heuristic is identical to that of STC: an es-policy $\mathcal{E}$ is extracted by the structure of an initial schedule $\boldsymbol{s}$. The second step of STC involves transforming the "unstable" schedule $\boldsymbol{s}$ into a "stable" schedule $\boldsymbol{t}$ with an iterative procedure, while keeping $\mathcal{E}$ fixed. In each iteration a one-unit time buffer is added at the start of that activity that "needs it the most" (as determined by a proposed "starting time criticality" measure) until adding more buffer time would not further reduce the instability of $\boldsymbol{t}$, which is measured by

$$\mathbb{E}\left[\sum_{i=1}^{n} w_i([\boldsymbol{S}((\mathcal{E}, \boldsymbol{t}), \boldsymbol{D})]_i - t_i)\right] \tag{6.19}$$

Here, w_i is a cost associated with the instability of activity i. Furthermore, t_n is kept fixed to a project deadline and therefore w_n represents the marginal cost of deviating from this project deadline. Note that by replacing α in (6.10) with individual weights w_i and choosing a fixed project deadline, it is straightforward to adapt our approach to the instability objective considered by van de Vonder et al. However, we felt that the choice of (6.10) as an objective is advantageous, as it underlines the tradeoff between expected makespan and instability more clearly and simplifies discussion by not involving a weight per individual activity and not requiring the choice of a project deadline.

Note that $\boldsymbol{t}$ is not guaranteed to be (precedence and resource) feasible with respect to mean activity durations (as required in the work of van de Vonder [89]). Enforcing $\boldsymbol{t}$ to hold this property in our approach can be accomplished by including the following constraint in the LP model:

$$t_j \geq t_i + \mathbb{E}[D_i] \qquad \forall (i,j) \in \mathcal{E}$$

However, this property only adds to the organizational value of $\boldsymbol{t}$ when mean values are reasonable estimates of activity durations.

Let us note that both our heuristic and STC have polynomial worst-case complexity (in the number of activities). However, in contrast with STC, our approach guarantees that $\boldsymbol{t}$ is chosen optimally when $\mathcal{E}$ is kept fixed and assuming the distribution of $\boldsymbol{D}$ is approximated with a sample. Therefore, if efficiency considerations enable us to choose a large-enough sample Γ (which is mostly the case due to the efficiency of existing LP solvers), our heuristic is expected to perform at least as well as STC. Finally, note that our heuristic is simpler to implement, requiring only the description of the presented LP model.

Leus et al. [56] assume as input a proactive schedule $\boldsymbol{t}$ (e.g. one that has been produced by STC). They propose a branch-and-bound search which returns the es-policy $\mathcal{E}$ which fits $\boldsymbol{t}$ optimally in minimizing an expression of expected instability similar to (6.19).

6.5 Exact MILP-based approach

PS-RCPSP (section 6.3) asks to find an es-policy and a proactive schedule $(\mathcal{E}, \boldsymbol{t})$ that together minimize the weighted sum of expected makespan and instability. Section 6.4 presented a heuristic approach according to which $\mathcal{E}$ is kept fixed while $\boldsymbol{t}$ is optimally paired with the policy by solving a LP. This section presents a Mixed Integer LP (MILP) model with which PS-RCPSP can be solved to optimality. However, it should be pointed-out that a solution is trully exact only if we assume stochastic duration distributions can be accurately described by the chosen sample Γ; for general probability distributions we obtain a lower bound. In fact, the problem of computing the exact expected makespan of a given es-policy (and assuming duration distributions with discrete support) has been shown by Hagstrom in [37] to be intractable. However, our notion of exactness is in line with the computational study of Stork [85] where "optimal" scheduling policies are computed by using a fixed sample of duration distributions.

This model includes binary variables representing the structure of $\mathcal{E}$ and linear variables representing $\boldsymbol{t}$. To our knowledge, no other exact approaches have been proposed in the literature for problems of similar type (i.e. asking for a scheduling policy and proactive schedule that together optimize some tradeoff between expected makespan and instability). To arrive at this PS-RCPSP model, we merge the LP model presented in the previous section with a MILP model that has been proposed by Artigues et al. [4] and which allows to solve the deterministic RCPSP by treating it as a flow-network problem. The model presented here is not entirely new, since a similar technique (repeating precedence constraints for each scenario of the chosen sample) has been proposed in [55] for minimizing the maximum regret of an es-policy.

6.5.1 The RCPSP model of Artigues et al.

Artigues et al. [4] represent a solution to the RCPSP as a so-called *resource-flow* $\boldsymbol{f} \in \mathbb{R}_0^{n \times n \times m}$; an assignment to variables f_{ijr} associated with each pair of activities $(i, j) \in N^2$ and each resource $r \in R$. A resource-flow describes the "passing" of resource units inbetween activities. More precisely, $\boldsymbol{f}$ is an indirect representation of every schedule $\boldsymbol{s}$ in which f_{ijr} units of resource r are released by activity i at its completion $s_i + d_i$ and then "picked up" by activity j at its start s_j, without another activity using these units between $s_i + d_i$ and s_j.

A resource-flow is feasible when it satisfies

$$\sum_{j \in N-\{i\}} f_{jir} = q_{ir} \quad \forall i \in N - \{1\} \tag{6.20}$$

$$\sum_{j \in N-\{i\}} f_{ijr} = q_{ir} \quad \forall i \in N - \{n\} \tag{6.21}$$

Eq. (6.20) asks that each activity i (except for the sink) receives as many resource units as it requires the moment it starts. Eq. (6.21) asks that each activity i (except for the source) releases as many resource units as it has used the moment it finishes.

The flow network $G(N, \phi(\boldsymbol{f}))$ associated with $\boldsymbol{f}$ is defined as $\phi(\boldsymbol{f}) := \{(i,j) \in N^2 : f_{ijr} > 0 \text{ for some } r \in R\}$; i.e. there is an arc from each activity to every other activity it passes at least one resource unit to. As shown by Leus [54, 55], feasible resource-flows and es-policies are interrelated: $\mathcal{E} = E \cup \phi(\boldsymbol{f})$ is a feasible es-policy if $\boldsymbol{f}$ is a feasible resource-flow (and $G(N, \mathcal{E})$ is acyclic). Therefore, every schedule which satisfies $G(N, \mathcal{E})$ is feasible. The following MILP model proposed by Artigues et al. enables one to find a feasible resource-flow $\boldsymbol{f}$ which minimizes the cost (described by function g) of a schedule $\boldsymbol{s}$ which satisfies the temporal constraints of $G(N, E \cup \phi(\boldsymbol{f}))$.

$$\begin{aligned}
\min \quad & s_n && && (6.22)\\
\text{s.t.} \quad & s_j \geq s_i + d_i - M(1 - z_{ij}) && \forall (i,j) \in N && (6.23)\\
& z_{ij} = 1 && \forall (i,j) \in E && (6.24)\\
& f_{ijr} \leq M z_{ij} && \forall (i,j) \in N^2, r \in R && (6.25)\\
& (6.20), (6.21) && && (6.26)\\
& f_{ijr} \geq 0, z_{ij} \in \{0,1\} && \forall (i,j) \in N^2, r \in R && (6.27)
\end{aligned}$$

Here M is a large constant. Due to (6.26) $\boldsymbol{f}$ is a feasible resource-flow. Due to (6.25), if $f_{ijr} > 0$ for one or more $r \in R$ then $z_{ij} = 1$, meaning that variables z_{ij} describe the flow-network $\phi(\boldsymbol{f})$ of the resource-flow (i.e. $\phi(\boldsymbol{f}) = \{(i,j) \in N^2 : z_{ij} = 1\}$). Due to (6.23) and (6.24), $\boldsymbol{s}$ describes a schedule which satisfies the temporal constraints in $G(N, E \cup \phi(\boldsymbol{f}))$. Since $\boldsymbol{f}$ is a feasible resource-flow, $\boldsymbol{s}$ is a feasible schedule.

6.5.2 Extension for S-RCPSP

This section presents a trivial extension to the RCPSP model of Artigues et al. which enables us to find optimal es-policies for the S-RCPSP. Considering a sample $\Gamma \subset \mathbb{R}^n$ of stochastic activity durations vector $\boldsymbol{D}$ allows us to present the following MILP model.

$$\min \quad \frac{1}{|\Gamma|}\sum_{\gamma\in\Gamma} s_n^\gamma \tag{6.28}$$

$$\text{s.t.} \quad s_j^\gamma \geq s_i^\gamma + d_i^\gamma - M(1-z_{ij}) \qquad \forall (i,j)\in N^2, \gamma\in\Gamma \tag{6.29}$$

$$(6.24-6.27) \tag{6.30}$$

Our extension is rather straightfoward. Each variable s_i is included here as variable s_i^γ for each sample scenario $\gamma \in \Gamma$. Precedence constraints (6.23) from before are now replicated for each scenario $\gamma \in \Gamma$ in condition (6.29). Objective (6.22) is now replaced with objective (6.28), which estimates the makespan expectation $\mathbb{E}[S(E \cup \phi(\boldsymbol{f}))]$ based on sample Γ.

6.5.3 Extension for PS-RCPSP

Here we extend the previous model by including a variable t_i for each $i \in N$, which determines the activity's proactive starting time. The resulting PS-RCPSP MILP model is presented below.

$$\min \quad \left[\alpha\left(\frac{1}{|\Gamma|}\sum_{\gamma\in\Gamma} s_n^\gamma\right) + (1-\alpha)\left(\frac{1}{|\Gamma|}\sum_{i=1}^{n}\sum_{\gamma\in\Gamma}(s_i^\gamma - t_i)\right)\right] \tag{6.31}$$

$$\text{s.t.} \quad s_j^\gamma \geq s_i^\gamma + d_i^\gamma - M(1-z_{ij}) \qquad \forall (i,j)\in N^2, \gamma\in\Gamma \tag{6.32}$$

$$(6.24-6.27) \tag{6.33}$$

$$s_i^\gamma \geq t_i \qquad i\in N, \gamma\in\Gamma \tag{6.34}$$

$$\boldsymbol{t} \geq 0 \tag{6.35}$$

The objective now becomes identical to that of the LP-based heuristic, measuring the weighted sum of expected makespan and expected instability. Condition (6.34) ensures that an activity may not start earlier than its proactive start time.

To summarize, by solving this model we obtain a PS-RCPSP solution $(\mathcal{E}, \boldsymbol{t})$ where $\mathcal{E} = \{(i,j) \in N^2 : z_{ij} = 1\}$ is a feasible es-policy and $\boldsymbol{t}$ defines a proactive schedule.

