The Flow Must Go On

Algorithms for Scheduling in Biomanufacturing

Kim van den Houten

The Flow Must Go On

Algorithms for Scheduling in Biomanufacturing

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology
by the authority of the Rector Magnificus, Prof.dr.ir. H. Bijl,
chair of the Board for Doctorates
to be defended publicly on
Wednesday 3, June 2026 at 12:30

by

Kim Cecilia VAN DEN HOUTEN

This dissertation has been approved by the promotor and the copromotors.

Composition of the doctoral committee:

Rector Magnificus,	chairperson
Prof.dr. M.M. de Weerdt,	Delft University of Technology, *promotor*,
Dr. D.M.J. Tax,	Delft University of Technology, *copromotor*.
Independent members:	
Prof.dr.ir. H.J. Noorman,	Delft University of Technology,
Dr. E. Freydell,	dsm-firmenich,
Dr. M. Lombardi,	University of Bologna, Italy,
Dr. Y. Zhang,	Eindhoven University of Technology,
Dr. J.M. van den Akker,	Universiteit Utrecht,
Prof.dr.ir. M.J.T. Reinders,	Delft University of Technology, reserve member.

The research in this thesis is supported by the AI4b.io program, a collaboration between TU Delft and dsm-firmenich, and is fully funded by dsm-firmenich and the RVO (Rijksdienst voor Ondernemend Nederland).

Keywords: scheduling, algorithms, biomanufacturing, fermentation, uncertainty, simulation, constraint programming, simple temporal networks

An electronic copy of this dissertation is available at
`https://repository.tudelft.nl/`.

Contents

ORGANISM– MECHANISM– PLAN

Summary

Biomanufacturing involves the large-scale production of bio-based products, for example, food ingredients. In fermentation-based factories, living organisms are used to produce the active components of these bio-based products through fermentation in large bioreactors. Modern biomanufacturing plants are highly complex, with many tanks and interconnected pieces of equipment. They are typically designed to make multiple products (in parallel), each with its own recipe and production process. As a result, it is not possible to repeat the same weekly schedule — customer demand changes over time, and different customer orders require different products and thus production schedules.

A feasible schedule should consider several scheduling rules essential for biomanufacturing. For example, in the food industry, it is crucial that tanks are cleaned before, after, or between specific production operations. The production of a single product involves multiple unit operations, for example, starting from fermentation, followed by several filtration steps. Between the different stages, there are rules about how long an intermediate product can wait at a single tank; otherwise, a product risks expiring. Due to the biological nature of fermentation processes, process durations are uncertain and can vary from batch to batch. Due to this complexity, limited resource capacities, and various sources of uncertainty, it poses a significant challenge for factory managers and production planners to fulfill all customer orders on time.

In this dissertation, we investigate how algorithms can support decision-making in operating complex biomanufacturing factories. An algorithm can be described as a step-by-step computational procedure that determines how to find a solution. We focus on the scheduling problem, which concerns deciding which tasks must be executed to meet all customer demand, assigning resources to tasks, and determining when and in what sequence all tasks occur.

Scheduling has been studied by researchers and practitioners for decades. Most scheduling problems are NP-hard, meaning that the computation time required to find an optimal solution can grow exponentially with the number of inputs. However, recent research shows that constraint programming—a paradigm for solving constrained optimization problems—can solve scheduling problems with up to a million tasks, as evaluated on several benchmark problems from the scheduling literature.

It remains unclear whether such performance can also be expected in the context of biomanufacturing factories, for which modeling the scheduling problem brings together many scheduling constraints that are often studied in isolation in the literature. Furthermore, constraint programming-based scheduling solutions typically do not account for uncertainties.

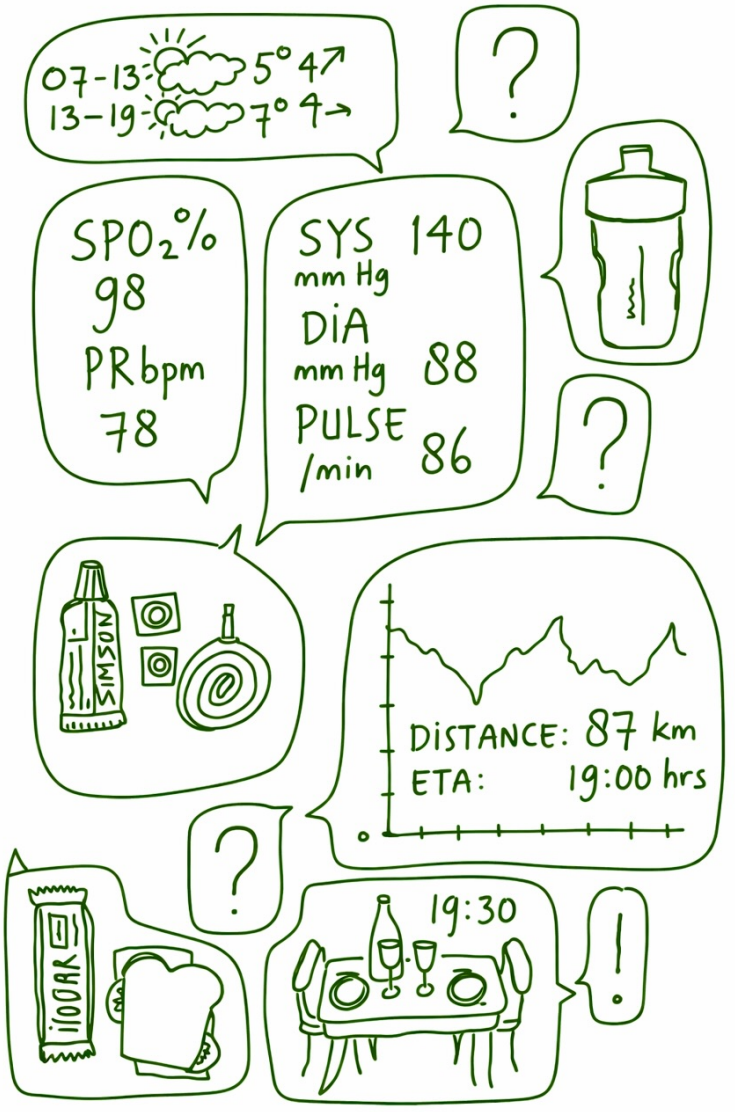
In this work, we investigate how solvers, such as constraint programming solvers, can be enhanced to be more robust against uncertainty in the optimization parameters. We propose an approach that integrates a learning-based method to find high-quality

schedules in the presence of uncertainty. We also explore a generic simulation-based framework for handling large-scale stochastic optimization problems such as scheduling.

We study efficient procedures for handling uncertainties in real-time scheduling. As scheduling problems are NP-hard, complete rescheduling would be impractical, as the algorithms might require computation times that are unacceptable for operators to make fast decisions in real-time. Therefore, we investigate algorithms suitable for the fast and real-time execution of schedules. We focus on one source of uncertainty, namely delays in processing times, and study these in combination with the presence of strict waiting constraints, which are typical for biomanufacturing due to the short shelf lives of food products. This combination of stochastic processing times and strict waiting constraints poses a challenging problem because delays can lead to infeasibilities, and a simple repair, such as postponing all tasks, is not always feasible. We demonstrate that combining simple temporal networks with uncertainty with constraint programming enables the efficient and robust online execution of schedules.

Furthermore, we contribute by considering a real factory, an enzyme production site. We explore the long-horizon production planning/scheduling problem using simulation-optimization techniques. Finally, we evaluate the performance of constraint programming for scheduling such a real factory. Our findings show that incorporating domain knowledge into the solving strategy substantially improves performance and enables the methods to scale to realistic biomanufacturing scenarios.

Altogether, this dissertation offers new insights and algorithms for solving scheduling problems in biomanufacturing.


07-13 5° 4↗
13-19 7° 4→
?
SPO2%
98
PRbpm
78
SYS 140
mm Hg
DiA
mm Hg 88
PULSE
/min 86
?
SIMSON
DISTANCE: 87 km
ETA: 19:00 hrs
?
19:30
!

Samenvatting

Biomanufacturing richt zich op het grootschalig produceren van biobased producten, zoals voedselingrediënten. In fermentatiefabrieken vindt fermentatie plaats in bioreactoren, waarbij levende organismen worden gebruikt om actieve componenten van deze biobased producten te laten groeien. Moderne biomanufacturing fabrieken zijn zeer complex, met vele tanks en onderling verbonden apparaten. Ze zijn meestal ontworpen om meerdere producten (tegelijk) te kunnen maken, elk met een eigen recept en productieproces. Daardoor is het niet mogelijk om simpelweg elke week dezelfde planning te herhalen — de klantvraag verandert in de tijd, en verschillende bestellingen vereisen verschillende producten en dus processen.

Een uitvoerbare planning moet rekening houden met diverse planningsregels die essentieel zijn voor biomanufacturing. In de voedingsindustrie is het bijvoorbeeld cruciaal dat tanks worden schoongemaakt vóór, na of tussen bepaalde productiehandelingen. De productie van één product omvat meerdere processtappen, bijvoorbeeld beginnend met fermentatie, gevolgd door verschillende filtratiestappen. Tussen de verschillende fasen bestaan regels over hoe lang een tussenproduct in een bepaalde tank mag blijven; anders loopt het risico te bederven. Door de biologische aard van fermentatie zijn de procestijden onzeker en kunnen ze per batch variëren. Vanwege deze complexiteit, de beperkte capaciteit van machines en verschillende bronnen van onzekerheid is het een grote uitdaging voor fabrieksmanagers en productieplanners om alle klantorders op tijd te leveren.

In dit proefschrift onderzoeken we hoe algoritmen de werking van zulke complexe biomanufacturingfabrieken kunnen ondersteunen. Een algoritme kan worden omschreven als een stapsgewijze procedure voor de computer die bepaalt hoe een oplossing wordt gevonden. We richten ons op het schedulingsprobleem, dat betrekking heeft op het beslissen welke taken moeten worden uitgevoerd om aan de klantvraag te voldoen, het toewijzen van machines aan taken, en het bepalen wanneer en in welke volgorde alle taken plaatsvinden.

Schedulingsproblemen worden al decennialang bestudeerd door onderzoekers en praktijkdeskundigen. De meeste schedulingsproblemen zijn NP-moeilijk, wat betekent dat de rekentijd die nodig is om een optimale oplossing te vinden exponentieel kan groeien met het aantal invoerparameters (bijvoorbeeld het aantal bestelde producten). Recent onderzoek toont echter aan dat constraint programming — een oplossingsmethode voor optimalisatieproblemen met constraints — schedulingsproblemen met tot wel een miljoen taken kan oplossen, zoals geëvalueerd op verschillende benchmarkproblemen uit de schedulingsliteratuur.

Het is echter onduidelijk of zulke prestaties ook verwacht kunnen worden in de context van biomanufacturingfabrieken, waar het modelleren van het schedulingsprobleem vele constraints samenbrengt die in de literatuur vaak afzonderlijk worden bestudeerd. Boven-

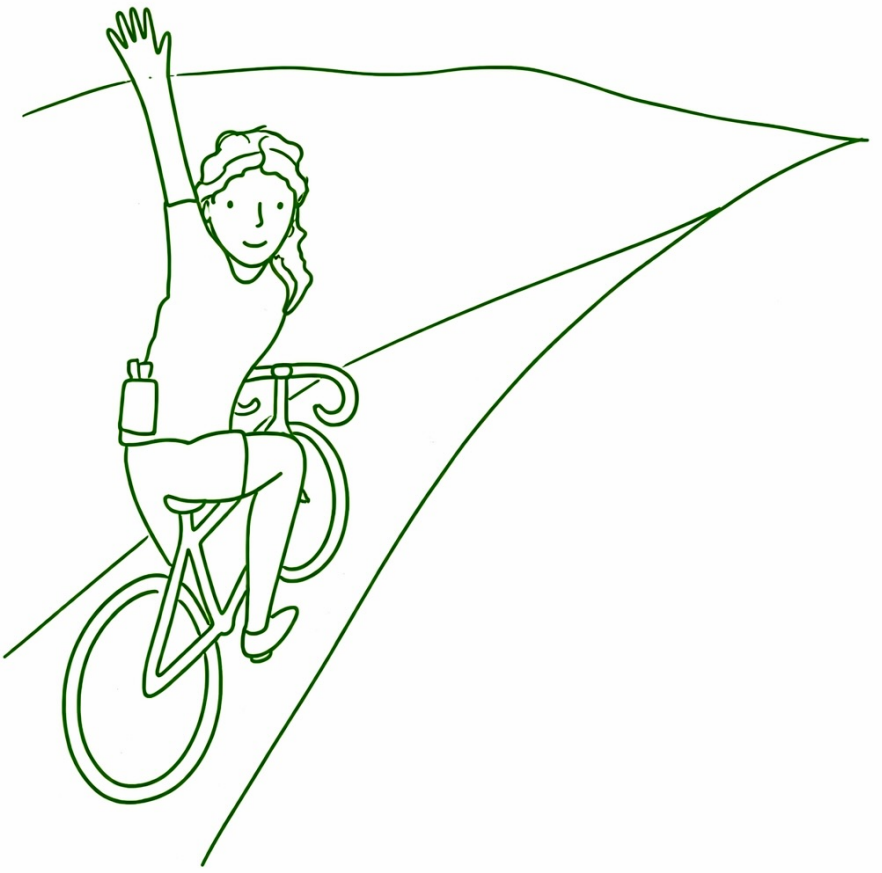
dien houden op constraint programming gebaseerde schedulingsoplossingen doorgaans geen rekening met onzekerheden.

In dit proefschrift onderzoeken we hoe constraint programming-technieken robuuster kunnen worden gemaakt tegen dergelijke variaties. We stellen verschillende aanpakmethodes voor die leermethoden integreren om fabrieksplanningen te optimaliseren. Ook onderzoek we hoe solvers, zoals constraint programming-solvers, robuuster gemaakt kunnen worden tegen onzekerheid in de optimalisatieparameters. We stellen een aanpak voor die een leergebaseerde methode integreert om kwalitatief hoogwaardige roosters te vinden in de aanwezigheid van onzekerheid. Daarnaast verkennen we een generiek simulatiegebaseerd raamwerk voor het omgaan met grootschalige stochastische optimalisatieproblemen, zoals planningsvraagstukken.

Ook bestuderen we efficiënte procedures om met variërende procesduren om te gaan in real-time. Doordat schedulingsproblemen NP-moeilijk zijn, is volledige herplanning bij inloop of uitloop vaak onpraktisch, aangezien de algoritmen rekentijden kunnen vereisen die te lang zijn voor operators om snel beslissingen te nemen. Hierdoor onderzoeken wij welke soorten algoritmen geschikt zijn voor snelle en real-time uitvoering van tevoren bedachte fabrieksplanning. We richten ons op één bron van onzekerheid, namelijk vertragingen in procestijden, en bestuderen deze in combinatie met strikte wachttijdbeperkingen, die typisch zijn voor biomanufacturing vanwege de beperkte houdbaarheid van voedingsproducten. Deze combinatie van onzekere procestijden en strikte wachttijdbeperkingen vormt een uitdagend probleem, omdat een vertraging tot onuitvoerbaarheid kan leiden, en een eenvoudige wijziging in de planning niet altijd mogelijk is — het simpelweg uitstellen van alle taken is immers niet altijd toegestaan. We onderzoeken het gebruik van simple temporal networks with uncertainty in combinatie met constraint programming en tonen aan dat deze aanpak efficiënte en robuuste uitvoering van schema's in real-time mogelijk maakt.

Daarnaast dragen we ook bij door een echte fabriek te beschouwen, namelijk een fabriek waarin enzymen voor de voedingsindustrie worden geproduceerd. We verkennen simulatie-gebaseerde optimalisatietechnieken om de fabrieksplanning voor de lange termijn te optimaliseren. Ten slotte evalueren we de prestaties van constraint programming voor het modelleren en oplossen van het planningsprobleem van zo'n echte fabriek. Onze bevindingen tonen aan dat het opnemen van domeinkennis in de oplossingsstrategie de prestaties aanzienlijk verbetert en de methoden schaalbaar maakt naar realistische scenario's.

Al met al draagt dit proefschrift nieuwe inzichten en praktische methoden bij aan het oplossen van complexe en onzekere schedulingsproblemen in de biomanufacturing.



1 Introduction

Biotechnology holds great promise for developing sustainable solutions to global food challenges. Fermentation-based biomanufacturing utilizes living organisms to produce active ingredients for various products, including food (Al-Maqtari, Waleed, and Mahdi 2019), and can play a crucial role in feeding the global population in a sustainable way.

Biomanufacturing sites are typically flexible multipurpose factories (G. P. Georgiadis, Elekidis, and M. C. Georgiadis 2019a; Harjunkoski *et al.* 2014). Such factories allow the production of a wide range of products by applying different processing rules on a shared set of often costly resources. Some bio-based products require a fermentation process, which typically takes place in a large tank (bioreactor) in which the active product is cultivated. This is, for example, the case for the production of ingredients (such as enzymes Al-Maqtari, Waleed, and Mahdi (2019)) for yoghurt or cheese, for which a simplification of the production process is visualized in Figure 1.1. After fermentation, a sequence of refining operations is needed, also referred to as downstream processing (while everything up to the fermentation step is referred to as upstream processing). These downstream operations refine the product to the desired specifications, and ultimately, packaging steps are required to deliver the final products to the customers.

Between the different processing steps, there is, in most cases, a need for a continuous flow, i.e., a product can only wait for a limited time before proceeding to the next piece of equipment. If such a constraint is violated (the zero-wait constraint), the quality of the product can be impacted, and thus it is essential that the flow must go on! For certain product–resource combinations, switching between products requires intermediate cleaning steps, which introduce additional scheduling steps. These cleaning steps are referred to as sequence-dependent setup times, which add additional complexity to the problem. These factors together lead to challenging scheduling problems in the daily operations of biomanufacturing plants (G. P. Georgiadis, Elekidis, and M. C. Georgiadis 2019a; Harjunkoski *et al.* 2014).

Biomanufacturing is further complicated by multiple sources of uncertainty, including product yield and processing times. The duration of a fermentation process is, for example, inherently stochastic. The combination of these stochastic processing times and the strict temporal constraints (zero-wait conditions) makes scheduling particularly challenging. These challenges are especially evident in the scheduling of an enzyme production facility at dsm-firmenich, which we examine in greater depth in this thesis.

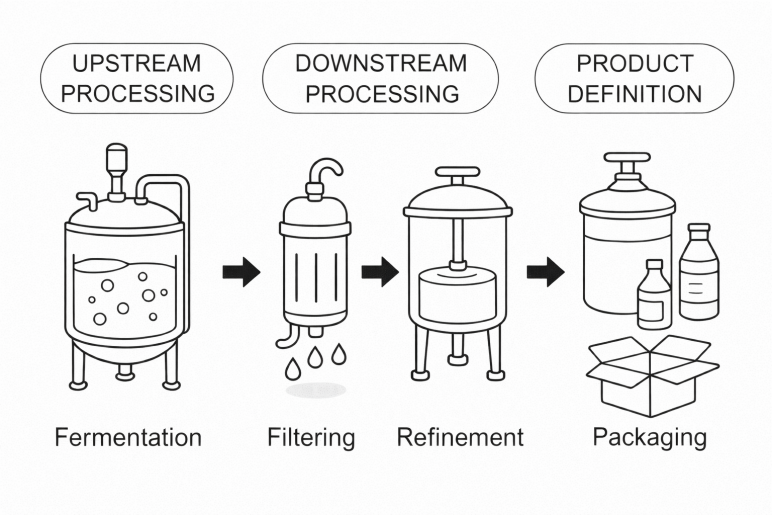


Figure 1.1 *Simplified overview of a fermentation-based production process of a bio-based product. This image is generated using an AI tool (ChatGPT).*

It is difficult to design standard schedules for such facilities. Additionally, since these factories typically produce a variety of products, it is impossible to simply repeat the same schedule week after week. Schedulers and planners must carefully balance robustness and high throughput, while also meeting customer deadlines. This often results in large-scale production schedules being created with spreadsheets or manual decisions, without any guarantee of optimality (G. P. Georgiadis, Elekidis, and M. C. Georgiadis 2019a).

The primary objective of this dissertation is to understand how such complex real problems as those in biomanufacturing can be solved. In this context, we develop and evaluate computational methods that support scheduling operations in biomanufacturing, with a focus on enhancing schedule efficiency and robustness. Therefore, the remainder of this introduction is structured as follows. Section 1.1 provides background on scheduling in biomanufacturing and on methods for solving general scheduling problems. Section 1.2 then outlines the structure of this dissertation, presenting the main research questions and contributions.

1.1. Scheduling

The efficiency of production industries is affected by production planning and scheduling (Rajkumar *et al.* 2010). Production planning is often determined before scheduling and does not consider scheduling objectives such as machine utilization and productivity. In general, planners focus on customer relations and demand prediction, whereas the schedulers focus on achieving high product quality and minimizing manufacturing time (Rajkumar *et al.* 2010). The scheduling questions in biomanufacturing are typically:

- What tasks must be executed to satisfy the given demand (batching)?

- How should the given resources be utilized (batch-resource assignment)?
- In what order are batches processed (sequencing and/or timing)?

(G. P. Georgiadis, Elekidis, and M. C. Georgiadis 2019a; Harjunkoski *et al.* 2014), and visualized in Figure 1.2.

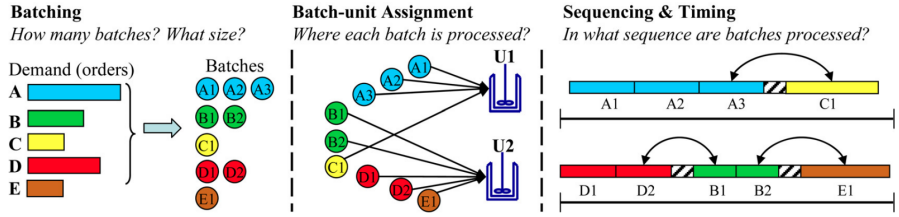


Figure 1.2 *Visualization of the different scheduling decisions, image copied from Harjunkoski* et al. *(2014). Given a customer demand, which specifies the required quantities for each product type, the first decision concerns how this demand is split into production batches. Next, for each batch, we must determine which machines will process it. Finally, we decide in which order and at which exact time each batch will be processed.*

Scheduling plays a crucial role in various domains, including healthcare and manufacturing, and has been a central topic of operations research for decades. Scheduling concerns the allocation of resources to tasks over specified time periods and represents a typical combinatorial optimization problem. Scheduling problems are typically NP-hard (Garey and Johnson 1990; Pinedo 2016), meaning that the computation time required to find an optimal solution can grow exponentially with the input size. A well-known way of visualizing schedules is through Gantt charts, such as visualized in Figure 1.2 under 'Sequencing & Timing', where you see on the horizontal axis the time horizon, and horizontal bars are used to show in what time period a task is occupying a certain resource. On the vertical axis you see different resources. In the remainder of this section, we will discuss common scheduling frameworks, solution approaches, and the impact of uncertainty on scheduling.

1.1.1. Frameworks

The broad range of industrial applications has led to an extensive body of literature on scheduling, which spans from relatively simple problems to far more complex scenarios. For example, an early work from 1954 by Johnson on machine scheduling presented two- and three-stage machine scheduling problems. The single-machine scheduling problem plays an important role in the scheduling literature because it is a special case of many other scheduling variants (Pinedo 2016). Chapter 3 of Pinedo's book is devoted entirely to single-machine scheduling, including optimal decision rules and related complexity analysis.

