TECHNISCHE UNIVERSITEIT DELFT

MASTER OF SCIENCE THESIS IN COMPUTER SCIENCE

Optimization for Production Planning using Probabilistic Simple Temporal Networks

Anna KALANDADZE

Supervisors:
Prof. Dr. M. M. DE WEERDT
PhD Candidate K. VAN DEN HOUTEN

25th April 2025

Delft University of Technology

Optimization for Production Planning using Probabilistic Simple Temporal Networks

Master's Thesis in Computer Science

Algorithmics group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

Anna Kalandadze

25th April 2025

Author
Anna Kalandadze

Title
Optimization for Production Planning using Probabilistic Simple Temporal Networks

MSc presentation
May 2, 2025

Graduation Committee

Prof. Dr. M. M. de Weerdt	Delft University of Technology
Dr. ir. J.T. van Essen	Delft University of Technology
PhD Candidate K. van den Houten	Delft University of Technology

Abstract

Production planning in the biomanufacturing sector presents significant challenges due to uncertainties in job durations caused by biological variability, environmental conditions, and raw material quality. Traditional scheduling methods typically fail to adapt to these uncertainties, leading to suboptimal outcomes. This research addresses this issue at DSM-Firmenich, focusing on optimizing production planning while maximizing profit, adhering to deadlines, and efficiently utilizing resources. We propose an integrated approach using Mixed-Integer Linear Programming (MILP) and Constraint Programming (CP) models, alongside Probabilistic Simple Temporal Networks (PSTNs) to handle uncertainty in real-time scheduling. The study introduces an offline optimization procedure for proactive scheduling decisions and a reactive real-time algorithm for adjustments of the planned schedule. This work showcases the potential of applying PSTNs in biomanufacturing and sets the stage for future research aimed at enhancing real-time execution strategies in factory environments.

.

Preface

This thesis represents the final step in my Master of Science journey in Computer Science at Delft University of Technology, a journey that began in 2020 when I started my studies. As I approach the conclusion of this phase, I reflect on the experiences and challenges that have shaped this work and my academic growth.

I would like to extend my sincere gratitude to my thesis coordinator, Prof. Dr. Mathijs de Weerdt, and my daily supervisor, Kim van den Houten, for their continuous support and invaluable guidance throughout this process. Their insightful feedback, encouragement, and dedication have been essential in shaping both my thesis and my personal development as a researcher.

I am also thankful to all those who participated in the Weekly Wednesday Meetings organized by the Algorithmics group. These meetings were invaluable for expanding my knowledge, exchanging ideas, and receiving feedback from a diverse group of fellow students. Presenting my work in these settings helped me grow academically and personally, and I am grateful for the enriching experience.

Finally, I want to express my deepest appreciation to my family. Their constant support has been a source of strength throughout this journey. Their belief in me, their encouragement during challenging moments, and their love have been crucial in helping me reach this point. I am truly grateful for everything. Thank you for being my anchor.

Anna Kalandadze

Delft, The Netherlands
25th April 2025

Contents

1 Introduction **1**

2 Background and Related Work **5**
2.1 Scheduling 5
2.1.1 RCPSP/max 6
2.1.2 Stochastic RCPSP/max 8
2.2 Approaches to solve SRCPSP/max problems 8
2.2.1 Proactive 8
2.2.2 Hybrid 9
2.3 Approaches for production planning and scheduling 17

3 Research questions **21**

4 Problem definition **25**
4.1 Production planning: DSM Firmenich case 25
4.2 MILP model 29
4.3 CP model 32

5 PSTN Stochastic Approach **35**
5.1 Method overview 35
5.2 Deterministic Approximation 36
5.3 PSTN construction 37
5.4 PSTN approximation 41
5.5 Real time execution 41

6 Evaluation **45**
6.1 Instance creation 45
6.1.1 Uncertainty in the instances 47
6.2 Comparing CP and MILP models to solve a deterministic problem 47
6.3 Influence of POS on the expected actual profit 49
6.3.1 Results: Using original constraints while varying deterministic job durations 51
6.3.2 Results: Using the soft deadline constraints while varying deterministic job durations 53

6.4 Evaluation of PSTN-based approach 55

7 Discussion **57**

8 Conclusion **61**

A Advanced algorithms **69**

B Detailed experiment results **73**

C Future RTE* Recovery Algorithm **75**

D CP model with soft deadlines **77**

E Reproducibility **81**

E.1 Data Access, Randomness and Seeding 81

Chapter 1

Introduction

Production planning in a factory involves organizing and coordinating all activities required for manufacturing products. This process includes estimating production volumes for upcoming periods, forecasting inventory levels, and identifying the workforce and resources needed to implement the plan. While estimating required products is typically based on historical data, decisions regarding which orders to accept and how to allocate resources are subject to optimization. These decisions directly impact profit.

Factories often manage multiple orders from clients, with each order specifying required products and deadlines for completion. Each product comprises a sequence of unit operations called "jobs." Factories must complete all necessary jobs to produce a product. Precedence or temporal constraints connect these jobs. Such constraints dictate the permissible time differences between various jobs' start and/or finish times. Every job has specific resource requirements; some jobs can run in parallel since multiple resources are available.

In a theoretical setting, each job has a defined duration. In the real world, however, such durations are often uncertain. While a planner can estimate job durations based on historical data, such as average time for completion, the actual time required may vary due to natural and operational factors. In the biomanufacturing industry, such factors include biological variability, temperature or humidity variability, and raw product quality. The variabilities affect job scheduling, as the actual durations of jobs may lead to deviations from the original schedule constructed using the predicted job duration. Such deviations may result in the inability to produce an order on time.

DSM-Firmenich, a global leader in Nutrition, Health, and Bioscience, faces challenges in biomanufacturing, which involves large-scale, long-horizon scheduling problems. Currently, their workflow involves planners estimating demand for upcoming orders and calculating production schedules based on predicted demand. Products are then scheduled for production according to resource availability and estimated deadlines. Optimizing such production schedules is challenging, as it is an NP-hard problem. The complexity increases further when uncertainties in

job durations appear, making it difficult to balance deadlines with efficient parallel execution.

Furthermore, DSM-Firmenich needs to maintain the desired inventory level of products. We define this combination of factors as an *Integrated Production Planning Problem*. The problem requires advanced algorithms to determine optimal schedules that minimize delays, maximize profit, and ensure effective resource utilization. Without such optimization, inefficiencies can increase operational costs and strain client relationships.

This research focuses on leveraging advanced computational methods to optimize production planning in manufacturing to address challenges faced by DSM-Firmenich. We aim to design and implement techniques that (1) maximize profit while adhering to the orders' deadline and optimizing resource usage and (2) dynamically adapt schedules to account for deviations in job durations.

We analyze Mixed-Integer Linear Programming (MILP) and Constraint Programming (CP) to assess their strengths and limitations in optimizing production scheduling. While researchers have widely applied both approaches in production planning [Bekrar et al., 2012] [Aguirre et al., 2018][Hosseini-Motlagh et al., 2021], key aspects such as inventory and deadline management remain underexplored. For example, the MILP model in [Belil et al., 2018] accounts for the inventory but primarily aims to minimize stock levels. However, maintaining adequate inventory is essential in the DSM-Firmenich factory setting. Therefore, we propose integrating inventory management into the model, enabling planners to monitor stock levels daily.

Additionally, we investigate Temporal Networks and their probabilistic extensions (PSTNs) to enable real-time schedule adjustments. Although the research [van den Houten et al., 2024] applied Simple Temporal Networks with Uncertainty (STNUs) to scheduling problems, demonstrating the ability of STNUs to execute schedules in real-time despite the presence of uncertainty efficiently, we did not find STNUs applications in production planning. STNUs assume bounded uncertainty values [Morris et al., 2001], which are unsuitable for production planning, where probability distributions better represent job durations. The primary advantage of PSTNs in production planning lies in their ability to model uncertainty explicitly. By representing job durations as probability distributions, PSTNs allow for a more accurate and flexible representation of the production planning problem, compared to traditional deterministic models that cannot account for these uncertainties. To our knowledge, researchers have not yet applied PSTNs to production planning. This gap presents an opportunity to explore how PSTNs can support planners by incorporating probabilistic reasoning to manage uncertainties in a factory setting better.

Simple Temporal Networks (STNs) can execute in real time. A similar execution algorithm exists for STNUs [Hunsberger and Posenato, 2024b]. The algorithm enables real-time schedule execution without rescheduling, ensuring rapid change adaptation. It dynamically responds to uncertainties during execution, not relying on pre-defined job start times. Instead, jobs are initiated as soon as the constraints

permit, allowing for more flexibility and responsiveness. This adaptability is particularly critical in dynamic environments such as factories, where quick decisions are essential to minimize downtime and achieve production goals. A drawback of the existing STNU real-time execution algorithm is that it fails when one of the problem constraints is not satisfied. In a factory setting, we do not want to terminate all jobs if one violates a constraint due to uncertainty. Therefore, we introduce a real-time execution algorithm that dynamically responds to uncertainties. Our implementation of the real-time execution algorithm adjusts the schedule when execution is disrupted and continues running to maximize expected profit.

This new application will assist planners in scheduling jobs required for products, achieving optimal results in deterministic scenarios, and estimating optimal outcomes for uncertain scenarios. The contributions of this paper can be summarized as follows:

1. We develop MILP and CP models for the integrated production planning and scheduling problem faced by DSM-Firmenich, which aims to maximize profit.

2. We present a novel PSTN-based approach consisting of (i) an offline optimization procedure that optimizes the scheduling decisions proactively and (ii) a new reactive real-time algorithm designed to react to uncertainty.

This research aims to provide tools for planners that reduce manual effort, improve resource utilization, and ensure the timely delivery of products. Furthermore, we provide a detailed analysis of the existing PSTN tools and their usage in practice.

The remainder of this paper is organized as follows. **Chapter 2** explains background needed to understand this research. Additionally, it presents a comprehensive literature review on MILP and CP in production planning. It discusses offline scheduling approaches, such as Partial Order Schedules (POS) and online scheduling methods utilizing temporal networks. **Chapter 3** formalizes the research questions addressed in the paper. **Chapter 4** models the DSM-Firmenich production planning problem using MILP and CP. **Chapter 5** introduces the proposed PSTN-based scheduling method, detailing both the offline optimization phase and the online real-time scheduling adjustments. **Chapter 6** presents the findings, evaluating a newly implemented PSTN-based approach. We discuss the results obtained in **Chapter 7**. Finally, we conclude the work, summarizing key contributions in **Chapter 8**.

By the end of this research, we aim to demonstrate how computational methods can transform production planning into a more efficient and adaptive process to maximize factory profit.

Chapter 2

Background and Related Work

This chapter provides the necessary background and reviews related work to establish core concepts and identify research gaps in production planning and scheduling.

Section 2.1 focuses on scheduling, outlining its role in production planning. It introduces a Resource-Constrained Project Scheduling Problem (RCPSP), a fundamental scheduling problem in manufacturing, and explores practical constraints that extend it, such as precedence constraints, known as a Resource-Constrained Project Scheduling Problem with Time Lags (RCPSP/max), and stochastic job durations, referred to as a Stochastic Resource-Constrained Project Scheduling Problem with Time Lags (SRCPSP/max).

Since the SRCPSP/max incorporates uncertainties and can represent the real-world DSM-Firmenich scheduling problem, Section 2.2 examines approaches to solve the SRCPSP/max problem, including proactive, reactive, and hybrid scheduling strategies. We focus on hybrid methods, such as Partial Order Schedules (POS), Simple Temporal Networks (STNs), and their extensions, which allow for flexible scheduling under uncertainty. We also highlight the recent advancements in Probabilistic Simple Temporal Networks (PSTNs), which model stochastic job durations more accurately than traditional bounded uncertainty models.

Lastly, Section 2.3 explores different optimization techniques applied in production planning, including Mixed Integer Linear Programming (MILP) and Constraint Programming (CP) models.

By the end of the chapter, readers should clearly understand the concepts relevant to this research in production planning, with the research gaps outlined to facilitate the formulation of research questions.

2.1 Scheduling

Scheduling involves assigning start and end times to the jobs outlined in the production plan, ensuring efficient resource utilization, and meeting deadlines.

In the DSM-Firmenich manufacturing setting, there is a list of orders, where

each order contains multiple products, and each product typically requires several stages (or jobs) to complete.

Each job has specific resource demands, and resources, such as machinery, labor, or materials, are limited. This scheduling scenario is often modeled as the Resource-Constrained Project Scheduling Problem (RCPSP) [Baar et al., 1999], a well-known scheduling problem.

The RCPSP has multiple products, each requiring multiple jobs, which must be scheduled. Each job requires certain resources. Jobs can run in parallel as long as resource constraints are satisfied. The typical objective in the RCPSP is to minimize the makespan, the total time required to complete all jobs. However, in manufacturing, objectives often include maximizing profit while adhering to strict deadlines and optimizing resource utilization.

Many real-world scheduling problems in production planning extend the RCPSP with additional constraints and complexities, such as:

- Precedence/Temporal Constraints: Certain jobs cannot start until specific preceding jobs are completed [Revesz, 2009]. These dependencies ensure that production sequences are logically and operationally feasible. In addition, minimum and maximum time lags often connect such jobs. For instance, a job might need to start within a certain timeframe after completing another job to maintain product quality. This alteration of the RCPSP is called a Resource-Constrained Project Scheduling Problem with Minimum and Maximum Time lags (RCPSP/max) [Neumann et al., 2006].

- Stochastic Job Durations: In manufacturing, job durations may vary due to equipment performance, material quality, or environmental conditions [Manzini and Urgo, 2015]. These uncertainties make scheduling more challenging, requiring robust approaches to accommodate variability while maintaining feasibility. This problem can be addressed as a Stochastic Resource-Constrained Project Scheduling Problem with Time Lags (SRCPSP/max) [Bruni et al., 2015].

DSM-Firmenich acknowledges both of these complexities. There are minimum and maximum time lags between jobs to produce a product successfully. Factors such as the required cleaning time of a machine after completing a job can determine such time lags. Additionally, uncertainty arises due to the biological nature of the jobs. We recognize that certain aspects of the defined integrated production planning problem align with the SRCPSP/max structure. Consequently, we leverage this scheduling problem as a framework to model and represent the DSM-Firmenich production planning case. Below, we provide a detailed explanation of the RCPSP/max and the SRCPSP/max.

2.1.1 RCPSP/max

The RCPSP/max variant extends the complexity of the RCPSP by incorporating minimum and maximum time lags between jobs [Neumann et al., 2006]. These

constraints create stricter temporal dependencies than standard precedence constraints by defining precise time windows within which a job must start or finish relative to another job. This additional layer of complexity requires considering both time-lag and resource restrictions simultaneously, significantly complicating the scheduling process.

We formally express the RCPSP/max can as follows [Neumann et al., 2006]:

- $A = \{0, \ldots, n, n+1\}$ represents jobs, where 0 represents the start and $n+1$ represents the end of the production time.
- Each job $i \in A$ has a duration d_i.
- There are $R = \{1, \ldots, k\}$ resources available.
- Each job i has a resource consumption r_{ik} of resource k.
- c_k indicates capacity for each resource k.
- lag_{ij}^{min} indicates a minimum time-lag between jobs i and j.
- lag_{ij}^{max} indicates a maximum time-lag between jobs i and j.
- $Temp_{min}$ represents jobs connected by minimum time-lags
- $Temp_{max}$ represents jobs connected by maximum time-lags

Let $start_i$ be the start time of each job i. Then, the start of the first job is zero, $start_0 = 0$. Precedence constraints must be respected:

$$start_j \geq start_i + lag_{ij}^{min} \quad \forall (i,j) \in (Temp_{min})$$

$$start_j \leq start_i + lag_{ij}^{max} \quad \forall (i,j) \in (Temp_{max})$$

Resource constraints must be satisfied:

$$\sum_{i \in A} r_{ik} * \alpha_{ik}(t) \leq c_k \quad \forall k \in R, t$$

Here, $\alpha_i(t)$ indicates if t is within the execution window of job i ($\alpha_i(t) = 1$). Otherwise, $\alpha_i(t) = 0$.

In the DSM-Firmenich case, scheduling involves assigning jobs to specific resource IDs. However, since an efficient algorithm exists for converting a start-time solution into a concrete machine assignment, we can first focus on solving the scheduling problem in terms of start times and subsequently apply this algorithm [Kleinberg and Tardos, 2005, p. 124] to determine the final machine allocations.

2.1.2 Stochastic RCPSP/max

Another challenge in the RCPSP/max is managing stochastic processing times. In the factory-based scenario, the durations of jobs are not fixed and may fluctuate. This can happen due to the biological nature of processes like fermentation, resembling the SRCPSP/max [Bruni et al., 2015]. This stochasticity introduces uncertainty into the scheduling process, as the actual duration of each job becomes known only upon its completion. Depending on the real execution time of the job, the received schedule may fluctuate. Furthermore, it can lead to violations of time lag and deadline constraints. Such violations may make previously feasible schedules infeasible under the new durations. Conversely, scenarios may arise where a schedule deemed infeasible under deterministic assumptions becomes feasible after accounting for stochastic variations if the actual job durations are shorter than expected.

When infeasibility arises due to uncertainty, the biomanufacturing factory may encounter customer dissatisfaction and profit loss. Relying solely on deterministic scheduling in such a setting is insufficient. Instead, incorporating and predicting uncertainty is essential to minimizing the risk of infeasibility. By integrating stochastic approaches, we can develop adaptive scheduling strategies that enhance resilience to uncertainty.

2.2 Approaches to solve SRCPSP/max problems

There are two main spectrums for stochastic scheduling in the literature: proactive and reactive scheduling. The primary objective of proactive scheduling is to develop a robust schedule in advance, accounting for potential uncertainties [Herroelen and Leus, 2005]. In contrast, reactive approaches focus on dynamically adjusting to uncertainties as they arise during execution. Recent studies demonstrate that combining proactive methods with online rescheduling can yield improved results [van den Houten et al., 2024], motivating further exploration of hybrid proactive-reactive approaches.

2.2.1 Proactive

In the DSM-Firmenich case, we assume job scheduling follows the SRCPSP/max. Proactive scheduling approaches for such stochastic problems often employ the Sample Average Approximation (SAA) method [Kleywegt et al., 2002], where samples are drawn from stochastic distributions and incorporated as scenarios within a stochastic programming formulation. The solver seeks a feasible solution across all scenarios while optimizing the average objective.

The current state-of-the-art proactive approach for the SRCPSP/max problem is `SORU` [Varakantham et al., 2016], an SAA-based Mixed Integer Programming (MIP) approach designed to minimize the α-robust makespan. The method constructs an SAA formulation using a subset of sampled durations. It determines start

times for each job for each sample while ensuring that precedence constraints with minimum and maximum time lags are satisfied. Additionally, it allows up to $\alpha\%$ of scenarios to violate resource feasibility constraints, aiming to minimize the sample average makespan across the selected samples.

The SAA method can be computationally expensive due to the large number of required samples. A heuristic variant, `SORU-H`, has been introduced to approximate the α-robust makespan [Varakantham et al., 2016]. Instead of relying on multiple sampled scenarios, `SORU-H` uses a single approximating sample representing a quantile of the duration distribution, significantly reducing computational efforts.

2.2.2 Hybrid

Partial Order Schedules (POS)

A Partial Order Schedule (POS) is a hybrid approach that represents a job network where every feasible temporal solution is guaranteed to be resource-consistent [Policella et al., 2007]. In a POS, jobs follow a relative ordering, ensuring that resource feasibility is always satisfied.

A single-point solution with chaining can construct a POS [Policella et al., 2004]. The algorithm divides each resource r_j with capacity cap_j into cap_j single-capacity sub-resources. Jobs are sorted by the start times and assigned to available capacity units at their respective start times. Once a sub-resource allocates a job [Kleinberg and Tardos, 2005, p. 124], a new precedence constraint is introduced between the job and all previous jobs using the same sub-resource. We later refer to this chaining algorithm as `get_resource_chains`.