Proposition 6.1. *Define $\mathcal{E} := \{(i,j) : z_{ij} = 1\}$ the arcs of the flow-network $G(N, E \cup \phi(\boldsymbol{f}))$ associated with resource-flow $\boldsymbol{f}$. Let $\boldsymbol{f}, \boldsymbol{z}, \boldsymbol{s}, \boldsymbol{t}$ be an optimal solution. For each scenario $\gamma \in \Gamma$, $\boldsymbol{s}^\gamma$ defines a schedule where each activity i starts as soon as allowed by its proactive release-time t_i and es-policy $\mathcal{E}$.*

Proof. For each scenario $\gamma \in \Gamma$, let $\bar{x}^\gamma$ denote the earliest allowed start time for activity i, allowed by the combination of $\mathcal{E}$ and proactive schedule $\boldsymbol{t}$. We want to prove that in an optimal solution, $\boldsymbol{s}^\gamma = \bar{x}^\gamma$ for all $\gamma \in \Gamma$.

Assume that $s_i^\gamma = \bar{x}_i^\gamma + \delta$ for some $\gamma \in \Gamma, i \in N$, with $\delta > 0$. Since (6.31) increases monotonically with s_i^γ, the objective can be improved by setting

$s_i^\gamma = \bar{x}_i^\gamma$, without violating any constraints. Therefore, in every optimal solution we have $s_i^\gamma = \bar{x}_i^\gamma$ for all $\gamma \in \Gamma$ and $i \in N$, meaning that each $\boldsymbol{s}^\gamma$ defines the earliest start times schedule allowed by the combination of by es-policy $\mathcal{E}$ and proactive schedule $\boldsymbol{t}$ under scenario γ. □ □

By proposition 6.1 it follows that an optimal solution to the MILP model presented above is, in fact, an optimal solution for the PS-RCPSP.

6.6 Heuristic MILP-based approach

Even for small instances (e.g. with 30 activities and 4 resources), solving the proposed model might take an inordinate amount of time. We propose an algorithm (Algorithm 2), the main idea of which was inspired by the *iterative flattening* heuristic of Oddi et al. [67]. The heuristic of Oddi et al. was developed for the deterministic RCPSP with minimum/maximum time-lag precedence constraints [40]. Every feasible schedule for the problem they study is compactly represented as a network of temporal constraints (known as a Simple Temporal Network [32]). It is the similarity with an es-policy (which is a network of zero-lag temporal constraints) that has inspired the development of the heuristic presented here.

The proposed heuristic involves solving a sequence of sufficiently small subproblems with non-increasing optimal objective values. Each iteration involves solving a partially solved instance to optimality. Thus, worst-case complexity is exponential in the number of activities. In practice, however, "good" solutions can be obtained with relative efficiency.

Algorithm 2 Iterative flattening for PS-RCPSP

1: $\boldsymbol{s} \leftarrow$ schedule for RCPSP $(N, R, E, \mathbb{E}[\boldsymbol{D}], \boldsymbol{q}, \boldsymbol{b})$
2: $\mathcal{E}^* \leftarrow E \cup \phi(\boldsymbol{f^s})$ with $\boldsymbol{f^s}$ extracted from $\boldsymbol{s}$
3: $\boldsymbol{t}^* \leftarrow (0, \ldots, 0)$
4: **while** termination criteria not met **do**
5: $\quad \mathcal{H} \leftarrow$ random subset of $\mathcal{E}^* - T(E)$ chosen by criticality probability
6: $\quad (\mathcal{E}, \boldsymbol{t}) \leftarrow$ optimal solution for PS-RCPSP $(N, R, \mathcal{E}^* - \mathcal{H}, \boldsymbol{D}, \boldsymbol{q}, \boldsymbol{b}, \alpha)$
7: $\quad$ **if** $(\mathcal{E}, \boldsymbol{t})$ has a lower objective than $(\mathcal{E}^*, \boldsymbol{t}^*)$ **then**
8: $\quad\quad (\mathcal{E}^*, \boldsymbol{t}^*) \leftarrow (\mathcal{E}, \boldsymbol{t})$
9: return $(\mathcal{E}^*, \boldsymbol{t}^*)$

Algorithm 2 assumes as input an instance of the PS-RCPSP. According to aforementioned notation, the instance is represented as $(N, R, E, \boldsymbol{D}, \boldsymbol{q}, \boldsymbol{b}, \alpha)$. An initial solution is obtained by solving a deterministic RCPSP (lines 1-3) which can be done efficiently with one of the various existing heuristics. This solution will serve as a starting point for the first iteration, which is described as follows. A partial solution is formed by removing a random subset of highly critical arcs from the current solution (line 6). The resulting subproblem is solved to optimality (by use of the proposed model) and a complete solution is obtained

(line 7). If this new solution is better, it becomes the starting point of the next iteration. The algorithm may terminate when, e.g., a chosen number of iterations have been performed, or the objective has failed to improve a certain number of times.

Further efficiency improvements. Note that the optimal solution $(\mathcal{E}, \boldsymbol{t})$ of the subproblem solved in each iteration cannot be worse than the best solution seen so far, $(\mathcal{E}^*, \boldsymbol{t}^*)$. To improve performance one may use $(\mathcal{E}^*, \boldsymbol{t}^*)$ as an initial solution when solving the model (line 6). Efficiency can be further improved by reducing the number of binary variables z_{ij} in the model. This can be accomplished by observing that z_{ij} for each $(i,j) \in T(\mathcal{E}^* - \mathcal{H})$ can be fixed to one and z_{ij} for each $(j,i) \in T(\mathcal{E}^* - \mathcal{H})$ can be fixed to zero.

6.7 Experiments

In [53], Vilches and Demeulemeester compare their method (CCP) with that by Van de Vonder et al. (STC) [89]. To the best of our knowledge, STC and CCP constitute the state-of-art as far as trading expected makespan for instability in stochastic project scheduling is concerned. In this section we extend this comparison by using the same experimental set-up and including results for our LP-based heuristic (Section 6.4) and the MILP-based heuristic (Section 6.6). As the results show, our approaches compare favorably with STC and CCP.

The set-up used in [53] was based on the `J30` deterministic RCPSP bench-set of the well-known PSPLIB [50], which comprises 480 deterministic RCPSP instances, each with $n = 30$ activities. Based on this bench-set, three stochastic RCPSP bench-sets were derived, namely `J30-low`, `J30-med`, and `J30-high`, corresponding to conditions of low, medium, and high project uncertainty, with activity durations following a discretized beta distribution. Specifically, each activity i with duration d_i in the deterministic RCPCP instance now has stochastic duration $D_i = [X_i d_i 0.5(l+h)]$ with $\mathbb{E}[D_i] \simeq d_i$ where

- X_i follows a beta distribution with shape parameters $\alpha = 2, \beta = 5$;
- $l = 0.75$ and $h = 1.625$ in the low variability bench-set;
- $l = 0.5$ and $h = 2.25$ in the medium variability bench-set;
- $l = 0.25$ and $h = 2.875$ in the high variability bench-set;
- operator $[\cdot]$ represents rounding to the closest integer.

Each of the evaluated methods (including STC and CCP) has a certain "tradeoff parameter" which determines whether more emphasis is put on minimizing expected makespan or minimizing expected instability. For our LP-based and MILP-based heuristic this tradeoff parameter is the weight α in expression (6.10). CCP and STC have corresponding parameters with a similar effect. By

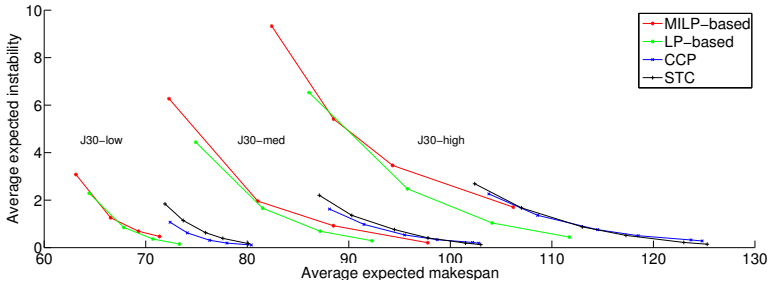


Figure 6.1: Trading expected makespan for stability.

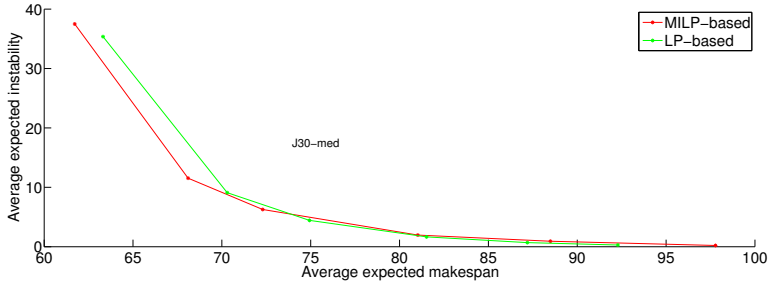


Figure 6.2: Trading expected makespan for stability for higher α.

varying the choice of the corresponding tradeoff parameter(s), a set of tradeoff data-points is obtained for each method, on each of the three bench-sets.

In Figure 6.1, the data-points for each method are displayed as a tradeoff curve, on each of the three bench-sets, resulting in three "clusters" of tradeoff curves. A tradeoff curve captures the average performance of the method on that bench-set. Specifically, each data-point is two-dimensional and records the average expected makespan and average expected instability for a certain choice of the tradeoff parameter(s), where the average is taken over all 480 instances of the bench-set. Data-points for CCP and STC are borrowed from the work of Vilches and Demeulemeester [53]. Data-points for our heuristics are obtained by setting $\alpha = 0.05$, 0.1, 0.2, and 0.4. Higher alpha values correspond to data-points closer to the upper left corner, with higher instability and lower makespan.

Figure 6.2 focuses on the medium variability case for higher α values, including additional data-points for $\alpha = 0.6$ and $\alpha = 0.9$. This allows us to compare the MILP-based and LP-based heuristics when more emphasis is put on minimizing expected makespan.

The expected makespan and expected instability of the solution provided by each of the methods on a particular instance is computed with a sample $\Gamma^{large} \subset \mathbb{R}^n$ comprising $|\Gamma^{large}| = 10^3$ realization of durations vector $\boldsymbol{D}$. Note that the data-points of Vilches and Demeulemeester were computed with a different sample of size 10^3. We assume that 10^3 is a sufficiently large sample

size to facilitate comparability with our results.

Configuration of heuristics. The sample Γ^{milp} used by our MILP-based heuristic during optimization (see line 6 of Algorithm 2) is of size $|\Gamma^{milp}| = 30$. Our MILP-based heuristic is configured to perform three (3) iterations and the number of highly critical arcs removed in each iteration (see line 5 of Algorithm 2) is $|\mathcal{H}| = 20$. Note that the criticality of the arcs is computed based on sample Γ^{large} (this is done efficiently, in time quadratic in n and linear in $|\Gamma^{large}|$). The solver we use is CPLEX version 12.6. Furthermore, we set a time-limit for the solver to 50 seconds (since each iteration starts from a feasible solution, the solver will always return with a solution within the time-limit). The polynomial-time complexity of our LP-based heuristic (no binary variables in the model) allows us to use a large sample during optimization. In fact, we use sample Γ^{large}. To find a deterministic schedule (as required in step 1 of this LP-based heuristic) we used a priority rule procedure recently proposed in [27]. Vilches and Demeulemeester use a sample of size 10^2 during optimization, for both STC and CCP. Furthermore, they limit the time spent in solving their CCP model on an instance to a maximum of 10 seconds.