Another important class of problems is job shop scheduling (for a recent survey see (Xiong *et al.* 2022)), which is widely used and offers a generic framework for modeling a broad variety of scheduling settings. In these formulations, jobs consist of a collection of tasks that must be executed, sometimes subject to deadlines. Each task represents a unit operation that requires specific resources, such as machines or personnel, to complete. Another widely studied framework is the resource-constrained project scheduling problem (RCPSP) (see (Hartmann and Briskorn 2022; Herroelen, De Reyck, and Demeulemeester 1998)). In this setting, each resource has a limited capacity that can be allocated to multiple tasks simultaneously, while each task may require one or more renewable or non-renewable resources. These two frameworks are closely related, as the RCPSP is a generalization of the classic job-shop scheduling problem.

A scheduling problem is a typical example of a combinatorial optimization problem (COP), where we want to find the best solution from a discrete solution space (for example because you want to choose from a finite set of resources), such that we minimize an objective function, like the time to execute the entire schedule (makespan). A solution to a scheduling problem is generally defined as an allocation of resources to tasks, together with start and completion times for each task, such that all scheduling constraints are satisfied. As most scheduling problems are NP-hard (Garey and Johnson 1990; Pinedo 2016), the computational effort required can grow exponentially with the problem size, and solving large real-world instances can become computationally expensive or even intractable, such that, within reasonable time limits, only suboptimal or no solutions can be obtained.

1.1.2. Solution approaches

A wide range of approaches has been developed for finding such solutions, spanning from exact methods, like mathematical programming and constraint programming, to heuristic approaches, including metaheuristics. An exact method is an algorithmic approach that is guaranteed to find an optimal solution to a problem if one exists (however, the runtime can grow exponentially). Heuristics are simple rules or procedures that quickly generate good, though not necessarily optimal, solutions, although for a subset of scheduling problems, optimal decision rules exist (Pinedo 2016). Metaheuristics are higher-level strategies that guide the search through large or complex solution spaces where an exhaustive search is infeasible (Chaudhry, A. M. Khan, and A. A. Khan 2013).

Mathematical programming offers a general framework for formulating and solving combinatorial optimization problems, including scheduling, in an exact way. In particular, mixed-integer linear programming (MILP) is widely used, which is a mathematical optimization model that combines integer and continuous decision variables (Conforti, Cornuéjols, and Zambelli 2014). MILP models capture linear constraints and objectives, and can be solved using methods such as branch-and-bound or cutting planes (Gomory 1958; Land and Doig 1960). MILP models have been used for modeling job shop and project scheduling problems for decades. However, surveys on scheduling in the process industries (including biomanufacturing) by G. P. Georgiadis, Elekidis, and M. C. Georgiadis (2019a) and Harjunkoski *et al.* (2014) note that, due to the frequent mixing and splitting of material flows, conventional job shop or project scheduling models discussed in the previous subsection are often unsuitable. Alternatively, material-based MILP mod-

els (Kondili, Pantelides, and Sargent 1993; Pantelides 1993) exist to model material flows in such factories. According to (G. P. Georgiadis, Elekidis, and M. C. Georgiadis 2019a), most of these models have been tested only on small to medium sizes and are not sufficient to address large-sized industrial problems, which is a major limitation of state-of-the-art MILP solutions to scheduling in biomanufacturing.

Constraint programming (CP) offers a flexible alternative for solving scheduling problems. CP can handle a wide variety of combinatorial constraints, and, unlike MILP, does not require linear constraints. Variables take values from specified domains, and propagation techniques systematically reduce the search space. CP, especially with interval variables (Laborie 2015), has become a powerful tool for solving deterministic scheduling problems. A recent comparison between solvers for CP and MILP demonstrated the strong performance of CP Optimizer (CP solver) (IBM 2024) on a set of scheduling problems. The authors highlight that their relative performance depends on the problem structure (Naderi, Ruiz, and Roshanaei 2023). In particular, the authors report that "CP models perform well in pure sequencing problems and are a superior alternative for solving joint sequencing–assignment problems, whereas MIP models are superior when scheduling problems only involve assignment decisions." Consequently, CP is often particularly effective for problems involving complex task sequencing, such as job-shop scheduling. However, these advances have mainly been demonstrated on standard job-shop scheduling benchmarks that do not incorporate all types of constraints, such as those in biomanufacturing. Only a few examples of CP solutions to biomanufacturing exist in the literature (Awad *et al.* 2022; Escobet Canal *et al.* 2019; Novara, Novas, and Henning 2016; Zeballos, Novas, and Henning 2011), however, none of them address the mixing and splitting of batches. This highlights a clear research gap regarding the applicability of CP to complex biomanufacturing systems, such as the real enzyme production studied in this research.

1.1.3. Scheduling under uncertainty

Scheduling in biomanufacturing is further complicated by uncertainty in key parameters arising from the biological nature of production processes. A stochastic combinatorial optimization problem (SCOP) can be formulated as the optimization of the expected value of an objective function. A particularly challenging aspect of SCOPs is that this expectation is generally intractable, requiring decisions on how to approximate the objective (Juan *et al.* 2015). A common, but rather simplistic approach is to replace uncertain quantities with their expected value, typically estimated from the empirical mean of historical data. However, such deterministic models disregard uncertainty and risk, and may thus yield poor solutions due to the "flaw of averages" (Savage 2012).

A more sophisticated approach for handling uncertainty is stochastic programming. So-called recourse models (Dantzig 1955; Higle 2005) make a distinction between decisions that must be made beforehand and those that can be made after the information about the uncertainty becomes available. A recourse action can thus also be seen as a repair. Those two-stage models (decide-observe-repair) are not always applicable, as in reality and the case of scheduling, uncertainty typically evolves over time, and a sequence of decisions must be made (decide-observe-decide- ..). This calls for multi-stage stochastic programming formulations that model this repeated pattern of decisions and observa-

tions (Higle 2005). Typically, scenario models are used to formulate both two-stage and multi-stage stochastic programming models, which comes down to including decision variables and constraints for each possible uncertainty scenario in one big linear program. A major problem is that these scenario models can grow extremely large, which aggravates the curse of dimensionality that is already present when solving deterministic problems.

Scaling stochastic programming relies on decomposition methods (Birge 1985; Van Slyke and Wets 1969) that are mainly developed and tailored to linear programming, whereas constraint programming (CP) has shown strong performance on deterministic scheduling problems. The literature on stochastic CP is limited (Hnich *et al.* 2009; Mercier-Aubin *et al.* 2020), and no generalized framework exists for incorporating uncertainty into CP. A recent contribution combines decomposition techniques with CP to address uncertainty in scheduling (Juvin, Houssin, and Lopez 2025), yet current approaches are restricted to two-stage settings and continue to face challenges such as large optimality gaps and significant computational effort. This motivates the exploration of alternative paradigms to enhance the robustness of CP-based scheduling under uncertainty.

Two directions are particularly relevant. First, the integration of learning techniques, such as machine learning (ML) and deep learning (DL), into optimization has gained interest in recent years (Bengio, Lodi, and Prouvost 2021; Lombardi and Milano 2018). Learning can help in addressing uncertainty in optimization problems (Elmachtoub and Grigas 2022; Sadana *et al.* 2025) while maintaining computational efficiency. This assumes the availability of (representative) data. Decision-focused learning (Elmachtoub and Grigas 2022; Mandi *et al.* 2023) is a relatively new approach to stochastic optimization. This approach embeds an optimization model, like CP, in a training procedure to minimize a decision goal (Elmachtoub and Grigas 2022). The direct application of such methods to stochastic scheduling problems is a promising research direction. Another direction is the broader idea of leveraging deterministic models to tackle more complex stochastic problems while preserving computational efficiency.

Second, insights from temporal constraint networks are promising for managing uncertainty in real-time, where achieving solutions within short runtimes is even more important. Temporal constraint networks represent temporal problems as graphs with time points (nodes) and temporal constraints (edges) (Dechter, Meiri, and Pearl 1991). An important concept is dynamic controllability (Morris, Muscettola, and Vidal 2001), which can be viewed as a guarantee that an online scheduling strategy will satisfy all constraints regardless of how uncertainty unfolds, provided a bounded uncertainty interval is given. We provide an example of a simple temporal network with uncertainty (STNU) (Morris, Muscettola, and Vidal 2001) and the concept of dynamic controllability in Figure 1.3. These methods, however, typically overlook other scheduling requirements, such as resource capacities—an area where constraint programming (CP) excels. Therefore, an interesting research direction is exploring how CP can be coupled with temporal networks for robust schedule execution.

1.1.4. The role of simulation

Simulation is both flexible and necessary for realistically capturing the complexity of biomanufacturing systems. A simulation model replicates the behavior of a real system, allowing experimentation in a virtual environment before being implemented in the

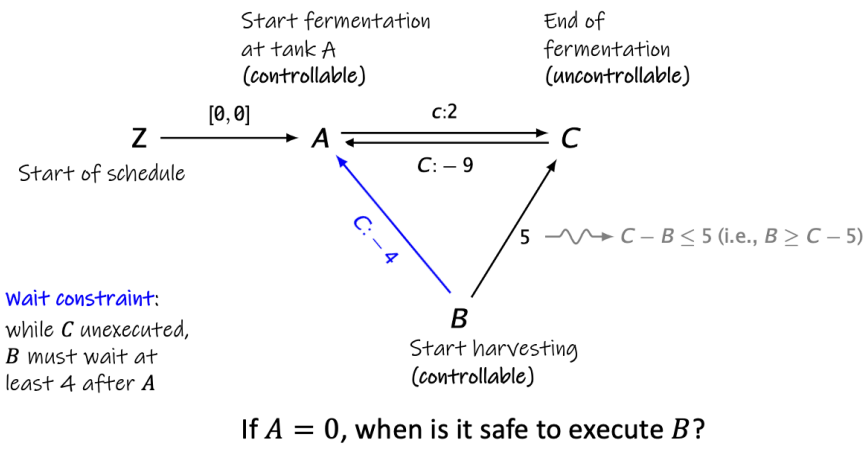


Figure 1.3 *Simple temporal network with uncertainty (STNU) and a wait constraint visualized which is required to enforce dynamic controllability. The example is adjusted from (Hunsberger and Posenato 2024). The labels A, B, C, Z, belong to nodes, i.e., time points, with Z being the start point at time 0. The link between A and Z means that the distance between A and Z should be exactly 0, so A will also be executed at time 0. A and B are controllable time-points, which means that a scheduler may choose their execution times. C is an uncontrollable even, we only know that the time distance between A and C will be between 2 and 9, indicated by a contingent link between A and C. There exists an ordinary link between B and C, which is a constraint that dictates that C must be executed at most 5 time units later than B. For this small example, there exists a simple execution strategy that will always result in a feasible assignment, namely by enforcing the blue wait constraints, which means that while C is unexecuted, B must wait at least 4 time units after A. The existence of such a strategy makes this small temporal network dynamically controllable.*

real world. Compared to mixed-integer linear programming or constraint programming, simulation models are generally less restrictive, allowing for the inclusion of detailed features, complex, nonlinear dynamics, and uncertainty. Two simulation paradigms are particularly relevant. Monte Carlo simulation is useful for analyzing systems or objective functions under uncertainty by repeated random sampling. Discrete-event simulation (Banks and Carson 1986) is well-suited for logistics and scheduling problems, as it represents a sequence of discrete events where each event changes the system state.

The combination of simulation techniques with optimization methods, such as simheuristics (Chica *et al.* 2020), offers a powerful and flexible framework for tackling optimization problems in complex and/or uncertain domains. In particular, for scheduling problems where stochastic programming may be less suitable, simheuristics can offer a more scalable and adaptable alternative. However, a research gap remains regarding how to design such simheuristics effectively, as there is currently no standard methodology for doing so.

Furthermore, it remains unclear how the integration of simulation and optimization can best support decision-making in real-world biomanufacturing environments.

1.2. Contributions of the dissertation

Prior studies have highlighted the challenges of developing generalizable mathematical models that accurately represent the highly detailed and complex nature of biomanufacturing environments. Furthermore, although constraint programming has shown great potential for scheduling, no standardized and scalable approach to handling uncertainty in scheduling exists. Motivated by these considerations, this dissertation explores new optimization approaches to effectively address scheduling challenges in biomanufacturing. This leads to the overarching research question:

How can optimization algorithms be designed to effectively solve scheduling problems in biomanufacturing?

This broad research question can be divided into smaller questions, which structure the contributions of this thesis. First, the lack of scalable and generalizable methods for handling uncertainty in scheduling leads to our subquestion:

How can we learn the parameters of deterministic constraint programming models that offer good solutions to stochastic scheduling problems with repair?

We make use of the efficiency of constraint programming solvers for scheduling, to answer this question in Chapter 2. We focus on a stochastic scheduling problem with a repair strategy. We explore a learning-based method as an alternative to scenario-based stochastic programming or using a deterministic simplification based on the expected values of the uncertain processing times.

In Chapter 2, we took the assumption that we have access to a solver that can offer optimal solutions to a deterministic problem within a reasonable amount of computation time. However, for large-scale stochastic combinatorial optimization problems, this assumption may not always hold true, and may already be difficult to find optimal solutions to the deterministic simplification. This leads to the following question that will be answered in Chapter 3.

How can we develop efficient algorithms for large-scale stochastic combinatorial optimization problems in which obtaining an optimal solution to a deterministic counterpart is already intractable, and solution evaluation relies on simulations?

We extend the idea of using tuned deterministic representations in Chapter 3. We focus on the simheuristic paradigm, which combines metaheuristics with simulation. The metaheuristic is typically designed to solve a deterministic counterpart, which is based on the expected values of the random components' parameters. This expected value simplification may not always be the best choice, and we challenge this approach in Chapter 3 by showing a counterexample and exploring alternatives.

We continue exploring strategies for handling uncertainty in scheduling in Chapter 4. However, in the two previous chapters, we assumed that a repair strategy or a simulation

model that takes as input the scheduling decisions always results in feasible solutions. Motivated by the biomanufacturing application, we identified a research gap in how to handle temporal uncertainties online in the presence of strict temporal constraints, such as maximum time lags. The challenge of this combination of constraints is that simple repair strategies, such as postponing tasks (as in Chapter 2), can lead to infeasibilities. This results in the following research question:

Which algorithms are most effective for scheduling problems that involve temporal uncertainty and strict temporal constraints?

Chapter 4 presents a general procedure for executing schedules under temporal uncertainty and with strict temporal constraints. Here, constraint programming is used for fixing a subset of the scheduling decisions offline. These decisions (resource assignment and sequencing) can then be embedded in so-called temporal networks, which are suitable for modeling temporal problems. In our contribution, we investigate how to utilize state-of-the-art techniques for solving these temporal problems, exploiting the concept of dynamic controllability to safely execute schedules online.

The final part of this dissertation focuses more explicitly on the case study of the enzyme production site of dsm-firmenich. The first question that we formulate is:

How can optimization approaches be designed to effectively plan production sequences in complex biomanufacturing environments over long planning horizons?

Chapter 5 targets this research question. In this chapter, metaheuristics are used to optimize production sequences, and we compare a global search strategy with a rolling horizon strategy. This contribution makes use of the high flexibility of discrete-event simulation and complements this with smart search strategies to provide tools for optimization that could directly be applied to real scheduling problems of the dsm-firmenich factory.

The contribution of Chapter 5 focuses on optimizing the production sequence only; the next chapter extends the scope by including additional scheduling decisions, such as machine assignment, sequencing, and task timing. Although constraint programming has demonstrated strong potential for scheduling problems, a clear research gap remains regarding its applicability to biomanufacturing—particularly given the numerous domain-specific characteristics involved, such as mixing and splitting operations, sequence-dependent setup times, and strict zero-wait constraints. This motivates the final chapter of the dissertation, which addresses an important research question aimed at further exploring the potential of constraint programming within the biomanufacturing context:

How effective are constraint programming solutions for the scheduling of a real biomanufacturing site that involves the mixing and splitting of batches into subbatches and has strict zero-wait constraints?

This question is considered in Chapter 6, which presents a constraint programming model of the enzyme production site of dsm-firmenich and a set of real benchmark instances. We focus on the performance of state-of-the-art constraint programming

solvers for these real instances. Furthermore, we provide an analysis of the impact of the zero-wait constraints on the runtime of the CP approach.

The main conclusions and reflections of this research are presented in Chapter 7. We have formulated suggestions for future research as precisely as possible to motivate other researchers and practitioners to continue along the line of this dissertation.

References

Awad, M., K. Mulrennan, J. Donovan, R. Macpherson, and D. Tormey (2022). “A constraint programming model for makespan minimisation in batch manufacturing pharmaceutical facilities”. In: *Computers & Chemical Engineering* 156, p. 107565.

Banks, J. and J. S. Carson (1986). “Introduction to discrete event simulation”. In: *Proceedings of the Winter Simulation Conference 1986*.

Bengio, Y., A. Lodi, and A. Prouvost (2021). “Machine learning for combinatorial optimization: a methodological tour d’horizon”. In: *European Journal of Operational Research* 290.2, pp. 405–421.

Birge, J. R. (1985). “Decomposition and partitioning methods for multistage stochastic linear programs”. In: *Operations Research* 33.5, pp. 989–1007.

Chaudhry, I. A., A. M. Khan, and A. A. Khan (2013). “A genetic algorithm for flexible job shop scheduling”. In: *Proceedings of the world congress on engineering*. Vol. 1, pp. 1–6.

Chica, M., A. A. Juan Pérez, O. Cordon, and D. Kelton (2020). “Why simheuristics? Benefits, limitations, and best practices when combining metaheuristics with simulation”. In: *Statistics and Operations Research Transactions* 44.2, pp. 311–334.

Conforti, M., G. Cornuéjols, and G. Zambelli (2014). “Integer programming models”. In: *Integer Programming*. Springer, pp. 45–84.

Dantzig, G. B. (1955). “Linear programming under uncertainty”. In: *Management Science* 1.3-4, pp. 197–206.

Dechter, R., I. Meiri, and J. Pearl (1991). “Temporal constraint networks”. In: *Artificial Intelligence* 49.1-3, pp. 61–95.

Elmachtoub, A. and P. Grigas (2022). “Smart "predict, then optimize"”. In: *Management Science* 68.1, pp. 9–26.

Escobet Canal, T., V. Puig Cayuela, J. J. Quevedo Casín, P. Palà Schönwälder, J. Romera Formiguera, W. Adelman, *et al.* (2019). “Optimal batch scheduling of a multiproduct dairy process using a combined optimization/constraint programming approach”. In: *Computers & Chemical Engineering* 124, pp. 228–237.

Garey, M. R. and D. S. Johnson (1990). *Computers and intractability; a guide to the theory of NP-completeness*. 1st. USA: W. H. Freeman Co.

Georgiadis, G. P., A. P. Elekidis, and M. C. Georgiadis (2019a). “Optimization-based scheduling for the process industries: from theory to real-life industrial applications”. In: *Processes* 7 (7), p. 438.

Gomory, R. (1958). “Outline of an algorithm for integer solutions to linear programs and an algorithm for the mixed integer problem”. In: *Bull. Am. Math. Soc* 64, pp. 275–278.

Harjunkoski, I., C. T. Maravelias, P. Bongers, P. M. Castro, S. Engell, I. E. Grossmann, J. Hooker, C. Méndez, G. Sand, and J. Wassick (2014). “Scope for industrial applications of production scheduling models and solution methods”. In: *Computers and Chemical Engineering* 62, pp. 161–193.

Hartmann, S. and D. Briskorn (2022). “An updated survey of variants and extensions of the resource-constrained project scheduling problem”. In: *European Journal of Operational Research* 297.1, pp. 1–14.

Herroelen, W., B. De Reyck, and E. Demeulemeester (1998). “Resource-constrained project scheduling: A survey of recent developments”. In: *Computers & Operations Research* 25.4, pp. 279–302.

Higle, J. L. (2005). “Stochastic programming: optimization when uncertainty matters”. In: *Emerging theory, methods, and applications*. Informs, pp. 30–53.

Hnich, B., R. Rossi, S. A. Tarim, and S. D. Prestwich (2009). “Synthesizing filtering algorithms for gobal chance-constraints”. In: *Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009, Lisbon, Portugal, September 20-24, 2009, Proceedings*. Ed. by I. P. Gent. Vol. 5732. Lecture Notes in Computer Science. Springer, pp. 439–453.

Hunsberger, L. and R. Posenato (2024). “Foundations of dispatchability for simple temporal networks with uncertainty”. In: *Proceedings of 16th International Conference Agents and Artificial Intelligence 2024*. Vol. 2, pp. 253–263.

IBM (2024). *IBM ILOG CPLEX Optimization Studio: CP Optimizer*. IBM. Armonk, NY. URL: https://www.ibm.com/products/ilog-cplex-optimization-studio.

Juan, A. A., J. Faulin, S. E. Grasman, M. Rabe, and G. Figueira (2015). “A review of simheuristics: Extending metaheuristics to deal with stochastic combinatorial optimization problems”. In: *Operations Research Perspectives* 2, pp. 62–72.

Juvin, C., L. Houssin, and P. Lopez (2025). “Flow-shop and job-shop robust scheduling problems with budgeted uncertainty”. In: *European Journal of Operational Research*.

Kondili, E., C. Pantelides, and R. Sargent (1993). “A general algorithm for short-term scheduling of batch operations”. In: *Computers & Chemical Engineering* 17 (2), pp. 212–227.

Laborie, P. (2015). *Modeling and solving scheduling problems with CP optimizer*.

Land, A. and A. Doig (1960). *An automatic method of solving discrete programming problems*, pp. 497–520.

Lombardi, M. and M. Milano (2018). “Boosting combinatorial problem modeling with machine learning”. In: *arXiv preprint arXiv:1807.05517*.

Mandi, J., J. Kotary, S. Berden, M. Mulamba, V. Bucarey, T. Guns, and F. Fioretto (2023). “Decision-focused learning: foundations, state of the art, benchmark and future opportunities”. In: *arXiv* 2307.13565.

Al-Maqtari, Q. A., A. Waleed, and A. A. Mahdi (2019). “Microbial enzymes produced by fermentation and their applications in the food industry-A review”. In: *International Journal of Agriculture Innovations and Research* 8.1, pp. 2319–1473.

Mercier-Aubin, A., L. Dumetz, J. Gaudreault, and C. Quimper (2020). “The confidence constraint: a step towards stochastic CP solvers”. In: *Principles and Practice of Constraint Programming - 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7-11, 2020, Proceedings*. Ed. by H. Simonis. Vol. 12333. Lecture Notes in Computer Science. Springer, pp. 759–773.

Morris, P., N. Muscettola, and T. Vidal (2001). *Dynamic control of plans with temporal uncertainty*.

Naderi, B., R. Ruiz, and V. Roshanaei (2023). “Mixed-integer programming vs. constraint programming for shop scheduling problems: new results and outlook”. In: *INFORMS Journal on Computing* 35.

Novara, F. M., J. M. Novas, and G. P. Henning (2016). “A novel constraint programming model for large-scale scheduling problems in multiproduct multistage batch plants: Limited resources and campaign-based operation”. In: *Computers & Chemical Engineering* 93, pp. 101–117.