Example 2.2.1 *In Figure 2.1, resource* 1*, with a capacity of* 5*, is divided into five sub-resources (machines). After job allocations on sub-resources, resource chains indicate that job* 0_0_0 *must execute before the jobs* 1_1_1 *and* 0_1_1*. Job* 1_1_1 *must finish before job* 0_1_1 *starts.*

Simple Temporal Networks

Temporal models, e.g., a Simple Temporal Network (STN), can complement a POS. Such temporal models include nodes representing time points, and edges, which represent temporal constraints [Dechter et al., 1991]. An STN includes only *ordinary* edges, meaning that a chosen value bounds the execution time between two nodes. Researchers formally express STN as (T, C), where T represents a set of real-valued variables (time-points) and C represents a set of temporal constraints [Dechter et al., 1991]. For example, $A \xrightarrow{3} B$ indicates a relation of two time points A and B. The ordinary link shows that $B - A \leq 3$. The edge $A \xrightarrow{-3} B$ indicates the relation $B - A \leq -3$, which results in a lower bound edge. We provide an example to help the readers understand STNs intuitively.

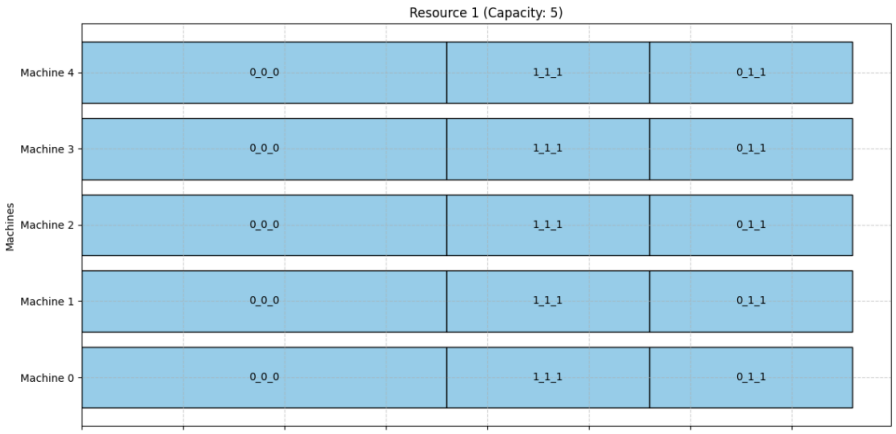

Figure 2.1: Resource chains

Example 2.2.2 *Suppose Anna must travel to an important meeting and arrive at her destination within 90 minutes after departing. An STN can model this situation: let t_1 and t_2 represent the time Anna departs from home and arrives at the meeting. Then, by adding $t_1 \xrightarrow{90} t_2$, we express temporal relation $t_2 \leq 90 + t_1$.*

Researchers have developed a flexible and efficient Real-Time Execution (RTE) algorithm for STNs, which maintains time windows for each time point [Muscettola et al., 1998]. As each time point X executes, the algorithm propagates constraints locally to the neighboring time points in the STN graph rather than across the entire network. We demonstrate RTE using Example 2.2.2. Suppose t_1 happens at timestamp 10. Then, the RTE algorithm propagates a constraint stating $t_2 \leq 100$. It executes t_2 before 100, satisfying the initial constraint.

Simple Temporal Networks with Uncertainty

STNs are extended into Simple Temporal Networks with Uncertainty (STNUs) to incorporate uncertainty. In STNUs, *contingent* links connect an activation point (controlled by the agent) to a contingent point (controlled by external factors). The duration of a contingent link varies within specified lower and upper bounds. An STNU is a triple, $S = (T, C, L)$ [Morris et al., 2001], where:

- (T, C) is an STN,
- L is a set of contingent links, each of the form (A, x, y, C), where A is the activation time point and C is the contingent time-point [Morris et al., 2001]. The duration is bounded: $C - A \in [x, y]$, but it is uncontrollable. Researchers also call these links `labeled` edges [Morris and Muscettola, 2005].

Some studies model the temporal structure using STNUs, where they represent each job with two nodes, start and finish, and create contingent links based on the job's duration bounds [Lombardi and Milano, 2009, Lombardi et al., 2013]. For example, $A \xrightarrow{[3,5]} B$ indicates the job with start time A and end time B. The contingent link shows that $B - A \in [3, 5]$. External factors determine the exact value of such an edge. Two labeled edges in the STNU often represent such a contingent link: lower bound $A \xrightarrow{3} B$ and upper bound $B \xrightarrow{-5} A$.

In recent work, STNUs are applied to the SRCPSP/max problem [van den Houten et al., 2024]. Solving the deterministic RCPSP/max problem with estimated job durations generates a fixed-point schedule. Then, the `get_resource_chains` algorithm constructs resource chains.

The construction of the STNU includes the following steps:

- Nodes represent the start and end times of each job.
- Contingent links are added between start and end nodes, with their bounds defined by the job's duration range.
- Researchers model minimal and maximal time lags as ordinary edges in the temporal graph to ensure precedence constraints are respected. They achieve this by inserting a directed edge from the successor start node B to the predecessor start node A as $B_{start} \xrightarrow{-lag} A_{start}$, where lag is a time lag between A and B, following the definition of the RCPSP/max time lag constraint.
- The work [van den Houten et al., 2024] incorporates resource chain dependencies derived from the chaining procedure as additional ordinary edges. From the example in Figure 2.1, ordinary edges from the start node of 1_1_1 to the finish node of 0_0_0 and from the start node of 0_1_1 to the finish node of 1_1_1 are inserted. In the STNU, this means that job 0_0_0 must finish before 1_1_1 starts, and job 1_1_1 must finish before 0_1_1 starts.

Example 2.2.3 *We consider a real-world scenario to present an STNU, where Anna still needs to commute to an important meeting across town. She relies on two transportation modes: a bus and a train. However, the bus route is subject to traffic delays, and the train's departure time is fixed at timestamp 40. The bus ride can last 25 to 35 minutes, meaning it could take longer or shorter depending on traffic. She must walk to the train station for 2 to 5 minutes upon arrival. The train ride can last 27 to 33 minutes, and then she must walk 10 to 15 minutes to reach her destination. She must reach her destination within 90 minutes to make it on time. STNUs can model this problem in Figure 2.2.*

Execution Strategies of STNU

The RTE algorithm suits STNs but cannot be directly applied to STNUs due to the unknown duration of jobs.

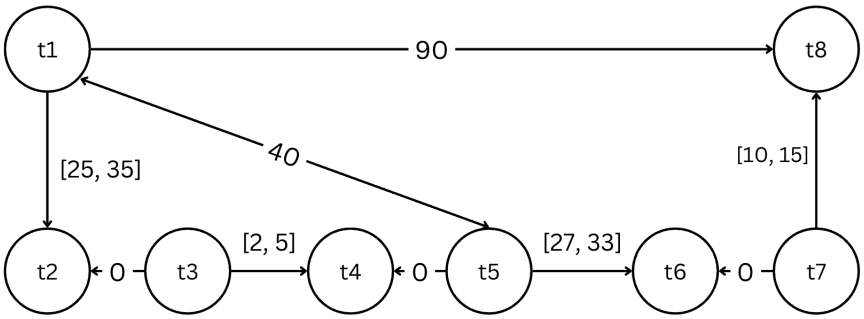

Figure 2.2: An STNU example, inspired by [Gao et al., 2020]

An STNU is dynamically controllable (DC) if there exists a dynamic, real-time execution strategy that ensures all constraints are satisfied, regardless of how the contingent durations unfold [Morris et al., 2001]. This means that even if each job would take its upper bound to execute, the schedule remains feasible.

During real-time execution, the algorithm applies constraint-propagation rules. These rules deduce implicit constraints from existing ones. For example, if two edges (A, B) and (B, C) exist, the algorithm introduces a new edge (A, C) with a derived constraint. A newly introduced edge bypasses an existing one if it creates an alternative connection that removes the influence of the original constraint.

Such constraint propagation rules can introduce a path P in an STNU, where newly derived edges bypass each lower constraint edge in P [Morris, 2006]. We call such a path semi-reducible.

A semi-reducible negative cycle (SRN cycle) is a cycle where applying constraint-propagation rules leads to a negative-weight cycle, meaning the sum of all constraint values in the cycle becomes negative. The dynamic controllability (DC) of an STNU is guaranteed if and only if the network contains no such SRN cycles [Morris, 2006].

Example 2.2.4 *We summarize an example of the SRN cycle from [Hunsberger and Posenato, 2024c]. In Figure 2.3, $C - A \leq 1$, and $D - C \leq -1$, which propagates to $D - A \leq 0$. The value of the cycle (A, C, D, B, A) is negative, resulting in the SRN cycle.*

The current state-of-the-art algorithm for checking the dynamic controllability of an STNU operates with a time complexity of $O(mn + k^2 n + kn \log n)$, where m represents the number of edges, n is the number of nodes, and k relates to contingent constraints [Hunsberger and Posenato, 2022].

If an STNU is DC, executing the STNU in real-time without risk of infeasibility is possible. However, the RTE algorithm cannot be directly applied to the STNU. Instead, the STNU is first converted to an Extended STNU (ESTNU), which generates wait edges representing a conditional constraint [Hunsberger and Posenato,

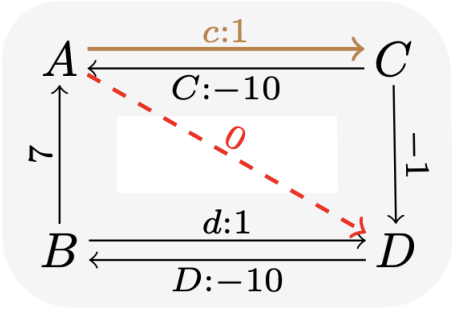

Figure 2.3: SRN cycle example, taken from [Hunsberger and Posenato, 2024c]

2024b]. DC-checking algorithms usually add the wait edges if an STNU is DC. The algorithms also add a time point Z, which is fixed at zero and must execute before all other time points. For example, suppose we have two contingent edges $A \to B$ and $A \to C$. A wait edge $(C, B, -weight, A)$ indicates that while B is not executed, C must wait at least $weight$ after A [Morris, 2014] [Hunsberger and Posenato, 2024b]. Such additional constraints indicate that a time point must wait until an activated contingent link executes.

Example 2.2.5 *We follow Example 2.2.3 and convert the given STNU into an ESTNU. The following wait edges are added:*

- $(t_3, t_2, -35, t_1)$*, which indicates that* t_3 *must wait for 35 units of time after* t_1*, while* t_2 *is not executed.*
- $(t_5, t_4, -5, t_3)$*, signifying that* t_5 *must wait for at least 5 units of time after* t_3*, while* t_4 *is not executed.*
- $(t_7, t_6, -33, t_5)$*, representing that* t_7 *must wait for at least 33 units of time after* t_5*, while* t_6 *is not executed.*

A time point Z *indicates that it must execute before all other time points. We present the ESTNU in Figure 2.4.*

The obtained ESTNU executes in real-time using the `RTE⋆` algorithm [Hunsberger and Posenato, 2024b], a modified version of the RTE algorithm for STNs. This algorithm operates iteratively by generating execution decisions, observing contingent time points (CTPs), and updating information when a CTP executes. Each time point has a lower and upper execution bound, initialized at 0 and $+\infty$, respectively.

The algorithm identifies activation time points without negative outgoing edges and marks them as `enabled`. These time points can execute. When one of them executes, the algorithm updates the bounds for its neighboring nodes based on the constraints. It then checks whether some of the CTPs execute. Since the CTP

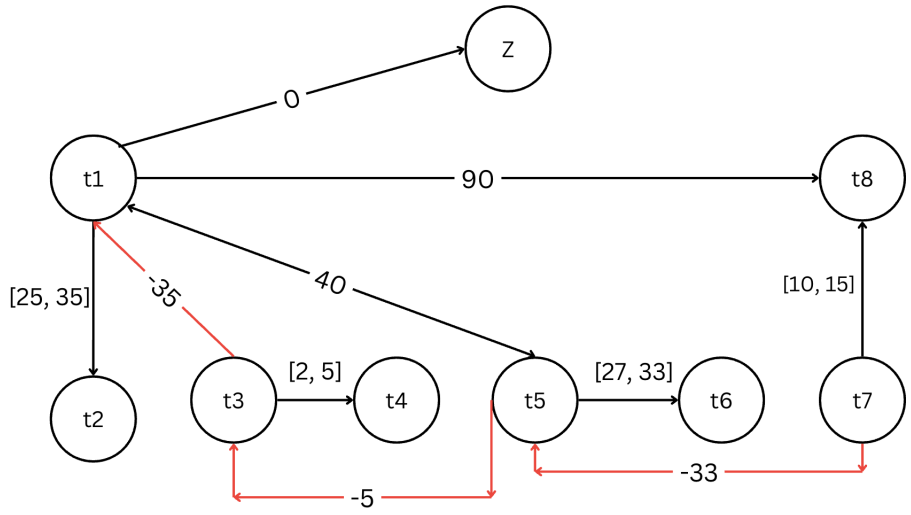

Figure 2.4: ESTNU example

execution is uncontrolled, an external environment handles observations. The algorithm then looks for activation time points with no outgoing negative edges, except the time points already executed, and marks them as `enabled`. The algorithm iterates till all time points execute. If the process succeeds, the algorithm returns a function that maps time points to real values [Hunsberger and Posenato, 2024b]. The benefit of the `RTE⋆` algorithm is that it propagates the information as soon as a CTP executes. Thus, it does not wait for the worst-case scenario to proceed, enabling efficient scheduling in polynomial time. While executing, the `RTE⋆` algorithm keeps an `rte_data` parameter, which includes a current propagated schedule `rte_data.f` and a current timestamp `rte_data.now`. The schedule maps each executed time point to its corresponding execution time, with executed time points as keys and their execution times as values.

Example 2.2.6 *We present the execution of the `RTE⋆` algorithm for Example 2.2.5. The execution begins with Z as it must execute before any other time point. At timestamp 0, `RTE⋆` executes Z. Then, t_1, becomes enabled. The execution of t_1 updates the upper bound for t_5 to 40 and t_8 to 90. The algorithm then identifies that t_2 must occur within the time window [25, 35]. Since external events govern contingent time points, `RTE⋆` remains idle until t_2 is observed. Due to the wait edge, t_3 must wait until t_2 executes.*

Assuming that t_2 occurs at timestamp 29, t_3 is enabled and executes at 29. Then, the next constraint dictates that t_4 must execute within [31, 34]. If t_4 executes at timestamp 34, t_5 waits and executes at 40. Then, t_6 is constrained to the interval [67, 73]. Suppose t_6 executes at timestamp 67; in that case, t_7 is enabled and executes at 68. t_8 must be executed within [78, 83]. Finally, if t_8 executes at timestamp 79 — within its upper bound of 90 — the algorithm successfully

completes its execution. The parameters `rte_data.now` *and* `rte_data.f` *are demonstrated in Table 2.1.*

Timestamp (`rte_data.now`)	Execution	Propagated Schedule (`rte_data.f`)
0	Z executed	$Z:0$
0	t_1 executed	$Z:0, t_1:0$
29	t_2 executed	$Z:0, t_1:0, t_2:29$
29	t_3 executed	$Z:0, t_1:0, t_2:29, t_3:29$
34	t_4 executed	$Z:0, t_1:0, t_2:29, t_3:29, t_4:34$
40	t_5 executed	$Z:0, t_1:0, t_2:29, t_3:29, t_4:34, t_5:34$
68	t_6 executed	$Z:0, t_1:0, t_2:29, t_3:29, t_4:34, t_5:34, t_6:64$
68	t_7 executed	$Z:0, t_1:0, t_2:29, t_3:29, t_4:34, t_5:34, t_6:64, t_7:64$
79	t_8 executed	$Z:0, t_1:0, t_2:29, t_3:29, t_4:34, t_5:34, t_6:64, t_7:64, t_8:79$

Table 2.1: Execution trace of the `RTE⋆` with `rte_data.now` and `rte_data.f`.

Probabilistic Simple Temporal Networks (PSTNs)

STNUs' contingent edges require defining lower and upper bounds for job durations. However, in the factory settings, these bounds are often unavailable. Instead, planners typically use historical data to estimate the mean and standard deviation of job durations. As a result, it is more natural to represent job durations as probability distributions rather than fixed bounds.

Probabilistic Simple Temporal Networks (PSTNs) are used to model unbounded distributions. Unlike bounded contingent links in STNUs, PSTNs represent each contingent link as a random variable with a specified probability density function [Fang et al., 2014]. This probabilistic representation enables the incorporation of uncertainties inherent in real-world manufacturing processes, making PSTNs a suitable model for factory scheduling challenges.

Example 2.2.7 *Consider again a scenario where Anna needs to commute from Figure 2.2 to understand PSTNs intuitively. However, now the bus and train routes are subject to unpredictable traffic delays. The bus ride duration follows a normal distribution with a mean of 30 minutes and a variation of 25 minutes ($\mathcal{N}(30, 25)$). The train ride has a duration drawn from $\mathcal{N}(30, 9)$. Given the probabilistic nature of travel times, Anna knows that delays are inevitable in some cases, potentially*

causing her to miss the meeting. This problem can be modeled using PSTNs in Figure 2.5, which helps analyze and adjust her schedule to balance minimizing total travel time and maximizing the probability of arriving on time for the meeting.

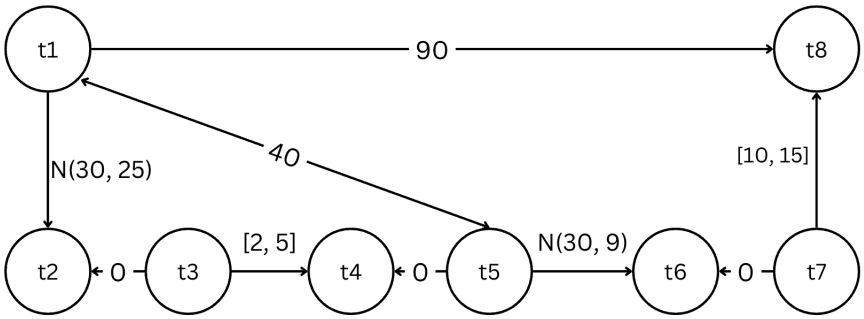

Figure 2.5: PSTN example, inspired by [Gao et al., 2020]

Execution Strategies of PSTNs

In a PSTN, underlying job distributions are unbounded. Referring back to the traffic example in Figure 2.5, fully controlling the network might be impossible since travel delays are not bounded. Instead of guaranteeing successful real-time execution, the model aims to balance execution efficiency with maximizing the probability of success. A PSTN must be approximated to an ESTNU to achieve this, which captures the maximum probability mass. This means that all contingent edges' approximated lower and upper bounds must represent the highest possible proportion of the probability distribution. A recent paper [Hunsberger and Posenato, 2024c] presents the following algorithm `buildApproxSTNU`. It takes a PSTN as the input and executes the following steps:

1. Approximates the PSTN as an STNU, capturing most of the probability distribution. All lower and upper bounds of contingent edges are assigned as $exp(\mu - 3.3 \cdot \sigma)$ and $exp(\mu + 3.3 \cdot \sigma)$ respectively, where μ is the mean of the log-normal probability distribution on the edge, and σ is its standard deviation. This captures 99.96% of the distribution.

2. Finds all SRN cycles in the obtained STNU. If no SRN cycles are present, the STNU is DC, and the process can be finished.

3. Attempts to resolve each SRN cycle by adjusting the bounds on the participating contingent links, making them more restrictive while preserving as much probability mass as possible from their respective probabilistic durations using a non-linear optimizer from `MatLab`. It determines new bounds for the contingent links that maximize the captured joint probability mass

while ensuring the SRN cycle remains non-negative [Hunsberger and Posenato, 2024c]. If no contingent links are in the SRN cycle, the algorithm cannot proceed, as it is not possible to adjust the durations of ordinary edges. If the solver fails to determine a new set of bounds for the contingent links needed to resolve the SRN cycle, the algorithm as a whole terminates unsuccessfully. Otherwise, new bounds are determined and pasted into STNU.