Observations. Figure 6.1 suggests that regardless of the mode of variability (low, medium, or high), when the purpose is to achieve near-zero instability, the LP-based heuristic yields the best results. This can be attributed to the efficiency of solving a LP model, which enables us to use a large sample (of size 10^3 in this case) during optimization.

Figure 6.2 suggests that even though the sample used during optimization is much smaller for the MILP-based heuristic (of size 30), it is more effective than the LP-based heuristic for higher α values (i.e. when minimizing instability is more important than minimizing makespan). Both the LP-based and the MILP-based heuristics start from the same es-policy (see step 1 in section 6.4 and line 2 of Algorithm 2, respectively). However, the MILP-based heuristic restructures the policy and this enables it to perform better at minimizing expected makespan.

Restructuring the policy comes at the cost of efficiency. With three iterations allowed per instance, this yields an average of 50 seconds per instance for the MILP-based heuristic. The LP-based heuristic is considerably more efficient, with an average of 1.5 seconds per instance. Vilches and Demeulemeester report that STC spends on average 0.2 seconds per instance, while their CCP approach spends on average 10 seconds per instance.

6.8 Conclusions and future work

This paper proposes the PS-RCPSP problem model which, assuming stochastic activity durations, asks to find a so-called earliest-start (es) policy and a proactive schedule that together minimize the weighted sum of expected project makespan and expected instability. Extending an existing MILP model for the RCPSP,

a MILP model for PS-RCPSP is presented, which allows us to find optimal (es-policy, proactive schedule) pairs. Solving this problem to optimality might require an impractical amount of time, even for instances with few activities (e.g. 30). Therefore, we propose a LP-based and a MILP-based heuristic for the PS-RCPSP. Our LP-based heuristic optimizes the proactive schedule by keeping the es-policy part of the solution fixed. Our MILP-based heuristic optimizes the structure of the policy together with the proactive schedule. The LP-based heuristic, which is rather efficient, seems to be more effective compared to the state-of-art (i.e. achieves smaller expected makespan for a certain level of expected instability) especially when the aim is to achieve close to zero instability. The MILP-based heuristic is rather effective when the aim is to achieve low expected makespan at the cost of moderate or high instability. In contrast to existing state-of-art approaches such as CCP [53] and STC [89], our heuristics rely on the idea of optimizing the proactive schedule together with the scheduling policy. This difference might in part explain observed performance differences.

Future work involves a thorough experimental analysis of the proposed heuristics, not for the purpose of comparing them to the state-of-art, but for a deeper understanding of their behavior and its dependence on problem characteristics. Furthermore, most existing stochastic project scheduling works are evaluated on instances where the deterministic RCPSP counterpart instance (formed by mean activity durations) serves as a good approximation of the stochastic instance. This is exploited by our heuristics and other heuristics such as STC. However, in certain practical domains (e.g. maintenance scheduling), the duration of some activities is known a-priori with accuracy, while the duration of other activities follows a distribution with very high variance. In maintenance scheduling, for example, "inspection" activities have known durations but "repair" activities might be (un)necessary with certain probabilities. We would like to investigate performance on such instances which cannot be approximated well by their determinitic counterpart.

Chapter 7

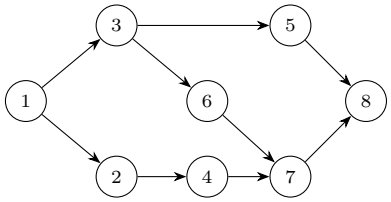
Stable dispatching with dynamic programming

7.1 Introduction

A *stochastic task network* is a directed acyclic graph $G(V, E)$ with each node in $V = \{1, \ldots, n\}$ representing a task with a random duration and each arc $(i, j) \in E$ representing a precedence-constraint between tasks i and j, specifying that task j cannot start unless task i has finished. Such networks appear in several domains like project scheduling [54], parallel computing [83], or even digital circuit design [12], where there is a need to model a partial order of events with uncertain durations. Postulating that a model of uncertainty is known, task durations are described by a random vector $D = (D_1, \ldots, D_n)$ with a known probability distribution. In project scheduling, for example, the duration D_i of task i may turn out to be shorter or longer than a nominal value according to a certain distribution.

A given task network is typically mapped to a *realized schedule* (i.e. an assignment of start-times to tasks) via *earliest-start dispatching*; i.e. observing outcome durations and starting a task immediately when precedence-constraints allow (i.e. not later than the maximum finish-time of its network predecessors). Random durations make the realized start-time of a task (and the overall realized schedule makespan) also random. Since *PERT networks* [59], a large body of literature focused on the problem of determining the makespan distribution [2], eventually shown to be a hard problem [38]. A variety of efficient heuristics have been developed so far (see [12]), among which Monte Carlo sampling remains, perhaps, the most practical.

Consider, for example, the stochastic task network in Fig. 7.1, detailing the plan of a house construction project, assuming task durations are random variables that follow the uniform distribution within respective intervals. With earliest-start dispatching, the overall duration of the project (i.e. the realized schedule makespan) will range between 12 and 20 days with an expected value



(a) Example construction plan.

	Tasks	Durations (days)
1.	Erect walls	2-4
2.	Finish walls	3-5
3.	Finish roof	2-6
4.	Install plumbing	3-5
5.	Finish exterior	6-8
6.	Install electricity	3-5
7.	Paint interior	2-4
8.	Finishing touches	1-1

(b) Estimated task durations.

Figure 7.1: A motivating example.

of a little over 16 days.

This paper addresses a problem which, to our knowledge, has not been addressed in existing literature. To motivate our problem, let us return to the earlier example and suppose task 7 ("Paint interior") is assigned to a painting crew charging \$100 per day. Assume we are willing to hire them for at least 4 days (the maximum number of days they will need) and for at most 6 days; i.e. we have a budget of \$600 for painting. With earliest-start dispatching, 7 may start within 8 to 15 days from the project start (the start-date of task 1). A challenge that arises in this situation is deciding when to hire the painting crew, because to allow for an expected makespan of a little over 16 days (as mentioned earlier), we must book the painting crew from the 8-th day and until the 19-th day, at the excessive cost of \$1100. The solution we examine here, is to use a different dispatching strategy, associating task 7 with a *planned release-time*, t_7, before which it may not start even if installing plumbing and electricity are finished earlier than t_7. If we choose that 7 may not start earlier than, e.g., $t_7 = 13$ days from the project start, we only need to book the painting crew on the 13-th day until the 19-th day, for an acceptable cost of \$600. However, the price to pay for this stability is an expected makespan increase to a little over 17 days.

Now suppose that after assessing our budget carefully it turns out that each task may deviate at most, say w days, from its respective planned release-time. The emerging question addressed in this paper is:

> Which planned release-times reach the desired level of stability[1] while minimizing the incurred performance penalty?

This problem does not involve resource-constraints. However, task networks are often used in the area of resource-constrained scheduling under uncertainty (see [10, 43]) to represent solutions (e.g. the *earliest-start policy* [46], the *partial-order schedule* [76, 36, 17]). Thus, our work is expected to be useful in dealing

[1] As in Bidot et al. [11], stability here refers to the extent that a predictive schedule (planned release-times in our case) is expected to remain close to the realized schedule.

with associated problems, such as distributing slack in a resource-feasible schedule to make it insensitive to durational variability [26].

Organization

A formal problem statement and its LP formulation are presented in Section 2. As the resulting LP can be quite costly to solve, Section 3 presents our main result, an efficient dynamic programming algorithm. Section 4 concludes the paper and outlines issues to be addressed in future work.

7.2 Problem definition

We are given a task network $G(V, E)$ and a stochastic vector $D = (D_1, \ldots, D_n)$ describing task durations. Let $\mathcal{Q}$ index the space of all possible realization scenarios for D such that d_{ip} denotes the realized duration of task i in scenario $p \in \mathcal{Q}$. We assume to know the probability distribution of D; i.e. the probability $\mathbb{P}[D = (d_{1p}, \ldots, d_{np})]$ for all $p \in \mathcal{Q}$. To limit the unpredictability of the realized schedule, we want to associate tasks with respective *planned release-times* $t = (t_1, \ldots, t_n)$ such that the realized schedule is formed by starting a task as early as permitted by precedence-constraints, but not earlier than its release-time. That is, the start-time s_{jp} of task j in scenario p will be determined as:

$$s_{jp} = \max[\max_{(i,j)\in E}(s_{ip} + d_{ip}), t_j] \tag{7.1}$$

Given a sample $\mathcal{P} \subseteq \mathcal{Q}$ of size m of the stochastic durations vector, this paper is devoted to the following problem:

$$\min_{t\geq 0} \quad F := \sum_{j\in V, p\in\mathcal{P}} s_{jp} \tag{P}$$

$$\text{subject to} \quad s_{jp} = \max[\max_{(i,j)\in E}(s_i + d_i), t_j] \qquad \forall j \in V, p \in \mathcal{P} \tag{7.2}$$

$$s_{jp} - t_j \leq w \qquad \forall j \in V, p \in \mathcal{P} \tag{7.3}$$

This problem tries to optimize a trade-off between stability and performance: release-times are sparsely spread in time in order to form a stable schedule, i.e. such that in every considered scenario a realized start-time will stay within w time-units from the corresponding release-time.

Since the whole space of possible duration realizations, $\mathcal{Q}$, may be too large, or even infinite, we only consider a manageable sample $\mathcal{P} \subseteq \mathcal{Q}$ during optimization.[2] At the same time, we want to ensure a minimal performance penalty $F - F^*$ where F^* denotes the throughput of earliest-start dispatching with no release-times. [3]

[2] Knowing the distribution of D, we assume to be able to draw $\mathcal{P}$.

[3] The reader can easily recognize the similarity of the proposed LP with a so-called Sample Average Approximation (SAA) of a stochastic optimization problem [48].

Instead of minimizing a standard performance criterion like expected makespan, we choose to maximize *expected throughput*, $\frac{1}{m}\frac{n}{\sum_{j,p} s_{jp}}$, which equals the average rate at which tasks finish over all scenarios. It can be shown that a schedule of maximum throughput is one of minimum makespan and/or tardiness (in case tasks are associated with deadlines). We maximize throughput indirectly by minimizing its inverse, with the constant $\frac{m}{n}$ omitted for simplicity.

LP formulation

The resulting problem is not easy to handle due to the equality constraint, but using a standard trick it can be rewritten as the following linear program (LP):

$$\begin{aligned}
\min_{s,t\geq 0} \quad & F := \sum_{j\in V, p\in\mathcal{P}} s_{jp} && && (P)\\
\text{subject to} \quad & s_{jp} \geq s_{ip} + d_{ip} && (i,j)\in E, p\in\mathcal{P} && (7.4)\\
& s_{jp} \geq t_j && j\in V, p\in\mathcal{P} && (7.5)\\
& s_{jp} - t_j \leq w && j\in V, p\in\mathcal{P} && (7.6)
\end{aligned}$$

Note that the solution-space of the resulting LP encompasses that of the original formulation. However, it is easy to show that both problems have the same set of optimal solutions, because a solution (s,t) for the LP cannot be optimal unless it satisfies (7.2).