Pantelides, C. (1993). “Unified frameworks for optimal process planning and scheduling”. In: *Proceedings of the Second International Conference on Foundations of Computer-Aided Process Operations*. 18-23 July, Crested Butte, United States, pp. 253–273.

Pinedo, M. (2016). *Scheduling: theory, algorithms, and systems*. 6th. Basel, Switzerland: Springer Nature.

Rajkumar, M., P. Asokan, T. Page, and S. Arunachalam (2010). “A GRASP algorithm for the integration of process planning and scheduling in a flexible job-shop”. In: *International Journal of Manufacturing Research* 5, pp. 230–251.

Sadana, U., A. R. Chenreddy, E. Delage, A. Forel, E. Frejinger, and T. Vidal (2025). “A survey of contextual optimization methods for decision-making under uncertainty”. In: *Eur. J. Oper. Res.* 320.2, pp. 271–289.

Savage, S. L. (2012). *The flaw of averages: why we underestimate risk in the face of uncertainty*. Engels. Wiley.

Van Slyke, R. M. and R. Wets (1969). “L-shaped linear programs with applications to optimal control and stochastic programming”. In: *SIAM journal on applied mathematics* 17.4, pp. 638–663.

Xiong, H., S. Shi, D. Ren, and J. Hu (2022). “A survey of job shop scheduling problem: The types and models”. In: *Comput. Oper. Res.* 142, p. 105731.

Zeballos, L. J., J. M. Novas, and G. P. Henning (2011). “A CP formulation for scheduling multiproduct multistage batch plants”. In: *Computers & Chemical Engineering* 35.12, pp. 2973–2989.

2

Learning from scenarios for stochastic repairable scheduling

When optimizing problems with uncertain parameter values in a linear objective, decision-focused learning enables end-to-end learning of these values. We are interested in a stochastic scheduling problem, in which processing times are uncertain, which brings uncertain values in the constraints, and thus, repair of an initial schedule may be needed. Historical realizations of the stochastic processing times are available. In this chapter, we answer the question: ***"How can we learn the parameters of deterministic constraint programming models that provide good solutions to stochastic scheduling problems with repair?"*** *We show how existing decision-focused learning techniques based on stochastic smoothing can be adapted to this scheduling problem. We include an extensive experimental evaluation to investigate in which situations decision-focused learning outperforms the state of the art, i.e., scenario-based stochastic programming.*

This chapter is based on the article: *van den Houten, K., Tax, D. M., Freydell, E., & de Weerdt, M. (2024, May). Learning from scenarios for repairable stochastic scheduling. In International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (pp. 234-242).*

2.1. Introduction

Decision-making can be challenging due to the stochastic nature of real-world processes. This complexity is evident across contexts, such as manufacturing, where unpredictable processing times make it difficult to meet strict customer deadlines. Formulating Constrained Optimization (CO) models for these problems is common, but unknown parameter values during decision-making pose challenges, as incorrect parameter estimates can lead to infeasibility.

In practice, such infeasibilities are repaired when reality unfolds. For instance, in a manufacturing system, tasks may be postponed in earlier stages to maintain the factory's flow. Various repair policies and schedule definitions are used across different contexts.

Historical data, represented as scenarios of unknown parameters like task duration, is often available. Simple averaging of these scenarios is a common yet naive approach that ignores uncertainty. Stochastic programming (Ruszczyński and Shapiro 2003) and robust optimization (Ben-Tal, Ghaoui, and Nemirovski 2009) offer alternatives, each with its challenges, such as scalability and overly conservative solutions. Moreover, modeling realistic repair possibilities exactly is not always possible in such two-stage optimization approaches.

Decision-focused learning (DFL), extensively reviewed by Mandi, Kotary, *et al.* (2023), introduces a novel paradigm for stochastic optimization. This approach embeds an optimization model, such as Constraint Programming (CP), into a training procedure to minimize regret loss (Elmachtoub and Grigas 2022). Challenges arise in backpropagation when dealing with combinatorial optimization problems, where solutions may change discontinuously. Recent research, including the score-function approach by Silvestri, Berden, Mandi, *et al.* (2023), shows promising directions for handling uncertainty in constraints. Both exploring DFL with uncertainty in constraints and analyzing the applicability of the score-function method are highlighted as valuable directions for further research (Mandi, Kotary, *et al.* 2023).

In this research, we explore various scenario-based approaches for stochastic scheduling. The contribution is threefold: 1) we apply DFL for the first time to a repairable stochastic scheduling problem with stochastic processing times, 2) we demonstrate how an existing DFL technique that uses stochastic smoothing can be used to serve a stochastic scheduling problem where historical realizations of processing times are used, and 3) we include an extensive experimental evaluation in which we assess differences in performance between deterministic, stochastic programming, and a DFL approach.

2.2. Illustrative example: scheduling with repair

We illustrate the effect of uncertainty in constraints with an example of scheduling two tasks on a single machine, where the average task lengths are $\bar{y}_1 = 4$, and $\bar{y}_2 = 5$. The machine is not available from $t = 5$ to $t = 10$. The task is to minimize the makespan. Using the mean values, the optimal decision is to schedule first task 2, and then task 1, which gives us a makespan of 14 (see Figure 2.1), while scheduling first task 1, and then task 2 results in a makespan of 15.

Now suppose that task 1 is deterministic, and task 2 is stochastic, following the discrete uniform distribution $y_2 \sim U(\{3,4,5,6,7\})$. It still holds that the expected task lengths are

$\bar{y}_1 = 4$, and $\bar{y}_2 = 5$. When we have $y_2 = 6$ and we schedule task 2 first, the effect of the repair strategy can be seen in Figure 2.2 and leads to a makespan of 20. Considering this repair, we can compute the expected values of the two alternative decisions and find that $\mathbb{E}$[first task 1, then task 2] = 15 and $\mathbb{E}$[first task 2, then task 1] = 16.6. So, given the underlying distributions, it is better to first schedule task 1 rather than task 2. We observe that relying solely on expected values to make a decision is not always a good idea when processing times are uncertain.

2.3. Problem statement.

The goal is to optimize an optimization (e.g. scheduling) problem

$$z^*(y) = \arg\min_z f(z, y) \; s.t. \; z \in C(y, z),$$

where $f(z, y)$ is the objective function given parameters y and decision z and the constraint set $C(y, z)$. However, the parameters (e.g., processing times) y are unknown at the time of solving. We see y as a realization of a random variable Y with distribution $P(Y)$. We are given a data set $\mathscr{D} = \{y_i\}_{i=1}^n$ with historical data on y (i.e., a set of scenarios), which we can use to optimize z. A naive approach would be to use the sample averages $\bar{y}$ to solve the deterministic model with estimation $\hat{y} = \bar{y}$ and use the solution $z^*(\bar{y})$ (which may require corrections when the true values become known). Section 2.2 shows that this naive approach may lead to suboptimality.

In this work, the aim is to minimize the so-called post-hoc regret task loss (Hu, J. C. H. Lee, and J. H. M. Lee 2023), which can be used to evaluate the quality of $z^*(\hat{y})$ based on realization y. This task loss comes from literature on *decision-focused learning* (Hu, J. C. H. Lee, and J. H. M. Lee 2023; Mandi, Kotary, *et al.* 2023; Silvestri, Berden, Mandi, *et al.* 2023; Silvestri, Berden, Signorelli, *et al.* 2026) for problems with unknown parameters in the constraints is- and is defined as:

$$PRegret(\hat{y}, y) = f(z_{corr}(\hat{y}, y), y) - f(z^*(y), y) + pen(z^*(\hat{y}), z_{corr}(\hat{y}, y)), \tag{2.1}$$

where y are the true coefficient values, $\hat{y}$ are the predicted values, $z^*(\hat{y})$ is the decision based on predicted values, and $z^*(y)$ is the optimal decision with perfect information such as defined by Bertsimas and Kallus (2019). Then, we have $f(z^*(\hat{y}), y)$, which are the costs for predicted decisions, and $f(z^*(y), y)$, which are true optimal costs. Due to uncertain parameters in the constraints, predicted decisions must sometimes be corrected using a repair function such that $z^*(\hat{y}) \to z_{corr}(\hat{y}, y)$. How this reparation is penalized is reflected in $pen(z^*(\hat{y}), z_{corr}(\hat{y}, y))$.

We formulate the problem as a learning problem, for which the idea is to predict the unknowns $\hat{y} = h_\theta(\mathscr{D})$ based on the data such that the task loss is minimized. The training goal is to minimize the expected post-hoc regret:

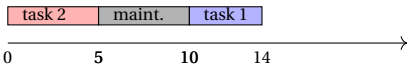


Figure 2.1 *Deterministic opt. schedule.*

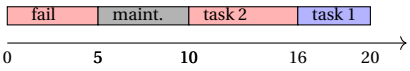


Figure 2.2 *Repair action when* $y_2 = 6$.

$$\arg\min_{\theta} \mathbb{E}_{y \sim P(Y)}[PRegret(\hat{y}, y)] \tag{2.2}$$

2.4. Decision-focused learning

In this section, we further elaborate on decision-focused learning (DFL) techniques that can be used to tackle Equation (2.3).

2.4.1. Zero-gradient problem.

DFL procedures that minimize the post-hoc regret loss by gradient-based optimization suffer from the zero-gradient problem. While minimizing the post-hoc regret with respect to θ to optimize the prediction $\hat{y} = h_\theta(\mathscr{D})$, a zero-gradient arises, because a combinatorial optimization solver is embedded in the loss computation (Elmachtoub and Grigas 2022), which is the $\frac{\delta z_{corr}(\hat{y},y)}{\delta \hat{y}}$ term in (2.3).

$$\frac{\delta PRegret(\hat{y}, y)}{\delta \theta} = \frac{\delta PRegret(z_{corr}(\hat{y}, y), y)}{\delta z_{corr}(\hat{y}, y)} \frac{\delta z_{corr}(\hat{y}, y)}{\delta \hat{y}} \frac{\delta \hat{y}}{\delta \theta}. \tag{2.3}$$

2.4.2. Stochastic smoothing

A novel approach (Silvestri, Berden, Mandi, *et al.* 2023; Silvestri, Berden, Signorelli, *et al.* 2026) shows that this zero-gradient problem can be solved with a stochastic smoothing trick. The crux is to use a stochastic estimator $\hat{y} \sim p_\theta(y)$ (where $p_\theta(y)$ is a parameterized distribution) instead of the point estimator $\hat{y} = h_\theta(\mathscr{D})$ (which is typical to decision-focused learning). Using a stochastic estimator makes the loss function an expectation, for which the gradient can be approximated with the score-function gradient estimator (also known as the likelihood ratio gradient estimator (Glynn 1990)) that uses:

$$\nabla_\theta \mathbb{E}_{\hat{y} \sim p_\theta(y)}[PRegret(\hat{y}, y)] = \mathbb{E}_{\hat{y} \sim p_\theta(y)}[PRegret(\hat{y}, y) \nabla_\theta \log(p_\theta(\hat{y}))] \tag{2.4}$$

for which the derivation can be found in (Silvestri, Berden, Mandi, *et al.* 2023). The most important assumption on $p_\theta(y)$ is that the probability density function must be differentiable with respect to θ. The right-hand side can be approximated using a Monte Carlo method (Mohamed *et al.* 2020). This score-function gradient-estimation approach is also the foundation of the REINFORCE algorithm (Williams 1992) and of various other reinforcement learning algorithms (Sutton and Barto 2018).

2.5. Method

In this section, we explain how the decision-focused learning technique with the score-function gradient estimation can be adapted to suit stochastic scheduling problems with a repair function.

2.5.1. Pseudocode

The pseudocode of the proposed method is presented in Algorithm 1. We refer to this method as DFL, although it differs from the traditional decision-focused learning setting, where a relation between features and the unknown values of an optimization problem is learned, as extensively surveyed in Mandi, Kotary, *et al.* (2023). The data $\mathscr{D} = \{y_i\}_{i=1}^{n}$ comprises historical examples of processing times y. We aim to learn which predictor $\hat{y}$ minimizes the expected post-hoc regret (Equation 2.3). We compute gradients by means of a stochastic estimator parameterized by θ, for which a common choice is the Normal distribution (Sutton and Barto 2018). During training, we sample $\hat{y} \sim \mathcal{N}(\mu = \theta_\mu \cdot \bar{y}, \sigma = \theta_\sigma \cdot \bar{\sigma})$, where both μ and σ are trainable, and initially set to the sample average $\bar{y}$ and sample standard deviation $\bar{\sigma}$. In each training step, we sample a point y_i and a prediction $\hat{y}$, compute the schedule $z^*(\hat{y})$, and update θ using the score-function gradient estimator that is provided in equation (2.4). After training, the stochastic estimator is treated as a point estimator by using $\hat{y} = \mu$. Note that the distribution is only needed during training for gradient computation with respect to the regret loss.

2.5.2. Illustration of the method

We explain the zero-gradient problem and smoothing technique with our example from Section 2.2. Suppose $(y_1, y_2) = (4, 6)$, but y_2 is unknown. Figure 2.3 shows how $\hat{y}_2$ affects the regret, which is the line with a discontinuity at $\hat{y}_2 = 5$, indicating a jump in scheduling priority. The blue curves show different stochastic estimators, and the small circles the expected regret values when we sample $\hat{y}_2$ from each distribution. The line through the circles represents the smoothed expected regret. We assess the applicability of Algorithm 1 using this example. The training data has an underlying distribution with $y_1 = 4$ and $y_2 \sim U(3, 4, 5, 6, 7)$. We expect the algorithm to find a scaling $\theta_2 > 1$ to prioritize scheduling task 1. A small experiment confirms that regret drops when μ_1 is above five as anticipated, see Figure 2.4.

Algorithm 1 DFL

Require: $\mathscr{D}_{train} = \{y_i\}_{i=1}^{n_{train}}$, $\mathscr{D}_{test} = \{y_i\}_{i=1}^{n_{test}}$

Initialize $\hat{y} \sim p_\theta(\hat{y})$ such that $\hat{y} \sim \mathcal{N}(\mu = \theta_\mu \cdot \bar{y}, \sigma = \theta_\sigma \cdot \bar{\sigma})$
for each epoch **do**
  for each batch in $\mathscr{D}_{train}$ **do**
    for each instance $(y_i, z^*(y_i))$ in batch **do**
      Sample $\hat{y}$ from $p_\theta(\hat{y})$
      Pass $\hat{y}$ to solver to get schedule
      Compute post-hoc regret$(\hat{y}, y_i)$
    Update θ with score-function:
    $\theta = \theta - \text{lr} \cdot \nabla_\theta PRegret(\hat{y}, y_i) \nabla_\theta \log(p_\theta(\hat{y}))$
Pass $\hat{y} = \mu$ to solver to get schedule
Evaluate post-hoc regret on $\mathscr{D}_{test}$

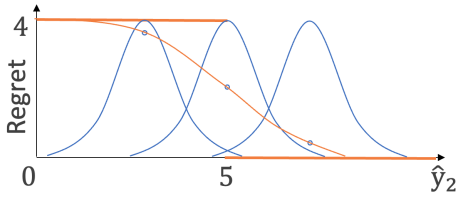
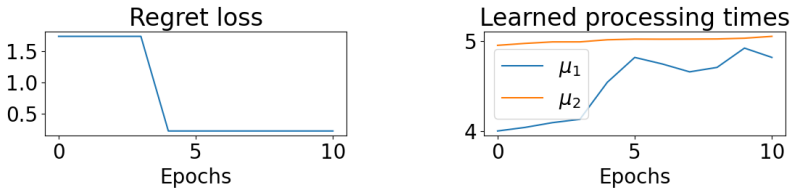


Figure 2.3 *Smoothing.*



(a) *Regret loss.* **(b)** μ_1 *vs* μ_2.

Figure 2.4 *Training curves.*

2.6. Experimental evaluation

To understand the potential of the proposed method DFL, we compare performance to deterministic and stochastic programming formulations (Section 2.6.2). We hypothesize that both stochastic programming and DFL outperform the more naive deterministic approach. Furthermore, we explore when DFL or stochastic programming performs better; we expect that DFL has better scalability to larger instances.

2.6.1. Problem instances and evaluation

We study a variant of the Stochastic Resource Constraint Project Scheduling Problem (RCPSP). We use two subsets of the PSPLib instances (being j301_1 to j303_10, and j901_1 - to j903_10) (Kolisch and Sprecher 1996). Furthermore, we use two sets of industry-inspired problem instances (small and large instances[1]), related to the factory of our industrial partner dsm-Firmenich. The data describing these instances is provided in our repository (van den Houten 2023). Originally deterministic, these instances are transformed into stochastic versions by sampling task durations from Normal distributions with mean d_j and standard deviation $\sqrt{d_j}$, where d_j is the deterministic processing time of task j from the original instance. We define a scenario as a vector of processing times for one problem instance. For each instance, three datasets are created: one with 100 training scenarios, another with 50 for validation and tuning, and a final set of 50 for evaluation.

The evaluation approach evaluates first-stage start times z_j decisions based on the schedule makespan. Tasks unable to start due to resource constraints or precedence relations undergo a repair policy with (multiple) one-unit time postponements resulting in

[1] 25 instances with 40 to 480 tasks, 13 resource groups with different capacities.

corrected start times z_j^{corr}. A penalty function measures the sum of start time deviations for all tasks j:

$$\text{pen}(z^*(\hat{y}), z^*_{corr}(\hat{y}, y)) = \rho \cdot \sum_{j \in J} z_j^{corr}(\hat{y}, y) - z_j(\hat{y}). \tag{2.5}$$

Here, ρ is the penalty coefficient, which we vary across experiments. Evaluation is conducted using the SimPy discrete-event simulation Python package (Matloff 2008).

2.6.2. Baseline methods

We study problem cases where historical realizations of stochastic processing times are available (without feature data). This section describes two other scenario-based methods that are included in the experiments.

Deterministic approach.

This is a simple baseline, where we compute scenario averages of the unknown optimization coefficients and solve the resulting deterministic model. We refer to this method as `deterministic`. The deterministic constraint programming (CP) model that uses these averages uses the following nomenclature: J: set of all tasks, R: set of all resources, S_j: set of successors of task j, j: subscript for tasks, r: subscript for resources, *parameters:* y_j: processing time of task j, $r_{r,j}$: resource requirement for task j, b_r: max capacity of resource r, $\text{minLag}_{j,i}$: min. difference between start times of tasks j and i, if i is a successor of j and *decision variables:* x_j: interval length for task j. The CP model is:

$$\text{Minimize } Makespan \qquad \text{s.t.} \tag{2.6a}$$

$$\text{Max(end\_of}(\text{x}_\text{j})) \leq Makespan \qquad j \in J \tag{2.6b}$$

$$\begin{aligned} \text{startOf}(\text{x}_\text{i}) \geq \text{endOf}(\text{x}_\text{j}); \quad & \forall j \in J\ \forall i \in S_j \quad or \\ \text{startOf}(\text{x}_\text{i}) \geq \text{minLag}_{\text{j,i}} + \text{startOf}(\text{x}_\text{j}); \quad & \forall j \in J\ \forall i \in S_j \end{aligned} \tag{2.6c}$$

$$\sum_{j \in J} \text{Pulse}(\text{x}_\text{j}, \text{r}_\text{r,j}) \leq \text{b}_\text{r} \qquad \forall r \in R \tag{2.6d}$$

$$x_j : \text{IntervalVar}(\text{J}, \text{y}_\text{j}) \qquad \forall j \in J \tag{2.6e}$$

In this model, (2.6b) defines the makespan which should be larger than the finish time of all tasks, and (2.6c) enforces precedence constraints between two tasks, where the $\text{minLag}_{\text{a,b}}$ is the minimal time difference needed between task a and b which is used for the industry instances. The CP pulse constraint (2.6d) models shared resource usage (Cplex, IBM ILOG 2009).

Stochastic programming.

The second baseline comprises a scenario-based stochastic programming formulation (again CP). We refer to this method as `stochastic`. The repair action is added to the stochastic model, which comprises the possibility to postpone activities, together with a penalty term for the deviations from the earliest-start-time decision that is included in the objective. Note that we use the same nomenclature as for the deterministic model, but we introduce the notion of scenarios $\omega \in \Omega$, and the first-stage earliest-start-time decision variable $z_j\ \forall j \in J$.

$$\text{Min} \frac{1}{|\Omega|} \sum_{\omega \in \Omega} Makespan(\omega) + \rho \cdot \sum_{\omega \in \Omega} \sum_{j} \text{startOf}(\text{x}_\text{j}(\omega)) - z_j \quad \text{s.t} \tag{2.7a}$$

$$\text{Max}_\text{j}(\text{end\_of}(\text{x}_\text{j}(\omega))) \leq Makespan(\omega) \qquad \forall \omega \in \Omega \tag{2.7b}$$

$$\text{startOf}(\text{x}_\text{i}(\omega)) \geq \text{endOf}(\text{x}_\text{j}(\omega)); \qquad \forall j \in J \; \forall i \in S_j \quad or$$

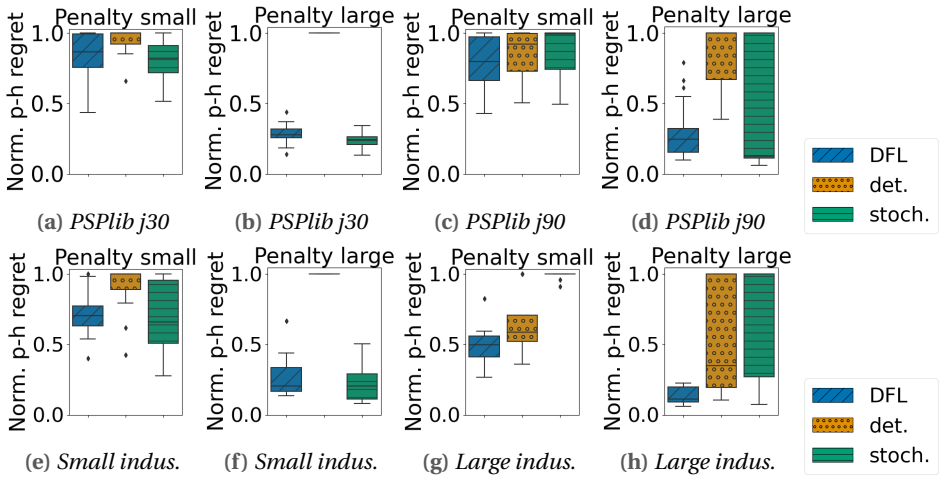
$$\text{startOf}(\text{x}_\text{i}(\omega)) \geq \text{minLag}_\text{j,i} + \text{startOf}(\text{x}_\text{j}(\omega)); \qquad \forall j \in J \; \forall i \in S_j \quad \forall \omega \in \Omega \tag{2.7c}$$

$$\sum_{j \in J} \text{Pulse}(\text{x}_\text{j}(\omega), \text{r}_\text{r,j}) \leq \text{b}_\text{r} \qquad \forall r \in R \quad \forall \omega \in \Omega \tag{2.7d}$$

$$x_j(\omega) : \text{IntervalVar}(\text{J}, \text{y}_\text{j}(\omega)) \qquad \forall j \in J \quad \forall \omega \in \Omega \tag{2.7e}$$

$$z_j \leq \text{startOf}(\text{x}_\text{j}(\omega)) \qquad \forall j \in J \quad \forall \omega \in \Omega \tag{2.7f}$$

2.6.3. Results



(a) *PSPlib j30* (b) *PSPlib j30* (c) *PSPlib j90* (d) *PSPlib j90*

(e) *Small indus.* (f) *Small indus.* (g) *Large indus.* (h) *Large indus.*

Figure 2.5 *Normalized post-hoc regret per instance set - penalty setting (smaller regret is better). The box spans from the 25th to the 75th percentile, visualizing the median and interquartile range.*

All experiments are done on a virtual server that uses an Intel(R) Xeon(R) Gold 6148 CPU with two 2.39 GHz processors and 16.0 GB RAM. All CP models are solved with a single-thread IBM CP solver (Cplex, IBM ILOG 2009). The runtime limits are set per problem size (max. 60min.) and provided in the README of the repository (van den Houten 2023), together with the tuned hyperparameters[2] for `deterministic`, `DFL`, and `stochastic`. Each boxplot in Figure 2.5 presents the results for the three methods on a single combination of instance set and penalty setting. The y-axis shows the distribution of normalized post-hoc regret across the test instances in that specific set. We use $\rho = \frac{1}{size}$ (small), where $size$ indicates the number of tasks, and $\rho = 1$ (large). We tested the

[2] Such as the number of scenarios.

significance of the performance differences among the algorithms for each setup (a-i) using a paired t-test with $\alpha = 0.05$, and the p-values are included in the repository (van den Houten 2023).