4. If all SRN cycles are resolved, the obtained STNU is DC and approximates the initial PSTN. The DC STNU and the probability mass obtained are saved.

The algorithm results in `status`, which contains the following fields:

1. **finished**: A Boolean value indicating whether the approximation function was completed successfully. If `false`, the PSTN is considered non-convertible to a DC STNU.

2. **exitFlag**: If `exitFlag` is greater than 0, the PSTN is convertible to a DC STNU. Otherwise, it is not.

3. **probabilityMass**: If the PSTN is convertible, this field represents the probability mass captured by the resulting DC STNU. The algorithm maximizes the probability mass.

4. **approximatingSTNU**: If the PSTN is convertible, this field contains the DC STNU to which the PSTN was transformed.

The implementation of this algorithm is publicly available in the CSTNU-tool [Posenato, 2022].

Although DC-checking combined with the `RTE`$\star$ has been applied to the SR-CPSP/max in [van den Houten et al., 2024], its application to production planning remains underexplored. Job durations also do not have defined bounds in the defined integrated production planning problem. Thus, instead of using STNUs, PSTNs are more suitable for the DSM-Firmenich problem. PSTNs have not yet been utilized in production planning. Therefore, we identify a research gap in applying recent advancements in the PSTN literature to the defined integrated production planning problem, where jobs follow the SRPSP/max pattern, but also include deadlines and inventory management.

2.3 Approaches for production planning and scheduling

There are multiple modeling approaches for production planning, with the most popular being Mixed-Integer Linear Programming (MILP) [Guzman et al., 2022]. A linear objective function, linear constraints, and integer variables characterize MILP [Castillo et al., 2011]. During the presolve phase, MILP solvers analyze whether they can tighten bounds or strengthen inequalities to reduce the number of variables and constraints. After presolve, combinatorial optimization algorithms,

e.g., local search [Crama et al., 2005] or branch and bound [Clausen, 1999], are applied to find the optimal solution [Huang et al., 2021].

Multiple MILP models exist to represent production planning. For instance, a case study in Algeria used a MILP model for multi-stage optimization in the agri-food supply chain, demonstrating significant performance gains over traditional planning heuristics [Bekrar et al., 2012]. Similarly, researchers have created efficient MILP-based models for multi-product, multi-stage continuous plants [Aguirre et al., 2018], addressing complexities such as demand uncertainty and the price elasticity of demand. Additionally, MILP models, incorporating costs, demand, and supply uncertainties, have outperformed deterministic approaches [Hosseini-Motlagh et al., 2021]. Researchers proposed a MILP model in agriculture and solved it using a CPLEX optimizer, effectively reducing production costs [Bayá et al., 2022]. Furthermore, a MILP model for a multi-product environment demonstrated its effectiveness in addressing real-sized problem instances [Belil et al., 2018]. These studies highlight MILP's versatility and effectiveness in optimizing production planning across diverse applications.

In recent years, Constraint Programming (CP), initially developed within artificial intelligence, has proven highly effective for solving production sequencing problems [Pinedo, 2022]. A key feature of CP is constraint propagation, which reduces the search space by inferring new constraints from existing ones [Rossi et al., 2008].

CP supports global constraints—complex constraints that capture common patterns in real-world problems and include specialized propagation algorithms to enhance computational efficiency [Van Hentenryck and Saraswat, 1996]. CP also utilizes interval variables, whose domains consist of time intervals [Laborie et al., 2012]. These variables effectively represent factory jobs, capturing a start time, job duration, and completion time.

Researchers proposed CP and MILP models for the distributed flexible job shop scheduling problem, showing that the CP model outperforms the MILP formulation [Meng et al., 2020]. Similarly, the CP model for parallel machine scheduling surpassed the results of MILP and genetic algorithm models by leveraging interval variables [Eray Cakici and Akdemir, 2024].

While researchers have extensively utilized MILP and CP models in production planning optimization, we found no research exploring their application to the defined integrated production planning problem. In this problem, each product consists of jobs following the SRCPSP/max pattern. The problem also involves additional complexities, such as deadline constraints and inventory management. The objective of DSM-Firmenich's problem is to maximize profit instead of minimizing makespan.

While the MILP model proposed in [Belil et al., 2018] incorporates inventory considerations, its primary focus is minimizing inventory levels. In contrast, in the DSM-Firmenich factory scenario, the factory aims to maintain a sufficient product inventory level. Multiple studies have explored inventory management within supply chains [Vicente, 2025] [Aktas and Temiz, 2020], where inventory is maintained

at a desired level. However, the supply chain context differs from the defined integrated production problem, where each job follows the SRCPSP/max schedule.

Thus, we identify a research gap in integrating inventory management, deadline constraints, and profit maximization into existing production planning models.

Chapter 3

Research questions

In this section, we present the main research question of this work. Then, we list the sub-questions that help answer the main question. The main goal of this work is to assist DSM-Firmenich in tactical and operational level decisions: offline scheduling, online scheduling, and optimization for the *defined integrated production planning problem*.

> *Research question: How can Probabilistic Simple Temporal Networks (PSTNs) be effectively utilized to maximize expected profit in the defined integrated production planning problem?*

Uncertainties in job durations, modeled as stochastic processing times, are inherent in real-world production systems. Proactive scheduling approaches aim to anticipate and mitigate these uncertainties by creating robust schedules that incorporate buffer times or flexible constraints. In contrast, reactive scheduling dynamically adjusts schedules in response to real-time deviations, providing planners with a more flexible and adaptive approach. Hybrid methods combine proactive and reactive strategies to balance efficiency and adaptability.

Simple Temporal Networks with Uncertainty (STNUs) combined with Partial Order Schedules (POS) have shown promise in optimization [van den Houten et al., 2024]. However, STNUs require bounded values, which may not fully capture the complexities of the defined integrated production planning. Probabilistic Simple Temporal Networks (PSTNs), which use probability distributions instead of fixed bounds, better align with the DSM-Firmenich case study. Since PSTNs have not yet been explored for production planning, we aim to investigate their potential for dynamically optimizing production schedules under uncertainty.

A PSTN needs to be complemented with a POS due to the presence of resource constraints, similarly to [van den Houten et al., 2024]. Thus, offline scheduling and online execution are required to answer the research question.

> *Sub-question 1: How can computational models like Mixed Integer Linear Programming (MILP) and Constraint Programming (CP) be*

utilized to model and optimize the deterministic version of the integrated production planning problem?

MILP and CP have been widely applied in production planning, including supply chain optimization, distributed flexible job shop scheduling, and parallel machine scheduling. While MILP is the more commonly used approach, CP has demonstrated superior performance in certain complex scheduling problems, highlighting its potential as an alternative.

We focus on modeling the DSM-Firmenich case with all relevant constraints, incorporating dynamic inventory management and demand forecasting – an aspect not extensively explored in the literature.

Using the CP and MILP models, we also evaluate whether CP can provide greater efficiency than MILP in solving the integrated production planning problem.

Sub-question 2: Offline Scheduling: How can a CP/MILP model be converted into a POS and a PSTN?

CP/MILP deterministic models produce a fixed schedule. However, for a PSTN representation, a POS must define the order of resource usage to ensure that the resource capacity constraints are met. Prior research has explored converting a Stochastic Resource-Constrained Project Scheduling Problem with Time Lags (SR-CPSP/max) into a POS [van den Houten et al., 2024]. However, the DSM-Firmenich case introduces additional complexity, incorporating optional orders—an aspect not yet mapped to POS in the literature.

Furthermore, existing work [van den Houten et al., 2024] focuses on a POS combined with STNUs, whereas the defined integrated production planning problem requires a PSTN representation to address our core research question. Thus, we investigate how to construct a PSTN while complementing it with a POS structure.

Sub-question 3: Online Execution: How do we adjust the real-time execution algorithm (`RTE*`) for the defined problem, given a constructed PSTN?

The original `RTE*` algorithm determines whether execution is successful or results in failure. However, in the DSM-Firmenich factory setting, production does not halt entirely even if a scheduling step fails. Instead, the factory adjusts the schedule to maximize expected actual profit dynamically. If execution were to stop at the first failure, the realized profit from already produced products could be significantly lower than expected.

By modifying the schedule in response to failures, the factory can continue production and potentially achieve a higher profit. This makes the original `RTE*` unsuitable for the defined integrated production planning problem. We propose an extension of `RTE*` with a recovery mechanism that enables adaptive rescheduling in case of failure, ensuring a more resilient and profitable execution strategy.

Sub-question 4: Optimization: How can the construction of a POS influence the expected actual profit?

Our approach complements PSTNs with a POS. The structure of the POS directly impacts how the PSTN is constructed, influencing its edges and values. As a result, different POS configurations lead to variations in the expected actual profit, which can either increase or decrease based on the flexibility and robustness of the generated PSTN.

Since the objective is to maximize profit, identifying an optimal POS that ensures a well-structured PSTN is crucial. To our knowledge, no existing research has explored the direct relationship between a POS and a PSTN. We address this gap by investigating whether a certain POS construction can lead to a higher expected actual profit, thereby enhancing production planning efficiency under uncertainty.

Chapter 4

Problem definition

This chapter explores the integrated production planning problem and the methods used to model it. First, we outline the DSM-Firmenich factory workflow and the key assumptions about its operations in Section 4.1. We then introduce a Mixed-Integer Linear Programming (MILP) model in Section 4.2, followed by a Constraint Programming (CP) model in Section 4.3. The objective is to develop a structured representation of the problem that can be efficiently solved using advanced algorithms.

4.1 Production planning: DSM Firmenich case

Production planning is the process of developing a detailed plan that specifies the anticipated production levels over a sequence of future periods within a defined timeframe, known as the planning horizon [Thomas and McClain, 1993]. DSM-Firmenich, a global leader in health, nutrition, and bioscience, faces the challenge of optimizing production plans. Within the factory, multiple orders must be managed under limited resource capacity, where each order can be represented as a list of products (product types). Every order has a specified deadline by which all its products must be produced, and each product is associated with a profit value. The total profit of an order is calculated as the sum of the individual profits from each product included in the order.

Each product consists of a series of unit operations called "jobs". These jobs are connected through precedence constraints, which define temporal dependencies between their execution times. These dependencies are quantified using a parameter called lag. For example, the start time of job k can be at most lag time units after the start time of job m if lag represents the maximum time lag. Conversely, if lag represents the minimum time lag, the start time of job k must be at least lag time units after the start time of job m.

Each job has specific resource requirements, and multiple resources are available, enabling jobs to run in parallel, thereby reducing overall production time and optimizing resource utilization.

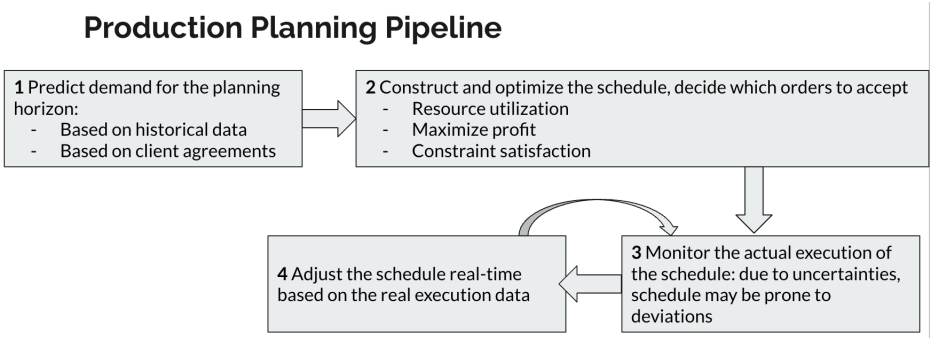

Figure 4.1: Four stages of production planning

Production planning at DSM-Firmenich spans multiple decision-making levels: strategic, tactical, and operational, each addressing distinct aspects of factory operations. Strategic decisions involve investments in equipment, product portfolio management, and customer portfolio strategy. This is done using market trends and historical data from the factory. Tactical decisions involve making a forecast for the planning horizon and constructing a schedule based on predicted demand. Operational decisions include optimizing the schedule and schedule adjustments due to uncertainties in job durations. In biomanufacturing, factors such as environmental conditions and the quality of raw materials influence job durations. The common practice by DSM-Firmenich is to use average durations from historical data. However, when executing the schedule in real-time, deviations can occur, requiring planners to adjust schedules dynamically.

The part of the pipeline of the planner can be seen in Figure 4.1. The pipeline consists of four stages. The first stage involves demand prediction by the planners, which refers to the tactical decision. The next stage focuses on offline scheduling, meaning that the products are scheduled without considering uncertainty. The final two stages, monitoring and adjusting the schedule, address the online part of the pipeline. They account for deviations arising from uncertainty in job durations. Offline scheduling, online scheduling, and optimization correspond to operational decisions.

An offline schedule generates an initial predicted profit. After executing the online schedule, the expected actual profit is calculated by summing the profit from all produced products.

The primary goal of the production planning model is to maximize the expected actual profit by fulfilling as many high-profit products as possible while ensuring all orders are completed on time, required constraints are satisfied, and resources are utilized efficiently.

Lastly, DSM-Firmenich maintains product inventory at a safety stock level set by the factory to increase flexibility in meeting order demands. This inventory ensures that predicted demand can always be met. Maintaining excessive inventory is generally undesirable in biomanufacturing. Many products have limited shelf

life; prolonged storage can lead to quality degradation, reducing their efficacy and value. Thus, inventory must be replenished only when requested.

We refer to the described production planning process as the *defined integrated production planning problem* further in the report.

We aim to support planners at the tactical and operational decision levels, including constructing and optimizing their schedules. Since scheduling is an NP-hard problem, advanced optimization algorithms can be employed. Offline scheduling, in particular, can be leveraged by mathematical models and advanced algorithms. The factories can use advanced algorithms during the planning stage to better estimate the time needed for a set of orders. During online scheduling, algorithms help adjust the schedule if a disruption occurs.

The interviews with DSM-Firmenich revealed that their scheduling approach does not account for the fact that certain product combinations may lead to more efficient schedules. We use advanced algorithms that consider such product combinations, possibly achieving higher profit and utilizing resources effectively.

We make the following assumptions about the *defined integrated production planning problem* in the models:

1. Each order contains products in the standard quantity. If an order contains more than the standard quantity of a product, this order is duplicated in the input.

2. The system categorizes orders as *required* or *optional* to address inventory replenishment. The model must produce required orders, while optional orders are not obligatory but desirable for maintaining inventory levels. If the factory capacity permits, the model can produce optional orders to replenish inventory. Optional orders generate profit, and accepting more of them increases the factory's overall profitability.

3. In the defined integrated production planning problem, profits are given per product. Our model assumes that the profit of an order is the sum of the profits of all products included in that order.

One can use the constructed models specifying the following input:

1. Define the orders and job structure. This involves specifying orders with their deadlines for the planning horizon. Each order consists of a list of products, and each product has a corresponding list of jobs. Define the time lags between jobs, their expected durations, and the required resources. The model automatically calculates the demand for each product based on the order deadline. For example, if order o has a product p with deadline t, the demand for p at timestamp t increases by 1.

2. Input the available resources and their capacities.

3. Plan inventory replenishment by creating optional orders for the products. The model prioritizes fulfilling mandatory orders first. If constraints allow, the model schedules optional orders, leading to inventory replenishment.

The output is an optimized schedule for all jobs that maximizes profit and the ability to monitor daily inventory levels. This approach ensures efficient planning while considering both mandatory and optional orders.

Example 4.1.1 *Deterministic example of the problem*
Suppose the factory receives two orders: o_1 and o_2 described in Table 4.1. There are two products: p_1 and p_2 summarized in Table 4.2. Two resources are available r_1 and r_2. Capacity of r_1 is 5, and capacity or r_2 is 10.

Order	**Type**	**Products**	**Deadline (days)**	**Profit ($)**
o_1	Required	p_1, p_2	14	20
o_2	Optional	p_2	14	20

Table 4.1: Details of orders o_1 and o_2

Product	**Jobs**	**Initial Inventory**
p_1	j_1, j_3	0
p_2	j_2	0

Table 4.2: Details of products

The resource requirements and durations for each job are shown in Table 4.3.

Job	**$\mathbb{E}$(duration)**	**Standard deviation**	**Resource r_1**	**Resource r_2**	Temporal
j_1	9	0.9	5	0	EMPTY
j_2	5	0.5	5	0	EMPTY
j_3	2	0.2	0	6	start(j_3) - start(j_1) $\leq$ 12

Table 4.3: Details of Jobs

Firstly, we allocate the required order o_1. For p_1, two jobs are required, and j_3 must start at least 12 days after j_1. Thus, the only possible start time for j_3 is day 12 to ensure it is completed by the deadline of 14. Job j_1 can start immediately at 0, finishing by day 9. Job j_2 can then start at day 9. Since j_1 and j_2 share the same resource (r_1) and cannot run in parallel, they must be scheduled sequentially.

There is no capacity left for the optional order o_2 because resource r_1 is fully occupied from day 0 to day 14. Hence, the maximum profit in this case is 20. The schedule is visualized in Figure 4.2.

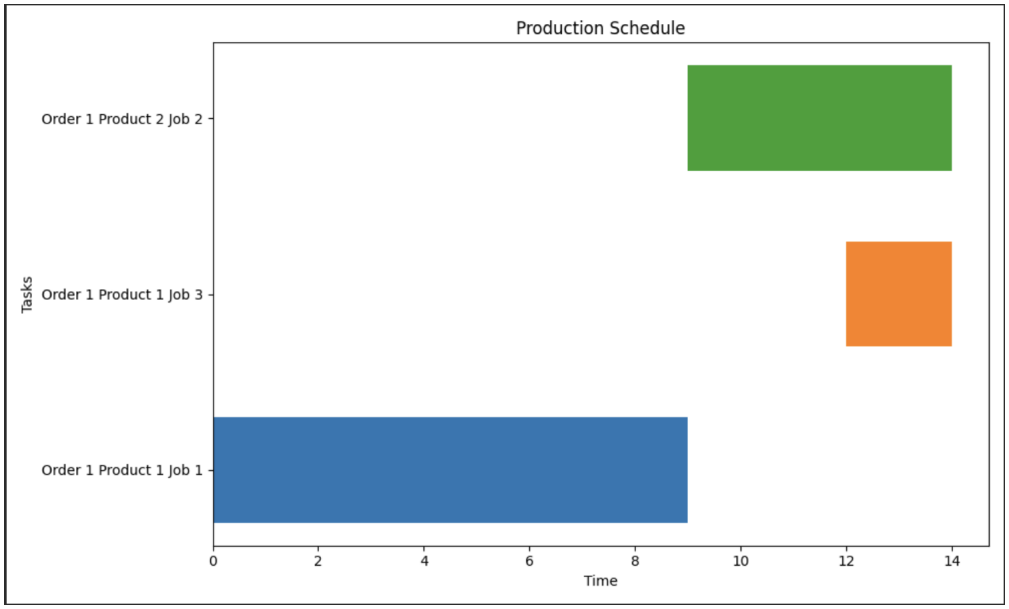

Figure 4.2: Visualization of the problem example schedule

4.2 MILP model

Below, we provide a mathematical definition of the defined integrated production planning problem.

Indices, Sets, and Parameters

The **indices** identified are:

- $i \in \{1, \ldots, n_o\}$: Represents one of the orders that should be completed.
- $j \in \{1, \ldots, n_p\}$: Represents one of the products.
- $k \in \{1, \ldots, n_j\}$: Represents one of the jobs.
- $l \in \{1, \ldots, n_r\}$: Represents one of the resources available.
- $t \in \{0, \ldots, n_t\}$: Represents the time (day) in the planning horizon.

Parameters

Based on the indices, we define the following **data**:

- $P(i)$: Set of products for order i.
- $J(j)$: Set of jobs for product j.
- $deadline_i \in \mathbb{Z}$: Deadline of order i.