Currently, the best (interior-point) LP solvers have a complexity of $O(N^3M)$ where N is the number of variables and M the input complexity [79]. Thus, letting $\delta \leq n$ denote the max in-degree in $G(V,E)$, the cost of solving (P) as an LP with nm variables and $O(n\delta m)$ constraints can be bounded by $O(n^4m^4\delta) \subseteq O(n^5m^4)$, which can be daunting even for small instances. Fortunately, as shown in the following section, we manage to obtain the substantially tighter bound of $O(n^2m)$ for solving (P), by exploiting its simple structure to devise a dynamic programming algorithm.

7.3 Fast computation of planned release-times

We first show that a fixed relationship between variables s_{jp} and t_j can be assumed while looking for an optimal (s,t). Based on this, a problem (P') is defined which can be solved instead of (P).

A tighter formulation

Begin by rewriting (7.6) as $t_j \geq \max_p s_{jp} - w\,, \forall j \in V$. Now, let Λ denote the set of all feasible (s,t) for problem (P) and let $\Lambda^* \subseteq \Lambda$ be that part of the solution-space that only contains (s,t) for which $t_j = \max_p s_{jp} - w$ for all j.

Lemma 7.1. *For every feasible $(s,t) \in \Lambda \setminus \Lambda^*$ there exists $(s',t') \in \Lambda^*$ with equal objective value.*

Proof. Consider feasible (s,t) with $t_j = \max_p s_{j^*p} - w + c$ with $c > 0$ for some j^*. Construct t' by letting $t'_j = t_j$ for all $j \neq j^*$ and $t'_{j^*} = t_{j^*} - c = \max_p s_{j^*p} - w$. Trivially, if (s,t) is feasible, so is (s,t'), with the same objective value. Keeping s fixed, we may repeat this construction to enforce that $t_j = \max_p s_{jp} - w$ for all j and have $(s,t') \in \Lambda^*$. □

The previous result allows us to consider the following problem, obtained by substituting $\max_{p' \in \mathcal{P}} s_{jp'} - w$ for t_j in (P):

$$\min_{s \geq 0} \quad \sum_p s_{np} \qquad (P')$$

$$\text{subject to} \quad s_{jp} \geq s_{ip} + d_{ip} \qquad (i,j) \in E, p \in \mathcal{P} \qquad (7.7)$$

$$s_{jp} \geq \max_{p' \in \mathcal{P}} s_{jp'} - w \qquad j \in V, p \in \mathcal{P} \qquad (7.8)$$

$$s_{jp} - (\max_{p' \in \mathcal{P}} s_{jp'} - w) \leq w \qquad j \in V, p \in \mathcal{P} \qquad (7.9)$$

Clearly, $(s,t) \in \Lambda^*$ iff s is feasible for (P').[4] In other words, the solution-space of P' comprises only those s that can be paired with t by letting $t_j = \max_p s_{jp} - w$ to form a feasible (s,t) for (P). By Lemma 7.1, if s is optimal for (P'), then (s,t) is optimal for (P). Also, if (P) has a solution (i.e. if $G(V,E)$ is acyclic),then (P') also has a solution.

The resulting STP

Formulation (P') is useful because it can be cast as a certain type of Temporal Constraint Satisfaction Problem (TCSP) [31]. We start by noting that (7.9) is always true and can be omitted. Moreover, (7.8) can be rewritten as (7.11), to obtain the following reformulation:

$$\min_{s \geq 0} \quad \sum_p s_{np} \qquad (P')$$

$$\text{subject to} \quad s_{ip} - s_{jp} \leq -d_{ip} \qquad (i,j) \in E, p \in \mathcal{P} \qquad (7.10)$$

$$s_{jp} - s_{jp'} \leq w \qquad (p,p') \in \mathcal{P}^2, j \in V \qquad (7.11)$$

Constraints (7.10) and (7.11) effectively represent the solution-space of a Simple Temporal Problem (STP) [31] with temporal variables $\{s_{jp} : j \in V, p \in \mathcal{P}\}$. The structure of the resulting STP (specifically, of its *distance graph* [31]) is demonstrated in Fig. 7.2.

The *earliest start time* (est) solution of any given STP (assuming it is consistent) assigns to each variable the smallest value it may take over the set of feasible solutions. Therefore, the est solution of the resulting STP optimally solves (P'), leading us to the following observation.

Observation 7.1. *By Lemma 7.1, an optimal solution (s,t) for (P) can be formed by finding the earliest start time solution s of the resulting STP and pairing it with t formed by letting $t_j = \max_{p \in \mathcal{P}} s_{jp} - w$ for all j.*

[4] Since $(s,t) \in \Lambda^*$ implies $\max_{p'} s_{jp'} - w = t_j$ for all j.

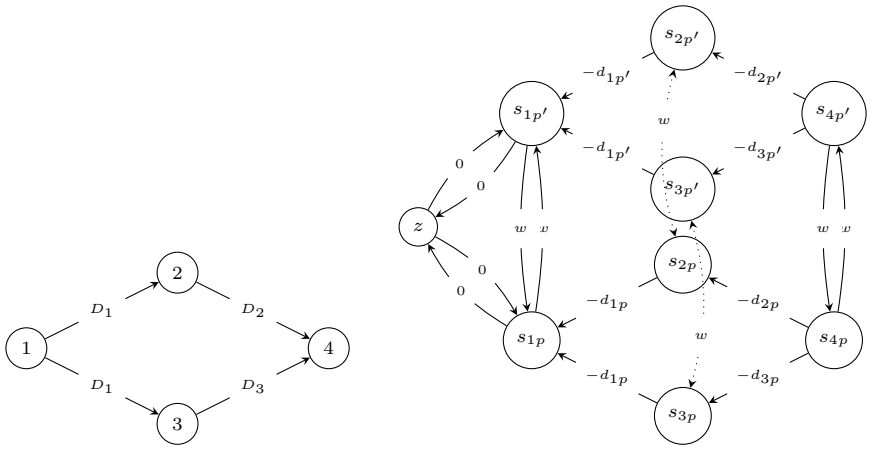


Figure 7.2: Example task network (a) and resulting STP (b) for a sample $P = \{p, p''\}$.

Algorithm 3 Optimal release-times via dynamic programming

1: $l(s_{1p}) \leftarrow 0$ for all $p \in \mathcal{P}$
2: **for** each tier j in a topological sort of $G(V, E)$ **do**
3: $\quad k_{jp} \leftarrow \min\{l(s_{ip}) - d_{ip} : (i, j) \in E\}$ for all $p \in \mathcal{P}$
4: $\quad p^* \leftarrow \arg\min\{k_{jp} : p \in \mathcal{P}\}$
5: $\quad l(s_{jp}) \leftarrow \max\{k_{jp}, k_{jp^*} + w\}$ for all $p \in \mathcal{P}$
6: $s_{jp} \leftarrow l(s_{jp})$ for all $j \in V, p \in \mathcal{P}$
7: $t_j \leftarrow \max_{p \in P} s_{jp} - w$ for each $j \in V$

The est value of s_{jp} is the length of the shortest-path (in the distance graph) from (the node corresponding to) s_{jp} to the special-purpose variable z which is fixed to zero. Those values can be found with a single-source shortest-path algorithm (e.g. Bellman-Ford [68]) in time $O(NM)$ where N is the number of nodes and M the number of arcs. In our case, $N = nm$ and $M = O(nm\delta)$, yielding $O(n^3m^2)$; already a better bound than that of solving (P) as an LP. However, in the following we obtain an even better bound with a dynamic programming algorithm.

Computing the est solution by dynamic programming

Let us associate each task $j \in V$ with a corresponding *tier* including all nodes $\{s_{jp} : p \in \mathcal{P}\}$ of the STP distance graph. A few remarks on the structure of the STP are in order. First, due to (7.11) the resulting STP is not acyclic, but each cycle only includes nodes that belong to the same tier. Second, due to (7.10) there is a path from each node in tier j to each node in tier i if and only if there

is a path from task i to j in $G(V, E)$.

Let $l(s_{jp})$ denote the shortest-path length from s_{jp} to z (i.e. the value of variable s_{jp} in an optimal solution of (P')). From the structure of the resulting STP, we have:

$$l(s_{jp}) = \min\left\{\min_{(i,j)\in E}(l(s_{ip}) - d_{ip}), \min_{p'\neq p} l(s_{jp'}) + w\right\} \tag{7.12}$$

The existence of cycles complicates solving subproblem $l(s_{jp})$ as it depends on (and is a dependency of) other subproblems $l(s_{jp'})$ in the same tier. However, we can "break" dependencies between subproblems in the same tier as shown below.

Define $k_{jp} := \min_{(i,j)\in E}(l(s_{ip}) - d_{ip})$ and $p^* := \arg\min_{p\in\mathcal{P}} k_{jp}$.

Lemma 7.2. $l(s_{jp}) = \min\{k_{jp}, k_{jp^*} + w\}$

Proof. Begin by noting that the shortest-path from s_{jp} to z visits at most one node $s_{jp'}$ from the same tier. As such, for every s_{jp} we have that: either $l(s_{jp}) = k_{jp}$, or $l(s_{jp}) = k_{jp'} + w < k_{jp}$ for some $p' \neq p$.

Now, note that $l(s_{jp^*}) = k_{jp^*}$, since if not (i.e. if $l(s_{jp^*}) \neq k_{jp^*}$), then $l(s_{jp^*}) = k_{jp'} + w < k_{jp^*}$ with $p' \neq p^*$, which contradicts the definition of p^*.

Last, we show that if $l(s_{jp}) \neq k_{jp}$ then $l(s_{jp}) = k_{jp^*} + w$. Suppose not. Since $l(s_{jp}) \neq k_{jp}$ then according to (7.12), $l(s_{jp}) = l(s_{jp'}) + w$ but with $p' \neq p^*$. Expanding $l(s_{jp'})$ according to (7.12),

$$\min\{k_{jp'}, \min_{p''\neq p'} l(s_{jp''}) + w\} + w < l(s_{jp^*}) + w = k_{jp^*} + w$$

and since $k_{jp'} \geq k_{jp^*}$,

$$\begin{aligned} &\min_{p''\neq p'} l(s_{jp''}) + w < k_{jp^*} \\ &\Leftrightarrow l(s_{jp^*}) + w < k_{jp^*} \end{aligned}$$

which contradicts that $l(s_{jp^*}) = k_{jp^*}$. □

The resulting recursion suggests a dynamic programming approach, summarized in Algorithm 3. It involves solving the subproblems of one tier at a time, visiting tiers according to a topological sort of $G(V, E)$ (recall that tiers correspond to tasks $j \in V$). Finding a topological sort takes $O(n\delta)$ [87], recalling that δ denotes the max in-degree of a task in the network. The overall complexity of Algorithm 3 is therefore $O(nm\delta) \subseteq O(n^2m)$.