In smaller instances with small penalties, both DFL and stochastic perform well, with no significant difference. For the large penalty, stochastic tends to outperform both deterministic and DFL methods significantly on PSPlib $j30$ instances. Notably, under $\rho = 1$ for $j30$ instances, no repairs were needed across all instances, emphasizing the robust performance of stochastic when the instances are small enough. For larger instances like PSPlib $j90$, DFL becomes better, even significantly, for the small penalty. For the large penalty, there is still a subset of the instances for which stochastic finds very robust solutions that do not need repairs, but because for some of the instances stochastic performs much worse than DFL (which is also visible in Figure 2.5d), we observe no significant difference between DFL and stochastic looking at all $j90$ instances. We see a somewhat similar pattern in the industrial instances, where stochastic is again most advantageous for the smaller instances and incurs a high penalty, although not significantly better than DFL. For larger instances, stochastic performs even worse, especially with a small penalty, and DFL is significantly better. We investigated optimality gaps of the outputs of the stochastic model and found that even with a time limit of three hours, the gaps are on average approximately around 30%, with outliers of more than 90 % (for the largest industry instances), which shows the scalability issue of scenario-based stochastic programming.

2.7. Related work

Previous studies (Berthet *et al.* 2020; Elmachtoub and Grigas 2022; Mandi, Kotary, *et al.* 2023) primarily compared prediction-focused versus DFL approaches for problems with uncertainty in (linear) objectives. The knapsack problem is the most prominent (Demirovic *et al.* 2019; Mandi, Demirović, *et al.* 2020). As far as we know, the set-up without feature data is not studied in earlier work (Mandi, Kotary, *et al.* 2023). However, the crux of our problem setting is that uncertain parameters occur in constraints, which can lead to infeasibilities. Hu et al. (Hu, J. C. H. Lee, and J. H. M. Lee 2022) were the first who introduced a post-hoc regret that penalizes for infeasibilities. The methods introduced in their work rely on specific conditions, such as being recursively and iteratively solvable (Hu, J. C. H. Lee, and J. H. M. Lee 2023; Hu, J. C. H. Lee, and J. H. M. Lee 2022). This work applies an adapted DFL approach (Silvestri, Berden, Mandi, *et al.* 2023; Silvestri, Berden, Signorelli, *et al.* 2026) to a pure stochastic repairable scheduling problem, for which both the repairable scheduling setting and the context without features are novel application domains for DFL. It is important to note that, before this study, we did not know whether DFL could work for repairable scheduling.

2.8. To conclude, and continue

This study explores a novel application of DFL to stochastic resource-constrained scheduling with repairs, where uncertainty lies in the constraints and the derivative is not smooth in itself. Results indicate that stochastic programming is dominant when it can find the

optimal solution, most prominently when the penalty factor is high and the instances are small enough to find robust solutions that do not need reparation. In contrast, we have shown that DFL scales better and is a promising alternative to stochastic programming, even in this pure stochastic scheduling setup. Furthermore, we highlight the potential of DFL due to its flexibility across various settings and repair strategies, providing a distinct advantage over stochastic programming, where modeling exact repair functions is not always possible. We hypothesize that in a setting with stochastic processing times, the benefits of DFL for stochastic scheduling are further enhanced, as shown in earlier research on uncertainty in a (linear) objective (Mandi, Kotary, *et al.* 2023). Further interesting directions are investigating alternative gradient estimators or reinforcement learning-inspired algorithms.

References

Ben-Tal, A., L. E. Ghaoui, and A. Nemirovski (2009). *Robust optimization*. 1st. Princeton: Princeton University Press, pp. 1–542.

Berthet, Q., M. Blondel, O. Teboul, M. Cuturi, J.-P. Vert, and F. Bach (2020). "Learning with differentiable pertubed optimizers". In: *Advances in neural information processing systems 2020*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin. Vol. 33. The MIT Press, pp. 9508–9519.

Bertsimas, D. and N. Kallus (2019). "From predictive to prescriptive analytics". In: *Management Science* 66(3), pp. 1025–1044.

Cplex, IBM ILOG (2009). "V12. 1: User's Manual for CPLEX". In: *International Business Machines Corporation* 46.53, p. 157.

Demirovic, E., P. J. Stuckey, J. Bailey, J. Chan, C. Leckie, K. Ramamohanarao, and T. Guns (2019). "An investigation into prediction + optimisation for the knapsack problem". In: *Integration of AI and OR Techniques in Constraint Programming*. Ed. by L.-M. Rousseau and K. Stergiou. Springer Cham.

Elmachtoub, A. and P. Grigas (2022). "Smart "predict, then optimize"". In: *Management Science* 68.1, pp. 9–26.

Glynn, P. W. (1990). "Likelihood ratio gradient estimation for stochastic systems". In: *Commun. ACM* 33.10, pp. 75–84.

Hu, X., J. C. H. Lee, and J. H. M. Lee (2023). "Branch and learn with post-hoc correction for predict+optimize with unknown parameters in constraints". In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Ed. by A. A. Cire. Springer Cham.

Hu, X., J. C. H. Lee, and J. H. M. Lee (2022). "Predict+optimize for packing and covering LPs with unknown parameters in constraints". In: *arXiv* 2209.03668.

Kolisch, R. and A. Sprecher (1996). "PSPLIB - a project scheduling problem library". In: *European Journal of Operational Research*, pp. 205–216.

Mandi, J., E. Demirović, P. Stuckey, and T. Guns (2020). "Smart predict-and-optimize for hard combinatorial optimization problems". In: *Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI-20*.

Mandi, J., J. Kotary, S. Berden, M. Mulamba, V. Bucarey, T. Guns, and F. Fioretto (2023). "Decision-focused learning: foundations, state of the art, benchmark and future opportunities". In: *arXiv* 2307.13565.

Matloff, N. (2008). *Introduction to discrete-event simulation and the SimPy Language*.

Mohamed, S., M. Rosca, M. Figurnov, and A. Mnih (2020). "Monte carlo gradient estimation in machine learning". In: *Journal of Machine Learning Research* 21.1, pp. 5183–5244.

Ruszczyński, A. and A. Shapiro (2003). "Stochastic programming models". In: *Handbooks in Operations Research and Management Science* 10, pp. 1–64.

Silvestri, M., S. Berden, J. Mandi, A. İ. Mahmutoğulları, M. Mulamba, A. D. Filippo, T. Guns, and M. Lombardi (2023). "Score function gradient estimation to widen the applicability of decision-focused learning". In: *arXiv* 2307.05213.

Silvestri, M., S. Berden, G. Signorelli, A. İ. Mahmutoğulları, J. Mandi, B. Amos, T. Guns, and M. Lombardi (2026). "Score function gradient estimation to widen the applicability of decision-focused learning". In: *Journal of Artificial Intelligence Research* 85.

Sutton, R. and A. Barto (2018). *Reinforcement learning: an introduction*. 2nd. Cambridge, Massachusetts, United States: The MIT Press.
Van den Houten, K. (2023). *Learning from scenarios for repairable stochastic scheduling*. https://github.com/kimvandenhouten/Learning-From-Scenarios-for-Repairable-Stochastic-Scheduling.
Williams, R. J. (1992). "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine Learning* 8, pp. 229–256.

3

Do not use expectations in simheuristics

This chapter focuses on stochastic combinatorial optimization problems for which it is hard to find the optimal solution and for which expensive simulations are required to evaluate solutions. A simheuristic framework typically deterministically finds multiple solutions for one or a few scenarios and complements this with simulation across a larger set of scenarios. The literature provides few guidelines on what a standard simheuristic approach is. Typically, the expected values of the random components of the parameters are used. The hypothesis in this chapter is that this may not be the best choice due to the flaw of averages. For example, in the context of scheduling problems with uncertain processing times and deadlines, using the average processing time may risk deadline violations. In this chapter, we thus answer the question: ***"How can simheuristic algorithms be improved by using alternative deterministic representations, beyond the standard expected-value approximation, for stochastic combinatorial optimization problems?"*** *We propose using alternative deterministic representations, namely optimized quantile values instead of expected values, within a simheuristic framework. In particular, we show that we can use a simheuristic approach that dynamically switches between different deterministic representations and gives better results to a representative set of benchmark problem instances. This work introduces a new building block for developing simheuristic implementations of stochastic combinatorial optimization problems. Finally, we provide theoretical insights for a set of ordering problems to support (i) that searching for a good deterministic representation can be effective and (ii) that quantiles are suitable for generating diverse deterministic representations.*

This chapter is submitted: *Van den Houten, K., Berkhout, J., Tax, D.M.J., Heidergott, B., De Weerdt, M.M.. Do not use expectations in simheuristics.* It is important to mention that the theorems and proofs that were provided in this chapter and its appendix were mainly contributed by Joost Berkhout.

3.1. Introduction

Many real-world problems from various domains, such as logistics, manufacturing, healthcare, and finance, can be stated as combinatorial optimization problems (COPs) of the form:

$$\min_{\pi \in S} f(x, \pi), \tag{3.1}$$

where π denotes a solution from the discrete solution space S, x are the parameters of problem and $f(\cdot)$ is the objective function. For example, π can be a schedule, x are the processing times, and $f(\cdot)$ is the tardiness.

These real-life COPs are often NP-hard, meaning there is little hope for an efficient algorithm that allows finding the optimal solution for all realistically-sized problem instances under a fixed time budget (Papadimitriou and Steiglitz 1998). Another complicating factor is that the (input) parameters of COPs are often uncertain in practice (A. A. Juan, Faulin, *et al.* 2015), i.e., the deterministic input parameters x are, in fact, random variables denoted by X.

In this work, we focus on these *stochastic* COPs (SCOPs) of the form:

$$\min_{\pi \in S} \mathbb{E}\left[f(X, \pi)\right], \tag{3.2}$$

where now X represents the random component of the problem for which we assume that either data is available or the distributions are known.

The objective function $f(\cdot)$ is either a closed-form expression or a performance metric that can be accessed with simulation, given a realization $X^{(\omega)}$ of the sample space. A challenging aspect of SCOPs is that the expected value $\mathbb{E}\left[f(X, \pi)\right]$ is generally intractable. Sample-average approximations (SAA) (A. A. Juan, Faulin, *et al.* 2015) is an approach that replaces the expected value as objective by a sample-average

$$\mathbb{E}\left[f(X, \pi)\right] \approx \frac{1}{N} \sum_{i=1}^{N} f(X^{(\omega_i)}, \pi), \tag{3.3}$$

which renders the problem in (3.2) tractable. However, SAA often involves time-consuming simulations and the analysis of N sub-objectives $f(X^{(\omega_i)}, \pi)$, which increases the complexity of the underlying optimization. To overcome these challenges of SAA-like approaches, one resorts to replacing the true stochastic objective $\mathbb{E}\left[f(X, \pi)\right]$ with a deterministic counterpart. A standard approach is using the expected value of the random component $\mathbb{E}[X]$ as an SAA with $N = 1$ and optimizing the corresponding *deterministic* COP (DCOP) (A. A. Juan, Faulin, *et al.* 2015):

$$\min_{\pi \in S} f(\mathbb{E}[X], \pi).$$

Indeed, evaluating a solution π in DCOP is done quickly via one function evaluation, whereas finding a good approximation to $\mathbb{E}\left[f(X, \pi)\right]$ requires many function evaluations for different samples of X. Although optimizing $f(\mathbb{E}[X], \pi)$ is computationally easier, the resulting solution may (and often will) perform poorly in the corresponding SCOP, because in general $\mathbb{E}\left[f(X, \pi)\right] \neq f(\mathbb{E}[X], \pi)$.

Ignoring this is also known as *flaw of averages* (Savage 2012). The newsvendor problem (Ruszczyński and Shapiro 2003) provides an example of the flaw of averages; in this

problem, the newsvendor should decide on the quantity of newspapers to purchase under stochastic demand. For illustration purposes, we show later in this paper that given the stochastic demand as a random variable X solving $f(\mathbb{E}[X],\pi)$ as a surrogate is suboptimal for the true objective $\mathbb{E}\left[f(X,\pi)\right]$.

The combination of NP-hardness and the time-consuming objective approximations via simulations makes SCOPs challenging to solve in practice. Fortunately, so-called *simheuristics* have shown to be able to find good solutions to practical SCOPs in recent years (Chica *et al.* 2020; A. A. Juan, Keenan, *et al.* 2021; Rabe, Deininger, and A. A. Juan 2020). Simheuristics provide a general framework to solve large-scale SCOPs by combining DCOP metaheuristics with simulation (A. A. Juan, Faulin, *et al.* 2015). In particular, the DCOP metaheuristic is used as a relatively fast way to generate candidate solutions for SCOP. Instead of simulating all newly found solutions, only the promising solutions are simulated with a relatively low number of realizations $X^{(\omega)}$ of the random variables to approximate their expected objective values. When approaching the computation time limit, the most promising solutions are awarded more simulations to identify the best solution to SCOP. One of the key challenges in designing an effective simheuristic is determining both which solutions to simulate and the number of samples to use for these simulations (Clapper, Berkhout, and Bekker 2024).

Simheuristics thrives on the assumption that $f(\mathbb{E}[X],\pi)$ is a reasonable proxy for $\mathbb{E}\left[f(X,\pi)\right]$ (A. A. Juan, Faulin, *et al.* 2015). While the mean value is indeed the best approximation to a random variable in the sense of the L^2-norm, replacing the order of f and $\mathbb{E}$ only provides a reasonable numerical approximation if the variance of X is (very) small and/or if f is approximately linear. As a result, optimizing an expected value DCOP often yields poor results in SCOP, an aspect that has been insufficiently addressed in the simheuristic literature. For example, in scheduling with uncertain processing times and a non-linear tardiness objective, it is important to hedge against delays; thus, we are interested in the right tail of the processing time distributions when optimizing. In capacitated vehicle routing with stochastic demand, using an upper quantile of the demand distribution yields more conservative solutions that reduce the risk of unmet demand and additional depot visits for replenishment. In Section 3.3, we show concretely on the newsvendor problem that the optimal solution for the expected value DCOP performs poorly in the SCOP and can even result in negative expected profits, whereas using a suitably chosen demand quantile in the DCOP recovers the SCOP-optimal solution when optimized.

The central philosophy of this work is to go beyond the standard deterministic representation $f(\mathbb{E}[X],\pi)$ of $\mathbb{E}[f(X,\pi)]$ and instead use deterministic counterparts that yield better candidate solutions for the SCOP when using a simheuristic. We advocate using representations based on quantiles of the random variables X, which can be implemented just as easily as $\mathbb{E}[X]$ and are sensitive to differences in the underlying parameter distributions. Unlike the newsvendor problem, where the optimal DCOP representation can be derived analytically, many simheuristic applications do not allow for such an analytical solution.

Our main contribution is a quantile simheuristic approach that dynamically learns effective quantile-based DCOP representations. For a given $p \in [0,1]$, a so-called p-quantile is defined for a random variable with cumulative distribution function $G(\cdot)$ as $G^{-1}(p) = \inf\{x \in \mathbb{R} : G(x) \geq p\}$. Our method alternates between different values of a single

$p \in [0,1]$, yielding distinct p-quantiles for the random variables X, which are then given as input to the metaheuristic. The metaheuristic generates candidate solutions until a predefined (intermediate) stop criterion is met, producing a sequence of solutions with fitness evaluations based on the current p-quantiles. Promising candidates are simulated briefly and ranked, after which a new (promising) level p is chosen to generate new candidate solutions, starting from the final solution of the previous metaheuristic run. When the computational budget is exhausted, the best SCOP solution found is returned. By dynamically adapting the level p used in the DCOP based on simulation feedback, the method effectively learns which aspects of the distributions of X are most relevant for obtaining high-quality SCOP solutions in a simheuristic.

This paper is structured as follows: in Section 3.3 we introduce motivational examples for which the proposed quantile simheuristic approach could lead to optimal SCOP solutions. We present relevant literature in Section 3.4. The general framework explaining the quantile simheuristic is included in Section 3.6, followed by our experimental evaluation in Section 3.7. In Section 3.8, we provide preliminary theoretical insights into the appropriateness of our approach's philosophy, before concluding our work in Section 3.9.

3.2. Formal problem statement

In this work, we focus on finding efficient and general procedures for solving stochastic combinatorial optimization problems (SCOPs) of the form:

$$\min_{\pi \in S} \mathbb{E}\left[f(X,\pi)\right], \tag{3.4}$$

where π denotes a solution from a discrete solution space S, where X represents the random component of the problem for which we assume that either data is available or the distributions are known, and $f(\cdot)$ is the objective function. The objective function $f(\cdot)$ can either be a closed-form expression or is typically a performance metric that can be accessed with simulation, given a realization $X^{(\omega)}$ of the sample space, which implies that the expected value $\mathbb{E}\left[f(X,\pi)\right]$ is generally intractable.

3.3. Motivation for our quantile simheuristic approach

We use the Newsvendor problem as a motivating example problem of the form of Equation (3.4) for our quantile approach. The classical newsvendor problem is a typical example of a problem for which it is undesirable to use the expected value of the stochastic demand and for which, instead, a p-quantile approach can find the optimal solution.

Example 3.3.1. *Let us consider the* newsvendor *problem, see for example the works by Qin* et al. *(2011), Ruszczyński and Shapiro (2003), and Winston and Goldberg (2004). A newsvendor has to decide on the quantity π of newspapers that she purchases from a distributor at the beginning of a day at the cost of c per unit. She can sell a newspaper for the price of s per unit, and unsold newspapers can be returned to the vendor at the price of r per unit. It is assumed that $0 < r < c < s$. The demand for newspapers on a particular day is described by a discrete random variable X, with a distribution function $G(\cdot)$. The profit is*

thus a function of π and X given by

$$\begin{aligned} f(X,\pi) &= s\min(X,\pi) + r\max(\pi - X, 0) - c\pi \\ &= (s-r)\min(X,\pi) - (c-r)\pi. \end{aligned}$$

The goal of the newsvendor is to maximize the expected profit, i.e., to solve SCOP with $\pi \in \mathbb{N}^+$. In this (simple) example, the optimal solution $\pi^\star$ can be characterized as

$$\pi^\star = G^{-1}\left(\frac{s-c}{s-r}\right) = \inf\left\{x \in \mathbb{R} : G(x) \geq \frac{s-c}{s-r}\right\}, \tag{3.5}$$

where $G^{-1}(\cdot)$ is the generalized inverse distribution function of $G(\cdot)$ (Winston and Goldberg 2004). In the example of the newsvendor problem, we show that a standard DCOP solution may behave poorly in SCOP. Suppose now that $X \sim \text{uniform}\{\mathbb{E}[X] - d, \mathbb{E}[X] + d\}$ with $d \in \{0, \ldots, \mathbb{E}[X]\}$. If the newsvendor solves DCOP as a surrogate for SCOP, she chooses $\pi^{(Q)} = \mathbb{E}[X]$, which gives an expected profit of

$$\begin{aligned} &\mathbb{E}\left[f(X, \pi^{(Q)})\right] \\ &= -\frac{2\mathbb{E}[X](c-r)(2d+1) + (r-s)\left(2\mathbb{E}[X]d + 2\mathbb{E}[X](d+1) - d^2 - d\right)}{4d+2} \\ &= (s-c)\mathbb{E}[X] - (s-r)\frac{d(d+1)}{2(2d+1)}, \end{aligned}$$

which is even negative when

$$\mathbb{E}\left[f(X, \mathbb{E}[X])\right] < 0 \iff d > 2\frac{s-c}{s-r}\mathbb{E}[X] - 0.5 + \sqrt{\left(2\mathbb{E}[X]\frac{s-c}{s-r}\right)^2 + \frac{1}{4}},$$

i.e., when d is large enough. This lower bound of d can be simplified by using that $\sqrt{x+0.25} - \sqrt{x} < 0.5$ for all $x > 0$, this gives

$$\sqrt{\left(2\mathbb{E}[X]\frac{s-c}{s-r}\right)^2 + \frac{1}{4}} < 2\frac{s-c}{s-r}\mathbb{E}[X] + 0.5$$

and thus

$$d > 4\frac{s-c}{s-r}\mathbb{E}[X] \implies \mathbb{E}\left[f(X, \mathbb{E}[X])\right] < 0.$$

In words, when d is larger than $4\frac{s-c}{s-r}\mathbb{E}[X]$, the expected profit will be even negative. To be precise, this requires that $4\frac{s-c}{s-r}\mathbb{E}[X] \leq \mathbb{E}[X] - 1 \iff 4\frac{s-c}{s-r} \leq 1 - 1/\mathbb{E}[X]$, that is, the loss of an unsold newspaper should be significantly (about four times) larger than the gain of a sold newspaper.

This example demonstrates that using the expected value for a DCOP instance does not always yield the optimal SCOP solution and can even result in a loss in certain instance scenarios. In contrast, a carefully chosen quantile yields the optimal SCOP solution, which motivates the use of quantile-based representations in our approach.

3.4. Background and related work

This section contains an overview of the simheuristics literature in Section 3.4.1. Further, since this work proposes not to use expectations in simheuristics, we provide an overview of related work in which similar ideas have been applied outside simheuristics domains, see Section 3.4.2. The method that we present in Section 3.6 falls under the broader class of hyperheuristics, which we discuss in Section 3.4.3.

3.4.1. Simheuristics

A. A. Juan, Faulin, *et al.* (2015) introduce "simheuristics" as a general framework to solve SCOPs by combining simulation with metaheuristics enhanced by problem-specific information. The metaheuristic generates candidate solutions from which the promising ones are awarded simulations. A schematic overview of a simheuristic approach is presented in Figure 3.1. The design of a simheuristic consists of a metaheuristic strategy and a simulation strategy on which we elaborate in the next two subsections.

Metaheuristic strategy

The starting point of designing an effective simheuristic is choosing an effective solver for the DCOP problem. The solver can be any algorithm that iteratively produces candidate solutions (exact or (meta)heuristic), but is typically a metaheuristic (A. A. Juan, Li, *et al.* 2022).