- $profit_i \in \mathbb{Z}$: Profit value of order i.
- $required_i \in \{0, 1\}$: Boolean indicator showing if order i is obligatory for production (1) or not (0).
- $inventory_j^0 \in \mathbb{Z}$: Initial inventory level for product j.
- $dem_j^t \in \mathbb{Z}$: Demand for product j at timestamp t.
- $\rho_k^l \in \mathbb{Z}$: Resource l requirements for job k.
- $duration_k \in \mathbb{Z}$: Duration of job k.
- $capacity_l \in \mathbb{Z}$: Capacity of resource l.
- $temporalConstraints$: Set of precedence relationships ($pred$, lag, $succ$), where $pred$ and $succ$ are jobs, while $lag \in \mathbb{Z}$.
- $M \in \mathbb{Z}$: Large integer to model if clauses linearly.

Decision Variables

We identify the following **decision variables**:

- $inv_j^t \in \mathbb{Z}$: Inventory level of product j at timestamp t.
- $s_{ijk}^t \in \{0, 1\}$: Boolean variable, equal to 1 if job $k \in J(j)$ for product $j \in P(i)$ for order i is started exactly at timestamp t, and 0 otherwise.
- $e_{ij}^t \in \{0, 1\}$: Boolean variable, equal to 1 if product $j \in P(i)$ for order i is finished exactly at timestamp t, and 0 otherwise.
- $y_i \in \{0, 1\}$: Boolean variable, equal to 1 if order i is accepted, and 0 otherwise.

Objective function

The **objective function** is to maximize the profit:

$$\text{Maximize:} \quad \sum_i y_i \cdot profit_i \tag{1}$$

Constraints

Construct decision variables:

$$inv_j^t = BooleanVar \quad \forall t, \forall j$$

$$s_{ijk}^t = BooleanVar \quad \forall t, \forall i, \forall j \in J(i), \forall k \in K(j)$$

$$e_{ij}^t = BooleanVar \quad \forall t, \forall i, \forall j \in J(i)$$

$$y_i = BooleanVar \quad \forall i$$

If the order is required, it must be accepted:

$$y_i \geq required_i, \quad \forall i$$

If the order is accepted, all corresponding jobs must be started within the planning horizon:

$$\sum_t s_{ijk}^t \geq y_i, \quad \forall i, j \in J(i), k \in K(j)$$

If the order is accepted, all products must be finished within the planning horizon. Each product can be finished only once.

$$\sum_t e_{ij}^t = y_i \quad \forall i, j \in J(i)$$

We connect the start time of each job within a product to the end time of the product's production:

$$\sum_t t \cdot e_{ij}^t \geq \sum_t t * s_{ijk}^t + duration_k - M * (1 - y_i) \quad \forall i, j \in J(i), k \in K(j)$$

We update the inventory according to demand:

$$inv_j^t = inv_j^{t-1} + \sum_i e_{ij}^t - demand_j^t, \quad \forall j, t > 0$$

$$inv_j^0 = inventory_j^0, \quad \forall j$$

$$inv_j^t \geq 0, \quad \forall j, t$$

All produced products must finish before their deadline:

$$\sum_t t \cdot e_{ij}^t + duration_k \leq deadline_i + M * (1 - y_i), \quad \forall i, j$$

All time lags must be respected:

$$\sum_t t \cdot s_{ijk}^t + lag_{km} \leq \sum_t t * s_{ijm}^t + M * (1 - y_i)$$

$$\forall i, j \in J(i); (k \in K(j), lag, m \in K(j)) \in temporalConstraints$$

The resource capacity cannot be exceeded:

$$\sum_i \sum_{j \in J(i)} \sum_{k \in K(j)} \sum_{\tau = t - duration_k + 1}^{t} \rho_k^l \cdot s_{ijk}^\tau \leq capacity_l + M * (1 - y_i), \quad \forall l, t$$

4.3 CP model

Sets and indices

- $O = \{1, 2, \ldots, n_o\}$: Set of n_o orders.
- $P = \{1, 2, \ldots, n_p\}$: Set of all possible n_p product types.
- $J = \{1, 2, \ldots, n_j\}$: Set of all possible n_j jobs.
- $R = \{1, \ldots, l, \ldots, n_r\}$: Set of resources.
- $T = \{0, \ldots, t, \ldots, n_t\}$: Set of time units (days).
- $P(i)$: Set of products for order i.
- $J(j)$: Set of jobs for product j.
- $S(k)$: Set of successors of job k.
- i: standard index for order
- j: standard index for product ‘
- k: standard index for job
- l: standard index for resource
- t: standard index for time

Parameters

- $deadline_i \in \mathbb{Z}$: Deadline for order i (day).
- $profit_i \in \mathbb{Z}$: Profit for order i.
- $required_i \in \{0, 1\}$: Binary variable (1) if order i is required, (0) otherwise.
- $inventory_{j0} \in \mathbb{Z}$: Inventory level for product j at the beginning of planning.
- $dem_{jt} \in \mathbb{Z}$: Demand for product j at the time t (the demand is increased only at orders' deadlines).
- $duration_k \in \mathbb{Z}$: Duration of job k.
- $\rho_k(l) \in \mathbb{Z}$: Amount of resource l required by job k.
- $lag_{km} \in \mathbb{Z}$: Time lag between jobs k and m.
- $capacity_l \in \mathbb{Z}$: Capacity of resource l.

Decision Variables

- x_{ijk}: (optional) interval CP variable for job k for product j for order i. It represents the interval of time during which ijk happens, with a duration $duration_k$. The optimizer should assign a start time and end time for this interval variable, which is notated with $startOf$ (start time of the job) and $endOf$ (end time of the job). If no interval is available, it stays empty.
- inv_{jt}: Inventory level of product j at time t
- q_{jt}: Quantity of product j produced at time t
- y_i: Binary variable, 1 if order i is completed, 0 otherwise

Objective Function

$$\text{Maximize} \sum_{i \in O} y_i \cdot profit_i$$

Constraints

Interval Variables

$$x_{ijk} \in \begin{cases} \text{IntervalVar}(i, j, k, duration_k) & \text{if } required_i = 1 \\ \text{OptionalIntervalVar}(i, j, k, duration_k) & \text{if } required_i = 0 \end{cases}, \quad \forall i \in O, j \in P(i), k \in P(j)$$

Inventory Balance

$$inv_{jt} = \begin{cases} inventory_{j0} & \text{if } t = 0 \\ inv_{j,t-1} + q_{jt} - dem_{jt} & \text{if } t > 0 \end{cases}, \quad \forall j \in P, t \in T$$

$$inv_{jt} \geq 0, \quad \forall j \in P, t \in T$$

Production Quantities

If the order is accepted, we check if the maximum end time of the jobs involved in each product's production is t.

$$q_{jt} = \sum_{i \in O \text{ if } y_i = 1} \left[\max_{k \in J(j)} \{\text{endOf}(x_{ijk}) = t\} \right], \quad \forall j \in P, t \in T$$

Task Scheduling

Presence Constraint: Order i is accepted only if all job intervals involved in order i are present.

$$y_i = \left(\bigwedge_{j \in P(i), k \in J(j)} \text{presenceOf}(x_{ijk}) \right), \quad \forall i \in O \text{ if } required_i = 0$$

Deadline constraint: If the order is accepted, it must be completed before the deadline.

$$\text{if } y_i = 1, \max_{k \in J(j)} \{\text{endOf}(x_{ijk}) \mid j \in P(i)\} \leq \text{deadline}_i, \quad \forall i \in O$$

Precedence Constraints

$$\text{if } y_i = 1, \text{startOf}(x_{ijk}) + lag_{km} \leq \text{startOf}(x_{ijm}), \quad \forall i \in O, j \in P(i), k \in J(j), m \in S(k)$$

Resource Capacity

$$\sum_{i \in O \text{ if } y_i = 1} \sum_{j \in P(i)} \sum_{k \in J(j)} \text{Pulse}(x_{ijk}, \rho_k(l)) \leq capacity_l, \quad \forall l \in R$$

Chapter 5

PSTN Stochastic Approach

This chapter outlines a method using Probabilistic Simple Temporal Networks (PSTNs) to deal with the defined integrated production planning problem. This method aims to develop a scheduling system that maximizes expected profit while respecting job dependencies, resource constraints, and deadlines under job duration uncertainty. Furthermore, the method adjusts the execution schedule online instead of strictly following a predefined schedule. We present an overview of the method in Section 5.1. Then, we explain the offline part of the method in Sections 5.2, 5.3, and 5.4. In these sections, we answer Sub-question 2, explaining how to convert a Partial Order Schedule (POS) into a PSTN. Lastly, we present a novel real-time execution approach applicable for the DSM-Firmenich scenario in Section 5.5, answering Sub-question 3.

5.1 Method overview

The method consists of two main components: offline and online execution. The offline phase creates a Partial Order Schedule (POS) over the given time horizon, ensuring it meets all constraints and can execute without conflicts given deterministic job durations. The online phase, on the other hand, is responsible for executing the schedule in real time. It continuously monitors job execution and adapts to deviations caused by uncertainties in job durations. Figure 5.1 shows the whole PSTN-based method.

The offline process begins by modeling the instance as deterministic, with fixed job durations. We first solve this deterministic instance using a Constraint Programming (CP) or a Mixed-Integer Linear Programming (MILP) solver. The solution provides an objective value (profit), a schedule for the deterministic scenario, and a set of selected orders for production.

Next, we apply the resource alignment algorithm (see Section 2.2.2) to the generated schedule, which produces resource chains. We construct a PSTN using the data it needs and invoke the `buildApproxSTNU` function, explained in Section 2.2.2. This function returns three key outputs: (i) the status of the approximation,

(ii) an approximating Simple Temporal Network with Uncertainty (STNU), and (iii) the probability mass associated with the approximation.

If the approximation is successful and the probability mass meets the acceptance criteria, we convert the approximating STNU into an Extended STNU (ESTNU). Subsequently, we start the online execution by running the Real-Time-Execution (`RTE*`) algorithm, which enables online scheduling by considering real job durations rather than their deterministic estimates. This step ultimately provides the realized profit of the factory.

However, if the approximation fails or the probability mass is too low, we modify the original deterministic instance. Adjustments may include removing certain products or modifying deterministic job durations. The process then returns to the deterministic modeling step and iterates until a satisfactory solution is found.

The objective of the method is to maximize the actual expected profit, defined based on the products produced.

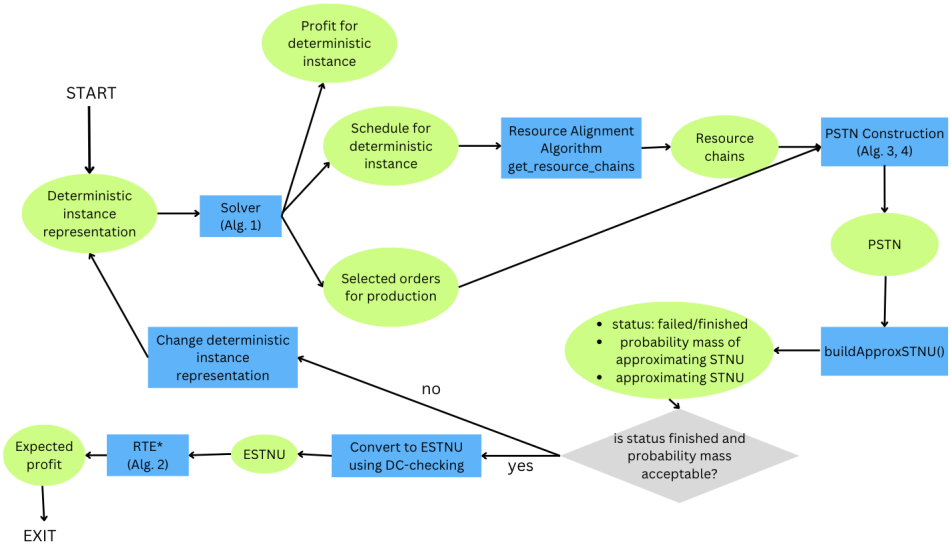

Figure 5.1: Method Overview

5.2 Deterministic Approximation

By problem definition, the factory receives both mandatory and optional orders. We must decide which orders $\mathcal{O}$ should be accepted for the planning horizon and which should be canceled. Additionally, we need to determine the sequence in which resources are utilized to process these orders.

We first solve a deterministic instance of the problem, where job durations are fixed. Let x represent an instance. We fix durations using the `SORU-H` robust

Algorithm 1: Construct a Partial Order Schedule

```
1 Function solve_deterministic_instance(instance, quantile):
      // Construct deterministic instance
2     deterministic_instance = SORU_H(instance, quantile)
      // Solve deterministic version of the problem
3     result, schedule ← solve(deterministic_instance)
4     if result ≠ None then
          // Construct resource chains
5         resource_chains ← get_resource_chains(schedule)
6         return result.objective_value, schedule, resource_chains
7     return None
```

optimization approach introduced in Section 2.2.1. This algorithm takes one approximating sample, representing a quantile of the distribution.

Solving deterministic x yields a deterministic schedule that only includes the accepted orders $\mathcal{O}_{accepted} \subseteq \mathcal{O}$. We apply the resource-chaining procedure from this schedule to determine how resources are allocated.

The objective of solving the deterministic instance is to construct a Partial Order Schedule (POS) that maximizes the expected actual profit.

We formalize this process in Algorithm 1. The function takes a problem instance and attempts to solve its deterministic variant.

In line 2, we determine the job durations for the deterministic instance. In line 3, the problem is formulated and passed to a solver. The solver outputs the solution status and the corresponding schedule if the problem is solved.

In line 4, we check whether the solver successfully found a solution. If a solution is found, we proceed to construct resource chains using `get_resource_chains` in line 5 (refer to Section 2.2.2). Algorithm 1 then returns the deterministic profit estimate $f^*(x)$, the generated schedule, and the resource chains.

If the problem is deemed unsolvable, the function instead returns `None`, indicating that no valid schedule could be found.

5.3 PSTN construction

The next step of the offline part of the pipeline is constructing a PSTN network by creating nodes and edges representing the problem.

We utilize the schedule obtained by solving the deterministic version of the problem in Algorithm 1 to determine which nodes to include. The obtained schedule only includes accepted orders, which are incorporated into the PSTN. If an order is rejected in the deterministic model, it is likewise excluded from the probabilistic model.

We represent all problem constraints by using edges. As a result, the following

structure is created below. The indices and sets used are identical to the indices and sets in the MILP and CP models (see Sections 4.2 and 4.3).

Nodes:

- EXECUTION_START: a node representing the beginning of the planning at timestamp 0.
- $\forall i \in O, j \in P(i), k \in J(j)$: we add $\texttt{START}_{ijk}$ and $\texttt{END}_{ijk}$: two nodes for each job in each product in each order, representing their start and end times, respectively.

Edges:

- **Ordinary Edges:**
  - An edge that represents a deadline constraint:

$$\begin{gathered}\forall i \in O, j \in P(i), k \in J(j) \\ \texttt{EXECUTION\_START} \xrightarrow{deadline_i} \texttt{END}_{ijk}\end{gathered} \tag{5.1}$$

    In the defined integrated production planning problem, every order has a deadline. Such an edge represents Equation 5.2.

$$\begin{gathered}\forall i \in O, j \in P(i), k \in J(j) \\ \texttt{END}_{ijk} - \texttt{EXECUTION\_START} \leq deadline_i\end{gathered} \tag{5.2}$$

  - An edge corresponding to the time lag constraints, connecting $\texttt{START}_{ijm}$ to $\texttt{START}_{ijk}$ $\forall i \in O, j \in P(i), k \in J(j), m \in S(k)$:

$$\texttt{START}_{ijm} \xrightarrow{-lag_{km}} \texttt{START}_{ijk} \tag{5.3}$$

    The edge can be translated to the following equation:

$$\begin{gathered}\forall i \in O, j \in P(i), k \in J(j), m \in S(k) \\ \texttt{START}_{ijk} - \texttt{START}_{ijm} \leq -lag_{km}\end{gathered} \tag{5.4}$$

  - For the resource constraints, we utilize resource chains constructed in Algorithm 1. Resource chains are a list of predecessor-successor pairs, where a predecessor and a successor are jobs for a certain product in an order. The jobs are indexed by three integers: order, product, and job. We add the ordinary edge $\forall (ijk, ijm) \in$ *resource chains* between $\texttt{END}_{ijk}$ and $\texttt{START}_{ijm}$:

$$\texttt{START}_{ijm} \xrightarrow{0} \texttt{END}_{ijk} \tag{5.5}$$

    We can represent this edge as an equation below:

$$\begin{gathered}\forall_{(ijk,ijm) \in resource\_chains} \\ \texttt{END}_{ijk} - \texttt{START}_{ijm} \leq 0\end{gathered} \tag{5.6}$$

– We ensure that all start nodes execute after the EXECUTION_START so that the EXECUTION_START executes at timestamp 0. A new ordinary edge is introduced from the $\texttt{START}_{ijk}$ to the EXECUTION_START:

$$\begin{gathered}\forall i \in O, j \in P(i), k \in J(j) \\ \texttt{START}_{ijk} \xrightarrow{0} \texttt{EXECUTION\_START}\end{gathered} \tag{5.7}$$

These edges are transformed into the following equations:

$$\begin{gathered}\forall i \in O, j \in P(i), k \in J(j) \\ \texttt{EXECUTION\_START} - \texttt{START}_{ijk} \leq 0\end{gathered} \tag{5.8}$$

- **Contingent Edges:**

  – An edge representing the probabilistic (uncertain) duration $\forall i \in O, j \in P(i), k \in J(j)$, connecting $\texttt{START}_{ijk}$ to $\texttt{END}_{ijk}$:

$$\texttt{START}_{ijk} \xrightarrow{\mu=\mathbb{E}(duration_{ijk}),\sigma=\sigma_{ijk}} \texttt{END}_{ijk} \tag{5.9}$$

  where μ is the mean of the job duration and σ refers to the standard deviation of the job duration.

Given the Partial Order Schedule (POS) constructed in Algorithm 1, we create the PSTN using the algorithm parse_instance_to_pstn, which adds all the nodes and ordinary and contingent edges described above. We then add the resource chain edges using the add_resource_chains algorithm. The detailed implementations of parse_instance_to_pstn and add_resource_chains can be found in Appendix A, in Algorithms 3 and 4, respectively.

Example 5.3.1 ***PSTN Construction Example***

I utilize the example 4.1.1 introduced earlier to illustrate a PSTN construction, modifying the deadline of o_1 from 14 to 20. The adjusted schedule appears in Figure 5.2. All jobs are scheduled, and both orders are accepted. Algorithm 1 returns the objective value 40 and the schedule visualized in Figure 5.2. The resource usage is visualized in Figure 5.3. The following resource chains are constructed and returned: $(o_1, p_1, j_1) \rightarrow (o_2, p_2, j_2)$, $(o_1, p_1, j_1) \rightarrow (o_1, p_2, j_2)$, $(o_2, p_2, j_2) \rightarrow (o_1, p_2, j_2)$.

This example produces the following Graph 5.4. In the PSTN, EXECUTION_START is connected to all finish nodes, representing deadline constraints. The time lag between $o1, p1, j1$ and $o1, p1, j3$ translates to the edge between their corresponding start nodes. Resource chains edges are added between the start and finish nodes with a value 0. Lastly, contingent edges are added to represent job duration distributions.