7.4 Conclusions and Discussion

Given a stochastic task network with n tasks we consider dispatching the tasks as early as possible, subject to (planned) release-times. Assuming a sample with m realizations of the stochastic durations vector is drawn, we defined an LP

for finding optimal release-times; i.e. that minimize the performance penalty of reaching a desired level of stability. The resulting LP is costly to solve, so pursuing a more efficient solution method we managed to show that optimal release-times can be expressed as a function of the earliest start time solution of an associated Simple Temporal Problem. Exploiting the structure of this STP, we were able to define a dynamic programming algorithm for finding optimal release-times with considerable efficiency, in time $O(n^2m)$.

Future work

Since we optimize according to a manageable sample $\mathcal{P}$, there is a (potentially non-zero) probability $\mathbb{P}_v$ that the realized start-time of a task deviates further than w time-units from its planned release-time. The question of how $\mathbb{P}_v$ (or $\mathbb{E}[\mathbb{P}_v]$ as in [22]) depends on m (the size of $\mathcal{P}$) should be addressed in future work. Furthermore, in an earlier paper [65], an LP similar to (P) was used it in a two-step heuristic for a flavor of the *stochastic resource constrained project scheduling problem* (stochastic RCPSP) [90, 52]. Given a resource allocation determined in a first step, in a second step a LP was used to find planned release-times that minimize the total expected deviation of the realized schedule from those release-times. This heuristic was found to outperform the state-of-the-art in the area of *proactive project scheduling*. In future work, we shall investigate using the algorithm presented here in order to stabilize the given resource-allocation, expecting gains in both efficiency and effectiveness. Finally, a potentially related problem, namely PERTCONVG, is studied by Chrétienne and Sourd in [23], which involves finding start-times for a task network so as to minimize the sum of convex cost functions. In fact, their algorithm bears structural similiarities to ours, since subproblems are solved in a topological order. It would be worth investigating if their analysis can be extended in order to enable casting the problem studied here as an instance of that problem.

Part III

Conclusion

Chapter 8

Conclusion

This dissertation was motivated by the challenge of scheduling and dispatching operations subject to uncertain and dynamic conditions prevailing in the NedTrain maintenance workshop. After introducing NedTrain as a maintenance company, in Chapter 1 we stated Research Problems I and II, to which Parts I and II of the thesis are devoted, respectively. After introducing the reader to related literature in Chapters 2 and 5, these problems were then broken-down into Research Questions, establishing the scope of our work. In this chapter we examine to which extent definitive answers were found for our research questions. From a high-level standpoint, we also examine whether Research Problems I and II were addressed adequately. Finally, we conclude the chapter with a list of topics related to our work that could be investigated in future work.

8.1 Answers to Research Questions

Research Question I.1, I.2 and I.3 were formulated at the end of Chapter 2 by breaking-down Research Problem I through the prism of STP-related literature. Research Questions II.1 and II.2 were formulated at the end of Chapter 5, this time by breaking-down Research Problem II through the prism of stochastic scheduling literature. The first three research questions were addressed in Chapters 3 and 4 and the last two research questions were addressed in Chapters 6 and 7.

8.1.1 Research Question I.1

The first approach examined in this thesis for dealing with uncertainty in the NedTrain workshop involves allowing people to (re)schedule themselves as they see fit, with as much flexibility as possible. The challenge in this case is to guarantee the resulting schedule will satisfy temporal and resource constraints, without relying on synchronous communication between independent parties (or work-teams) in the workshop. Flexible interval schedules originally proposed by

Wilson et al. [96] could enable such a scheduling process. The high worst-case complexity of $O(n^5)$ associated with their LP-based approach, however, is expected to hit a performance barrier when n is in the order of several hundreds or even thousands of tasks, as is the case with problem instances representing a NedTrain workshop. Reasoning about resource constraints significantly exacerbates this problem since a large number of candidate solutions must be evaluated. In pursuit of a way around performance limitations, we formulated the following question:

> *How to efficiently compute concurrent flexibility in a given STP, in low-order polynomial time?*

This question was answered in Chapter 3 by showing how to compute the maximum achievable amount of flexibility for a given STN as a min-cost matching problem on a bipartite graph with $2n$ nodes. The use of, e.g. the Hungarian method, yields a worst-case complexity of $O(n^3)$, which is much better than $O(n^5)$. Our approach was further improved in the first part of Chapter 4 where, beyond just measuring the amount of achievable flexibility, we show how an actual interval schedule offering maximum flexibility can be constructed from a min-cost matching, again in cubic time. In effect, we exploited the special structure of the LP considered by Wilson et al. in order to come-up with a custom and more efficient solution method. An obvious implication of our results is showing that concurrent flexibility is not only more accurate, but also at least as easy to compute as (the more prominent in existing literature) naive flexibility. As such, existing approaches (e.g. POS-generation procedures; see Section **??**) can be adapted to a more accurate flexibility metric without having to trade accuracy for efficiency.

8.1.2 Research Question I.2

As task execution unfolds, more dispatching times become fixed, or known. Once a dispatching time becomes fixed, the associated time interval prescribed in the interval schedule is no longer needed. As task execution unfolds, we would like to keep redistributing unneeded flexibility over yet-undispached tasks. Care should be taken, however, to avoid causing disruptions. If redistributing flexibility results in a drastically different interval schedule, or decoupling, then the latter becomes a "moving target" that cannot be relied on to make commitments. As such, redistributing flexibility should be performed incrementally, i.e. with respect to an already existing interval schedule. Moreover, if redistributing flexibility is a slow operation then disruptive delays might be introduced. In pursuit of such a dynamic decoupling operation, we raised the following question:

> *How to incrementally recompute a concurrent flexibility interval schedule during dispatching?*

We answer this question in Chapter 4 by developing an extension of the LP by Wilson et al. for computing a flexibile interval schedule from scratch. This

new LP can be used to construct an interval schedule of maximum flexibility with respect to an existing one, by widening (if possible) the time intervals of yet-undispatched tasks and narrowing-down those of already dispatched tasks into the chosen timepoints. The updated interval schedule is guaranteed to offer at least as much flexibility as the given interval schedule. Moreover, this rescheduling operation is non-disruptive since whatever dispatching options were available to work-teams by the initial interval schedule will continue to be valid until all tasks have been dispatched. In fact, potentially more attractive options might become available later as the interval schedule is kept up-to-date with the progressing task execution.

8.1.3 Research Question I.3

Keeping the interval schedule up-to-date with new information about already dispatched tasks is, in effect, a rescheduling operation. For situations with a large number of tasks, a rescheduling operation should be computable as efficiently as possible. Otherwise, there is a risk of impeding the task execution process with disruptive delays. As such, we were led to the formulation of the following question:

> *How to redistribute concurrent flexibility as fast as possible (using heuristic methods if necessary)?*

This question was answered in two steps in Chapter 4. We managed to improve upon the computationally expensive LP-based approach for redistributing unused flexibility mentioned earlier, by showing that updating a given interval schedule can be cast as the basic problem of finding a flexibile interval schedule from scratch. Since the latter can be cast as a min-matching problem, the updating problem can also be solved efficiently in $O(n^3)$ with a min-cost matching algorithm. In a second step, we managed to lower the cost of updating the interval schedule even further, by developing a heuristic that performs each time window update in near-linear time, with almost no loss of optimality (as indicated by experiments).

In conclusion, our contributions with respect to concurrent flexibility are manifold. We managed to improve the efficiency of computing flexible interval schedules. We also managed to add a 'dynamic' dimension to the existing 'static' framework, allowing the fast and incremental adaptation of flexible interval schedules as task execution progresses.

8.1.4 Research Question II.1

The second approach considered in this thesis amounts to constructing a schedule with sufficient slack to absorb the effects of variable task durations. In contrast with the existing approach at NedTrain, which is mostly manual and relies on the domain expertise of operational planners, our approach involves a sophisticated modeling of uncertainty in the workshop. Based on observations and data

collected over past maintenance sessions, the duration of each task is modelled as a (random variable with a known) probability distribution. Based on such a stochastic model of uncertainty, our approach involves constructing a predictive schedule with the right amount of slack at the right place. Despite the insertion of slack, the amount of time allocated for a task might turn-out to be insufficient during execution. For this reason, a so-called earliest-start scheduling policy (a stochastic task network) is coupled with the predictive schedule. The policy prescribes a set of rules for reacting to possible buffer-overruns, i.e. how to update the predictive schedule in order to retain feasibility with temporal and resource constraints. Our objective is to find such a (policy, schedule) pair that trades favorably between robustness (i.e. good expected performance w.r.t. deadlines) on one hand, and stability (i.e. tendency of outcome dispatching times to stay near the predictive schedule) on the other hand. Existing approaches, however, mostly avoid treating the policy and the schedule together as a whole during optimization. In pursuit of potentially better results by enabling access to a solution-space of higher dimensionality, we stated the following research question:

> *How to optimize a scheduling policy and a predictive schedule together as a pair?*

We managed to answer this question in Chapter 6 by developing stochastic extensions of existing mathematical programming models for RCPSP, a deterministic scheduling problem. First we provide a LP formulation of the problem of fitting a predictive schedule to a given earliest start policy, using a sample of the random durations. We then extend this LP into a MILP by introducing binary variables that enable reasoning about the structure of the earliest start policy at the same time as the structure of the predictive schedule. As expected, this integrated approach outperforms existing ones in trading-off robustness for stability. The computational cost, however, becomes prohibitive for instances of practical size. The cost can be lowered by using a smaller sample (i.e. fewer duration realization scenarios), but this compromises solution quality. Our experiments reveal that under a limited computation time budget, it is more effective to use a two-step approach (i.e. find a policy first and fit a schedule to it) than an integrated approach but with a small sample of task durations. Our two-step approach relying on the LP formulation mentioned earlier seems to outperform existing two-step approaches significantly. In conclusion, we were able to develop an integrated approach, therefore answering the research question adequately. Moreover, we were able to deduce that such an integrated approach is beneficial only with a sufficiently large time-budget available for computation. As a by-product of developing this integrated approach we managed to outperform existing two-step approaches, for the more practical case of dealing with a limited time-budget.

8.1.5 Research Question II.2

As task execution progresses the durations and the dispatching times of certain tasks become known. In effect, then, whatever stochastic modeling of uncertainty was used to generate an initial (schedule, policy) pair becomes obsolete during task execution. As such, we would like to have a rescheduling operation for adapting the allocation of slack and the structure of the policy to new information about outcome durations and dispatching times. Modifying the structure of the earliest-start policy would involve reasoning about resource constraints. To avoid the associated combinatorial explosion (since a rescheduling operation should be fast), it would make more sense to quickly adapt the allocation of slack while keeping the policy fixed. The LP-based approach for constructing a predictive schedule based on a fixed scheduling policy, mentioned earlier, is not efficient enough for use as a rescheduling operation. As such, we decided to seek an answer for the following question:

> *How to efficiently update the predictive schedule by reacting to outcome durations, keeping pace with execution?*

In Chapter 7 we answer this question by developing a very efficient alternative to the LP-based approach, based on dynamic programming. This fast approach for finding a stable predictive schedule enables us to handle instances with a large number of tasks. It also enables us to use a large sample of task durations, i.e. a high-resolution representation of uncertainty. This latter advantage is crucial for establishing stable dispatching times in situations where task durations exhibit aggressive variability, as is the case with conditional repair tasks in the NedTrain workshop.