(A. A. Juan, Faulin, *et al.* 2015) makes a correlation assumption between $\mathbb{E}\left[f(X,\pi)\right]$ and $f(\mathbb{E}[X],\pi)$, which is the main motivation for using a metaheuristic procedure that solves $f(\mathbb{E}[X],\pi)$. $f(\mathbb{E}[X],\pi)$ can be seen as an *ad hoc approximation* for $\mathbb{E}\left[f(X,\pi)\right]$ (Bianchi *et al.* 2009). In the standard simheuristic framework, solutions π are evaluated based on $f(\mathbb{E}[[X],\pi)$ (i.e., an SAA with $N=1$).

Simheuristics differ from generic simulation optimization in which all solutions are simulated; instead, only promising solutions are awarded with more simulation (SAA with $N>1$), which we briefly explain in the next subsection.

Simulation strategy

Next to the metaheuristic, a simheuristic implementation should define a simulation strategy in two aspects.

First, in simheuristics, the trade-off between exploration and exploitation is crucial, and it is influenced by the selection policy of promising solutions that will be simulated. According to (A. A. Juan, Faulin, *et al.* 2015), typically, the simulation strategy is that every new best DCOP solution is simulated, an approach also reported by Rabe, Deininger, and A. A. Juan (2020). In an overview article about best practices in simheuristics (Chica *et al.* 2020), it is suggested to use an elite set that can "provide decision-makers a range of alternative solutions".

A second design aspect is deciding on the simulation budget used per candidate solution. In the literature (Chica *et al.* 2020), heuristics are used, such as looking at the confidence interval and looking for a requested precision (number of simulations) to decide on how many simulation runs should be awarded to a candidate solution. More

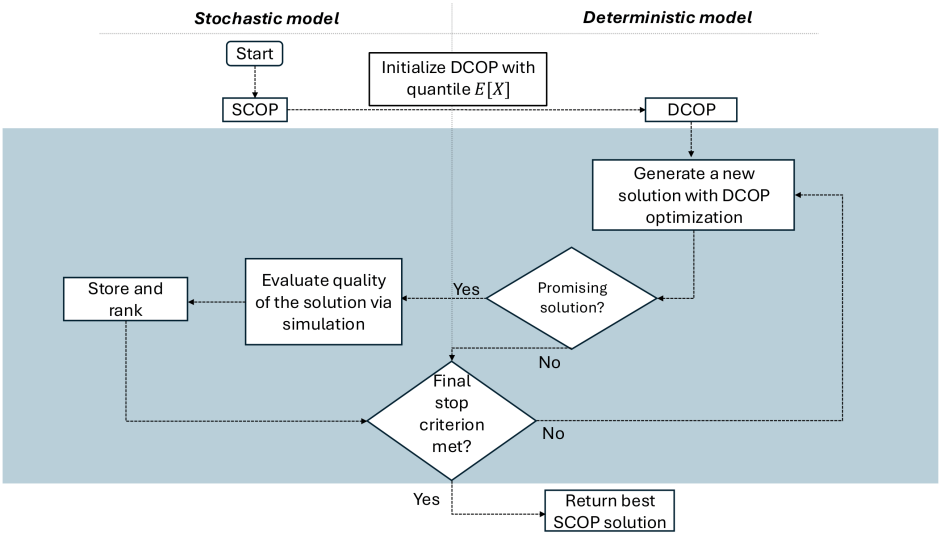


Figure 3.1 *Overview of a Simheuristic Framework. . Visualization is adapted from (A. A. Juan, Li,* et al. *2022).*

recent works on simheuristics showcase how the computation budget can be allocated smartly between optimization and simulation (Clapper, Berkhout, and Bekker 2024).

A standard simheuristic implementation

Although it is not exactly clear from the literature what a standard simheuristic is, we present a typical implementation in this section. We include the pseudocode of this 'standard' simheuristic approach to make the schematic overview from Figure 3.1 more concrete.

We introduce the notation of `standard` to refer to the simheuristic implementation that simulates all new best DCOP solutions with a fixed simulation budget per solution η and a maximum total budget of β. We use sensitivity analysis to determine how to set η, given the total budget β that determines when the stop criterion of the entire algorithm is met. Algorithm 2 shows the pseudocode of `standard` (in line with Figure 3.1). In this overview, the algorithm keeps track of π^b, but alternatively, an elite set could be used and possibly simulated more upon termination, which we indicated in line 7.

In the remainder of this paper, the focus is on challenging the use of $f(\mathbb{E}[X], \pi)$ and seeing if there exist better alternatives to $\mathbb{E}[X]$, rather than on designing new ideas for how to divide the budget between simulation and optimization. Therefore, we make some standard choices for designing the simheuristic framework, which we will enhance with alternative DCOP representations. We open a new dimension of designing simheuristic algorithms, which will be explained in Section 3.6.

Algorithm 2 `standard`

Require: A feasible initial solution π^{init}, objective function/simulator $f(\cdot)$, simulation budget per solution set to η, stop criterion: max total number of $f(\cdot)$ evaluations set to β.

$\pi^b \leftarrow \pi^{init}$ ▷ *Initialize the best solution.*
repeat
  Generate a new candidate solution π^c using metaheuristic
  if π^c is new best in terms of $f(\mathbb{E}[X], \pi)$ **then**
    Simulate π^c for random parameter settings $X^{(1)}, X^{(2)}, \ldots, X^{(\eta)}$
    Compute mean of the simulations $\bar{M}(\pi^c) = \frac{1}{\eta}\sum_{i=1}^{\eta} f(X^{(i)}, \pi^c)$
    Rank and store solution π^c ▷ *An elite set could be used.*
    if $\bar{M}(\pi^c) < \bar{M}(\pi^b)$ **then**
      $\pi^b \leftarrow \pi^c$ ▷ *Update best SCOP.*
until stop criterion is met ▷ *When β evaluations are used.*
return π^b ▷ *Return solution with best SCOP objective.*

3.4.2. Tuning deterministic representations

In this work, we propose a generic framework that uses alternative deterministic representations other than the expected value to find better solutions. To the best of our knowledge, this has not been done in the context of simheuristics before. However, we have seen similar approaches in other contexts.

As illustrated with the newsvendor problem in Section 3.3, using the expected values of the random variables as a deterministic representation may not always lead to the best solutions to the stochastic problem. A similar philosophy can be observed in the work by Powell and Ghadimi (2022) (in a different problem setting, however), who argues that the best functional form (i.e., the deterministic representation) can be application-dependent and formed based on user experience. Powell and Ghadimi (2022) provides an example where people look for the shortest path in a deterministic setting and depart a bit early to account for unforeseen issues. Their parameter choice is learned by looking at what parameter value gives the best performance in the stochastic problem (Ghadimi and Powell 2024). Similar examples of such an approach in which the parameters of a deterministic representation are tuned have been presented by Römer, Hagemann, and Porrmann (2023) and van den Houten *et al.* (2024).

Relevant implementations of simheuristics that propose tuning the deterministic representation to find better solutions for the stochastic problem do exist. A. Juan *et al.* (2011) optimize the Vehicle Routing Problem with Stochastic Demands (VRPSD) by solving the (deterministic) Capacitated Vehicle Routing Problem (CVRP) with reduced/conservative capacity levels so that the solution for VRPSD has some slack capacity. González *et al.* (2012) conducted similar research, but on the arc routing problem with random demands. Again, they use safety stocks to solve the deterministic version of the problem. Halvorsen-Weare and Fagerholt (2011) addresses the supply vessel planning problem, in which one should determine the optimal fleet size and mix of offshore supply vessels and their routing and scheduling to different installations. The authors generate robust solutions

by adding time slack to each route.

Such a slack-based approach is challenged by Van Den Akker, Van Blokland, and Hoogeveen (2013), who propose a simulation-based local search framework for the stochastic job shop scheduling problem, in which each candidate solution is evaluated via repeated discrete-event simulation using common random numbers. Their experimental results show that their sampling-based approach consistently outperforms pure average or slack-based methods based on percentile values of the stochastic processing times. Also interesting is a recent approach by Passage, van den Akker, and Hoogeveen (2025), which proposes an approximation method to speed up local search for stochastic planning problems with wait relations. Their method approximates the makespan by estimating the distribution of the maximum and is compared to a standard simheuristic approach that runs many simulations per candidate solution, resulting in a drastic speedup.

In this chapter, we take inspiration from such examples. Our goal is to design a generic framework that can be applied to multiple applications and integrated with different metaheuristics. We account for the underlying data distribution using a quantile-based approach. This is complemented by a dynamic algorithmic structure that alternates between different quantile strategies. In this sense, our approach can be seen as a variant of the method proposed by Van Den Akker, Van Blokland, and Hoogeveen (2013), where we switch between different quantile representations within a single run of the search algorithm. In contrast, Van Den Akker, Van Blokland, and Hoogeveen (2013) compare a purely percentile-based local search approach with a simulation-based local search approach that uses a relatively small number of samples per solution and common random numbers for stability.

In the next subsection, we discuss how our approach relates to hyperheuristics.

3.4.3. Hyperheuristics

We can draw inspiration from the field of hyperheuristics to learn how to utilize various deterministic representations in solving our stochastic problem.

Burke *et al.* (2013) define survey hyperheuristics as a "(high-level) methodology which, when given a particular problem instance or class of instances, and a number of low-level heuristics (or its components), automatically produces an adequate combination of the provided components to effectively solve the given problem(s)." Each heuristic rule results in a different neighborhood of candidate solutions. When an optimization in one neighborhood of candidate solutions stagnates, one could switch to an alternative neighborhood (Rousseau, Gendreau, and Pesant 2002). Such an approach is also known as a Random Gradient approach and randomly samples the heuristic rules. For an extensive overview on hyperheuristics, we refer to (Burke *et al.* 2013).

A renowned example of a hyperheuristic is Adaptive Large Neighborhood Search (Pisinger and Ropke 2007). This method is defined by a set of destroy and repair heuristics. The adaptiveness is reflected in scores for the destroy and repair heuristics, which are updated online and track how well each heuristic performs. The simheuristic we present in Section 3.6.4 can be complemented with such an adaptive layer. However, an important meta-analysis on the adaptive layer of ALNS by Turkeš, Sörensen, and Hvattum (2021) studied the impact of the adaptive nature on a wide range of ALNS implementations

and applications. Surprisingly, they conclude that it only improves efficiency by 0.41%. Therefore, it should be carefully evaluated whether an adaptive layer brings value to our setting.

More specifically, one could say that using different DCOP representations results in exploring different neighborhoods of the search space, potentially leading to better SCOP solutions. A more nuanced reasoning is as follows: the neighborhood that can be reached with the metaheuristic remains the same, but the evaluation of solutions changes. One evaluation may be more favorable than another for finding good solutions to the SCOP, and this can vary across different parts of the search space. In our simheuristics approach, these different evaluations lead to a similar effect as hyperheuristics/ALNS, i.e., more exploration and potentially more exploitation.

In a way, we can draw a parallel between our approach and using several heuristics in a hyperheuristic framework. Therefore, we find it important to draw a connection between our approach and the field of hyperheuristics. In the next section, we show an motivating example for such an approach.

3.5. Motivating example for dynamic simheuristics

This section presents an example that shows (i) that different neighborhoods in the solution search space require different DCOP representations to find the local SCOP optimum and (ii) that different DCOP representations allow for escaping a local optimum and finding the global optimum.

3.5.1. About the example

The example considers a single-machine scheduling problem with three jobs. The job durations are modeled as normal random variables with mean values $1, 2, 2$ and variances $1, 0.1, 0.1$. The jobs have importance weights $2, 2, 1$, and due dates $3, 3, 3$. There is a sequence-dependent setup time of one time unit when switching to or from job 2; the other setup times are 0. The goal is to find a schedule $\pi \in S_3$ that minimizes

$$f(X, \pi) = 100 * \text{“total weighted tardiness } \pi\text{”} + \text{“total completion time } \pi\text{”},$$

where X_i denotes the RV of job i's duration. Table 3.1 gives an overview of all schedules and their (different) objectives. The SCOP objectives are approximated based on 10^7 simulations. The optimal solution for DCOP with mean durations [1, 2, 2] does not coincide with the global optimum, but DCOP optimization with fixed durations [1, 0.9, 2] does. DCOP with fixed durations [a, b, c] may be abbreviated as [a, b, c]-DCOP. Note that these fixed durations are reachable with a carefully chosen quantile value since the support is assumed to be $\mathbb{R}$. Furthermore, it shows the chosen quantile value does not always have to be (conservatively) larger than its mean to find the (typically more robust) SCOP optimum. Due to the sequence-dependent setup times, starting or ending with job 2 is efficient for all objectives.

Table 3.1 *Enumeration of schedules. Overview of all schedules and their objectives. The best row/schedule per objective is marked in* ***bold****.*

Schedule	SCOP Objective	[1, 0.9, 2]-DCOP Objective	[1, 2, 2]-DCOP Objective
(0, 1, 2)	395.41	197.8	**310**
(0, 2, 1)	923.15	690.9	912
(1, 0, 2)	**394.73**	**197.7**	311
(1, 2, 0)	1013.99	680.7	1014
(2, 0, 1)	831.15	590.9	812
(2, 1, 0)	1013.21	570.8	1013

3.5.2. Used simheuristic for example

The simheuristic to optimize the example makes use of a local search algorithm. This algorithm uses a local search operator to switch two neighbor elements in a schedule's permutation; for example, (0, 2, 1) and (1, 0, 2) can be reached in 1 step from (0, 1, 2), but (2, 1, 0) cannot be reached in 1 step from (0, 1, 2) (but it can be reached in 3 steps). The arcs in Figure 3.2 show all possible local search steps and an arc's direction indicates in which direction the objective gets better. Thus, every objective gives a specific direction to an arc. The blue arcs in Figure 3.2 show that there are two local optima in SCOP: (1, 0, 2) and (2, 0, 1).

A typical strategy in simheuristics is only to simulate "promising" solutions as expressed by its objective in a deterministic representation. In the simheuristic for the example, a solution is only simulated if it gives a new best DCOP objective. Furthermore, the search only moves to better DCOP objectives and stores the best SCOP along the way.

3.5.3. Different neighborhoods require different deterministic representations

When the simheuristic's search is in (2, 1, 0), optimizing DCOP with mean durations [1, 2, 2] leads to the local SCOP optimum. In contrast, when DCOP with [1, 0.9, 2] is the deterministic representation, the search will stop in (2, 1, 0), and the local SCOP optimum will not be uncovered. So in this neighborhood where job 2 starts, DCOP with mean durations [1, 2, 2] is superior: it allows for quick evaluations and leads to the local SCOP optimum. However, when the search is in (0, 1, 2), DCOP with durations [1, 0.9, 2] leads to the other local SCOP optimum (1, 0, 2), while DCOP with durations [1, 2, 2] gets stuck in (0, 1, 2). So, in this neighborhood where job 2 is last, DCOP with mean durations [1, 0.9, 2] is superior.

The following analysis applies to the two neighborhoods:

- **Neighborhood with job 2 *first*:** In this case, jobs 0 and 1 will likely be tardy. So it is best to mainly focus on the total completion times of jobs 0 and 1 (which is a linear objective so that optimization with expectations will work). This leads to the local SCOP optimum of (2, 0, 1).
- **Neighborhood with job 2 *last*:** In this case, jobs 0 and 1 are likely to meet their due dates. In that case, it is better for robustness to put the more uncertain job 0 as

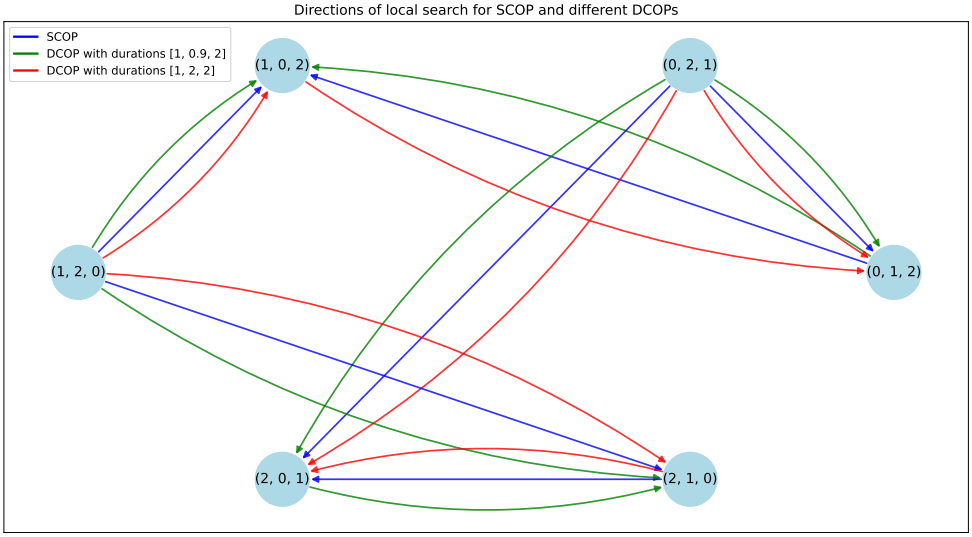


Figure 3.2 *Illustration local search directions. Illustration that different DCOP representations are effective in different search neighborhoods. The graph nodes indicate solutions, and the edges indicate the direction of a better solution in terms of a specific objective. In particular, from solution (2, 1, 0), DCOP with durations [1, 2, 2] gives the same (correct) direction as SCOP leading to the local (and global) SCOP optimum, whereas DCOP with durations [1, 0.9, 2] gets stuck. In contrast, starting in solution (0, 1, 2), DCOP with durations [1, 0.9, 2] leads to the correct search direction leading to the local SCOP optimum, whereas DCOP with durations [1, 2, 2] does not.*

second, leading to the global SCOP optimum of (1, 0, 2).

This example illustrates that it may depend on the current search neighborhood, which deterministic representation for SCOP is effective. In other words, different search regions may require different DCOP representations to solve SCOPs effectively with simheuristics.

3.5.4. Escaping local optima and reaching a global optimum in example

Figure 3.3 shows that optimizing DCOP with durations [3, 2, 2] gives a possibility to escape the local SCOP optimum (2, 0, 1), which allows finding the global SCOP optimum. This shows that different DCOP representations may help escape local DCOP and SCOP optima.

3.5.5. Demonstration of Reindexing

This section showed that DCOP with durations [1, 2, 2] does not find the optimal solution in the example. Theorem 1 states that [1, 2, 2]-DCOP can be reindexed so that its best solution coincides with the optimal SCOP solution. This example can be reindexed as follows to illustrate Theorem 1. Applying $\texttt{reindex}(X, \pi_{[2]}, \pi_{[1]})$, where $\pi_{[2]} = (1,0,2)$ is the

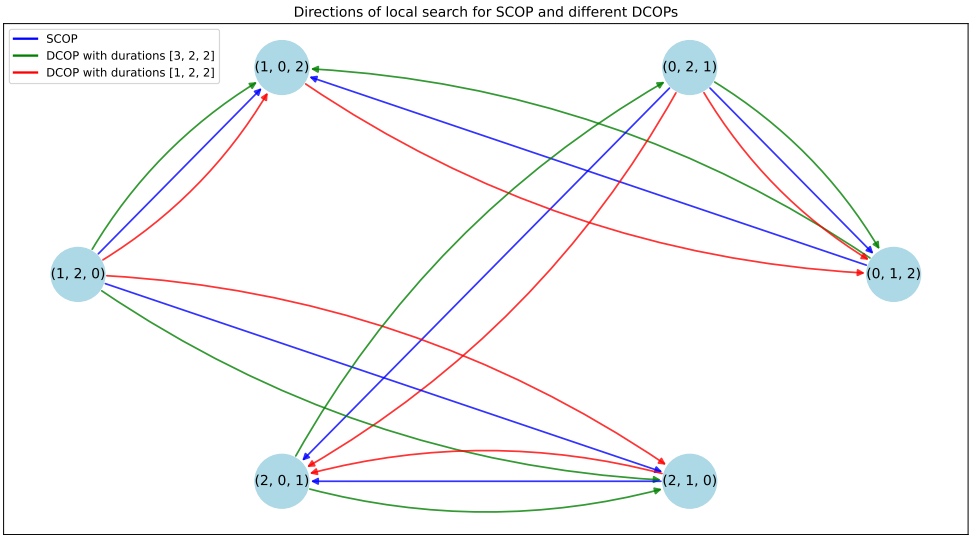


Figure 3.3 *Illustration with escape possibility. DCOP with durations [3, 2, 2] gives a possibility to escape the local SCOP optimum (2, 0, 1), which allows us finding the global SCOP optimum.*

Table 3.2 *Enumeration of schedules after reindexing. Overview of all schedules and their objectives after reindexing. In particular, [2, 1, 2]-DCOP is obtained after reindexing [1, 2, 2]-DCOP and shows its best solution coincides with the optimal SCOP solution. The best row/schedule per objective is marked in* ***bold****.*

Schedule	SCOP Objective	[1, 2, 2]-DCOP Objective	[2, 1, 2]-DCOP Objective
(0, 1, 2)	390.61	310	311
(0, 2, 1)	916.99	912	1014
(1, 0, 2)	389.88	311	310
(1, 2, 0)	1009.05	1014	912
(2, 0, 1)	823.18	812	1013
(2, 1, 0)	1007.74	1013	812

second best solution in [1, 2, 2]-DCOP and $\pi_{[1]} = (0,1,2)$ the best, leads to durations [2, 1, 2] which gives the results in Table 3.2, which also includes the SCOP objective and DCOP objective for [1, 2, 2] for comparison. After the reindexation, the optimal SCOP solution is found.

3.6. Quantile simheuristic approach

Let π^* be an overall best solution for SCOP, i.e., the solution that solves formulation (3.2). The goal of our proposed methods is to find an alternative deterministic representation that is as computationally cheap as an expected value representation $f(\mathbb{E}[x], \pi)$ (an SAA

with $N = 1$). We propose a p-quantiles deterministic representation that yields the optimal solution for SCOP. Thus, our method solves:

$$\min_{p \in [0,1]} \mathbb{E}[f(X, \arg\min_{\pi \in S} f(G_X^{-1}(p), \pi)] \tag{3.6}$$

where $G_X^{-1}(p)$ refers to the inverse cumulative distribution functions of the elements in X for probability p, and the resulting p is the best proxy of the distributions of X in light of the given optimization problem. In a way, this is the problem-dependent projection of distributions to scalars.

3.6.1. Using quantiles

To solve a SCOP efficiently using simheuristics, it is crucial that solving the representative DCOP yields relatively good solutions for the SCOP. To that end, we propose `quantile` (see Section 3.6.4 later in the text), a simheuristic implementation using several representative DCOPs, arguing that this can lead to more effective exploration, resulting in better SCOP solutions.

In the following, we assume that the metaheuristic is given an additional input p, which determines the values used as the DCOP representation, based on the p-quantiles of the random variables. In other words, every time we give the metaheuristic a new p-setting, it evaluates candidate solutions π based on $f(G^{-1}(p), \pi)$. For sake of notation, we introduce DCOP($\mathbb{E}[X]$) to refer to the DCOP counterpart that minimizes $f(\mathbb{E}[X], \pi)$ and DCOP(p) to the alternative counterpart that minimizes $f(G^{-1}(p), \pi)$. While Equation 3.6 allows for a distinct p_i per element of X, we employ a scalar $p \in [0, 1]$ in our algorithm. From a practical perspective, a shared quantile level substantially reduces the search space (see also Sections 3.6.2 and 3.6.3 for how this choice is implemented), which is particularly desirable in high-dimensional settings. The extension to a p_i for each element in the vector X is suggested for further research. We further elaborate on this in Section 3.8.