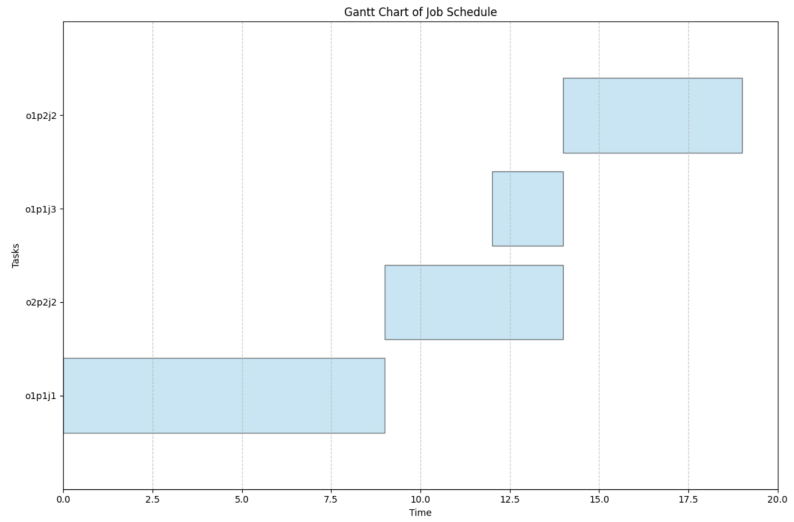

Figure 5.2: Visualization of the problem example schedule

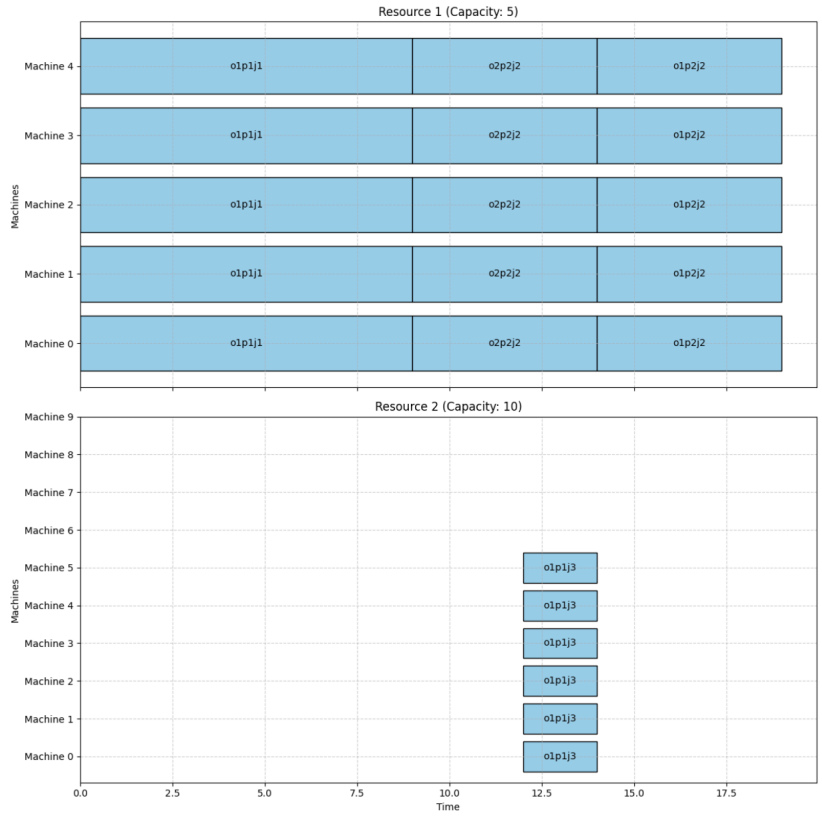

Figure 5.3: Use of resources (upper: resource 1; lower: resource 2;)

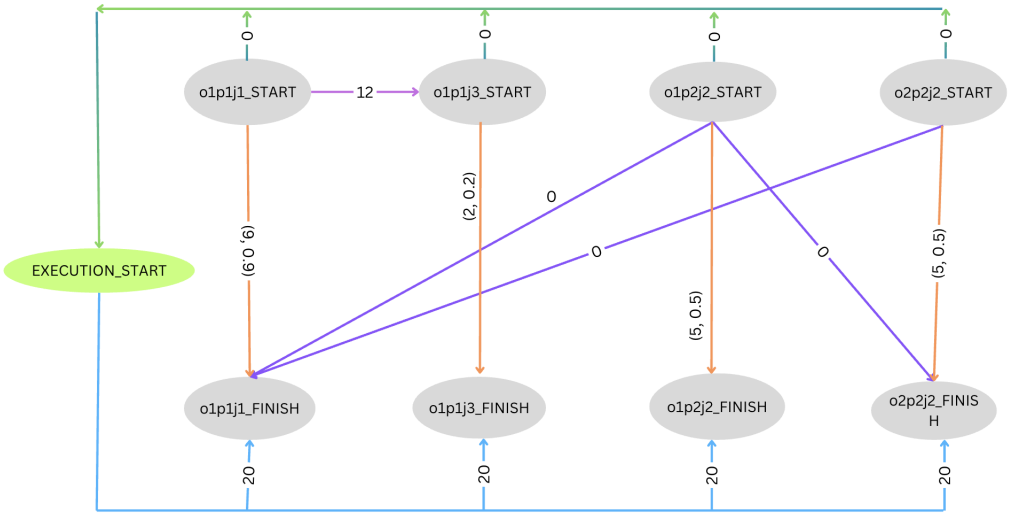

Figure 5.4: PSTN Construction Example

5.4 PSTN approximation

The PSTN Construction (Section 5.3) corresponds to the PSTN Construction Algorithm step in Figure 5.1. The next step is to approximate the received PSTN with a Dynamically Controllable Simple Temporal Network with Uncertainty (DC STNU) using the `buildApproxSTNU` algorithm [Hunsberger and Posenato, 2024c] explained in Section 2.2.2. The `PSTN` obtained from the PSTN Construction is used as input for the `buildApproxSTNU`. Then, if the `buildApproxSTNU` succeeds, we save the obtained approximating DC STNU (`approximatingSTNU`) and its probability mass.

5.5 Real time execution

The DC STNU received from the PSTN Approximation maximizes the probability mass of the PSTN. We have operated offline until now, meaning we use predefined job durations and translate the obtained Partial Order Schedule (POS) into the PSTN. We want to employ the `RTE`$\star$ algorithm to execute jobs in real-time using their actual durations. However, the `RTE`$\star$ can only be successfully applied to an Extended Simple Temporal Network with Uncertainty (ESTNU). Consequently, we convert the received DC STNU into an ESTNU using the DC checking algorithm [Hunsberger and Posenato, 2024a].

A received ESTNU approximates the PSTN; hence, it does not capture the entire range of the underlying duration distribution. As a result, there exists a risk that actual job durations may fall outside the predefined bounds, causing the `RTE`$\star$ to fail.

This scenario is particularly relevant in manufacturing environments, where strict adherence to predefined time bounds may not always be feasible. Instead of halting the entire production line and performing a full rescheduling, an alternative approach is to allow the factory to continue operations by selectively canceling jobs that contribute to disruptions. This strategy has the potential to enhance overall efficiency while still maintaining acceptable levels of profitability.

We present a novel approach to executing `RTE*`, allowing the execution to proceed even if the `RTE*` fails. We use the output parameters of the `RTE*` – the schedule `rte_data.f` and the current timestamp `rte_data.now` (see Section 2.2.2) to recover from failure.

We propose the following procedure: run the classical `RTE*` algorithm while it does not fail. As soon as it fails, we introduce modifications to the schedule.

Suppose we have an ESTNU for a given factory instance. During the execution of `RTE*`, some nodes become active while the algorithm propagates information to other nodes. If the `RTE*` halts at a timestamp t, we can categorize the jobs into three distinct sets:

- Completed Jobs (J_C): a set of jobs already started and finished before timestamp t.
- Ongoing Jobs (J_O): a set of jobs that have already started before timestamp t but have not ended.
- Pending Jobs (J_P): a set of jobs that have not started/finished before timestamp t.

Regarding the factory, completed jobs are unaffected by the crash and do not require any rescheduling as they have already finished. Ongoing jobs, on the other hand, are disrupted and thus need to be adjusted.

Completed jobs are found by checking which jobs are fully present in the current schedule (`rte_data.f`) – the schedule of jobs until t. This means that each job with both `START` and `END` in the schedule is present. We also find pending jobs; their `START` nodes are present in the schedule, while the `END` nodes are absent. The detailed algorithm for finding completed and pending jobs can be found in Appendix A (Algorithm 5).

We can translate the obtained sets of jobs into the product sets:

- Completed Products (P_C): a set of products that have been produced, meaning that all jobs required to make a product belong to J_C.
- Ongoing Products (P_O): a set of partially started products that have not been produced yet. This means that while some of the jobs required for the product belong to J_C, some jobs remain in J_O or J_P.
- Pending Products (P_P): a set of products that have not started/finished before timestamp t. This means that none of the jobs required for products are part of J_C or J_P.

Completed products are found by checking if all jobs required for a product are present in the completed jobs. Each product in this category is already produced; thus, we can collect the profit: $current\_profit = \sum_{p \in P_C} profit_p$.

If the product's jobs are partially present in completed jobs or ongoing jobs, the product is added to the ongoing products. The detailed implementation for finding current profit, completed products, and ongoing products can be found in Algorithm 6 in Appendix A.

The DC STNU received from the `buildApproxSTNU` needs to be modified. At the time t, when the `RTE*` fails, the DC STNU includes all initial jobs from POS. A new DC STNU must include only jobs for products that are not produced or canceled. The following modification steps are introduced:

Step 1: Removal of completed products

Since completed products are finished before the `RTE*` failure, they are already counted for the profit, and we do not schedule them again. Thus, we remove all jobs required for completed products from the DC STNU, as well as their corresponding edges.

Step 2: Removal of ongoing products

Ongoing products were halted between the execution stages; thus, they could not be entirely produced. We cancel such products, and they do not contribute to the profit. We represent it in the DC STNU by removing all nodes and edges from the DC STNU that correspond to jobs required to produce ongoing products.

This step is strict, as not necessarily all of the ongoing products disrupted the `RTE*` algorithm. However, the current version of the algorithm does not support finding exactly the jobs that caused the disruption. Furthermore, if the new DC STNU consists of only a subset of jobs for certain products, the time lag edges of such jobs need to be updated. This happens because some jobs have already been executed, but they still might have time lag dependencies on pending jobs. This can cause the newly adjusted STNU to no longer be DC. We propose modifying time lag edges in Appendix C – a technique that should be explored in the future. In the current version of the algorithm, only products that have not yet started (their jobs are not completed or ongoing) remain in the DC STNU.

Step 3: Modification of deadlines

`EXECUTION_START` now marks the beginning of the new execution, requiring an adjustment of all deadlines accordingly. The `RTE*` halted at timestamp t, meaning that some time has already passed in the execution. If we use initial deadlines, the pending products have more time to complete. An alternative would be to set `EXECUTION_START` to t. However, due to the current implementation of the `RTE*` algorithm, it always starts with a timestamp 0. We decide to stay consistent with the logic and mark `EXECUTION_START` as 0. Thus, we need to alter the ordinary edges for the deadlines to be the following:

$$\texttt{EXECUTION\_START} \xrightarrow{deadline_k - t} \texttt{EVENT\_END}_k \tag{5.10}$$

which represents that

$$\texttt{EVENT\_END}_k - \texttt{EXECUTION\_START} \leq deadline_k - t \tag{5.11}$$

We now present Algorithm 2, which describes the online execution of a DC STNU and utilizes functions presented above. The Algorithm takes three input parameters: (i) a DC STNU generated by the `buildApproxSTNU`, (ii) the instance containing job and product information, and (iii) the sample, which provides actual job durations during execution. Since we aim to track the actual profit obtained in real-time, the profit is stored at the beginning (line 2).

The DC STNU is converted to an ESTNU to execute the `RTE*` algorithm. The algorithm attempts to run `RTE*` (line 4), returning a `result` (indicating success or failure) and `rte_data` (data generated by `RTE*`). If execution fails, the algorithm then identifies all jobs and products that have started and finished and calculates the current profit (lines 7-9, refer to Algorithms 5 and 6 for detailed implementation).

Subsequently, the DC STNU is altered in lines 10-11. The STNU is then converted back into an ESTNU, and the process repeats until execution is either successfully completed or fails due to infeasibility. This iterative approach ensures that the schedule adapts dynamically to real-time uncertainties while maintaining temporal consistency and maximizing the expected profit. The algorithm operates fully online, meaning that no offline rescheduling is needed, as the resource chains are reused from the initially constructed POS. As a result, the runtime of the proposed `RTE*` remains polynomial, rather than exponential as in traditional rescheduling approaches. This efficiency makes the method particularly suitable for factory settings, where real-time responsiveness is essential and computational delays can lead to costly downtime or suboptimal decisions.

Algorithm 2: Novel RTE* mechanism with recovery: Directly modifying ESTNU after RTE* failure

```
1  Function run_estnu_online (stnu: STNU, sample, instance):
2      profit ← 0
3      estnu ← dc_check(stnu)
4      result, rte_data = rte_star(estnu, sample)
5      while (!result) do
6          t ← rte_data.now
7          J_C, J_O ← find_current_jobs(rte_data.f, estnu)
8          profit_r, P_C, P_O ←
             find_corresponding_products(J_C, J_O, instance)
9          profit = profit + profit_r
10         remove_nodes_and_edges(P_C ∪ P_O, stnu)
11         modify_deadline_edges(t, stnu)
12         estnu ← dc_check(stnu)
13         result, rte_data = rte_star(estnu, sample)
14     return profit
```

Chapter 6

Evaluation

This chapter evaluates a novel Probabilistic Simple Temporal Network (PSTN)-based approach for the integrated production planning problem. Section 6.1 describes the process of generating the instances used in the experiments. Then, we aim to verify the correctness and compare Mathematical Integer Linear Program (MILP) and Constrained Programming (CP) models in Section 6.2. This evaluation aims to understand whether one model is preferred over the other based on the objectives and scalability. Section 6.3 provides insight into which deterministic instances to use in order to maximize expected profit. Lastly, we evaluate the PSTN-based method compared to a simple proactive method based on the expected profit in Section 6.4.

6.1 Instance creation

We use the data from the benchmark test sets J10[1] for the single-mode Resource-Constrained Project Scheduling Problem with Time Lags (RCPSP/max) to construct experimental instances. We chose this RCPSP/max benchmark due to the assumption that all jobs follow the RCPSP/max pattern in the defined integrated production planning problem. Each of these instances represents a single product type in a standard quantity. Each test set consists of twelve jobs, where ten jobs have a specific duration, and two jobs serve as sink and source with a duration of 0. We assume that ten jobs for each product can represent a real product in the DSM-Firmenich factory. The instances contain the necessary job information, including job durations and resource usage, which we use without modification. We merge the resources defined in the individual instances and adjust accordingly to simulate a shared resource environment between product types. We also exclude infeasible instances to ensure the experiments' feasibility.

The procedure for creating experimental instances involves the following steps:

[1] https://www.wiwi.tu-clausthal.de/en/ueber-uns/abteilungen/betriebswirtschaftslehre-insbesondere-produktion-und-logistik/research/research-areas/project-generator-progen/max-and-psp/max-library/single-mode-project-duration-problem-rcpsp/max

1. Determining the number of orders and the number of product types per order: the number of orders and the number of product types per order are predefined. Each order is randomly assigned a set of product types, constrained by a predetermined maximum number of product types per order. This approach aims to replicate a factory's operations, where a fixed number of orders are processed, each containing specific products. Since the recipe for each product is predefined and remains unchanged within an order, we utilize product data from the J10 benchmark.

2. Combining and adjusting resources: each instance initially contained exactly five resources with defined capacities due to the J10 benchmark data. We fix the total number of resources at five, matching the original setup. We track the minimum demand for each resource across all products to establish a reasonable lower bound for each resource capacity. Once the demands are collected, we determine the final resource capacities using a randomized approach, setting each resource's availability between five and six times the observed minimum requirement. This adjustment ensures that resources are adequately allocated across products, balancing sufficient availability and limited capacity. Doing so prevents the unrealistic scenario where all jobs can run in parallel, effectively enforcing resource constraints and ensuring the scheduling problem remains feasible.

3. Determining the deadline based on the makespan of its constituent product types: we define the makespan of a product type as the minimum time required to complete all its associated tasks. Deadlines are assigned randomly within a range, where the lower bound is defined as the maximum makespan of the products within the order, extended by 20%. The upper bound is then set as an additional 20% beyond this lower bound. For example, suppose the makespan is 100. Then, the lower bound for the deadline is extended by 20%, meaning that it is 120. The upper bound extends the lower bound by 20% leading to the value of 144.

   This deadline extension accounts for potential variations in job durations. If deadlines are too strict, even minor delays can result in infeasibility. We introduce this slack to enhance scheduling flexibility since deviations are common in the factory setting.

   The overall timespan for the experiment is then set as the maximum deadline across all orders, ensuring sufficient time for task scheduling.

4. We randomly assign each product's profit a value between 0 and 100 to simulate varying levels of importance and profit contribution. We assume that in the real factory setting, each product maintains a fixed profit regardless of the order it belongs to. Since the factory aims to maximize overall profit, a random assignment is sufficient for conducting experiments while preserving realistic variability.

5. We assume that the inventory level of each product is zero at the start of the planning. Although this may not reflect the actual conditions in the real factory, our primary goal is to evaluate the PSTN-based method, which operates independently of the initial inventory state. We leave a detailed analysis of inventory tracking and its impact on the model for future work.

6.1.1 Uncertainty in the instances

The instances given in the J10 benchmark are deterministic. However, the factory faces uncertainty in job durations. We transform the deterministic instances into stochastic ones using the following assumptions:

1. We set the expected job durations (μ) according to the initial durations from the J10 instance. Using these initial durations ensures that the instance remains feasible while respecting both time lags and resource constraints. The J10 benchmark is also widely recognized as a representative dataset for realistic RCPSP/max instances. Assuming that the integrated production planning problem reflects the RCPSP/max pattern, we proceed with the initial durations.

2. The standard deviation σ incorporates variability in job durations. We set it to the following values:

$$\begin{gathered} \sigma = \epsilon \cdot \sqrt{\mu} \\ \text{where } \epsilon \text{ influences noise level} \\ \epsilon \in 0.05, 0.1, 0.2, 0.25, 0.3, 0.4, 0.5, 0.7 \end{gathered} \tag{6.1}$$

   Higher values of standard deviation introduce greater uncertainty in job durations. We select ϵ values that represent both minimal and substantial deviations to evaluate the PSTN-based method's robustness under different uncertainty levels. Small standard deviations ensure that actual job durations remain close to their expected values, while larger standard deviations introduce significant variability. By testing multiple cases, we gain insights into how well the PSTN-based approach handles uncertainties in scheduling.

We assume that job durations follow a log-normal distribution. This choice is motivated by two key factors. First, the log-normal distribution only produces positive values, making it well-suited for modeling job durations, which cannot be negative. Second, our approach relies on the CSTNU tool for PSTN support, which exclusively supports log-normal distributions.

6.2 Comparing CP and MILP models to solve a deterministic problem

The primary goal of this experiment is to verify the correctness and compare the performance of the Constraint Programming (CP) and Mixed Integer Linear Pro-

gramming (MILP) models in solving a deterministic variant of the factory scheduling problem. Performance is evaluated based on scalability and feasibility.

Three key parameters are varied systematically to construct the instances:

1. Number of orders: the experiments consider scenarios with 3, 5, 10, 15, and 20 orders. This allows for assessing how increasing orders affect the models' performance, particularly regarding computation time and feasibility. We choose these numbers to reflect the potential demand of the factory, starting from a very small number of orders and increasing it.

2. Number of products per order: this parameter is set to 5, 10, 20, and 30, creating varying levels of complexity in the structure of individual orders. Higher values introduce greater diversity in product combinations and increase the instance size. This variation helps evaluate how the models handle increasing internal complexity within individual orders.

We randomly assign each order as *required* or *optional*. While we do not have access to the actual proportion of required versus optional orders in the factory, this randomized assignment allows us to evaluate the model's ability to handle both critical and flexible demands within a single planning instance.

The models under evaluation were introduced in Sections 4.2 and 4.3. For the CP model, we utilize *CP Optimizer*, a robust and widely recognized solver provided by IBM ILOG CPLEX Optimization Studio, [2] which recently demonstrated superiority over CPLEX across a set of benchmark scheduling problems [Naderi et al., 2023]. For the MILP model, we select the *Gurobi Solver*,[3] as it outperformed CPLEX Optimizer on multiple benchmarks [Anand et al., 2017]. We execute all the problem instances using the CP and MILP models and compare their performance, focusing on objective values and execution times. We also verify that the obtained solution satisfies all problem-related constraints.