8.2 Solutions for Research Problems

Based on our answers to the research questions, now from a higher-level standpoint we evaluate whether we addressed the research problems.

8.2.1 Research Problem I

For the motivation behind this problem, the reader may refer to Section 1.4. For convenience, the problem statement is repeated below.

> *How to compute flexible schedules for independent work-teams that can be easily adapted to changes in the environment?*

To deal with this problem we turned to the area of simple temporal reasoning, which mostly relates to Artificial Intelligence (AI). Based on the concept of a flexible interval schedule (or temporal decoupling), we managed to form the following solution for this problem.

Task execution in the workshop is orchestrated by an interval schedule, which can be computed based on our answers to Research Questions I.1, I.2, and

I.3. Such an interval schedule prescribes, for each team in the workshop, a time-window for each of their tasks. A team may pick any suitable time to dispatch a task within the given time-window, without having to worry about the decisions of other teams (i.e. the decisions of different teams in the workshop are decoupled). The interval schedule guarantees a joint schedule will be formed by individual team decisions, satisfying all workshop constraints: due-dates and precedence relations between tasks, but also resource constraints over shared equipment, platforms and so on. Moreover, the time-windows available for each team will get even wider as task execution goes on and more dispatching options will be available. In other words, each team in the workshop can freely (re)schedule themselves within the margins of given time-windows, knowing that: i) their plans need not be shared with other teams, nor will they get invalidated by disruptions down the line, ii) shared resources like equipment and platforms will be available without having to negotiate with other teams, iii) even more (potentially more attractive) options for rescheduling might become available later.

Despite its advantages, this approach also suffers from the following limitation. In contrast with the approach proposed in Part II, is that it relies on the assumption of deterministic task durations. As such, the generation and continuous adaptation of an interval schedule must rely on worst-case estimates about the time needed to complete a task. As a result, teams might be given more leeway than absolutely necessary in order to meet train delivery due-dates.

8.2.2 Research Problem II

For the motivation behind this problem, the reader may refer to Section 1.4. For convenience, the problem statement is repeated below.

> *How to compute robust and stable schedules for work-teams in order to deal with uncertainty in the duration of maintenance tasks.*

To deal with this problem we turned to the research area of stochastic project scheduling, which mostly relates to Operational Research (OR). Assuming that uncertain task durations behave like random variables with known distributions and based on the concept of robust and stable scheduling (or proactive/reactive scheduling), we managed to form the following solution for this problem.

Observing the fluctuations of task durations over past maintenance sessions, we build a statistical model of uncertainty that suits the NedTrain workshop. We assume to be able to derive information such as the probability that a particular repair task will have to be performed on a particular train, given its past workshop visits and its forecasted condition. Based on our answers to Research Questions II.1 and II.2, this model of uncertainty is used in order to compute a "predictive" schedule with the right amount of slack and at the right places. Teams in the workshop can plan-ahead based on the given schedule, knowing that scheduled times are unlikely to change in the future. In case allocated slack turns-out to be insufficient, a scheduling policy (computed alongside the schedule) tells

operational planners which tasks to shift forward in order to repair the schedule. The schedule and the policy guarantee that adaptations of the schedule during execution do not compromise our chances of delivering trains on-time.

The main downside of this approach is that, in contrast with the one examined in Part I, teams are not allowed to determine the dispatching times of their tasks on-the-fly, during execution. Moreover, this approach does not guarantee full team independence in the workshop. Independence is established to some degree, however, since scheduled dispatching times are expected to remain mostly unchanged during task execution and thus no negotiation over dispatching decisions is necessary.

8.3 Recommendations for future work

Combining uncertain durations with STNs

A possible extension of current work would involve combining stochastic task durations with flexible interval schedules, therefore bringing together the best of the two approaches examined in Part I and Part II. Such a technique could be based on the framework of so-called Probabilistic Simple Temporal Networks, or PSTNs [81]. A PSTN is a STN in which the maximum (or minimum) temporal distance between a pair of time variables can be a random variable. Future work could focus on the potential to define a stochastic generalization of flexibility, which would allow, in addition, the representation of stochastic task durations. Interestingly, some work in the direction proposed here is already underway [20, 58].

Partial temporal decoupling

The "total decoupling" techniques considered in Part I represent an extreme form of decoupling, in that (the dispatching time of) every event is decoupled from that of every other event. In other words, our approach covers the extreme case in which each event (or STP variable) is dispatched by a different actor. Perhaps in most practical cases, however, there are fewer actors (or parties) than variables and each actor controlls a subset of the STP variables. In the NedTrain workshop, for example, we may assume that members of the same team can cooperatively determine a schedule for the subset of tasks they control. As such, it would be interesting to develop a generalization of the flexible interval schedules framework, such that the partial schedules of groups of variables are decoupled. Then, the total decoupling technique examined in Part I would emerge as a special case when every group contains a single variable. Another special case emerges when there is only one group containing all variables. It is interesting to consider what the total width of the time-windows in the interval schedule represents, in each of the two cases. In the first case, it represents concurrent flexibility. In the second case, since no pair of variables is decoupled, it represents the amount of naive flexibility. That is, such a hypothetical partial decoupling

framework would allow the unification of concurrent and naive flexibility as the two concepts become special cases of a more general definition of flexibility.

Provisional schedules

Stochastic scheduling techniques for computing a reliable predictive schedule is the main point of the approach examined in Part II of the thesis. An interesting alternative would be to consider an STP-based approach for computing reliable schedules. The term *provisional schedule* shall be used as an alternative to a predictive schedule. Given such a schedule, actors in the workshop strive to execute tasks at provisional dispatching times. Occassional deviations are unavoidable, e.g. because of equipment break-downs or unforeseen late/early deliveries. Consider that a provisional schedule is essentially a point within the solution-space of the given STP (or POS), which is essentially a polytope. Because of possible disruptive events, then, the outcome schedule (i.e. the outcome dispatching times) is expected to be another, nearby point. An interesting question to be addressed in future work, asks how to pick a provisional schedule such that our chances of ending-up with a feasible realized schedule (i.e. one within the solution-space) are maximized. In other words, how to minimize the risk of failing to dispatch our tasks successfully because of disruptions. One could consider, for example, taking the *point of gravity*, or the *center of mass*, of a polytope, which can be easily approximated by sampling random points from the polytope [47].

POS generation targetting concurrent flexibility

The approach examined in Part I assumes the computation of a POS (by translating resource constraints into additional temporal constraints) that offers high concurrent flexibility. POS generation heuristics like *solve-and-robustify* [75], however, only target the maximization of naive flexibility. As Staats et al. observed [84], such heuristics mostly yield poor results with respect to concurrent flexibility. Future work could therefore focus on devloping heuristics that specifically target concurrent flexibility. Note that our work "paved the way" for such work by establishing that concurrent flexibility is not only more accurate, but also can be computed at least as fast as naive flexibility.

Summary

The research presented in this thesis is part of the *Rolling Stock Life Cycle Logistics* applied research and development program, conducted by NedTrain. As a company, NedTrain belongs to Nederlandse Spoorwegen (NS; the principal railway company in the Netherlands) and provides maintenance services for the NS train-fleet. The aim of this program is to enhance NedTrain's competitiveness as a rolling-stock maintenance services provider. Our work focuses on the operational aspect of this R&D program, motivated by the challenge of scheduling tasks (or operations) in a NedTrain maintenance workshop, such that trains are delivered on-time for circulation in the rail network. Most tasks in the workshop have uncertain durations (or processing times), which complicates the scheduling process.

After introducing NedTrain as a company, Chapter 1 identifies the main issues with scheduling in the workshop. The point is made that scheduling under uncertainty is not so much about finding a good (or timely) schedule, as it is about continously adapting to outcome task durations without violating scheduling constraints and without compromising timeliness. Continuously changing the schedule of a workshop is not an option as it confuses and disorganizes human resources, impeding performance. As such, we consider two options that management faces in order to cope with uncertainty:

(I) Instead of using a schedule that changes frequently, give people the flexibility to (re)schedule themselves at will.

(II) Insert sufficient slack in the schedule to avoid frequent changes during task execution.

Each option is mapped to a corresponding research problem shown below:

(I) How to compute flexible schedules for independent work-teams that can be easily adapted to changes in the environment?

(II) How to compute robust and stable schedules for work-teams in order to deal with uncertainty in the duration of maintenance tasks?

Pursuing option I above, the aim is to provide as much flexibility as possible to independent decision-making parties in the workshop. The main difficulty is ensuring scheduling constraints (i.e. precedences between tasks, due-dates,

resource availability constraints) will be satisfied by a schedule that is formed gradually from decisions taken independently by different parties. The aim of pursuing option II, on the other hand, is to provide as much stability (i.e. a schedule that is not expected to change) as possible, without compromising performance (i.e. timely train deliveries). The difficulty in pursuing option II is determining how much slack to insert and at which points in the schedule, according to how uncertainty accumulates at different parts of the schedule.

Research Problem I is treated in Part I of the thesis. In the tradition of earlier work in the context of RSLCL, we consider the research area of Simple Temporal Problem (STP) constraints. The main idea behind our approach is modelling the situation in a NedTrain workshop (i.e. temporal and resource constraints) as a STP. This STP is mapped into an interval schedule. In contrast with a regular schedule (which prescribes a dispatching time per task), an interval schedule prescribes a time-interval per task. So long as each task is dispatched within its time-interval, constraint satisfaction in the workshop is guaranteed. Part I is devoted to algorithms for finding a maximum flexibility interval schedule (prescribing as wide time-intervals as possible) and for keeping it up-to-date with new information about already dispatched tasks, as it becomes available during execution.

Research Problem II is treated in Part II of the thesis. The main idea now is to model the situation in the workshop (i.e. temporal and resource constraints) as a Stochastic Task Network, i.e. a network of precedence relations between tasks with random durations. Our main assumption is that we can estimate the probability distribution of each task duration based on historical data from past maintenance sessions. Using this precise image of uncertainty in the workshop, the task network is mapped into a predictive (or stable) schedule. Human resources at the workshop are asked to simply dispatch tasks at the dispatching times prescribed in that schedule. The predictive schedule is constructed with the right amount of slack at the right places, such that it will remain mostly unchanged during task execution. Part II is devoted to algorithms for finding such a stable predictive schedule without compromising robustness (i.e. the chance of meeting train-delivery due-dates) and (optionally) for adapting it to new information during task execution.

Part I comprises Chapters 2,3,4. Chapter 2 functions as a prelude to Part I. The chapter begins with a summary of important concepts from STP-related literature. Then, gaps from existing literature not allowing us to address Research Problem I are identified. Finally, a series of Research Questions associated with those gaps are formulated and those questions are addressed in Chapters 3 and 4.