3.6.2. Random gradient

We propose to alternate between different levels of p, to which we refer as p-strategies throughout the algorithm, i.e., every iteration i, we select a new p-strategy p_i and we continue with a DCOP(p_i) metaheuristic (i.e., a metaheuristic that optimizes $\min_{\pi \in S} f(G^{-1}(p_i), \pi)$). If the DCOP($p_i$)-metaheuristic continues from the current solution π, it must evaluate the quality of that solution with respect to $f(G^{-1}(p_i), \pi)$ because in the previous iteration π was evaluated with $f(G^{-1}(p_{i-1}), \pi)$. Therefore, using the new p-strategy to generate only one new solution is inefficient. To overcome this inefficiency, we include an intermediate stop criterion in our approach, and only if the stop criterion is met, we move to the next iteration and a new p-strategy. In other words, we generate several new solutions using one strategy before moving on to the next iteration. This intermediate stop criterion is related to a Random Gradient approach (Burke *et al.* 2013), i.e., we continue as long as we improve with one quantile level p_i, and only move to the next iteration if we do not get any more DCOP(p_i) improvement for a certain number of iterations τ.

Each p-strategy can be seen as a different neighborhood evaluation of the search space, which allows for escaping a local optimum and finding a local/global SCOP optimum.

Section 3.5 presents an example that shows (i) that different neighborhoods in the solution search space require different DCOP representations to find the local SCOP optimum and (ii) that different DCOP representations allow for escaping a local optimum and finding the global optimum. This serves as an additional motivation for our dynamic approach.

3.6.3. An adaptive layer

Instead of choosing each new p-strategy randomly, an adaptive variant of our quantile approach can utilize adaptive selection strategies commonly employed in the context of ALNS. This approach stimulates the use of p-quantiles that have been successful before in optimizing SCOP. We refer to this approach as `adaptive`.

A typical example is the Roulette Wheel approach, which alternates between strategies by random sampling from a weighted distribution. Implementing the Roulette Wheel for our case, we define the p-strategies as neighborhoods ($\omega \in \Omega$), so each ω-neighborhood corresponds to a p-setting, that is used to set-up a DCOP(p) metaheuristic, i.e., a DCOP metaheuristic that minimizes $f(G^{-1}(p), \pi)$. The weights (ρ_ω) for each neighborhood are initialized as $\rho \leftarrow (1, \ldots, 1)$ and get updated according to its performance.

Then, in each iteration, a neighborhood (p-strategy) is selected using the weights as probabilities; see the following definition.

Definition 1. `RouletteWheelSelect`(ρ, Ω) *samples from a discrete distribution with probabilities set to* $\frac{\rho_\omega}{\sum_{\omega \in \Omega} \rho_\omega}$ *for neighborhood* ω *and returns a* p*-strategy.*

The weights are updated following the Roulette Wheel scheme, where first a score σ is assigned to the outcome of the iteration, such as described in Table 3.3. Then, the weight for the current neighborhood ω is updated by a convex combination of the current weight ρ_ω and the score σ, using α as a weight decay parameter:

Definition 2. `RouletteWheelUpdate` (ρ, σ) *denotes the Roulette Wheel update function following the formula* $\rho_\omega = \alpha\rho_\omega + (1-\alpha)\sigma$*, where* ω *refers to the neighborhood that was used in the iteration.*

Note that our approach is not limited to the Roulette Wheel and/or Random Gradient, and users of our method could use alternative methods, such as by taking inspiration from bandit theory, e.g., using the α-Upper Confidence Bound bandit scheme (Hendel 2022).

Table 3.3 *Scoring Scheme Quantile Strategies in* `adaptive` *Approach. Note that* σ_4 *only makes sense when an elite set is used with a size larger than 1.*

Score	Description
σ_1	We found a new global best SCOP.
σ_2	We found a new best DCOP, but not a new best SCOP.
σ_3	We did not find a new best DCOP.
σ_4	We did not find a new best SCOP, but the new solution is an elite solution.

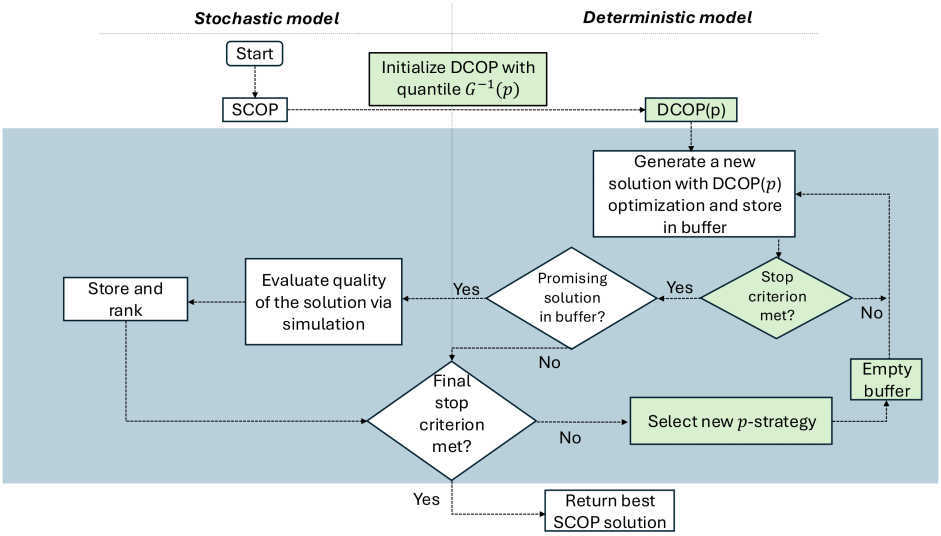


Figure 3.4 *Overview of the Quantile Simheuristic. The green boxes indicate the components that are unique for* `quantile` *and that are not present in* `standard`*.*

3.6.4. Pseudocode `quantile` simheuristic

We present a schematic overview of our novel simheuristic approach `quantile` in Figure 3.4. The approach `adaptive`, follows the same structure but is complemented with the `RouletteWheelSelect` and `RouletteWheelUpdate` functions.

An outer loop of the simheuristic controls the p-quantiles setting. The rhombus with "*Stop criterion met?*" determines how long to continue optimization in one p-quantiles DCOP representation. During optimization in one representation, we store solutions in a buffer $\mathscr{B}$.

The inner loop continues until the intermediate stop criterion is met, i.e., there has been τ number of DCOP(p) metaheuristic iterations without improvement in terms of $f(G^{-1}(p),\pi)$), following Section 3.6.2. Then, the best solution from the buffer $\mathscr{B}$ is selected, and if it is promising, i.e., a new best solution in terms of $f(G^{-1}(p),\pi)$, it is sent to the simulation.

Then, the outer loop selects a new random p-strategy, and importantly, the p-quantiles are used to set the new DCOP representation and continue the metaheuristic. In case adaptive is applied, the scores are updated, and the p-strategy is chosen according to the Roulette Wheel scheme. When the total budget β is exhausted, we return the best SCOP solution π^b. The pseudocode is included in Algorithm 3. In an extended version of this algorithm, an elite set could be used, complemented with an intensive simulation end phase in which the elite solutions are awarded with additional simulation.

It is important that `quantile` utilizes a different simulation strategy compared to `standard` (in which all new best DCOP solutions are simulated). We introduced the intermediate stop criterion to allow for continuing for multiple iterations with the current

Algorithm 3 `quantile` Simheuristic

Require: A feasible initial solution π^{init}, objective function/simulator $f(\cdot)$, short simulation budget η, set of p-strategies Ω, inner loop stop criterion: max number of $f(\cdot)$ evaluations without improvement set to τ, outer loop stop criterion: max total number of $f(\cdot)$ evaluations set to β.

Initialize weights $\rho \leftarrow (1,\dots,1)$
Initialize current solution $\pi \leftarrow \pi^{init}$
Initialize best solution $\pi^b \leftarrow \pi^{init}$
repeat
    Select new p-strategy from Ω.
    Evaluate $f(G^{-1}(p),\pi)$ ▷ *Evaluate the current solution with the new p-setting.*
    Empty solution buffer $\mathscr{B} \leftarrow \{\}$
    repeat
        Generate a new candidate solution π using metaheuristic search with strategy p
        $\mathscr{B} \leftarrow \mathscr{B} \cup \{\pi\}$ ▷ *Store solutions from inner loop.*
    until inner loop stop criterion is met ▷ *Max τ iterations without DCOP(p) improvement.*
    Find candidate $\pi^c \in \mathscr{B}$, where $\pi^c = \min_{\pi \in \mathscr{B}} f(G^{-1}(p),\pi)$
    if π^c is new best in terms of on $f(G^{-1}(p),\pi^c)$ **then**
        Simulate π^c for random parameter settings $X^{(1)}, X^{(2)}, \dots, X^{(\eta)}$
        Compute mean of the simulations $\bar{M}(\pi^c) = \frac{1}{\eta}\sum_{i=1}^{\eta} f(X^{(i)},\pi^c)$
        Rank and store solution π^c ▷ *An elite set could be used.*
        if $\bar{M}(\pi^c) < \bar{M}(\pi^b)$ **then**
            $\pi^b \leftarrow \pi^c$ ▷ *Update best SCOP.*
    Update selection scheme. ▷ *Only for `adaptive`.*
until outer loop stop criterion is met
return π^b ▷ *Return solution with best SCOP objective.*

p-strategy and avoid inefficiency caused by warm-up costs after each new p-strategy, Of course, comparing `quantile` and `standard` would be unfair if the simulation strategy is different; thus, we later introduce another baseline `mean`, which follows the same simulation strategy as `quantile` but only uses a DCOP metaheuristic with the $\mathbb{E}[X]$ as deterministic representation.

3.7. Experimental evaluation

In this research, the goal is to investigate whether we can improve the standard simheuristic framework by exploring alternative deterministic representations. In our experimental evaluation, we target this goal by setting up experiments that help answer the following research questions (RQs):

1. How does a simheuristic approach using deterministic metaheuristics with quantile representations compare to baseline simheuristics that only use the expected value?

2. How much does an adaptive layer help to improve our simheuristic approach?

3. What is the advantage of a quantile approach compared to running in parallel the simheuristic with only a single p-strategy for the quantiles?

4. What is the effect of the underlying distribution on the performance of our approach compared to the baselines?

The research questions above motivate the choices for experimental settings that are presented in Section 3.7.1. We present a set of benchmark problems representative of real-world problems in Section 3.7.1 and a set of baseline methods utilizing the expected value in Section 3.7.1. Our evaluation procedure is described in Section 3.7.1. We present all relevant hyperparameter configurations in 3.7.1, including the differences between `quantile` and `adaptive`, which are relevant for answering RQ2. Section 3.7.2 includes our experimental results and answers to the research questions.

3.7.1. Experimental settings

Benchmark problems

For the problems we use, we distinguish between stochastic parameters X and deterministic parameters θ. The deterministic parameters of all benchmark instances are taken from existing instances from the literature. For the stochastic parameters, we use a generic data generation procedure which will be described below. Note that the experiments in this research are designed in such a way that we study the effectiveness of using alternative deterministic representations in an isolated fashion; we are not claiming that we are designing a simheuristic implementation that is the state-of-the-art for the problems described in this section.

Stochastic permutation flow shop scheduling problem First, we consider the stochastic permutation flow shop scheduling problem with tardiness minimization (`spfsp`). In the permutation flow shop scheduling problem, a collection of jobs must be processed on a series of machines. Each job requires a specific amount of processing time on each machine. Additionally, all jobs must follow the same sequence on each machine, beginning with the first machine and concluding with the last machine in the series. We use Taillard instances `http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html` of the permutation flowshop scheduling problem. The deterministic parameters θ are the following: the number of jobs n, the number of machines m, and the deadlines for each job d_j*.

In the Taillard instances, deterministic values for the processing times are given: p_{jk} is the processing time of job j on machine k. In the stochastic variant of the problem, the processing times are the stochastic parameters X. We assume the processing times are i.i.d. as: $X_{jk} \sim \text{Uniform}\left\{\min\{1, \lfloor p_{jk} - \epsilon\sqrt{p_{jk}}\,\rceil\}, \lfloor p_{jk} + \epsilon\sqrt{p_{jk}}\rceil\right\}$,
where ϵ is the noise level. The solution π consists of a set of starts s_{jk} and finish times f_{jk} for all jobs on all machines given the constraints. The objective is to minimize the maximum tardiness, which defines $f(X,\theta,\pi)$:

$$f(X,\theta,\pi) = \max_j \max\{f_{jm} - d_j, 0\}.$$

Stochastic capacitated vehicle routing problem Second, we consider the stochastic capacitated vehicle routing problem where the objective is to minimize the costs (`scvrp`). The customer demands are random variables. The Capacitated Vehicle Routing Problem (CVRP) is described with a depot, a set of customers, and a distance graph between the depot and all customers (also across different customers). Each customer has a demand q_i. There are unlimited vehicles available, but the goal is to minimize the total distance that is traveled by these vehicles. Each vehicle has a maximum capacity. We use deterministic problem instances that come from the benchmark library `http://vrp.atd-lab.inf.puc-rio.br/index.php/en/` that describe the problem parameters, summarized by θ, being: the number of customers n, the distance matrix D between each customer and the depot, the maximum capacity c of each vehicle.

In the VRP instances, the demands q_i for customer i are given as deterministic parameters. In our experimentation, we use these to define independent demand distributions as $X_i \sim \text{Uniform}\left\{\min\{1, \lfloor q_i - \epsilon\sqrt{q_i}\,\rceil\}, \lfloor q_i + \epsilon\sqrt{q_i}\,\rceil\right\}$, where ϵ defines the noise level.

A feasible solution π comprises a set of routes for which each route belongs to one vehicle. For that reason, the total demand of the customers on one route cannot exceed the vehicle capacity. In the stochastic CVRP (SCVRP), the demands are stochastic. A feasible solution to the DCOP can become infeasible for the SCOP because the demand assigned to one route can now exceed the capacity. To that end, a penalty is introduced for exceeding capacity.

The objective of the SCOP is defined as the total distance plus a penalty for excessive capacity, such that:

$$f(X,\theta,\pi) = \; d(X,\theta,\pi) + \text{penalty} \cdot e(X,\theta,\pi),$$

where $d(X,\theta,\pi)$ is the total distance and $e(X,\theta,\pi)$ is the total excessive demand. The problem instances data is further described in Appendix 3.A.

Baseline methods

We consider three baseline methods in the evaluation.

We introduce `dcop` to indicate the approach of spending the full computation budget β on DCOP optimization and returning the best DCOP solution (based on $f(\mathbb{E}[X], \pi)$).

A second baseline follows the more standard approach from A. A. Juan, Keenan, *et al.* (2021), to which we refer with `standard`, see also Algorithm 2. This `standard` approach uses (similar to `dcop`) a DCOP metaheuristic that optimizes optimizes $f(\mathbb{E}[X], \pi)$. All new best DCOP solutions obtained with the metaheuristic are simulated briefly, and the best so far in terms of SCOP objective is stored and returned when the budget β is exhausted (following the standard framework from Figure 3.1).

The third baseline is the `mean` approach: here, we use the same simulation strategy as is used for `quantile`, see Algorithm 3, but we set $\Omega = \{\mathbb{E}[X]\}$, i.e., we only use the expected value as DCOP representation. This is the fairest comparison between our approach and a simheuristic approach using expected values because `mean` employs the same simulation strategy as `quantile`, whereas the simulation strategies of `dcop` and `standard` differ.

Evaluation

We run the experiments with multiple seeds (for the `scvrp` instances, we used five seeds, and for the `spfsp` instances, we used two seeds per instance and noise level). The reason that the seeds are relatively low is the high computation time for each experiment. To give an approximation: the total computation time needed to construct Table 3.5 was 56 hours, and to construct Table 3.6 was 121 hours (using a virtual server that uses an Intel(R) Xeon(R) Gold 6148 CPU with two 2.39 GHz processors and 16.0 GB RAM). To prove the significance of our results, we performed a pairwise t-test as follows (see Test 10 in the book by Kanji (2006)): each seed, instance, budget β, and noise level configuration together form one setting on which the methods are compared pairwise. Per method pair, the comparison is made based on the solution returned by the method, which we simulate with $N = 5,000$. We report on the significance of our results in Section 3.7.2. A correction to the t-test is applied based on the multiple comparison principle (Abdi 2007).

Configurations

The configurations that are important for all methods, including all baselines and `quantile` and `adaptive`, are presented in this section.

We use an existing metaheuristic implementation for the deterministic problem (DCOP), which is provided in the ALNS package (Wouda and Lan 2023) (we slightly adjusted it to the case with deadlines). The ALNS implementation for solving `spfsp` is based on (Stützle and Ruiz 2018), which is an iterated greedy approach. We used the random removal and adjacent removal as repair operators, and we used the greedy repair operation (see (Wouda and Lan 2023)). Used selection strategy: As selection strategy, we used Alpha Upper Confidence Bound with scores $\sigma = (5, 1, 1, 0.5)$ and $\alpha = 0.05$ and as acceptance strategy simulated annealing with $T^{start} = 500$ and $T^{end} = 1$ and a linear decay with step $\gamma = 0.99$.

The final stop criterion is set to β as the maximum total budget (i.e., # $f(\cdot)$ evaluations). We vary β in the experiments. The simulation budget η, i.e., the number of simulations per solution in the different simheuristic implementations, is set to $\eta = 50$ based on a sensitivity analysis; see also Appendix 3.D. For simplicity, the size of the elite set is set to one across all methods, i.e., we always use the best SCOP so far based on the brief simulations (thus, in the pseudocodes, we show that only the best SCOP is stored and returned). Of course, a user of our approach could use a larger elite set and implement a final phase with intensive simulation of the promising elite solutions.

Settings specific to `quantile` and `adaptive` are presented in Table 3.4. We set Ω as the set of DCOP representations to select from, i.e., a set with quantile representations. The intermediate stop criterion is a maximum number of iterations without improvement set to τ. We configure the Roulette Wheel selection scheme settings (scores σ and the weight decay α) based on some initial exploration for `adaptive`, which we later compare with `quantile`. We did not perform an extensive tuning of the Roulette Wheel, as we are mainly concerned with whether an adaptive layer could bring some advantage within our approach.

	Settings `quantile`	Settings `adaptive`
η	50	50
τ	50	50
σ, α	N.A.	σ_1=20, σ_2=5, σ_3=0.5, $\alpha = 0.8$
Ω	$\{\mathbb{E}[X], G^{-1}(0.1k) \mid k = 1,2,\ldots,9\}$	$\{\mathbb{E}[X], G^{-1}(0.1k) \mid k = 1,2,\ldots,9\}$

Table 3.4 *This table summarizes the configuration of our `quantile` and `adaptive` algorithms used in the experiments unless indicated differently. A sensitivity analysis motivating these choices is included in 3.D.*

3.7.2. Results

Research question 1:

How does a simheuristic approach using deterministic metaheuristics with quantile representations compare to baseline simheuristics that only use the expected value?

In Table 3.5 and Table 3.6, we provide the results for the `quantile` approach and the baseline methods. We include the outcomes of our statistical tests in Table 3.7 and Table 3.8. In Table 3.5, we observe that for `scvrp`, `quantile` significantly (see also Table 3.7) outperforms with substantial relative differences all other methods that only use the mean, being the `dcop` method (taking the best DCOP solution), the `standard` (simulating all new best DCOP solutions), and the `mean` method (only simulating the best solution from the buffer after each τ iterations without improvement). For the `scvrp` instances, we see that with a budget of $\beta = 50,000$, the differences from the baselines are already substantial, and all methods show only marginal improvement with higher budget settings. For the `spfsp` instances, however, we observe that increasing the budget has a greater effect and benefits our method the most. We believe this can be explained by the fact that the approaches with expected value are more prone to get stuck, while the `quantile` approach can be more effective given a longer budget as it finds more candidate solutions.

We conclude from the above observations that the proposed `quantile` approach works better than the baseline methods.

Research question 2:

How much does an adaptive layer help to improve our simheuristic approach?

As suggested by Turkeš, Sörensen, and Hvattum (2021), we perform an isolated analysis to see whether an adaptive layer can help improve the `quantile` simheuristic approach.

We compare the `quantile` approach with an extended variant `adaptive` that uses a Roulette Wheel selection scheme such as described in Section 3.6.3. We use the hyperparameter settings from Table 3.4. The results are presented in Table 3.5 and Table 3.6. We see lower normalized averages for the `spfsp` instances that are obtained with `adaptive` compared to `quantile`, but we cannot prove a significant improvement when using the adaptive layer. For `scvrp`, the differences in the normalized averages are even negligible. From these results, we conclude that an ALNS-like adaptive layer can improve the perfor-

Table 3.5 *Normalized average costs for* `scvrp`*, bold results signify the lowest costs. Experiments with* $\epsilon \in [0.5, 1]$.

β **(in thousands)**	**50**	**100**	**150**	**200**
quantile methods				
`quantile`	0.08	**0.06**	**0.05**	**0.05**
`adaptive`	**0.07**	0.07	0.06	0.06
baseline methods				
`dcop`	0.33	0.29	0.29	0.29
`mean`	0.24	0.24	0.23	0.22
`standard`	0.23	0.20	0.20	0.20
single strategies				
0.1	1.00	1.00	1.00	1.00
0.2	1.00	1.00	1.00	1.00
0.3	0.97	0.97	0.97	0.97
0.4	0.81	0.81	0.81	0.81
0.5	0.64	0.63	0.62	0.62
0.6	0.21	0.21	0.20	0.20
0.7	0.09	0.08	0.07	0.07
0.8	0.10	0.09	0.08	0.08
0.9	0.19	0.17	0.15	0.15
1	0.30	0.28	0.26	0.26

mance of a `quantile` approach, but that it is not certain that this always holds and/or is significant. In general, we think that our approach mostly benefits from the exploration caused by using different quantile representations. However, learning which p-quantiles settings are more promising can be useful. Therefore, we suggest performing an isolated analysis and tuning of this adaptive component when using our method for a specific application.

Research question 3:

What is the advantage of a quantile approach compared to running in parallel the simheuristic with only a single p-strategy for the quantiles?

We set up this experiment as follows: we run each experiment with each strategy in $\Omega = \{\mathbb{E}[X], G^{-1}(0.1k) \mid k = 1, 2, \ldots, 9\}$ for the given β and compare this with the `quantile` and `adaptive` algorithms that are given the same budget β. Therefore, the existence of a strategy that outperforms the `quantile` strategy is a bit of an unfair comparison, as running each strategy with budget β requires in total a budget of $\beta' = |\Omega| \cdot \beta$, where $|\Omega|$ equals the number of strategies (p-settings). However, if the `quantile` outperforms all strategies or is at least competitive, this provides an even stronger argument for using our approach.

For the problem instances of `scvrp`, we see in Table 3.5 that the single strategy with

Table 3.6 *Average tardiness for* `spfsp`*, bold results signify the lowest tardiness. Experiments with* $\epsilon \in [0.5, 1]$.