The objective values obtained from the MILP and CP approaches are identical for all feasible instances, meaning that both models can find a feasible solution or result in a timeout. However, in the case of an infeasible problem, the CP model results in a timeout (set to one hour), whereas the MILP model identifies infeasibility. This suggests that both models are suitable when constraints are flexible, but if the problem is potentially infeasible, the MILP model is preferred. Furthermore, running both models in parallel could provide the optimal solution in the shortest possible time.

Figure 6.1 shows that the CP model consistently solves feasible instances faster than the MILP model in this application. Since the integrated production planning problem is NP-hard, the solving time increases exponentially with input size. Although the CP model's execution time appears to grow more rapidly, this effect stems from the one-hour timeout limit — MILP reaches the timeout for smaller

[2] https://www.ibm.com/docs/en/icos/22.1.0?topic=cp-optimizer

[3] https://www.gurobi.com/

instances, while CP manages to find an optimal solution. Despite this, both models still produce identical feasible solutions. However, the MILP model cannot always verify optimality due to the timeout. Overall, the CP model achieves lower solving times for feasible instances, confirming findings from prior research that highlight the efficiency of CP-based methods for scheduling and resource allocation problems [Meng et al., 2020, Eray Cakici and Akdemir, 2024]. In contrast, Figure 6.2 illustrates the results for infeasible problems, where the MILP model outperforms the CP model by successfully identifying infeasibility rather than timing out. Previous studies did not account for optional interval variables, which we use in our CP model, and these variables may contribute to the observed inefficiencies. This needs investigation in future research.

Given these results and the need for efficiency, we use both the MILP and CP models in subsequent experiments.

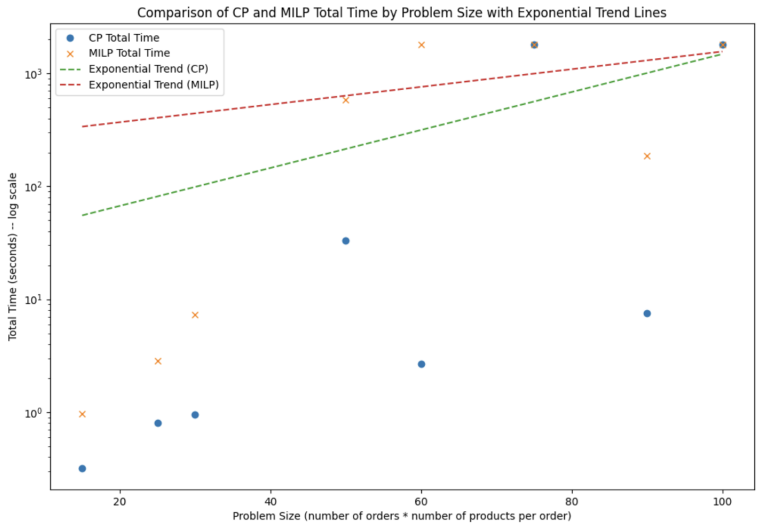

Figure 6.1: CP vs MILP execution times for feasible instances

6.3 Influence of POS on the expected actual profit

The experiment aims to determine how constructing deterministic instances for a Partial Order Schedule (POS) affects the expected actual profit. Since we construct a POS offline before execution begins, its structure can influence the order in which products are scheduled. This scheduling order can lead to either frequent infeasibilities or a smooth execution. As a result, the choice of job durations and constraints used in the deterministic instance can have a direct impact on the actual execution and, consequently, on the overall expected profit.

We conduct the experiments on seven specific instances of average size, poten-

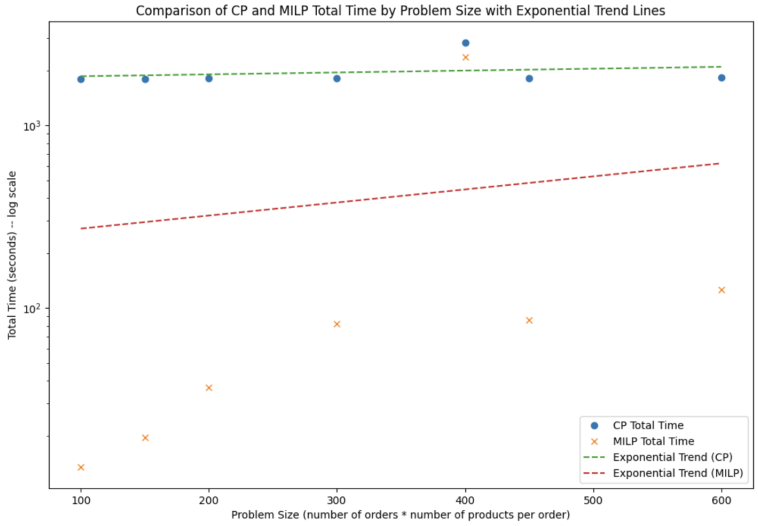

Figure 6.2: CP vs MILP execution times for infeasible instances

tially representing the factory's needs. Each instance contains ten orders, each consisting of five products. The instances differ in their job duration standard deviations, explained in Equation 6.1. Each instance contains exactly seven mandatory and three optional orders. This setup ensures that mandatory orders remain a priority. At the same time, the system retains the flexibility to reject optional orders if the deterministic job durations are too long, thus preserving feasibility by guaranteeing the completion of the most critical orders. Note that the current models do not support partial fulfillment of orders — if an order is rejected, all associated products are canceled.

Our proposed PSTN-based method involves solving a deterministic instance to obtain a Partial Order Schedule (POS). We address the following research question:

> *Sub-question 4: Optimization: How can the construction of a POS influence the expected actual profit?*

We approach this sub-question in two parts:

1. Using original constraints: we construct a POS using the original problem constraints while varying deterministic job durations.

2. Introducing flexible constraints: we modify the original constraints to incorporate soft deadlines (see the model introduced in Appendix D).

We construct deterministic instances using the state-of-art proactive method `SORU-H` for the RCPSP/max (see Section 2.2.1).

We modify the deterministic job durations as outlined in Section 5.2, varying the quantile parameter:

$$q \in 0.5, 0.75, 0.9 \tag{6.2}$$

Each q represents a quantile of the distribution. For example, for $q = 0.5$, the deterministic durations correspond to the median of the job duration distributions.

Altering the job durations in deterministic instances allows us to analyze how increasing conservatism during offline scheduling impacts the expected profit.

We simulate real-time execution by sampling each job's specified distribution. 100 samples are generated for the `RTE*` per instance. We report the probability mass that Dynamically Controllable (DC) Simple Temporal Network with Uncertainty (STNU) captures and the expected profit obtained for each quantile q. The expected profit is computed for each instance and averaged across all samples using Sample Average Approximation (SAA). We apply the SAA method to simulate multiple possible scenarios for the actual job durations.

In the second part of the experiment, we also relax the original strict deadline constraint by introducing soft deadlines. Instead of requiring the deterministic model to meet each order's deadline strictly, we allow for violations and penalize large lateness in the objective. The motivation behind this approach is that it can lead to more resource-efficient schedules and fewer disruptions when executing the schedule online. With soft deadlines, the schedule cannot fail due to deadline violations, leading to smoother execution. When using conservative deterministic job durations, actual execution times may be shorter in practice, allowing the system to still meet the original deadlines during the real-time execution. Moreover, late products can be used for inventory and future orders.

6.3.1 Results: Using original constraints while varying deterministic job durations

We solve each instance using different quantile values. Since some products in the optional orders may be rejected by the deterministic schedule, Table 6.1 reports the number of products accepted under this schedule. The table also includes the mean job durations computed using each quantile value (Mean Quantile Duration). When the standard deviation scales by $\epsilon = 0.05, 0.10, 0.20$, the resulting quantile durations are similar and become identical when rounded to integers. This suggests that the quantile value has almost no impact on the POS and the expected profit for these instances. However, for larger scales in standard deviation ($\epsilon > 0.20$), the quantile-based durations begin to diverge. This indicates that the choice of the quantile parameter may influence the expected profit in these cases.

We present the detailed real-time execution experiment results in Tables B.2.

Figure 6.3 illustrates the relationship between the quantile parameter q and the standard deviation in job durations. The highest expected profit is achieved at $q = 0.5$, where the deterministic durations match the median of the expected durations for the instances with the standard deviation factor $\epsilon < 0.7$. In these instances, most of the products accepted in the deterministic scenario are also accepted during the

Table 6.1: Summary of Quantile-Based Duration Sampling

Epsilon	Quantile	Mean Duration (initial instance)	Mean Quantile Duration	Products Accepted
0.05	0.50	4.741	5.688	50
0.05	0.75	4.741	5.766	50
0.05	0.90	4.741	5.837	50
0.10	0.50	4.741	5.685	50
0.10	0.75	4.741	5.841	50
0.10	0.90	4.741	5.985	50
0.20	0.50	4.741	5.670	50
0.20	0.75	4.741	5.985	50
0.20	0.90	4.741	6.285	45
0.25	0.50	4.741	5.659	50
0.25	0.75	4.741	6.054	50
0.25	0.90	4.741	6.436	45
0.30	0.50	4.741	5.645	50
0.30	0.75	4.741	6.121	45
0.30	0.90	4.741	6.587	45
0.40	0.50	4.741	5.612	50
0.40	0.75	4.741	6.249	45
0.40	0.90	4.741	6.890	40
0.50	0.50	4.741	5.571	50
0.50	0.75	4.741	6.368	45
0.50	0.90	4.741	7.192	40
0.70	0.50	4.741	5.466	50
0.70	0.75	4.741	6.575	45
0.70	0.90	4.741	7.783	35

real-time execution. As the quantile parameter increases, the deterministic model becomes more conservative, accepting fewer products. Nevertheless, the PSTN can schedule most of the accepted products deterministically, but this results in lower profit if fewer products are accepted during planning. This suggests that, for these instances, setting $q = 0.5$ is sufficient to maximize the expected profit while maintaining feasible execution under uncertainty. We also observe that for most cases, the expected profit is the same for all samples. This reflects that the approximated STNUs captured most of the distribution, and the `RTE`$\star$ algorithm never disrupts. For our last instance, with the standard deviation scale $\epsilon = 0.7$, while most products are accepted deterministically when $q = 0.5$ and $q = 0.75$, most products fail during the real-time execution. When taking $q = 0.9$, we are more conservative in planning, meaning fewer products are planned. However, that gives enough flexibility for the real-time execution, which leads to the expected profit increase.

Conclusion 1: The deterministic schedule significantly impacts the expected profit in real-time execution. The choice of the quantile parameter q, which determines the level of conservatism in the schedule, plays a crucial role in this outcome. A poorly chosen q can lead to overly optimistic or overly conservative schedules,

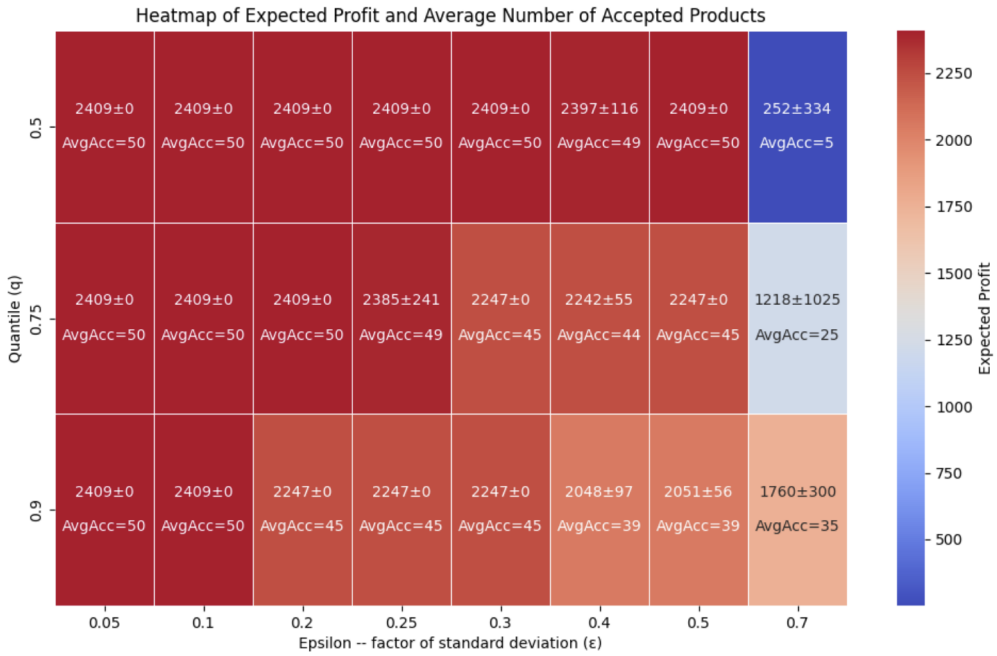

Figure 6.3: Combination of Standard Deviation and Quantile versus Expected Profit and Average Number of Accepted Products (reported for each instance)

reducing profitability. Therefore, tuning the quantile value appropriately for the given instance characteristics is essential. Our research suggests that using the median values when planning ensures optimal expected profit for jobs with low standard deviation. Taking more conservative durations is required for the jobs with higher deviation to ensure optimal expected profit. Tuning the deterministic durations ensures that the offline plan aligns well with the uncertainties present in the real world, ultimately leading to maximal expected profit.

6.3.2 Results: Using the soft deadline constraints while varying deterministic job durations

In our implementation of the real-time execution, strict adherence to hard deadlines may lead to unnecessary disruptions, especially under uncertainty. We explore soft deadline constraints, where lateness is penalized rather than strictly prohibited. The key hypothesis is that soft deadlines can reduce disruptions during real-time execution using `RTE`$\star$, improving the overall schedule robustness and expected profit.

Given that most of our benchmark instances exhibited limited disruptions (particularly when the uncertainty scale factor $\epsilon < 0.7$), we focus this analysis on a more uncertain scenario: the instance with standard deviation scale $\epsilon = 0.7$. In

Table 6.2: Expected profit and probability mass: PSTN approach with soft deadlines

q	ϵ	Expected Profit $f(x)$	Probability mass	Makespan t	Throughput $f(x)/t$	Average Accepted	Lateness
0.50	0.70	601.81 $\pm$852.45	0.17	130.15 $\pm$361.36	4.62	12	105.96
0.75	0.70	670.24 $\pm$873.549	0.19	125.59 $\pm$340.45	5.3	13	95.70
0.90	0.70	2326.61 $\pm$358.59	0.99	355.54 $\pm$21.04	6.54	49	165.98

Table 6.3: Expected profit and probability mass: PSTN approach with hard deadlines

q	ϵ	Expected Profit $f(x)$	Probability mass	Makespan t	Throughput $f(x)/t$	Average Accepted
0.50	0.70	251.78 $\pm$333.78	$6.89 \cdot 10^{-3}$	78.77 $\pm$12.33	3.19	5
0.75	0.70	1217.74 $\pm$1025.17	0.39	134.64 $\pm$303.25	9.04	25
0.90	0.70	1759.74 $\pm$299.59	0.99	185.02 $\pm$96.95	9.51	35

this experiment, we solve the deterministic version of the problem using the soft-deadline model proposed in Appendix D, incorporating different levels of conservatism through job durations derived from quantile values $q = 0.5$, 0.75, and 0.9.

As shown in Table 6.2, increasing the quantile value q leads to notable gains in performance across several metrics: expected profit, throughput, and the number of accepted products all improve. At the highest quantile level ($q = 0.9$), the soft deadline model accepts 49 out of 50 products and achieves a near-perfect probability mass ($P = 0.99$), demonstrating strong robustness under high conservatism. This flexibility, however, comes at the cost of increased average lateness (165.98), reflecting the trade-off inherent in relaxing strict deadline constraints.

In contrast, the results from the hard deadline model (Table 6.3) exhibit a more restrictive profile. At lower quantiles ($q = 0.5, 0.75$), significantly fewer products are accepted and disruptions occur more frequently, leading to reduced expected profit and throughput. While both models achieve high probability mass at $q = 0.9$, the deterministic schedule under hard deadlines accepts fewer products overall. This leads to lower expected profit, albeit with better adherence to deadlines.

These results underscore a key trade-off: soft deadlines promote higher profits, but result in deadline violations. Hard deadlines enforce stricter conditions during the real-time execution, but satisfy all initial constraints.

Conclusion 2: Incorporating soft deadlines into the scheduling framework leads to higher expected profits by allowing more products to be completed under uncertainty. However, this comes at the cost of increased average lateness, which may negatively impact customer satisfaction and potentially incur penalty costs. There-

fore, a nuanced strategy is required: manufacturers should differentiate between orders with strict contractual deadlines and those with more temporal flexibility, and integrate this distinction into the planning model to balance profitability with service-level commitments.

6.4 Evaluation of PSTN-based approach

The main research question of this work is:

> *Research question: How can Probabilistic Simple Temporal Networks (PSTNs) be effectively utilized to maximize expected profit in the defined integrated production planning problem?*

We answer this research question by introducing the novel methodology in Chapter 5. We want to evaluate the proposed method by comparing it to a proactive baseline – the `SORU-H` method. This evaluation is essential for understanding the practical potential of the PSTN-based approach within real-world factory settings.

The results for the PSTN-based method are drawn from Table B.2, while detailed results for the `SORU-H` baseline are presented in Table B.1.

We present a comparative analysis in Figure 6.4, where each instance is represented by two bars indicating the best expected profit obtained across all tested quantile values. The PSTN-based method consistently outperforms the proactive `SORU-H` baseline in terms of expected profit across all instances. In terms of makespan, both approaches yield comparable results for most instances, with a notable divergence in the final instance. There, the PSTN-based approach results in a longer makespan, which is accompanied by a higher number of accepted products. Interestingly, the PSTN-based method also achieves a shorter makespan in several cases, highlighting the potential efficiency of the `RTE*` execution algorithm. However, the underlying factors contributing to the observed makespan variations in the PSTN-based method require further investigation in future research.

Conclusion 3: The results support our central research question by demonstrating that PSTNs can effectively enhance production planning under uncertainty: the PSTN-based approach consistently achieves higher expected profit than the proactive `SORU-H` method.

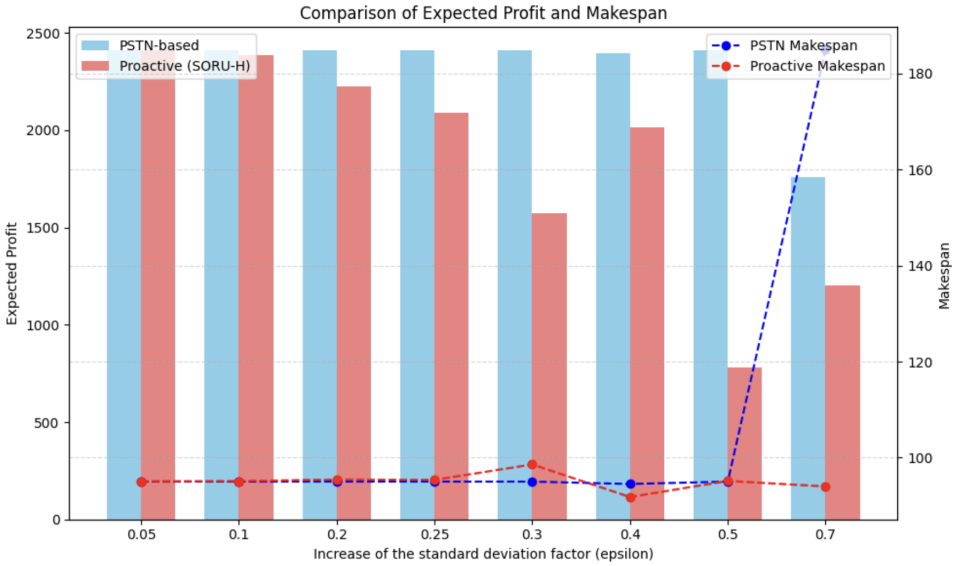

Figure 6.4: Comparison of proactive SORU−H and the PSTN method based on the expected profit and makespan

Chapter 7

Discussion

In this chapter, we analyze the results and assess the applicability of the Probabilistic Simple Temporal Networks (PSTN)-based approach to the defined production planning problem. Additionally, we discuss the limitations of our work and outline potential directions for future research.