Gaps in existing literature essentially stem from efficiency and dynamicity considerations that were not addressed by the earlier work of Wilson et al. We raise the following questions:

Research Question I.1

Research Question I.2

Research Question I.3

Part II comprises Chapters 5,6,7. Chapter 5 is functionally equivalent to Chapter 2, but for Part II. The chapter begins with a summary of important concepts and problems from the literature related to Stochastic Task Networks. Then, gaps from existing literature not allowing us to address Research Problem II are identified and again, a series of Research Questions associated with those gaps are formulated. Those questions are addressed in Chapters 6 and 7.

Chapter 3

We discuss two flexibility metrics for Simple Temporal Networks (STNs): the so-called naive flexibility metric based on the difference between earliest and latest starting times of temporal variables, and a recently proposed concurrent flexibility metric. After an analysis based on a geometric interpretation of both metrics, we discuss an alternative method to compute these flexibility metrics. We establish an interesting connection between the computation of these flexibility metrics and properties of the minimal distance matrix D_S of an STN S: the concurrent flexibility metric can be computed by finding a *minimum* weight matching of a weighted bipartite graph completely specified by D_S, while the naive flexibility metric corresponds to computing a *maximum* weight matching in the same graph. From a practical point of view this correspondence offers an advantage: instead of using an $O(n^5)$ LP-based approach, to compute the concurrent flexibility metric whose worst-case complexity is bounded by $O(n^5)$, reducing the problem to a matching problem we derive an $O(n^3)$ algorithm for computing the concurrent flexibility metric.

Chapter 4

Temporal decoupling is a method to distribute a temporal constraint problem over a number of actors, such that each actor can solve its own part of the problem. It then ensures that the partial solutions provided can be always merged to obtain a complete solution. This paper discusses static and dynamic decoupling methods offering maximal flexibility in solving the partial problems. Extending previous work, we present an exact $O(n^3)$ flexibility-maximizing static decoupling method. Then we discuss an exact $O(n^3)$ method for updating a given decoupling, whenever an actor communicates a commitment to a particular set of choices for some temporal variable. This updating method ensures that: (i) the flexibility of the decoupling never decreases and (ii) every commitment once made is respected in the updated decoupling. To ensure an efficient updating process, we introduce a fast heuristic to construct a new decoupling given an existing decoupling in nearly linear time. We present some experimental results showing that, in most cases, updating an existing decoupling in case new commitments for variables have been made, significantly increases the flexibility of making commitments for the remaining variables.

Chapter 6

This paper addresses a problem of practical value in project scheduling: trading expected makespan for stability, under stochastic activity duration uncertainty. We present the formal statement of a problem that we name Proactive Stochastic RCPSP (PS-RCPSP). Assuming activity durations follow known probability distributions, PS-RCPSP asks to find a so-called earliest-start (es) policy and a proactive schedule that together minimize the weighted sum of expected project makespan and expected instability (deviation of the realized from the proactive schedule). Extending an existing MILP model for the well-known deterministic Resource-Constrained Project Scheduling Problem (RCPSP), we present a MILP model for PS-RCPSP, which allows us to find optimal (es-policy, proactive schedule) pairs. To deal with instances of practical size, we propose a Linear Programming (LP)-based and a Mixed-Integer LP (MILP)-based heuristic. Our LP-based heuristic optimizes the proactive schedule while keeping the es-policy part of the solution fixed. Our MILP-based heuristic aims to optimize the structure of the policy together with the proactive schedule. In contrast to existing state-of-art approaches such as CCP [53] and STC [89], our heuristics rely on optimizing the proactive schedule together with the scheduling policy. Experiments show that the LP-based heuristic is efficient and compares favorably with the state-of-art (i.e. achieves smaller expected makespan for a certain level of expected instability) when the aim is to achieve near-zero instability at the cost of higher makespan. The MILP-based heuristic seems more effective (albeit not as efficient) when the aim is to achieve low expected makespan at the cost of moderate or high instability.

Chapter 7

This paper concerns networks of precedence constraints between tasks with random durations, known as stochastic task networks, often used to model uncertainty in real-world applications. In some applications, we must associate tasks with reliable start-times from which realized start-times will (most likely) not deviate too far. We examine a dispatching strategy according to which a task starts as early as precedence constraints allow, but not earlier than its corresponding *planned release-time*. As these release-times are spread farther apart on the time-axis, the randomness of realized start-times diminishes (i.e. *stability* increases). Effectively, task start-times becomes less sensitive to the outcome durations of their network predecessors. With increasing stability, however, performance deteriorates (e.g. expected makespan increases). Assuming a sample of the durations is given, we define an LP for finding release-times that minimize the performance penalty of reaching a desired level of stability. The resulting LP is costly to solve, so, targeting a specific part of the solution-space, we define an associated Simple Temporal Problem (STP) and show how optimal release-times can be constructed from its earliest-start-time solution. Exploiting the special structure of this STP, we present our main result, a dynamic programming algorithm that finds optimal release-times with considerable efficiency

gains.

Bibliography

[1] Joseph Adams, Egon Balas, and Daniel Zawack. The shifting bottleneck procedure for job shop scheduling. *Management science*, 34(3):391–401, 1988.

[2] VG Adlakha and Vidyadhar G Kulkarni. A classified bibliography of research on stochastic pert networks: 1966-1987. *INFOR*, 27, 1989.

[3] Christian Artigues, Sophie Demassey, and Emmanuel Neron. *Resource-constrained project scheduling: models, algorithms, extensions and applications*. John Wiley & Sons, 2013.

[4] Christian Artigues, Philippe Michelon, and Stéphane Reusser. Insertion techniques for static and dynamic resource-constrained project scheduling. *European Journal of Operational Research*, 149(2):249–267, 2003.

[5] JJ Arts. Spare parts planning and control for maintenance operations. *(PhD), Eindhoven University of Technology, Eindhoven.(D175)*, 2013.

[6] Behzad Ashtiani, Roel Leus, and Mir-Bahador Aryanezhad. New competitive results for the stochastic resource-constrained project scheduling problem: exploring the benefits of pre-processing. *Journal of Scheduling*, 14(2):157–171, 2011.

[7] Francisco Ballestín. When it is worthwhile to work with the stochastic rcpsp? *Journal of Scheduling*, 10(3):153–166, 2007.

[8] Francisco Ballestin and Roel Leus. Resource-constrained project scheduling for timely project completion with stochastic activity durations. *Production and Operations Management*, 18(4):459–474, 2009.

[9] J-H Bartels and Jürgen Zimmermann. Scheduling tests in automotive r&d projects. *European Journal of Operational Research*, 193(3):805–819, 2009.

[10] C. Beck and A.J. Davenport. A survey of techniques for scheduling with uncertainty, 2002.

[11] Julien Bidot, Thierry Vidal, Philippe Laborie, and J Christopher Beck. A theoretic and practical framework for scheduling in a stochastic environment. *J. Scheduling*, 12, 2009.

[12] David Blaauw, Kaviraj Chopra, Ashish Srivastava, and Lou Scheffer. Statistical timing analysis: From basic principles to state of the art. *IEEE transactions on computer-aided design of integrated circuits and systems*, 27, 2008.

[13] Jacek Blazewicz, Jan Karel Lenstra, and AHG Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5(1):11–24, 1983.

[14] J.C. Boerkoel and E.H. Durfee. Distributed algorithms for solving the multiagent temporal decoupling problem. In *Proceedings of the 10th International Conference on Autonomous Agents and Multi-agent Systems (AAMAS-11)*, pages 141–148. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[15] J.C. Boerkoel and E.H. Durfee. Distributed reasoning for multiagent simple temporal problems. *Journal of Artificial Intelligence Research (JAIR)*, 47:95–156, 2013.

[16] Felix Bomsdorf and Ulrich Derigs. A model, heuristic procedure and decision support system for solving the movie shoot scheduling problem. *Or Spectrum*, 30(4):751–772, 2008.

[17] Alessio Bonfietti, Michele Lombardi, and Michela Milano. Disregarding duration uncertainty in partial order schedules? Yes, we can! In *CPAIOR*, 2014.

[18] Kristof Braeckmans, Erik Demeulemeester, Willy Herroelen, and Roel Leus. Proactive resource allocation heuristics for robust project scheduling. *DTEW Research Report 0567*, pages 1–22, 2005.

[19] A. Brambilla. *Artificial Intelligence in Space Systems: Coordination Through Problem Decoupling in Multi Agent Planning for Space Systems*. Lambert Academic Publishing, 2010.

[20] Jeb Brooks, Emilia Reed, Alexander Gruver, and James C Boerkoel. Robustness in probabilistic temporal planning. In *AAAI*, pages 3239–3246, 2015.

[21] Peter Brucker, Sigrid Knust, Arno Schoo, and Olaf Thiele. A branch and bound algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 107(2):272–288, 1998.

[22] Giuseppe Calafiore and Marco C Campi. Uncertain convex programs: randomized solutions and confidence levels. *Mathematical Programming*, 102, 2005.

[23] Philippe Chrétienne and Francis Sourd. Pert scheduling with convex cost functions. *Theoretical Computer Science*, 292, 2003.

[24] HSM Coxeter. Regular and semi-regular polytopes. i. *Mathematische Zeitschrift*, 46(1):380–407, 1940.

[25] Stefan Creemers. Minimizing the expected makespan of a project with stochastic activity durations under resource constraints. *Journal of Scheduling*, 18(3):263–273, 2015.

[26] Andrew Davenport, Christophe Gefflot, and Chris Beck. Slack-based techniques for robust schedules. In *Sixth European Conference on Planning*, 2014.

[27] Frits de Nijs and Tomas Klos. A novel priority rule heuristic: Learning from justification. In *Twenty-Fourth International Conference on Automated Planning and Scheduling*, 2014.

[28] Filip Deblaere, Erik Demeulemeester, and Willy Herroelen. Proactive policies for the stochastic resource-constrained project scheduling problem. *European Journal of Operational Research*, 214(2):308–316, 2011.

[29] R. Dechter. *Constraint processing*. Morgan Kaufmann, 2003.

[30] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Knowledge Representation*, 49:61–95, 1991.

[31] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial intelligence*, 49, 1991.

[32] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial intelligence*, 49(1):61–95, 1991.

[33] R.W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6), 1962.

[34] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34(3):596–615, July 1987.

[35] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.

[36] Daniel Godard, Philippe Laborie, and Wim Nuijten. Randomized large neighborhood search for cumulative scheduling. In *ICAPS*, volume 5, 2005.

[37] Jane N Hagstrom. Computational complexity of pert problems. *Networks*, 18(2):139–147, 1988.

[38] Jane N Hagstrom. Computing the probability distribution of project duration in a pert network. *Networks*, 20, 1990.

[39] Sönke Hartmann and Dirk Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207(1):1–14, 2010.

[40] Willy Herroelen, Erik Demeulemeester, and Bert De Reyck. A note on the paper resource-constrained project scheduling: Notation, classification, models and methods by brucker et al. *European Journal of Operational Research*, 128(3):679–688, 2001.