β **(in thousands)**	**50**	**100**	**150**	**200**
quantile methods				
`quantile`	0.40	0.33	0.28	0.28
`adaptive`	**0.23**	**0.19**	**0.18**	**0.11**
baseline methods				
`dcop`	0.57	0.52	0.51	0.51
`mean`	0.58	0.57	0.57	0.52
`standard`	0.57	0.57	0.51	0.51
single strategies				
0.1	0.63	0.61	0.60	0.58
0.2	0.68	0.66	0.63	0.63
0.3	0.70	0.69	0.67	0.67
0.4	0.62	0.59	0.59	0.59
0.5	0.56	0.56	0.50	0.50
0.6	0.66	0.57	0.57	0.57
0.7	0.72	0.63	0.61	0.61
0.8	0.64	0.60	0.60	0.59
0.9	0.75	0.75	0.64	0.63
1	0.61	0.61	0.60	0.60

Table 3.7 *Statistical test results* `scvrp`*. Based on a pairwise t-test with* $\alpha = 0.05$ *based on results from Table 3.5. * means significant.*

Strategy Pair	**t-stat.**	**p**
`adaptive` vs `dcop`	-21.145	0.0*
`adaptive` vs `mean`	-16.44	0.0*
`adaptive` vs `standard`	-14.262	0.0*
`dcop` vs `quantile`	20.685	0.0*
`mean` vs `quantile`	15.559	0.0*
`quantile` vs `standard`	-13.498	0.0*
`adaptive` vs `quantile`	1.307	0.195

$p = 0.7$ is the best choice and that the difference with `quantile` is not significant (t-test). We suspect that this is caused by the exploration of using the different p-quantiles to generate candidate solutions, allowing the escape of local optima (see also Section 3.5 for a concrete example).

For `spfsp` (Table 3.6), we see a problem-specific optimal of $p = 0.5$, but `adaptive` has the lowest mean values and is significantly better. This shows that the `adaptive` strategy is useful here. We also see that the gap between the best results from all individual strategies separately is quite large. We again think that due to the adaptiveness of the algorithm, the simheuristic is less prone to getting stuck in a DCOP local optimum of one

Table 3.8 *Statistical test results* `spfsp`*. Based on a pairwise t-test with* $\alpha = 0.05$ *based on results from Table 3.6. * means significant.*

Strategy Pair	t-stat.	p
`adaptive` vs `dcop`	-7.127	0.0*
`adaptive` vs `mean`	-7.365	0.0*
`adaptive` vs `standard`	-7.146	0.0*
`dcop` vs `quantile`	3.859	0.001*
`mean` vs `quantile`	4.354	0.0*
`quantile` vs `standard`	-4.035	0.0*
`adaptive` vs `quantile`	-1.904	0.066

single strategy and, therefore, finds better solutions to the SCOP.

To summarize, we never see that one of the strategies that used the expected values led to the best performance, which motivated our title “Do Not Use Expectations in Simheuristics".

Research question 4:

What is the effect of the underlying distribution on the performance of our approach compared to the baselines?

In simheuristic literature, the consensus is that in cases with moderate uncertainty (i.e., if the variances of the random variables are not too large), using the ad-hoc approximation $f(\mathbb{E}[X], \pi)$ is in general a good idea (A. A. Juan, Faulin, *et al.* 2015). Therefore, we want to analyze now whether using our approach is more advantageous in case of larger variances. Our second hypothesis is that not using expectations in simheuristics is extra important when distributions are (right-)skewed.

Effect of variance We want to understand the effect of variance in the underlying distribution of X on the performance of the `quantile` approach compared to the baselines. Thus, we compare the results obtained for different noise levels ϵ. We extract the results for $\epsilon \in \{0.001, 0.5, 1\}$ and we include the results in Table 3.9 and Table 3.10. However, surprisingly, if we reduce the variance in the problem to a $\epsilon = 0.001$, we still see the advantages of our proposed approach. We explain this as follows: as long as the set of discrete quantiles is larger than one, using several deterministic counterparts for the generation of candidate solutions, more exploration causes better solutions than a simheuristic using only the expected value.

Then, we observe the differences in results for $\epsilon \in \{0.5, 1\}$. For the `scvrp` problem instances (Table 3.9), we see that for the higher variance setting, we observe larger relative differences between the `quantile` approach and the baselines than for the lower variance setting. Analyzing the results for the problem instances of `spfsp`, we do not see particular differences for $\epsilon = 0.5$ and $\epsilon = 1$ (Table 3.10). We still see that the `adaptive` approach generally yields the best results.

Table 3.9 *Effect of variance on* `scvrp` *problem instances. Obtained costs for* `scvrp` *with noise levels set to* $\epsilon \in \{0.001, 0.05, 1\}$.

	$\epsilon = 0.001$				$\epsilon = 0.5$				$\epsilon = 1$			
β (in thousands)	50	100	150	200	50	100	150	200	50	100	150	200
`adaptive`	0.51	0.32	0.31	0.28	0.29	0.23	0.17	0.15	0.10	0.10	0.06	0.05
`dcop`	0.72	0.58	0.58	0.49	0.96	0.78	0.78	0.76	0.98	0.89	0.89	0.88
`mean`	0.82	0.72	0.62	0.58	0.71	0.71	0.64	0.63	0.67	0.67	0.65	0.64
`quantile`	0.53	0.48	0.33	0.32	0.26	0.10	0.04	0.04	0.17	0.09	0.08	0.07
`standard`	0.76	0.58	0.58	0.49	0.69	0.60	0.60	0.58	0.58	0.54	0.54	0.54

Table 3.10 *Effect of variance on* `spfsp` *problem instances. Obtained costs for* `spfsp` *with noise levels set to* $\epsilon \in \{0.001, 0.05, 1\}$.

β (in thousands)	$\epsilon = 0.001$				$\epsilon = 0.5$				$\epsilon = 1$			
`adaptive`	0.68	0.00	0.00	0.00	0.50	0.43	0.32	0.30	0.33	0.26	0.25	0.00
`dcop`	0.73	0.55	0.53	0.53	0.96	0.88	0.87	0.87	0.77	0.70	0.68	0.68
`mean`	0.84	0.73	0.55	0.55	1.00	0.96	0.96	0.88	0.79	0.77	0.76	0.70
`quantile`	0.41	0.28	0.27	0.25	0.60	0.40	0.29	0.28	0.79	0.63	0.54	0.54
`standard`	0.73	0.55	0.53	0.53	0.96	0.96	0.87	0.87	0.77	0.76	0.69	0.69

Effect of skewness Our hypothesis is that not using expectations in simheuristics becomes even more important when distributions are (right-)skewed. We now use instead of a uniform discrete distribution based on the original deterministic parameters d:

$$X \sim \text{Uniform}\left\{\min\{1, \lfloor (d - \epsilon\sqrt{d}\,\rceil\}, \lfloor (d + \epsilon\sqrt{d}\,\rceil\right\},$$

we use a Geometric discrete distribution where $X = d + G$ and G is the sample from a Geometric distribution with shape parameter q, and for which the probability mass function is given as $P(G = k) = (1-q)^{k-1}q$ and we set $q = 0.1$. The distributions are right-skewed ($\gamma = (2-q)/\sqrt{1-q}$)). For the geometric distribution, the formula for the variance is given by $\sigma^2 = (1-q)/q^2$. The results are included in Table 3.11 and Table 3.12. We see that for the `spfsp` instances, the adaptive layer in `adaptive` works better than the random selection in `quantile`.

The results in Tables 3.11–3.12 indicate that using quantiles in simheuristics also benefits the problem instances with skewed underlying distributions.

Table 3.11 *Skewness* `scvrp`, $q = 0.1$. *Obtained normalized costs for* `scvrp` *with skewness setting* $q = 0.1$.

β	50	100	150	200
`adaptive`	0.02	0.01	0.01	0.01
`dcop`	0.99	0.99	0.99	1.00
`mean`	0.87	0.87	0.87	0.87
`quantile`	0.02	0.01	0.01	0.01
`standard`	0.84	0.84	0.84	0.84

Table 3.12 *Skewness* `spfsp`, $q = 0.1$. *Obtained normalized costs for* `spfsp` *with skewness setting* $q = 0.1$.

β	50	100	150	200
`adaptive`	0.34	0.24	0.23	0.23
`dcop`	0.63	0.60	0.60	0.60
`mean`	0.77	0.63	0.63	0.61
`quantile`	0.67	0.47	0.47	0.41
`standard`	0.77	0.61	0.61	0.61

3.8. Supporting theoretical insights

The proposed approach is primarily motivated by empirical performance and practical considerations. The newsvendor problem motivates the use of quantiles in our method (see Section 3.3). Although the method itself is not derived from a theoretical framework, in this section, we provide theoretical insights for a set of ordering problems to support (i) that searching for a good DCOP representation can be effective and (ii) that quantiles are suitable for generating diverse DCOP representations. These results offer an initial step toward a more formal understanding of why the proposed adaptive simheuristic framework is effective. We emphasize that this analysis is not intended as a full theoretical justification of the method, but rather as a preliminary contribution that complements the computational study and points towards future theoretical developments.

3.8.1. Optimal DCOP representations for ordering problems

This section defines a set of so-called ordering problems for which there exist p-quantiles for all random parameters such that the optimal solution to the DCOP with those p-quantiles is also optimal for the SCOP. Consequently, for those problems, focusing on a DCOP instantiated with such quantiles does not rule out finding the SCOP optimum, while retaining the computational advantages of a deterministic formulation.

Definition 3. *An* ordering problem *is defined as follows. Given a set of indices* $\{1, 2, \ldots, n\}$ *representing cities, jobs, items, et cetera, a set of parameters* X*, including constant* X_0 *and index-dependent values* X_i*,* $X_{i,j}$*, and* $X_{i_1,i_2,\ldots,i_d}$*, find a permutation* $\pi \in S_n$ *where* S_n *is the set of all possible permutations, that minimizes a cost function* $f(X, \pi)$*, i.e.,*

$$\min_{\pi \in S_n} f(X, \pi),$$

where the ordering π *is only used as an index to the parameters* X*. We use* $\pi(i)$ *to indicate the* i*-th element in the permutation* π*. More specifically,* $f(X, \pi)$ *is then defined such that there exists a function* $f'(y_1, \ldots, y_r) = f(X, \pi)$ *where each parameter* y *is derived from* X *using the indexes prescribed by* π*.*

Remark 1. *If multiple parameters need to be defined, we can add a superscript, i.e.,* X^k*. So, for example, multiple parameters with a single index can be defined as* X_i^1*,* X_i^2*, . . . , et cetera. Then, for example,* X_i^1 *may represent the weight of item* i*, and* X_i^2 *its value. For simplicity, the main body of this article uses only one parameter; however, all arguments*

can be generalized to multiple parameters. See Section 3.C for an example with multiple parameters.

Let us examine a concrete problem that can be formulated as an ordering problem. We include more concrete problems in Section 3.C.

Example 3.8.1. ***Traveling Salesman Problem (TSP):*** *The indices represent cities. The parameters are $X = (X_{j,k})_{j,k}$ where $X_{j,k}$ denotes the distance from city j to k. For simplicity and without loss of generality, suppose there are $n+1$ cities, and the tour starts and ends at city $n+1$. Consequently, the goal is to find a permutation $\pi \in S_n$ over the remaining cities that minimizes*

$$f(X,\pi) = X_{n+1,\pi(1)} + \sum_{i=1}^{n-1} X_{\pi(i),\pi(i+1)} + X_{\pi(n),n+1}.$$

Now assume the parameters X are stochastic with a corresponding set of independent distribution functions $G_X(\cdot)$, which has the same indexation as X, for example, $G_X(\cdot)_{i,j}$ is the distribution function of parameter $X_{i,j}$. The following assumptions are made regarding the support of the parameters to ensure sufficient flexibility in adjusting them later on.

Assumption 1. *The supports of random variables in X with the same index-dimension are equal: for each relevant index-dimension $d \in \mathbb{N}$ of X (often just for $d = 1$ and $d = 2$), the support of X_i is equal to the support of $X_{i'}$ for each pair of index-tuples $i = (i_1, i_2, \ldots, i_d)$ and $i' = (i'_1, i'_2, \ldots, i'_d)$.*

To identify alternative choices for the DCOP parameters, we utilize the quantile functions of the distributions of X. Let the corresponding p-quantiles be denoted by $G_X^{-1}(p)$, where p is of the same dimension as X so that each X-component has its unique quantile. For example, the p_{ij}-quantile for X_{ij} is $G_X^{-1}(p_{ij})_{ij}$, or with slight abuse of notation just $G_X^{-1}(p)_{ij}$.

The following theorem makes use of a carefully chosen reindexation to obtain a quantile-based DCOP representation whose optimal solution equals the optimal solution of the SCOP.

Theorem 1. *Let $\pi^\star$ be the optimal SCOP solution for an ordering problem. Under Assumption 1, it holds that:*

$$\exists p : \operatorname*{arg\,min}_{\pi \in S_n} f(G_X^{-1}(p), \pi) = \pi^\star.$$

In fact, there exist many p's for which it holds (of the order of the maximum support size).

Proof: We provide a sketch of the proof here. For a full proof, we refer to Appendix 3.B. First, observe that for a given X, each ordering has an objective value. One of these orderings is the optimal ordering of the SCOP denoted by $\pi^\star$. By changing X through what we define as re-indexing, we can obtain this $\pi^\star$ as the outcome of a DCOP optimization for this adapted X. Given the assumption on the support (Assumption 1), we can guarantee that this adapted X can be obtained by particular p-quantiles, providing an argument for using p-quantiles in the search for an optimal SCOP solution. □

3.8.2. Theory vs. method

Finding a p for which Theorem 1 holds is non-trivial in practice. However, it motivates an approach that seeks effective p-quantiles for DCOP, which leads to relatively good SCOP solutions when using many scenarios is computationally infeasible.

An important limitation of our method is that, for practical reasons, we limited the search space for different p-strategies to a single scalar rather than a p_i setting for every single random parameter. Situations in which this is undesirable may occur in cases where random parameters are correlated. As future research, it is therefore interesting to study approaches with different p-settings for stochastic parameters X. We thought of a gradient-based approach similar to (van den Houten *et al.* 2024), but initial attempts have been unsuccessful so far.

3.9. Conclusion and discussion

In conclusion, using alternatives to the expected value in a simheuristic approach improves performance for stochastic combinatorial optimization problems. We proposed a new generic component to improve the simheuristic framework by dynamically utilizing quantiles, drawing inspiration from best practices in hyperheuristics. We analyzed the effectiveness of these different components in comparison to baseline methods that use a deterministic representation based on expected values under various budget settings.

Of course, a simheuristic implementation, like metaheuristics, requires extensive tuning to improve performance in a specific application. In this research, we constructed our experiments such that we show when the use of quantiles and/or an adaptive layer (that updates weights for different quantiles settings based on performance so far) can be beneficial, following the advice from (Turkeš, Sörensen, and Hvattum 2021) to focus more on gaining insight into the contribution of metaheuristic components than the "horse-racing" (Johnson *et al.* 1999) focus on competitive performance. Therefore, it is essential to note that we do not claim to present the best possible algorithms for the problem instances used in our evaluation; instead, we primarily focused on comparing standard and quantile simheuristics.

We suggest that future research focus on narrowing the gap between theory and practice. In this work, we provide an initial theoretical justification showing that, for a class of ordering problems, p_{ij}-quantiles exist for which the DCOP optimal solution is also optimal in the SCOP. A crucial next step is to further develop this theory and apply it in practice by developing methods that effectively search for quantiles to improve SCOP optimization.

References

Abdi, H. (2007). "Bonferroni and Šidák corrections for multiple comparisons". In: *Encyclopedia of measurement and statistics*. Ed. by N. J. Salkind. Sage, pp. 103–107.

Bianchi, L., M. Dorigo, L. M. Gambardella, and W. J. Gutjahr (2009). "A survey on metaheuristics for stochastic combinatorial optimization". In: *Natural Computing* 8.2, pp. 239–287.

Burke, E. K., M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu (2013). "Hyper-heuristics: a survey of the state of the art". In: *Journal of the Operational Research Society* 64.12, pp. 1695–1724.

Chica, M., A. A. Juan Pérez, O. Cordon, and D. Kelton (2020). "Why simheuristics? Benefits, limitations, and best practices when combining metaheuristics with simulation". In: *Statistics and Operations Research Transactions* 44.2, pp. 311–334.

Clapper, Y., J. Berkhout, and R. Bekker (2024). "Adaptive budget allocation in simheuristics applied to stochastic home healthcare routing and scheduling". In: *Computers & Industrial Engineering* 198, 1–16 (110651).

Ghadimi, S. and W. B. Powell (2024). "Stochastic search for a parametric cost function approximation: Energy storage with rolling forecasts". In: *European Journal of Operational Research* 312.2, pp. 641–652.

González, S., D. Riera, A. A. Juan, M. G. Elizondo, and P. Fonseca (2012). "Sim-RandSHARP: a hybrid algorithm for solving the arc routing problem with stochastic demands". In: *Proceedings of the 2012 Winter Simulation Conference (WSC)*, pp. 3123–3133.

Halvorsen-Weare, E. E. and K. Fagerholt (2011). "Robust supply vessel planning". In: *International conference on network optimization*. Springer, pp. 559–573.

Hendel, G. (2022). "Adaptive large neighborhood search for mixed integer programming". In: *Mathematical Programming Computation* 14.2, pp. 185–221.

Johnson, D. S. *et al.* (1999). "A theoretician's guide to the experimental analysis of algorithms." In: *Data Structures, Near Neighbor Searches, and Methodology* 5, pp. 215–250.

Juan, A., J. Faulin, S. Grasman, D. Riera, J. Marull, and C. Mendez (2011). "Using safety stocks and simulation to solve the vehicle routing problem with stochastic demands". In: *Transportation Research Part C: Emerging Technologies* 19.5, pp. 751–765.

Juan, A. A., J. Faulin, S. E. Grasman, M. Rabe, and G. Figueira (2015). "A review of simheuristics: Extending metaheuristics to deal with stochastic combinatorial optimization problems". In: *Operations Research Perspectives* 2, pp. 62–72.

Juan, A. A., P. Keenan, R. Martí, S. McGarraghy, J. Panadero, P. Carroll, and D. Oliva (2021). "A review of the role of heuristics in stochastic optimisation: from metaheuristics to learnheuristics". In: *Annals of Operations Research* 320, pp. 1–31.

Juan, A. A., Y. Li, M. Ammouriova, J. Panadero, and J. Faulin (2022). "Simheuristics: an introductory tutorial". In: *Proceedings of the 2022 Winter Simulation Conference (WSC)*, pp. 1325–1339.

Kanji, G. K. (2006). *100 statistical tests*. Sage Publications.

Papadimitriou, C. H. and K. Steiglitz (1998). *Combinatorial optimization: algorithms and complexity*. Dover Publications.

Passage, G., M. van den Akker, and H. Hoogeveen (2025). “A new, efficient approach to speed up local search by estimating the solution quality: an application to stochastic, parallel machine scheduling: G. Passage et al.” In: *Journal of Heuristics* 31.3, p. 26.
Pisinger, D. and S. Ropke (2007). “A general heuristic for vehicle routing problems”. In: *Computers & Operations Research* 34.8, pp. 2403–2435.
Powell, W. B. and S. Ghadimi (2022). “The parametric cost function approximation: a new approach for multistage stochastic programming”. In: *ArXiv* abs/2201.00258.
Qin, Y., R. Wang, A. J. Vakharia, Y. Chen, and M. M. H. Seref (2011). “The newsvendor problem: review and directions for future research”. In: *European Journal of Operational Research* 213.2, pp. 361–374.
Rabe, M., M. Deininger, and A. A. Juan (2020). “Speeding up computational times in simheuristics combining genetic algorithms with discrete-event simulation”. In: *Simulation Modelling Practice and Theory* 103, p. 102089. (Visited on 05/24/2022).
Römer, M., F. Hagemann, and T. F. Porrmann (2023). “Predict, tune and optimize for data-driven shift scheduling with uncertain demands”. In: *International Conference on Learning and Intelligent Optimization*. Springer, pp. 254–269.
Rousseau, L.-M., M. Gendreau, and G. Pesant (2002). “Using constraint-based operators to solve the vehicle routing problem with time windows”. In: *Journal of Heuristics* 8.1, pp. 43–58.
Ruszczyński, A. and A. Shapiro (2003). “Stochastic programming models”. In: *Handbooks in Operations Research and Management Science*. Vol. 10, pp. 1–64.
Savage, S. L. (2012). *The flaw of averages: why we underestimate risk in the face of uncertainty*. Engels. Wiley.
Stützle, T. and R. Ruiz (2018). “Iterated greedy.” In: *Handbook of Heuristics*, pp. 547–577.
Turkeš, R., K. Sörensen, and L. M. Hvattum (2021). “Meta-analysis of metaheuristics: Quantifying the effect of adaptiveness in adaptive large neighborhood search”. en. In: *European Journal of Operational Research* 292.2, pp. 423–442. (Visited on 10/12/2024).
Van Den Akker, M., K. Van Blokland, and H. Hoogeveen (2013). “Finding robust solutions for the stochastic job shop scheduling problem by including simulation in local search”. In: *International symposium on experimental algorithms*. Springer, pp. 402–413.
Van den Houten, K., D. M. Tax, E. Freydell, and M. de Weerdt (2024). “Learning from scenarios for repairable stochastic scheduling”. In: *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, pp. 234–242.
Winston, W. L. and J. B. Goldberg (2004). *Operations research: applications and algorithms*. Thomson Brooks/Cole.
Wouda, N. A. and L. Lan (2023). “ALNS: a Python implementation of the adaptive large neighbourhood search metaheuristic”. In: *Journal of Open Source Software* 8.81, p. 5028.

Appendix

3.A. Problem instances

In the experimental evaluation, we use the following instances for `spfsp`:

- `taillard_20_1` with deterministic deadlines: [1302 1859 27 839 695 2112 1174 1665 1579 265 1053 1486 1062 1208 1578 951 455 696 639 1816 226 767 1305 976 1289 259 1519 1210 1473 1445 330 590 1831 1213 1050 1573 1153 934 1039 1077 680 2282 1173 593 227 896 2048 1845 270 1118]*.
- `taillard_20_8` with deterministic deadlines: [1167 1698 1447 1223 1338 406 272 866 967 2209 1554 524 57 101 1149 1641 969 1872 1656 1500 894 547 1265 1897 379 1111 620 739 1513 933 1716 786 868 1787 504 1010 1472 1562 755 1462 2041 209 818 1113 425 1327 335 1439 885 1074]*.

* The deadlines for each job are randomly generated as integers within a specified range. Specifically, for each job j, the deadline d_j is drawn from a uniform distribution over the interval $[0, 2 \cdot \sum_{k=1}^{m} p_{jk}]$, where p_{jk} is the processing time of job j on machine k and m is the total number of machines, i.e., operations of job j. This is implemented as: $d_j = \text{randint}(0, 2 \cdot \sum_{k=1}^{m} p_{jk}) \quad \text{for} \quad i = 1, 2, \ldots, n$, where $\text{randint}(a, b)$ denotes a random integer between a and b, and $\sum_{k=1}^{m} p_{jk}$ is the total sum of the processing times on all machines job j.