> *Research question: How can Probabilistic Simple Temporal Networks (PSTNs) be effectively utilized to maximize expected profit in the defined integrated production planning problem?*

We propose a novel approach in Chapter 5 to address this research question, to maximize expected profit. We model the integrated production planning problem as a PSTN and apply the `RTE*` algorithm, which does not simply fail but reacts to the failures and adapts.

This work represents a significant step toward addressing the integrated production planning problem. While it does not capture many constraints of a real-world factory, it establishes a solid foundation that can be extended to incorporate additional complexities in future research. One particularly impactful constraint is the presence of time lags between jobs. Although the current model assumes a Resource-Constrained Project Scheduling Problem with Time Lags (RCPSP/max) setting, some jobs may also require constraints between their finish times or their start and finish times. For example, machine cleaning tasks often need to begin only after the completion of a specific job. Such requirements can be modeled as time lags between the end of one job and the start of another. However, since job finish times correspond to contingent nodes in the PSTN, introducing time-lag constraints involving finish times could increase the risk of execution-time disruptions. Currently, start-to-start time lags affect only activation nodes, which are deterministic and not subject to temporal uncertainty. These constraints are always satisfiable within PSTNs. In contrast, adding start-to-finish or finish-to-finish constraints could introduce potential infeasibilities, making it more challenging to maintain temporal consistency during execution. This issue similarly affects the proactive `SORU-H` approach. In this method, job start times are fixed in advance,

while job end times are subject to stochastic variability. Introducing additional time-lag constraints involving uncertain finish times may also lead to infeasible schedules under this approach. This raises the question of whether the PSTN-based methods can be effectively extended to accommodate more complex time-lag constraints while maintaining their advantages over proactive approaches.

We use eight exact instances that mimic factory conditions to simulate real-world scenarios. For these instances, the proposed PSTN-based method outperforms the proactive approach. While the proposed approach demonstrates promising results on the instances considered, real-world factory environments may present structural and operational variations. They may not reflect the tested instances. The PSTN-based method resulted in capturing 99% of the distribution when converting PSTNs into Simple Temporal Networks with Uncertainty (STNUs) in many scenarios. However, this probability might be lower for other instances with higher standard deviations. This can cause more failures in the Real-Time Execution (RTE$\star$), potentially lowering profit. As one of the causes of failure is violating a deadline, we propose a new model with soft deadlines, which penalizes lateness in Appendix D. This approach increases the expected profit while sacrificing the deadline constraint. For future research, we suggest tuning the importance of the lateness objective to meet the real factory needs.

Furthermore, in the RTE$\star$ recovery mechanism, we disrupt jobs in the execution process. We assume that resources are immediately available. However, in the DSM-Firmenich case, some machines require cleaning/idle time and cannot be used immediately. Deterministically, such disruptions can be included as additional jobs or time-lags. However, in real-time, such pauses can lead to longer makespan, as they were not encountered during planning, as the RTE$\star$ was disrupted unexpectedly. The factory would then need to fully re-plan with remodeling to achieve the optimal objective or sacrifice on the makespan.

The proposed RTE$\star$ algorithm is also strict, as we remove all products executing at the timestamp when the RTE$\star$ failed. This means we remove all partially produced products. Referring back to the proposed RTE$\star$ (see Section 5.5), if some jobs of the product are completed and some are pending, the product is still removed, although no jobs are ongoing. We propose removing only the products with ongoing jobs in Appendix C. A better alternative is to find specific jobs that caused RTE$\star$ disruptions and remove only products associated with those jobs. However, as the RTE$\star$ is rather novel, we could not find the literature around this issue. While finding the jobs that are disrupted is straightforward, future research can look into finding the job that caused disruptions.

Moreover, we chose not to resolve the deterministic schedule after the RTE$\star$ alterations due to the goal of making real-time execution fast. However, by resolving the deterministic model again with only products left in the schedule, we can further optimize the solution and reduce the total makespan, thus utilizing the resources more effectively. This alteration can be explored in future work to evaluate the trade-off between the time it takes to solve real-time execution with rescheduling and the expected profit with makespan.

Additionally, we calculate the expected total profit by summing up the individual earnings of the produced products. However, profits are typically tied to complete order fulfillment in a factory setting. Partial completion of an order may lead to reduced profit due to lower client satisfaction. Furthermore, the current model assumes constant product profits, whereas, in reality, profits can be dynamic and may vary depending on factors such as deadlines. Such aspects warrant further investigation in future work to understand their influence on the expected profit and make the problem reflect a real factory instance.

Beyond evaluating the methodology within the studied test instances, the proposed research also offers value across the broader decision-making hierarchy in manufacturing:

1. Tactical planning: By constructing a deterministic schedule before execution, planners gain valuable insights into short-term factory capacity and resource allocation. The framework also accommodates optional orders, which can be strategically used for inventory replenishment or demand smoothing. Unlike manual planning, in the current setup, planners only need to input the instance with the relevant parameters, and the model generates the optimal schedule. This approach offers a more optimal solution than manual scheduling (as the problem is NP-hard) and alleviates the time burden on planners by automating the scheduling process.

2. Operational decision-making: The `RTE`$\star$ mechanism enables rapid, on-the-fly adaptation of schedules in response to execution-time disruptions, with computational complexity bounded in polynomial time. This allows the system to proceed with job execution without waiting for pre-defined start times or initiating a rescheduling process.

These observations suggest that probabilistic temporal reasoning can be a powerful enabler across planning levels, provided that future work continues to bridge the gap between theoretical scheduling models and real-world operational constraints. Lastly, for the factory to fully leverage these techniques, the system needs to be developed into a user-friendly tool that creates an interface that allows planners to interact easily and use these advanced scheduling techniques in their daily operations.

The code and reproducibility instructions can be found in Appendix E.

Chapter 8

Conclusion

In this research, we tackled the challenges of optimizing production planning under uncertainty within the biomanufacturing context of DSM-Firmenich. Our focus was on enhancing decision-making at both tactical and operational levels through computational methods capable of handling complex scheduling constraints and variability in job durations.

To this end, we introduced the Integrated Production Planning Problem, which incorporates multiple production dimensions: order deadlines, inventory requirements, time-lag constraints, uncertain job durations, and resource limitations. We developed and compared Mixed-Integer Linear Programming (MILP) and Constraint Programming (CP) models to solve deterministic instances of this problem and proposed a novel probabilistic framework using Probabilistic Simple Temporal Networks (PSTNs) to address uncertainty.

Our approach consists of both offline optimization and online execution. In the offline phase, we convert schedules generated by MILP or CP into Partial Order Schedules (POS), which serve as the basis for constructing PSTNs. In the online phase, we introduced a real-time execution algorithm that extends the existing `RTE*` method with a recovery mechanism, enabling the system to adapt dynamically to deviations caused by uncertainty rather than failing outright.

Despite their clear suitability in addressing the uncertainty commonly faced in production environments, we could not find any prior application of PSTNs specifically in production planning. Furthermore, we found that the initial `RTE*` algorithm was not designed to account for real-world scenarios and required adjustments. Thus, our work fills a significant gap in the literature by introducing practical applications of PSTNs in production planning and addressing the limitations of existing real-time execution algorithms.

We demonstrated that both MILP and CP can effectively model the deterministic version of the integrated planning problem. When evaluating the models, we found that the CP model outperformed the MILP model regarding computational time for feasible scenarios. Both models produced identical feasible solutions. However, in cases where the problem was infeasible, the MILP model showed better perform-

ance, as the CP model failed to deliver solutions within the time limits, resulting in timeouts.

Additionally, we developed a method to transform a POS derived from a deterministic model into a PSTN that encapsulates all specified constraints. Our findings revealed that the quality of the POS significantly impacts the expected profit, emphasizing the importance of fine-tuning the deterministic problem to enhance outcomes. Job durations can be fine-tuned depending on the uncertainty level of the jobs. If the jobs are less uncertain, their predicted durations can lead to optimal schedules. Taking more conservative durations may result in higher expected profit if the job durations deviate. Constraints also impact the derived POS. For example, incorporating soft deadlines may lead to higher expected profits. Depending on client agreements and specific order requirements, the factory can strategically apply soft deadlines to some orders while using hard deadlines for others.

We utilized existing CSTNU tools to convert PSTNs into Dynamically Controllable (DC) Simple Temporal Networks with Uncertainty (STNUs) to enable real-time, uncertainty-aware scheduling. We then designed a reactive execution strategy that tolerates failures and adapts to changing conditions to maintain profitability. While our algorithm currently cancels all partially produced products at the moment of failure, we observed that it may be overly conservative. This can lead to unnecessary product cancellations and should be addressed in future research.

By combining deterministic optimization with probabilistic execution, our contributions provide both practical tools and theoretical insights for production planners. This approach reduces manual workload, improves resource utilization, and sustains high service levels — even amid unpredictable operational conditions. Ultimately, this research lays the groundwork for more adaptive and robust systems in uncertain production environments.

Bibliography

A. M. Aguirre, S. Liu, and L. G. Papageorgiou. Optimisation approaches for supply chain planning and scheduling under demand uncertainty. *Chemical Engineering Research and Design*, 138:341–357, 2018.

A. Aktas and I. Temiz. Multi-period mixed integer programming model for supply chain planning under safety stock. *Mersin University Journal of Maritime Faculty*, 2, 11 2020. doi: 10.47512/meujmaf.816402.

R. Anand, D. Aggarwal, and V. Kumar. A comparative analysis of optimization solvers. *Journal of Statistics and Management Systems*, 20(4):623–635, 2017.

T. Baar, P. Brucker, and S. Knust. *Tabu Search Algorithms and Lower Bounds for the Resource-Constrained Project Scheduling Problem*, pages 1–18. Springer US, Boston, MA, 1999. ISBN 978-1-4615-5775-3. doi: 10.1007/978-1-4615-5775-3_1. URL https://doi.org/10.1007/978-1-4615-5775-3_1.

G. Bayá, E. Canale, S. Nesmachnow, F. Robledo, and P. Sartor. Production optimization in a grain facility through mixed-integer linear programming. *Applied Sciences*, 12(16), 2022. ISSN 2076-3417. doi: 10.3390/app12168212. URL https://www.mdpi.com/2076-3417/12/16/8212.

A. Bekrar, D. Trentesaux, B. Beldjilali, et al. Multi-stage optimization in supply chain: An industrial case study. In *9th International Conference on Modeling, Optimization & SIMulation*, 2012.

S. Belil, S. Kemmoé-Tchomté, and N. Tchernev. Milp-based approach to mid-term production planning of batch manufacturing environment producing bulk products. *IFAC-PapersOnLine*, 51(11):1689–1694, 2018. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2018.08.213. URL https://www.sciencedirect.com/science/article/pii/S240589631831334X. 16th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2018.

M. E. Bruni, P. Beraldi, and F. Guerriero. *The Stochastic Resource-Constrained Project Scheduling Problem*, pages 811–835. Springer International Publishing,

Cham, 2015. ISBN 978-3-319-05915-0. doi: 10.1007/978-3-319-05915-0_7. URL https://doi.org/10.1007/978-3-319-05915-0_7.

E. Castillo, A. J. Conejo, P. Pedregal, R. Garcia, and N. Alguacil. *Building and solving mathematical programming models in engineering and science*. John Wiley & Sons, 2011.

J. Clausen. Branch and bound algorithms-principles and examples. *Department of computer science, University of Copenhagen*, pages 1–30, 1999.

Y. Crama, A. W. Kolen, and E. Pesch. Local search in combinatorial optimization. *Artificial Neural Networks: An Introduction to ANN Theory and Practice*, pages 157–174, 2005.

R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial intelligence*, 49(1-3):61–95, 1991.

I. K. Eray Cakici and M. Akdemir. Advanced constraint programming formulations for additive manufacturing machine scheduling problems. *Journal of the Operational Research Society*, 0(0):1–16, 2024. doi: 10.1080/01605682.2024.2382867. URL https://doi.org/10.1080/01605682.2024.2382867.

C. Fang, Y. Peng, and B. C. Williams. Chance-constrained probabilistic simple temporal problems. In *AAAI Conference on Artificial Intelligence*, 2014. URL https://api.semanticscholar.org/CorpusID:8423862.

M. Gao, L. Popowski, and J. Boerkoel. Dynamic control of probabilistic simple temporal networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(06):9851–9858, Apr. 2020. doi: 10.1609/aaai.v34i06.6538. URL https://ojs.aaai.org/index.php/AAAI/article/view/6538.

E. Guzman, B. Andres, and R. Poler. Models and algorithms for production planning, scheduling and sequencing problems: A holistic framework and a systematic review. *Journal of Industrial Information Integration*, 27:100287, 2022. ISSN 2452-414X. doi: https://doi.org/10.1016/j.jii.2021.100287. URL https://www.sciencedirect.com/science/article/pii/S2452414X21000844.

W. Herroelen and R. Leus. Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, 165(2):289–306, 2005. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2004.04.002. URL https://www.sciencedirect.com/science/article/pii/S0377221704002401. Project Management and Scheduling.

S.-M. Hosseini-Motlagh, M. R. G. Samani, and F. Abbasi Saadi. Strategic optimization of wheat supply chain network under uncertainty: a real case study. *Operational research*, 21(3):1487–1527, 2021.

L. Huang, X. Chen, W. Huo, J. Wang, F. Zhang, B. Bai, and L. Shi. Branch and bound in mixed integer linear programming problems: A survey of techniques and trends, 11 2021.

L. Hunsberger and R. Posenato. Speeding up the rul$^-$ dynamic-controllability-checking algorithm for simple temporal networks with uncertainty. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(9):9776–9785, Jun. 2022. doi: 10.1609/aaai.v36i9.21213. URL https://ojs.aaai.org/index.php/AAAI/article/view/21213.

L. Hunsberger and R. Posenato. Converting simple temporal networks with uncertainty into minimal equivalent dispatchable form. *Proceedings of the International Conference on Automated Planning and Scheduling*, 34(1):290–300, May 2024a. doi: 10.1609/icaps.v34i1.31487. URL https://ojs.aaai.org/index.php/ICAPS/article/view/31487.

L. Hunsberger and R. Posenato. Foundations of dispatchability for simple temporal networks with uncertainty. pages 253–263, 01 2024b. doi: 10.5220/0012360000003636.

L. Hunsberger and R. Posenato. Robust Execution of Probabilistic STNs. In P. Sala, M. Sioutis, and F. Wang, editors, *31st International Symposium on Temporal Representation and Reasoning (TIME 2024)*, volume 318 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:19, Dagstuhl, Germany, 2024c. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-349-2. doi: 10.4230/LIPIcs.TIME.2024.12. URL https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.TIME.2024.12.

J. Kleinberg and E. Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., USA, 2005. ISBN 0321295358.

A. J. Kleywegt, A. Shapiro, and T. Homem-de Mello. The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization*, 12(2):479–502, 2002. doi: 10.1137/S1052623499363220. URL https://doi.org/10.1137/S1052623499363220.

P. Laborie, J. Rogerie, P. Shaw, P. Vilím, and F. Katai. *Interval-Based Language for Modeling Scheduling Problems: An Extension to Constraint Programming*, pages 111–143. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-23592-4. doi: 10.1007/978-3-642-23592-4_6. URL https://doi.org/10.1007/978-3-642-23592-4_6.

M. Lombardi and M. Milano. A precedence constraint posting approach for the rcpsp with time lags and variable durations. In I. P. Gent, editor, *Principles and Practice of Constraint Programming - CP 2009*, pages 569–583, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-04244-7.

M. Lombardi, M. Milano, and L. Benini. Robust scheduling of task graphs under execution time uncertainty. *IEEE Transactions on Computers*, 62(1):98–111, 2013. doi: 10.1109/TC.2011.203.

M. Manzini and M. Urgo. Makespan estimation of a production process affected by uncertainty: Application on mto production of nc machine tools. *Journal of Manufacturing Systems*, 37:1–16, 2015. ISSN 0278-6125. doi: https://doi.org/10.1016/j.jmsy.2015.10.001. URL https://www.sciencedirect.com/science/article/pii/S0278612515000783.

L. Meng, C. Zhang, Y. Ren, B. Zhang, and C. Lv. Mixed-integer linear programming and constraint programming formulations for solving distributed flexible job shop scheduling problem. *Computers & Industrial Engineering*, 142:106347, 2020. ISSN 0360-8352. doi: https://doi.org/10.1016/j.cie.2020.106347. URL https://www.sciencedirect.com/science/article/pii/S0360835220300814.

P. Morris. A structural characterization of temporal dynamic controllability. In F. Benhamou, editor, *Principles and Practice of Constraint Programming - CP 2006*, pages 375–389, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-46268-2.

P. Morris. Dynamic controllability and dispatchability relationships. In H. Simonis, editor, *Integration of AI and OR Techniques in Constraint Programming*, pages 464–479, Cham, 2014. Springer International Publishing. ISBN 978-3-319-07046-9.

P. Morris and N. Muscettola. Temporal dynamic controllability revisited. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 3*, AAAI'05, page 1193–1198. AAAI Press, 2005. ISBN 157735236x.

P. Morris, N. Muscettola, and T. Vidal. Dynamic control of plans with temporal uncertainty. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'01, page 494–499, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1558608125.

N. Muscettola, P. H. Morris, and I. Tsamardinos. Reformulating temporal plans for efficient execution. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning*, KR'98, page 444–452, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc. ISBN 1558605541.

B. Naderi, R. Ruiz, and V. Roshanaei. Mixed-integer programming vs. constraint programming for shop scheduling problems: New results and outlook. *INFORMS Journal on Computing*, 35, 03 2023. doi: 10.1287/ijoc.2023.1287.

K. Neumann, C. Schwindt, and J. Zimmermann. *Resource-Constrained Project Scheduling with Time Windows*, pages 375–407. Springer US, Boston, MA, 2006. ISBN 978-0-387-33768-5. doi: 10.1007/978-0-387-33768-5_15. URL https://doi.org/10.1007/978-0-387-33768-5_15.

M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer International Publishing, 2022. ISBN 9783031059216. doi: 10.1007/978-3-031-05921-6. URL http://dx.doi.org/10.1007/978-3-031-05921-6.

N. Policella, S. Smith, A. Cesta, and A. Oddi. Generating robust schedules through temporal flexibility. pages 209–218, 06 2004.

N. Policella, A. Cesta, A. Oddi, and S. Smith. From precedence constraint posting to partial order schedules: A csp approach to robust scheduling. *AI Commun.*, 20:163–180, 01 2007.

R. Posenato. Cstnu tool: A java library for checking temporal networks. *SoftwareX*, 17:100905, 2022. ISSN 2352-7110. doi: https://doi.org/10.1016/j.softx.2021.100905. URL https://www.sciencedirect.com/science/article/pii/S2352711021001564.

P. Revesz. *Temporal Constraints*, pages 2945–2948. Springer US, Boston, MA, 2009. ISBN 978-0-387-39940-9. doi: 10.1007/978-0-387-39940-9_391. URL https://doi.org/10.1007/978-0-387-39940-9_391.

F. Rossi, P. van Beek, and T. Walsh. Chapter 4 constraint programming. In F. van Harmelen, V. Lifschitz, and B. Porter, editors, *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 181–211. Elsevier, 2008. doi: https://doi.org/10.1016/S1574-6526(07)03004-0. URL https://www.sciencedirect.com/science/article/pii/S1574652607030040.