[41] Willy Herroelen and Roel Leus. The construction of stable project baseline schedules. *European Journal of Operational Research*, 156(3):550–565, 2004.

[42] Willy Herroelen and Roel Leus. Robust and reactive project scheduling: a review and classification of procedures. *International Journal of Production Research*, 42(8):1599–1620, 2004.

[43] Willy Herroelen and Roel Leus. Project scheduling under uncertainty: Survey and research potentials. *EJOR*, 165, 2005.

[44] L. Hunsberger. Algorithms for a temporal decoupling problem in multi-agent planning. In *Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI-02)*, pages 468–475, 2002.

[45] L. Hunsberger. *Group Decision Making and Temporal Reasoning*. PhD thesis, Harvard University, Cambridge, Massachusetts, 2002.

[46] G Igelmund and Franz Josef Radermacher. Preselective strategies for the optimization of stochastic project networks under resource constraints. *Networks*, 13, 1983.

[47] Ravi Kannan, László Lovász, and Miklós Simonovits. Random walks and an o*(n5) volume algorithm for convex bodies. *Random structures and algorithms*, 11(1):1–50, 1997.

[48] Anton J Kleywegt, Alexander Shapiro, and Tito Homem-de Mello. The sample average approximation method for stochastic discrete optimization. *SIAM J. Optimization*, 12, 2002.

[49] Rainer Kolisch and Sönke Hartmann. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European journal of operational research*, 174(1):23–37, 2006.

[50] Rainer Kolisch and Arno Sprecher. Psplib-a project scheduling problem library: Or software-orsep operations research software exchange program. *European Journal of Operational Research*, 96(1):205–216, 1997.

[51] Oumar Koné, Christian Artigues, Pierre Lopez, and Marcel Mongeau. Event-based milp models for resource-constrained project scheduling problems. *Computers & Operations Research*, 38(1):3–13, 2011.

[52] Patricio Lamas and Erik Demeulemeester. A purely proactive scheduling procedure for the resource-constrained project scheduling problem with stochastic activity durations. *J. Scheduling*, 2015.

[53] Patricio Lamas and Erik Demeulemeester. A purely proactive scheduling procedure for the resource-constrained project scheduling problem with stochastic activity durations. *Journal of Scheduling*, pages 1–20, 2015.

[54] Roel Leus. Resource allocation by means of project networks: dominance results. *Networks*, 58(1):50–58, 2011.

[55] Roel Leus, Christian Artigues, and Fabrice Talla Nobibon. Robust optimization for resource-constrained project scheduling with uncertain activity durations. In *Industrial Engineering and Engineering Management (IEEM), 2011 IEEE International Conference on*, pages 101–105. IEEE, 2011.

[56] Roel Leus and Willy Herroelen. Stability and resource allocation in project planning. *IIE transactions*, 36(7):667–682, 2004.

[57] Michele Lombardi and Michela Milano. A min-flow algorithm for minimal critical set detection in resource constrained project scheduling. *Artificial Intelligence*, 182:58–67, 2012.

[58] Kyle Lund, Sam Dietrich, Scott Chow, and James C Boerkoel. Robust execution of probabilistic temporal plans. In *AAAI*, pages 3597–3604, 2017.

[59] Donald G Malcolm, John H Roseboom, Charles E Clark, and Willard Fazar. Application of a technique for research and development program evaluation. *Operations research*, 7, 1959.

[60] Laurent Michel and Pascal Van Hentenryck. Iterative relaxations for iterative flattening in cumulative scheduling. In *ICAPS*, volume 4, pages 200–208, 2004.

[61] Rolf H Möhring, Franz Josef Radermacher, and Gideon Weiss. Stochastic scheduling problems i – general strategies. *Zeitschrift für Operations Research*, 28(7):193–260, 1984.

[62] Rolf H Möhring, Franz Josef Radermacher, and Gideon Weiss. Stochastic scheduling problems ii – set strategies. *Zeitschrift für Operations Research*, 29(3):65–104, 1985.

[63] Kiriakos Simon Mountakis. Stochastic Scheduling of Train Maintenance Projects. Master's thesis, Delft University of Technology, the Netherlands, 2013.

[64] S. Mountakis, T.B. Klos, and C. Witteveen. Temporal flexibility revisited: Maximizing flexibility by computing bipartite matchings. In *Proceedings Twenty-Fifth International Conference on Automated Planning and Scheduling*, 2015.

[65] Simon Mountakis, Tomas Klos, Cees Witteveen, and Bob Huisman. Exact and heuristic methods for trading-off makespan and stability in stochastic project scheduling. In *MISTA*, 2015.

[66] Nederlandse Spoorwegen. Ns annual report. Technical report, 2016.

[67] Angelo Oddi and Riccardo Rasconi. Iterative flattening search on rcpsp/max problems: Recent developments. In *Recent Advances in Constraints*, pages 99–115. Springer, 2009.

[68] Stefano Pallottino. Shortest-path methods: Complexity, interrelations and new propositions. *Networks*, 14, 1984.

[69] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982.

[70] L.R. Planken. *Algorithms for Simple Temporal Reasoning*. PhD thesis, Delft University of Technology, 2013.

[71] L.R. Planken, M.M. de Weerdt, and R. van der Krogt. Computing all-pairs shortest paths by leveraging low treewidth. *Journal of Artificial Intelligence Research*, 43(1):353–388, January 2012.

[72] L.R. Planken, M.M. de Weerdt, and C. Witteveen. Optimal temporal decoupling in multiagent systems. In VanderHoek, Kaminka, Lesperance, Luck, and Sen, editors, *Proceedings of the Ninth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-10)*, pages 789–796. IFAAMAS, May 2010.

[73] N. Policella, A. Cesta, A. Oddi, and S.F. Smith. From precedence constraint posting to partial order schedules: A CSP approach to robust scheduling. *AI Communications*, 20(3):163–180, 2007.

[74] N. Policella, S. F. Smith, A. Cesta, and A. Oddi. Generating robust schedules through temporal flexibility. In S. Zilberstein, J. Koehler, and S. Koenig, editors, *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling (ICAPS 2004)*, pages 209–218. AAAI, 2004.

[75] Nicola Policella, Amedeo Cesta, Angelo Oddi, and Stephen F Smith. Solve-and-robustify. *Journal of Scheduling*, 12(3):299–314, 2009.

[76] Nicola Policella, Angelo Oddi, Stephen F Smith, and Amedeo Cesta. Generating robust partial order schedules. In *CP*, 2004.

[77] M. E. Pollack and I. Tsamardinos. Efficiently dispatching plans encoded as simple temporal problems. In I. Vlahavas and D. Vrakas, editors, *Intelligent Techniques for Planning*, pages 296–319. Idea Group Publishing, 2005.

[78] F.A. Potra and S.J. Wright. Interior-point methods. *Journal of Computational and Applied Mathematics*, 124(1–2):281 – 302, 2000.

[79] Florian A Potra and Stephen J Wright. Interior-point methods. *J. Computational and Applied Mathematics*, 124, 2000.

[80] J.E. Parada Puig. *Serviceability of passenger trains during acquisition projects*. PhD thesis, University of Twente, 2015.

[81] Eugene Santos and Joel D Young. Probabilistic temporal networks: A unified framework for reasoning with time and uncertainty. *International Journal of Approximate Reasoning*, 20(3):263–291, 1999.

[82] Andreas Schutt, Thibaut Feydy, Peter J Stuckey, and Mark G Wallace. Solving rcpsp/max by lazy clause generation. *Journal of Scheduling*, 16(3):273–289, 2013.

[83] Vladimir Shestak, Jay Smith, Anthony A Maciejewski, and Howard Jay Siegel. Stochastic robustness metric and its use for static resource allocations. *J. Parallel and Distributed Computing*, 68, 2008.

[84] JJ Staats. Improving pcp algorithms using flexibility metrics: Creating flexible schedules for technical maintenance. 2014.

[85] Frederik Stork. Branch-and-bound algorithms for stochastic resource-constrained project scheduling. *Technical rep*, pages 702–2000, 2000.

[86] Frederik Stork and Marc Uetz. On the generation of circuits and minimal forbidden sets. *Mathematical programming*, 102(1):185–203, 2005.

[87] Robert Endre Tarjan. Edge-disjoint spanning trees and depth-first search. *Acta Informatica*, 6, 1976.

[88] Stijn Van de Vonder, Francisco Ballestin, Erik Demeulemeester, and Willy Herroelen. Heuristic procedures for reactive project scheduling. *Computers & Industrial Engineering*, 52(1):11–28, 2007.

[89] Stijn Van de Vonder, Erik Demeulemeester, and Willy Herroelen. Proactive heuristic procedures for robust project scheduling: An experimental analysis. *European Journal of Operational Research*, 189(3):723–733, 2008.

[90] Stijn Van de Vonder, Erik Demeulemeester, and Willy Herroelen. Proactive heuristic procedures for robust project scheduling: An experimental analysis. *EJOR*, 189, 2008.

[91] Stijn Van de Vonder, Erik Demeulemeester, Willy Herroelen, and Roel Leus. The use of buffers in project management: The trade-off between stability and makespan. *International Journal of production economics*, 97(2):227–240, 2005.

[92] Stijn Van de Vonder, Erik Demeulemeester, Willy Herroelen*, and Roel Leus. The trade-off between stability and makespan in resource-constrained project scheduling. *International Journal of Production Research*, 44(2):215–236, 2006.

[93] P. van Leeuwen and C. Witteveen. Temporal decoupling and determining resource needs of autonomous agents in the airport turnaround process. In *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 02*, WI-IAT '09, pages 185–192. IEEE Computer Society, 2009.

[94] Vineeth Veetil, Kaviraj Chopra, David Blaauw, and Dennis Sylvester. Fast statistical static timing analysis using smart monte carlo techniques. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30, 2011.

[95] Chandramouli Visweswariah, Kaushik Ravindran, Kerim Kalafala, Steven G Walker, Sambasivan Narayan, Daniel K Beece, Jeff Piaget, Natesan Venkateswaran, and Jeffrey G Hemmett. First-order incremental block-based statistical timing analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(10):2170–2180, 2006.

[96] M. Wilson. *Robust Scheduling in an Uncertain Environment*. PhD thesis, Delft University of Technology, 2016.

[97] M. Wilson, C. Witteveen, T.B. Klos, and B. Huisman. Enhancing flexibility and robustness in multi-agent task scheduling. In *Proceedings OPTMAS workshop*, 2013.

[98] Michel Wilson, Tomas Klos, Cees Witteveen, and Bob Huisman. Flexibility and decoupling in Simple Temporal Networks. *Artificial Intelligence*, 214:26–44, 2014.

[99] Michel Wilson, Nico Roos, Bob Huisman, and Cees Witteveen. Efficient workplan management in maintenance tasks. In *BNAIC 2011: 23rd Benelux Conference on Artificial Intelligence, Ghent, Belgium, 3-4 November 2011*, 2011.