L. J. Thomas and J. O. McClain. Chapter 7 an overview of production planning. In *Logistics of Production and Inventory*, volume 4 of *Handbooks in Operations Research and Management Science*, pages 333–370. Elsevier, 1993. doi: https://doi.org/10.1016/S0927-0507(05)80187-2. URL https://www.sciencedirect.com/science/article/pii/S0927050705801872.

K. van den Houten, L. Planken, E. Freydell, D. M. J. Tax, and M. de Weerdt. Proactive and reactive constraint programming for stochastic project scheduling with maximal time-lags, 2024. URL https://arxiv.org/abs/2409.09107.

P. Van Hentenryck and V. Saraswat. Strategic directions in constraint programming. *ACM Comput. Surv.*, 28(4):701–726, Dec. 1996. ISSN 0360-0300. doi: 10.1145/242223.242279. URL https://doi.org/10.1145/242223.242279.

P. Varakantham, N. Fu, and H. C. Lau. A proactive sampling approach to project scheduling under uncertainty. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), Mar. 2016. doi: 10.1609/aaai.v30i1.10404. URL https://ojs.aaai.org/index.php/AAAI/article/view/10404.

J. J. Vicente. Optimizing supply chain inventory: A mixed integer linear programming approach. *Systems*, 13(1), 2025. ISSN 2079-8954. doi: 10.3390/systems13010033. URL https://www.mdpi.com/2079-8954/13/1/33.

Appendix A

Advanced algorithms

Algorithm 3: Generating PSTN from the RCPSP/max instance

```
1  Function parse_instance_to_pstn(instance, schedule):
2      pstn ← PSTN()
3      horizon ← pstn.add_node("execution_start")
4      foreach i ∈ instance.orders do
           // Only iterate through accepted orders
5          if i ∈ schedule then
6              foreach j ∈ i.products do
7                  foreach k ∈ j.jobs do
                       // Add start of the job
8                      job_start ← pstn.add_node(i.id_j.id_k.id_START)
                       // Add finish of the job
9                      job_finish ← pstn.add_node(i.id_j.id_k.id_END)
                       // Add job duration contingent edge
10                     pstn.add_contingent_link(job_start, job_finish, k.location, k.sigma)

                       // Add deadline ordinary edge
11                     pstn.set_ordinary_edge(horizon, job_finish, i.deadline)
12                     pstn.set_ordinary_edge(job_start, horizon, 0)
               // Add time lag ordinary edge
13             foreach k ∈ j.jobs do
14                 foreach succ ∈ k.successors do
15                     i_idx ← pstn.find(i.id_product.id_k.id_START)
16                     j_idx ← pstn.find(i.id_j.id_succ.id_START)
17                     pstn.set_ordinary_edge(j_idx, i_idx, −succ.lag)
18     return pstn
```

Algorithm 6 is designed to classify products into completed and ongoing categories based on the completed jobs J_C and pending jobs J_P. We initialize the profit (set to zero) to track the profit from P_C, along with empty lists for completed products, ongoing products, and a visited set to track processed products (lines 2-3). In line 4, the algorithm then iterates through the set of J_C, where each job is

Algorithm 4: Add resource chains to a PSTN

```
1  Function add_resource_chains (pstn, resource_chains) :
2      foreach pred, succ ∈ resource_chains do
3          pred_idx ← pstn.find(pred_START) ;
4          succ_idx ← pstn.find(succ_START) ;
5          pstn.set_ordinary_edge(succ_idx, pred_idx, 0) ;
```

Algorithm 5: Find completed and ongoing products from the execution schedule

```
1  Function find_current_jobs (all_scheduled_jobs, estnu) :
2      if all_scheduled_jobs == None then
3          return False
4      start_jobs, finish_jobs ←
         get_start_and_finish_jobs(all_scheduled_jobs, estnu)
5      J_C ← start_jobs ∩ finish_jobs
6      J_O ← start_jobs − finish_jobs
7      return J_C, J_O
8  Function get_start_and_finish_jobs (all_scheduled_jobs) :
9      start_jobs, finish_jobs ← set(), set()
10     foreach job ∈ all_scheduled_jobs do
           // If job corresponds to START
11         if job includes "start" then
               // Append name of the job in the format of i_j_k
                  leaving "start" or "end"
12             start_jobs.append(job_name)
           // If job corresponds to END
13         if job includes "end" then
14             finish_jobs.append(job_name)
15     return start_jobs, finish_jobs
```

represented in the format (i, j, k) (line 4), which are extracted as integers in line 5 for further processing. In this setup, i corresponds to the order, j indicates the product, and k represents the job.

In line 6, the algorithm checks if (i, j) has already been visited to avoid redundant processing of the same product. If not, it marks the product as visited and retrieves the actual product information from the instance (line 8).

Next, in lines 9-14, the algorithm determines whether all jobs required for the product have been completed. If every job associated with the product belongs to J_C, the product is marked as completed and added to the completed products list (line 16). Additionally, the product's value (profit contribution) is added to the total profit on line 17. Otherwise, if some jobs remain unfinished, the product is classified as ongoing and added to the ongoing products list (line 18).

After processing J_C, the algorithm then iterates through J_O. Using the same procedure as for J_C, in lines 24-27 the algorithm retrieves the product information,

marks the product as ongoing (line 28), and adds it to the visited set (line 29).

Finally, in line 32, the algorithm returns the total profit, the list of P_C, and the list of ongoing products P_O.

Algorithm 6: Extract profit, completed and ongoing products based on completed and ongoing jobs

```
1  Function find_corresponding_products(J_C, J_O, instance):
2      profit ← 0;
3      completed_products ← [] ongoing_products ← [] visited ← [];
4      foreach job ∈ J_C do
5          i, j, k = map(job, int);
6          if (i, j) ∉ visited then
7              visited.append((i, j));
               // finds actual product based on (i, j)
8              product = find_product_in_instance(i, j, instance);
9              all_jobs_in_JC = True;
10             for j_p ∈ product.jobs do
11                 if j_p ∉ J_C then
12                     all_jobs_in_JC = False
13                 end
14             end
15             if all_jobs_in_JC then
16                 completed_products.append(product);
17                 profit ← profit + product.value;
18             end
19             else
20                 ongoing_products.append(product);
21             end
22         end
23     end
24     foreach job ∈ J_O do
25         i, j, k = map(job, int);
26         if (i, j) ∉ visited then
27             product = find_product_in_instance(i, j, instance);
28             ongoing_products.append(product);
29             visited.append((i, j));
30         end
31     end
32     return profit, completed_products, ongoing_products
```

Appendix B

Detailed experiment results

Table B.1: Expected profit and probability mass: SORU-H approach

q	ϵ	Expected Profit ($f(x)$)	Makespan t	Throughput $f(x)/t$
0.50	0.05	2409.00 ± 0.00	$95.06(\pm 0.09)$	25.34
0.50	0.10	2264.46 ± 572.11	$95.12(\pm 0.17)$	23.80
0.50	0.20	144.54 ± 572.11	$95.30(\pm 0.43)$	1.52
0.50	0.25	0.00 ± 0.00	0	–
0.50	0.30	0.00 ± 0.00	0	–
0.50	0.40	0.00 ± 0.00	0	–
0.50	0.50	0.00 ± 0.00	0	–
0.50	0.70	0.00 ± 0.00	0	–
0.75	0.05	2409.00 ± 0.00	$95.05(\pm 0.09)$	25.34
0.75	0.10	2312.64 ± 472.07	$95.12(\pm 0.18)$	24.31
0.75	0.20	337.26 ± 835.89	$97.22(\pm 0.29)$	3.47
0.75	0.25	24.09 ± 239.69	$98.35(\pm 0.44)$	0.24
0.75	0.30	561.75 ± 972.98	$91.32(\pm 0.53)$	6.15
0.75	0.40	426.93 ± 881.50	$93.84(\pm 0.85)$	4.55
0.75	0.50	696.57 ± 1039.22	$96.95(\pm 0.97)$	7.19
0.75	0.70	89.88 ± 440.32	$96.95(\pm 0.97)$	0.91
0.90	0.05	2409.00 ± 0.00	$95.08(\pm 0.09)$	25.33
0.90	0.10	2384.91 ± 239.69	$97.10(\pm 0.14)$	24.57
0.90	0.20	2224.53 ± 223.57	$95.44(\pm 0.35)$	23.31
0.90	0.25	2089.71 ± 573.32	$95.36(\pm 0.49)$	21.91
0.90	0.30	1572.90 ± 1029.70	$98.57(\pm 0.57)$	15.96
0.90	0.40	2012.92 ± 287.56	$91.79(\pm 0.78)$	21.93
0.90	0.50	780.52 ± 996.98	$95.14(\pm 0.94)$	8.20
0.90	0.70	1200.54 ± 861.67	$94.52(\pm 1.28)$	12.77

Table B.2: Expected profit and probability mass: PSTN approach

q	ϵ	Expected Profit ($f(x)$)	Makespan t	Throughput $f(x)/t$	
0.50	0.05	2409.00 ± 0.00	0.99	$95.00(\pm 0.0)$	25.36
0.50	0.10	2409.00 ± 0.00	0.99	$95.00(\pm 0.0)$	25.36
0.50	0.20	2409.00 ± 0.00	0.99	$95.00(\pm 0.0)$	25.36
0.50	0.25	2409.00 ± 0.00	0.99	$95.00(\pm 0.0)$	25.36
0.50	0.30	2409.00 ± 0.00	0.99	$95.00(\pm 0.0)$	25.36
0.50	0.40	2397.37 ± 116.30	0.99	$94.51(\pm 15.69)$	25.37
0.50	0.50	2409.00 ± 0.00	0.99	$94.81(\pm 0.54)$	25.41
0.50	0.70	251.78 ± 333.78	$6.89 \cdot 10^{-3}$	$78.77(\pm 12.33)$	3.19
0.75	0.05	2409.00 ± 0.00	0.99	$95.00(\pm 0.0)$	25.36
0.75	0.10	2409.00 ± 0.00	0.99	$95.00(\pm 0.0)$	25.36
0.75	0.20	2409.00 ± 0.00	0.99	$95.00(\pm 0.0)$	25.36
0.75	0.25	2384.91 ± 240.90	0.99	$94.11(\pm 44.28)$	25.34
0.75	0.30	2247.00 ± 0.00	0.99	$94.91(\pm 1.20)$	23.67
0.75	0.40	2241.51 ± 54.90	0.99	$94.92(\pm 3.98)$	23.61
0.75	0.50	2247.00 ± 0.00	0.99	$94.96(\pm 0.39)$	23.66
0.75	0.70	1217.74 ± 1025.17	0.39	$134.64(\pm 303.25)$	9.04
0.90	0.05	2409.00 ± 0.00	0.99	$95.00(\pm 0.0)$	25.36
0.90	0.10	2409.00 ± 0.00	0.99	$95.00(\pm 0.0)$	25.36
0.90	0.20	2247.00 ± 0.00	0.99	$95.00(\pm 0.0)$	23.65
0.90	0.25	2247.00 ± 0.00	0.99	$95.00(\pm 0.0)$	23.65
0.90	0.30	2247.00 ± 0.00	0.99	$95.00(\pm 0.0)$	23.65
0.90	0.40	2048.34 ± 97.40	0.99	$94.83(\pm 15.72)$	21.59
0.90	0.50	2051.49 ± 56.04	0.99	$94.80(\pm 6.74)$	21.64
0.90	0.70	1759.74 ± 299.59	0.99	$185.02(\pm 96.95)$	9.51

Appendix C

Future RTE* Recovery Algorithm

All completed jobs (J_C) can be eliminated from the DC STNU, as they do not require further rescheduling. However, nodes corresponding to completed jobs may still have edges connecting them to nodes corresponding to pending jobs (J_P). These edges cannot simply be removed, as doing so would result in the loss of critical time lag constraints between jobs. Instead, they must be modified appropriately.

An example case of this scenario is illustrated in Figure C.1. At timestamp t, the RTE* algorithm fails. Here, $start_1$ belongs to J_C, while $start_2$ belongs to J_P. If we were to delete the ordinary edge from $start_1$ to $start_2$, we would eliminate the time constraint that governs the sequence of these jobs. To preserve the structure of the scheduling constraints, we need to modify this edge instead of removing it entirely.

In the PSTN, the `EXECUTION_START` initially represents the beginning of the planning. Since the new execution of RTE* will begin from the moment of its failure, the `HORIZON` must now correspond to timestamp t. All constraints must be updated relative to t.

We need to calculate the minimal and maximal time that should elapse before it begins to determine the updated constraints for $start_2$. Since job_1 has already been observed, its exact duration, $duration_1$, is known. Thus, the distance between $start_1$ and t can be computed using the following equation:

$$\texttt{time\_past} = t - finish_1 + duration_1 = t - start_1 \tag{C.1}$$

To get the remaining lag after timestamp t, the following equation is followed:

$$\texttt{remaining\_lag} = lag - \texttt{time\_past} \tag{C.2}$$

This means that there must be at least/most $remaining\_lag$ after t before executing job_2.

To resemble such modification on the network, we add the following edge, where

$\texttt{START}_{suc}$ represents the start of the job that was connected to the job in J_C by a time lag:

$$\texttt{START}_{suc} \xrightarrow{-remaining\_lag} \texttt{HORIZON} \tag{C.3}$$

This introduces the following equation:

$$\texttt{START}_{suc} - \texttt{HORIZON} \leq -remaining\_lag \tag{C.4}$$

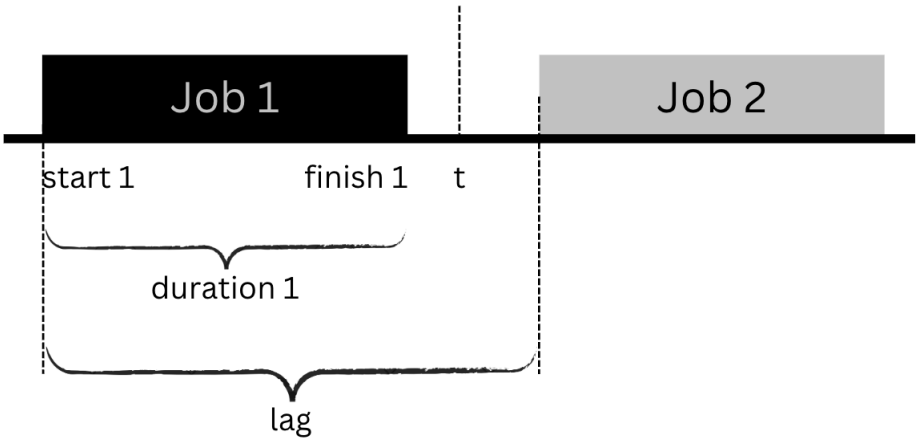

Figure C.1: Critical case for time lag edges

After adding new time lag edges, completed jobs can be safely removed from the network with edges involving their nodes.

Appendix D

CP model with soft deadlines

Sets and indices

- $O = \{1, 2, \ldots, n_o\}$: Set of n_o orders.
- $P = \{1, 2, \ldots, n_p\}$: Set of all possible n_p product types.
- $J = \{1, 2, \ldots, n_j\}$: Set of all possible n_j jobs.
- $R = \{1, \ldots, l, \ldots, n_r\}$: Set of resources.
- $T = \{0, \ldots, t, \ldots, n_t\}$: Set of time units (days).
- $P(i)$: Set of products for order i.
- $J(j)$: Set of jobs for product j.
- $S(k)$: Set of successors of job k.
- i: standard index for order
- j: standard index for product ‘
- k: standard index for job
- l: standard index for resource
- t: standard index for time

Parameters

- $deadline_i \in \mathbb{Z}$: Deadline for order i (day).
- $profit_i \in \mathbb{Z}$: Profit for order i.
- $required_i \in \{0, 1\}$: Binary variable (1) if order i is required, (0) otherwise.
- $duration_k \in \mathbb{Z}$: Duration of job k.

- $\rho_k(l) \in \mathbb{Z}$: Amount of resource l required by job k.
- $lag_{km} \in \mathbb{Z}$: Time lag between jobs k and m.
- $capacity_l \in \mathbb{Z}$: Capacity of resource l.

Decision Variables

- x_{ijk}: (optional) interval CP variable for job k for product j for order i. It represents interval of time during which ijk happens, with a duration $duration_k$. The optimizer should assign a start time and end time for this interval variable, which is notated with $startOf$ (start time of the job) and $endOf$ (end time of the job). If no interval is available, it stays empty.
- y_i: Binary variable, 1 if order i is completed, 0 otherwise
- $lateness_i$: Lateness of order i

Objective Function

$$\text{Maximize } (\sum_{i \in O} y_i \cdot profit_i - c * \sum_{i \in O} lateness_i)$$

where c is a hyperparameter for the importance of lateness. For the purpose of the experiment, the hyperparameter is set to $c = 0.5$.

Constraints

Interval Variables

$$x_{ijk} \in \begin{cases} \text{IntervalVar}(i, j, k, duration_k) & \text{if } required_i = 1 \\ \text{OptionalIntervalVar}(i, j, k, duration_k) & \text{if } required_i = 0 \end{cases}, \quad \forall i \in O, j \in P(i), k \in P(j)$$

Task Scheduling

Presence Constraint: Order i is accepted only of all job intervals involved in order i are present.

$$y_i = \left(\bigwedge_{j \in P(i), k \in J(j)} \text{presenceOf}(x_{ijk}) \right), \quad \forall i \in O \text{ if } required_i = 0$$

Precedence Constraints

$$\text{if } y_i = 1, \text{startOf}(x_{ijk}) + lag_{km} \leq \text{startOf}(x_{ijm}), \quad \forall i \in O, j \in P(i), k \in J(j), m \in S(k)$$

Lateness Constraints

$$max\_end\_time_i = max_{j \in O(i), k \in J(j)}(endOf(x_{ijk}))\ \forall i \in O$$

$$lateness_i \geq max\_end\_time_i - deadline_i \forall i \in O$$

$$lateness_i \geq 0\ \forall i \in O$$

Resource Capacity

$$\sum_{i \in O \text{ if } y_i = 1} \sum_{j \in P(i)} \sum_{k \in J(j)} \text{Pulse}(x_{ijk}, \rho_k(l)) \leq capacity_l, \quad \forall l \in R$$

Appendix E

Reproducibility

We have made our code publicly available to support reproducibility and encourage the adoption of similar temporal network methodologies. Detailed instructions regarding dependencies and experiment scripts can be found in the `README` and `BUILDING` file of our repository PSTN for RCPSP-MAX GitHub Repository. Note that the repository serves as an extension to already existing code in STNU for RCPSP-MAX GitHub Repository, which utilized Simple Temporal Networks with Uncertainty. Furthermore, we extended the CSTNU tool and uploaded our extension to CSTNU Tool extension.

The implementation of the Python code was done in Python 3.9.6.

Java code implementation was done using Java 21.0.2. Java 21 is required to run the project. Furthermore, MATLAB with "Optimization Toolbox" and "Statistics and Machine Learning Toolbox" is needed to run PSTN algorithms.

E.1 Data Access, Randomness and Seeding

The experiments use benchmark J10 data [1].

We treat job durations from the J10 benchmark as the mean of the distributions. We normalize these parameters using the following formulas to sample from a log-normal distribution based on these means:

$$\mu = \ln\left(\frac{\text{mean}}{\sqrt{1+\left(\frac{\text{std}}{\text{mean}}\right)^2}}\right)$$

$$\sigma = \sqrt{\ln\left(1+\left(\frac{\text{std}}{\text{mean}}\right)^2\right)}$$

Where:

[1] https://www.wiwi.tu-clausthal.de/en/ueber-uns/abteilungen/betriebswirtschaftslehre-insbesondere-produktion-und-logistik/research/research-areas/project-generator-progen/max-and-psp/max-library/single-mode-project-duration-problem-rcpsp/max

1. μ is the location of the log-normal distribution.
2. σ is the sigma of the log-normal distribution
3. mean is the mean of the job durations from the J10 benchmark.
4. std is the standard deviation we set for the job durations.

Randomness plays a role in the sampling of durations from probability distributions. Fixed random seeds are used as 42 to ensure consistency across runs.