In the experimental evaluation, we use and adjust the instances from `http://vrp.atd-lab.inf.puc-rio.br/index.php/en/` for `scvrp`. We used two instances from Christofides, Mingozzi and Toth (1979), being `CMT_1.vrp`, with penalty = 10 and `CMT_2.vrp`, with penalty = 10.

3.B. Proof of theorem

In this appendix, we provide the detailed proof of Theorem 1.

We represent the objective $f(X, \pi)$ of ordering problems defined in Section 3.8.1 by a function $f'(y_1, y_2, \ldots, y_r)$ for arguments $(y_1, y_2, \ldots, y_r)$. The particular tuple of parameters from X that are used as these arguments are dictated by π and denoted as $X\langle\pi\rangle$. The function f' and tuple $X\langle\pi\rangle$ are defined such that it holds for all π and X:

$$f'(X\langle\pi\rangle) = f(X, \pi). \tag{3.7}$$

An ordering problem can then be reformulated as

$$\min_{\pi \in S_n} f'(X\langle\pi\rangle).$$

The proof of the main theorem requires a reindexation, which uses the following definition.

Definition 4. $\texttt{reindex}(X, \pi, \pi')$ *denotes a reindexation of* X *where all occurrences of index* $\pi(i)$ *are replaced by index* $\pi'(i)$ *for all* $i \in \{1, 2, \ldots, n\}$.

For example, $X_{\pi(i)}$ becomes $X_{\pi'(i)}$ in $\texttt{reindex}(X, \pi, \pi')$ for all $i \in \{1, 2, \ldots, n\}$. Reindexing allows the change of the parameter set's perspective, thereby effectively shifting the problem orientation. This is illustrated in the following example.

Example 1. *Consider a TSP of* $n+1 = 3$ *cities where the distance from* i *to* j *is denoted by* $X_{i,j}$ *and given by fixed parameters*

$$X = \begin{bmatrix} 0 & 2 & 1 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{bmatrix}.$$

By assumption (and without losing on optimality since any city must be visited anyway), the tour will start and end at city $n+1 = 3$. *Then, the remaining tour is constructed by a permutation over* $\{1, 2\}$ *(*$\{1, n\}$ *with* $n = 2$*), i.e., cities* 1 *and* 2 *(thus, there are only two solutions possible). For example,* $\pi = (2, 1)$ *denotes the tour* $3 \to 2 \to 1 \to 3$. *To encode this as an ordering problem, let*

$$f(y_1, y_2, y_3) = y_1 + y_2 + y_3$$

denote the calculation of the tour length by adding the three edge distances traversed. Which edges are traversed are determined by the (remaining) permutation π *over 1 and 2. The tuple of parameters which are "unpacked" as* y_1, y_2, y_3 *are determined by* π *via*

$$X\langle\pi\rangle = (X_{3,\pi(1)}, X_{\pi(1),\pi(2)}, X_{\pi(2),3}). \tag{3.8}$$

Indeed, in this symmetric case, all orderings of $X_{3,\pi(1)}, X_{\pi(1),\pi(2)}, X_{\pi(2),3}$ *for* $X\langle\pi\rangle$ *are feasible, just choose one. Let* $\pi_{[1]}$ *denote solution* $(2, 1)$ *and let* $\pi_{[2]}$ *denote solution* $(1, 2)$. *For these two solutions, we get:*

- *For* π *equal to* $\pi_{[1]} = (2, 1)$*:*

  $$X\langle\pi\rangle = (X_{3,\pi(1)}, X_{\pi(1),\pi(2)}, X_{\pi(2),3}) = (X_{3,2}, X_{2,1}, X_{1,3}) = (1, 1, 1)$$

  so that

  $$f(X\langle\pi\rangle) = 1 + 1 + 1 = 3.$$

- *For* π *equal to* $\pi_{[2]} = (1, 2)$*:*

  $$X\langle\pi\rangle = (X_{3,\pi(1)}, X_{\pi(1),\pi(2)}, X_{\pi(2),3}) = (X_{3,1}, X_{1,2}, X_{2,3}) = (2, 2, 1)$$

  so that

  $$f(X\langle\pi\rangle) = 2 + 2 + 1 = 5.$$

So $\pi_{[1]} = (2, 1)$ *is the best solution, and* $\pi_{[2]} = (1, 2)$ *the second-best (and worst) solution. Reindexing reorients the* X *matrix to*

$$\texttt{reindex}(X, \pi_{[2]}, \pi_{[1]}) = \begin{bmatrix} 0 & 1 & 1 \\ 2 & 0 & 1 \\ 1 & 2 & 0 \end{bmatrix}$$

since index 1 and 2 are switched. Denote for simplicity, $\texttt{reindex}(X, \pi_{[2]}, \pi_{[1]})$ *as* $X^{\texttt{reindex}}$*. Let us reconsider the two solutions after the reindexation; for these two solutions, we get:*

- *For* π *equal to* $\pi_{[1]} = (2,1)$*:*

  $$X^{\texttt{reindex}}\langle\pi\rangle = (X^{\texttt{reindex}}_{3,\pi(1)}, X^{\texttt{reindex}}_{\pi(1),\pi(2)}, X^{\texttt{reindex}}_{\pi(2),3}) = (X^{\texttt{reindex}}_{3,2}, X^{\texttt{reindex}}_{2,1}, X^{\texttt{reindex}}_{1,3}) = (2,2,1)$$

  so that

  $$f(X^{\texttt{reindex}}\langle\pi\rangle) = 2+2+1 = 5,$$

  which is equal to $f(X\langle\pi_{[2]}\rangle)$*.*

- *For* π *equal to* $\pi_{[2]} = (1,2)$*:*

  $$X^{\texttt{reindex}}\langle\pi\rangle = (X^{\texttt{reindex}}_{3,\pi(1)}, X^{\texttt{reindex}}_{\pi(1),\pi(2)}, X^{\texttt{reindex}}_{\pi(2),3}) = (X^{\texttt{reindex}}_{3,1}, X^{\texttt{reindex}}_{1,2}, X^{\texttt{reindex}}_{2,3}) = (1,1,1)$$

  so that

  $$f(X^{\texttt{reindex}}\langle\pi\rangle) = 1+1+1 = 3,$$

  which is equal to $f(X\langle\pi_{[1]}\rangle)$*.*

In conclusion, solving TSP with $\texttt{reindex}(X, \pi_{[2]}, \pi_{[1]})$ *gives* $\pi_{[2]} = (1,2)$ *as optimal solution and* $\pi_{[1]} = (2,1)$ *as the second-best (and worst) solution.*

The reindexation has the following properties for all $\pi, \pi' \in S_n$:

- Getting X back again:

  $$X = \texttt{reindex}(\texttt{reindex}(X, \pi, \pi'), \pi', \pi).$$

- Getting the same X after reindexing:

  $$X\langle\pi\rangle = \texttt{reindex}(X, \pi, \pi')\langle\pi'\rangle \tag{3.9}$$

- The reindexing order matters: in general

  $$\texttt{reindex}(X, \pi, \pi') \neq \texttt{reindex}(X, \pi', \pi).$$

The proof then goes as follows.

Proof. Proof Let the optimal SCOP solution be

$$\pi^\star = \arg\min_{\pi \in S_n} E[f(X, \pi)] = \arg\min_{\pi \in S_n} E[f'(X\langle\pi\rangle)].$$

Recall from (3.7) that $f(X, \pi) = f'(X\langle\pi\rangle)$, so in the remaining proof, we will just work with the latter. Consider the DCOP with parameters $\bar{X}$ from the support, i.e.,

$$\min_{\pi \in S_n} f'(\bar{X}\langle\pi\rangle),$$

and order all its solutions $\pi \in S_n$ from best to worst and denote them, respectively, as

$$\pi_{[1]}, \pi_{[2]}, \ldots, \pi_{[n!]}$$

where $n!$ is the number of solutions. So $\pi_{[1]}$ is the best solution to $\min_{\pi \in S_n} f'(\bar{X}\langle\pi\rangle)$.

Since the $\pi_{[i]}$'s span all permutations, there must exist a k for which

$$\pi_{[k]} = \pi^\star.$$

Now reindex the problem so that $\pi_{[k]}$ becomes the best $\bar{X}$-DCOP solution. In particular, the optimal solution to the reindexed DCOP problem with $\texttt{reindex}(\bar{X}, \pi_{[1]}, \pi_{[k]})$, i.e.,

$$\min_{\pi \in S_n} f'(\texttt{reindex}(\bar{X}, \pi_{[1]}, \pi_{[k]})\langle\pi\rangle),$$

is equal to $\pi^\star$. This follows from two observations:

- Due to symmetry, the reindexation does not change the set of all objective values:

$$\{f'(\bar{X}\langle\pi\rangle) \mid \forall \pi \in S_n\} = \{f'(\texttt{reindex}(\bar{X}, \pi', \pi'')\langle\pi\rangle) \mid \forall \pi \in S_n\}, \quad \forall \pi', \pi'' \in S_n.$$

- After reindexation, the definition of the ordering problem (stating that $f(\cdot)$ is only dependent on π via the parameter choices) and Property (3.9) guarantee that $\pi_{[k]}$ has the same objective as $\pi_{[1]}$ before reindexing:

$$f(\bar{X}\langle\pi_{[1]}\rangle) = f(\texttt{reindex}(\bar{X}, \pi_{[1]}, \pi_{[k]})\langle\pi_{[k]}\rangle).$$

Consequently, solution $\pi_{[k]}$ must be best after reindexation.

The last part of this proof is about setting the right parameter values so that the indexation effectively occurs. Since fixed parameters $\bar{X}$ are chosen from their supports, Assumption 1 guarantees a quantile $p^\star$ can be found for which

$$G_X^{-1}(p^\star) = \texttt{reindex}(\bar{X}, \pi_{[1]}, \pi_{[k]}).$$

So the optimal solution of $G_X^{-1}(p^\star)$-DCOP will equal $\pi^\star$, i.e.,

$$\arg\min_{\pi \in S_n} f'(G_X^{-1}(p^\star)\langle\pi\rangle) = \pi^\star. \tag{3.10}$$

The reindexing was done based on solving $\bar{X}$-DCOP. In principle, any $\bar{X}$ from the supports can be used in DCOP, each leading to different quantile values for which the corresponding DCOP gives the SCOP-optimal solution $\pi^\star$. Consequently, there are as many $p^\star$'s in the order of the maximum support size for which (3.10) holds. □

Remark 2. *Note that the first part of the proof of Theorem 1 proves that for any DCOP representation, there always exists a specific reindexing of the DCOP parameters so that any solution may become the optimal DCOP solution.*

Finding a p-quantile-DCOP representation for which Theorem 1 holds is hard in practice. However, it does motivate the use of a dynamic simheuristic, which can be effective in finding a DCOP representation that leads to relatively good SCOP solutions. This method is detailed in the body of the paper.

3.C. Examples of ordering problems

This section enumerates more problems that can be classified as ordering problems.

1. **Knapsack Problem:** The objective depends on items' weights $(X_i^1)_i$ and items' values $(X_i^2)_i$ which are indexed by the ordering. Furthermore, a (constant) capacity of X_0 is given. The items are packed in order of permutation π till they exceed capacity. Let $\mathbf{1}\{A\}$ indicate an indicator function which is 1 if A is true, and 0 else. Then choosing

$$f(y_1^1, y_2^1, \ldots, y_n^1, y_1^2, y_2^2, \ldots, y_n^2) = \max_{j \in \{1,\ldots,n\}} \mathbf{1}\left\{\sum_{i=1}^{j} y_i^1 \leq X_0\right\} \sum_{i=1}^{j} y_i^2$$

and

$$X\langle\pi\rangle = (X_{\pi(1)}^1, \ldots, X_{\pi(n)}^1, X_{\pi(1)}^2, \ldots, X_{\pi(n)}^2), \tag{3.11}$$

leads to

$$f(X\langle\pi\rangle) = \max_{j \in \{1,\ldots,n\}} \mathbf{1}\left\{\sum_{i=1}^{j} X_{\pi(i)}^1 \leq X_0\right\} \sum_{i=1}^{j} X_{\pi(i)}^2,$$

here j denotes the number of items taken in order of π. The indicator checks whether the total weight $\sum_{i=1}^{j} X_{\pi(i)}^1$ does not exceed capacity X_0, whereas the total value of the packed items is measured via $\sum_{i=1}^{j} X_{\pi(i)}^2$. The maximization will pack as many items as possible in order of the π till the capacity is exceeded.

2. **Single-Machine Scheduling Problem with Weighted Completion Times Objective:** The objective depends on job weights $(X_i^1)_i$ and job durations $(X_i^2)_i$ which are indexed by the ordering:

$$f(X\langle\pi\rangle) = \sum_{i=1}^{n} X_{\pi(i)}^1 \sum_{j=1}^{i} X_{\pi(j)}^2.$$

In this case, tuple $X\langle\pi\rangle$ could be the same as (3.11) for the knapsack problem and

$$f(y_1^1, y_2^1, \ldots, y_n^1, y_1^2, y_2^2, \ldots, y_n^2) = \sum_{i=1}^{n} y_i^1 \sum_{j=1}^{i} y_j^2.$$

3. **Single-Machine Scheduling Problem with Tardiness Objective:** The objective depends on job deadlines $(X_i^1)_i$, job durations $(X_i^2)_i$ and sequence-dependent setup times $(X_{j,k}^3)_{j,k}$ which are indexed by the ordering:

$$f(X\langle\pi\rangle) = \sum_{i=1}^{n} \max\left(\underbrace{\sum_{j=1}^{i-1}\left(X_{\pi(j)}^2 + X_{\pi(j),\pi(j+1)}^3\right)}_{\text{start time job } \pi(i)} + X_{\pi(i)}^2 - X_{\pi(i)}^1,\ 0\right).$$

In this case, tuple $X\langle\pi\rangle$ could be

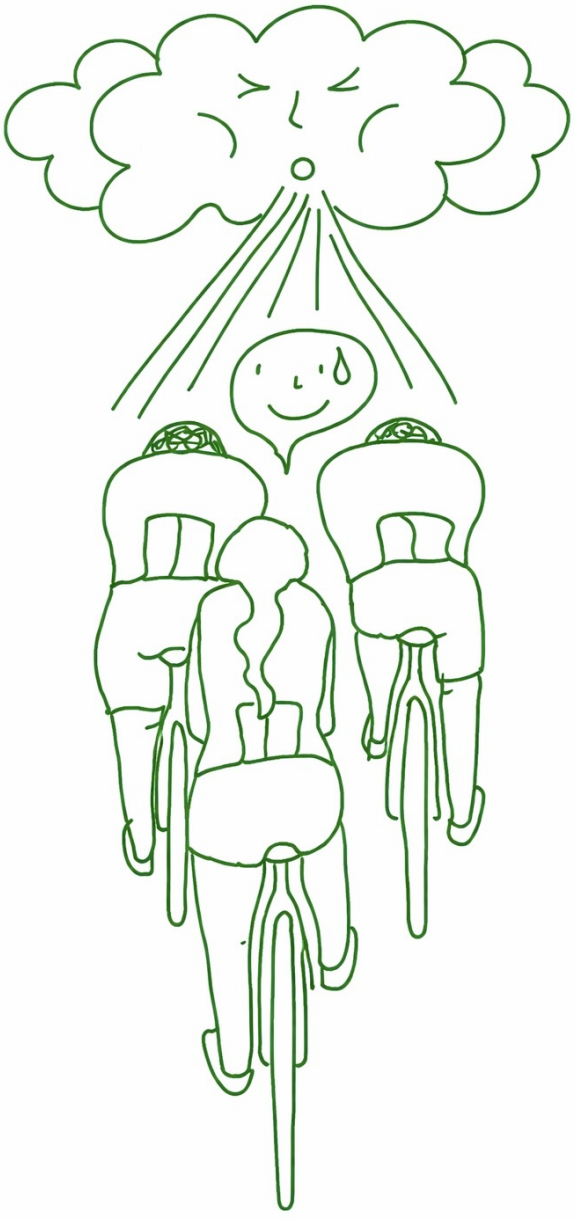
$$X\langle\pi\rangle = \{X^1_{\pi(1)}, \ldots, X^1_{\pi(n)}, X^2_{\pi(1)}, \ldots, X^2_{\pi(n)}, X^3_{\pi(1),\pi(2)}, X^3_{\pi(2),\pi(3)}, \ldots, X^3_{\pi(n-1),\pi(n)}\}.$$

and

$$\begin{aligned} f(y^1_1, y^1_2, \ldots, y^1_n, y^2_1, y^2_2, \ldots, y^2_n, y^3_{1,2}, y^3_{2,3}, \ldots, y^3_{n-1,n}) \\ = \sum_{i=1}^{n} \max\left(\sum_{j=1}^{i-1}\left(y^2_j + y^3_{j,j+1}\right) + y^2_i - y^1_i,\, 0\right). \end{aligned}$$

3.D. Sensitivity analysis

We perform a sensitivity analysis on the effect of η for different β settings. We used $\eta \in \{10, 50, 100, 200\}$. There is no overall best setting for η. However, we observe that $\eta = 50$ is a robust choice for both the `quantile` and the `mean` algorithm and for different β settings. Furthermore, we investigated the effect of τ for different β settings. We found that it is mostly important that τ is not too low (in our analysis, too low was approximately $\tau \leq 20$). Based on our sensitivity analysis, we set $\tau = 50$ in our experiments.



4

Proactive and reactive constraint programming

This chapter investigates scheduling strategies for the stochastic resource-constrained project scheduling problem with maximal time lags (SRCPSP/max). Recent advances in Constraint Programming (CP) and Temporal Networks have re-invoked interest in evaluating the advantages and drawbacks of various proactive and reactive scheduling methods. In this chapter, we answer the question: ***"Which algorithms are most effective for resource-constrained scheduling problems that involve temporal uncertainty and strict temporal constraints?"*** *First, we present a new, CP-based, fully proactive method. Second, we show how a reactive approach can be constructed using an online rescheduling procedure. A third contribution is based on partial order schedules and uses Simple Temporal Networks with Uncertainty (STNUs). Our analysis shows that the STNU-based algorithm performs best in terms of solution quality, while also showing good relative computation time.*

This chapter is based on the article: *Van den Houten, K., Planken, L., Freydell, E., Tax, D. M., & De Weerdt, M. (2025, April). Proactive and reactive constraint programming for stochastic project scheduling with maximal time-lags. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 39, No. 25, pp. 26534-26541).*

4.1. Introduction

In real-world scheduling applications, durations of activities are often stochastic, for example, due to the inherent stochastic nature of processes in biomanufacturing. At the same time, hard constraints must be satisfied: e.g. once fermentation starts, a cooling procedure must start at least (minimal time lag) 10 and at most (maximal time lag) 30 minutes later. The combination of maximal time lags and stochastic durations is especially tricky: a delay in duration can cause a violation of a maximal time lag when a resource becomes available later than expected. Such constraints are reflected in the Stochastic Resource-Constrained Project Scheduling Problem with Time Lags (SRCPSP/max). This problem has been an important focus of research due to its practical relevance and the computational challenge it presents, as finding a feasible solution is NP-hard (Bartusch, Möhring, and Radermacher 1988).

Broadly speaking, there are two main schools of thought regarding solution approaches for stochastic scheduling in the literature: 1) proactive scheduling and 2) reactive scheduling. The main goal of proactive scheduling is to find a robust schedule offline, whereas reactive approaches adapt to uncertainties online. Proactive and reactive approaches can be considered as opposite ends of a spectrum. Both in practice and literature, it is often observed that methods are hybrid, such as earlier work on the SRCPSP/max.

Hybrid approaches appear for example in the form of a *partial order schedule* (POS), which is a temporally flexible schedule in which resource feasibility is guaranteed (Policella, Smith, *et al.* 2004). The state-of-the-art POS approach for SRCPSP/max is the algorithm BACCHUS (Fu, Varakantham, and Lau 2016), although the comparison provided by the authors themselves shows that their earlier proactive method SORU-H (Varakantham, Fu, and Lau 2016) performs better. Partial order schedules are often complemented with a temporal network (Lombardi and Milano 2009), in which time points (nodes) are modeled together with temporal constraints (edges). Recent advances in temporal networks with uncertainties (Hunsberger and Posenato 2024a) pave the way for improvements in POS approaches for SRCPSP/max.

State-of-the-art methods like BACCHUS and SORU-H use Mixed Integer Programming (MIP), whereas Constraint Programming (CP), especially with interval variables (Laborie 2015), has become a powerful alternative for scheduling. This is evidenced by a CP model for resource-constrained project scheduling provided by Laborie *et al.* (2018). Moreover, a recent empirical comparison between CP and MIP solvers on a broad set of benchmark scheduling problems found that CP Optimize often outperformed CPLEX on the considered instances (Naderi, Ruiz, and Roshanaei 2023) (whereas MIP tends to be advantageous for problems that primarily involve assignment decisions). These advances, however, have not yet been explored for SRCPSP/max, despite potential applications. CP can be used for finding robust, proactive schedules or for a reactive approach with rescheduling during execution, which has been viewed as too computationally intensive (Van de Vonder, Demeulemeester, and Herroelen 2007).

A proper benchmarking paper that performs a statistical analysis of the results of the different methods for SRCPSP/max is lacking. Comparing different stochastic scheduling techniques for this problem should be done carefully. Due to the maximum time lags and stochastic durations, methods can fail to satisfy resource or precedence constraints. Therefore, not only the solution quality and computation time but also the

feasibility should be taken into account. Fu, Varakantham, and Lau (2016) criticize their own evaluation metric and indicate that future work should provide a more extensive comparison. We argue that statistical tests that correctly account for infeasibilities are needed to properly compare different methods.

This paper proposes new versions of a proactive approach and a hybrid approach using the latest advances in Constraint Programming (Laborie *et al.* 2018) and Temporal Networks (Hunsberger and Posenato 2024a). A new reactive approach with complete rescheduling, which, as far as we know, has not been investigated before, is included in the comparison, too. We aim to address the existing research gap by providing a clear, informative comparison of various methods, examining aspects such as infeasibility, solution quality, and computation time (both offline and online). In doing so, this work provides a practical guide for researchers and industrial schedulers when selecting from various methods for their specific use cases.

4.2. The scheduling problem

The RCPSP/max problem (Kolisch and Sprecher 1996) is defined as a set of activities J and a set of resources R, where each activity $j \in J$ has a duration d_j and requires from each resource $r \in R$ a certain amount indicated by $r_{r,j}$. A solution to the RCPSP/max is a start-time assignment s_j for each activity, such that the constraints are satisfied: (i) at any time the number of resources used cannot exceed the max resource capacity c_r; and (ii) for some pairs (i, j) of activities, precedence constraints are defined as minimal or maximal time lags between the start time of i and the start time of j. The goal is to minimize the makespan.

We use the following example from (Schutt *et al.* 2013) of a simple RCPSP/max instance. There are five activities: a, b, c, d, and e, with start times s_a, s_b, s_c, s_d, and s_e. The activities have durations of 2, 5, 3, 1, and 2 time units, respectively, and resource requirements of 3, 2, 1, 2, and 2 on a single resource with a capacity of $c_r = 4$. The precedence relations are as follows: activity b starts at least 2 time units after a starts, activity b starts at least 1 time unit before c starts, activity c cannot start later than 6 time units after a starts, and activity d starts exactly 3 time units before e starts. These constraints can be visualized in a project graph (Figure 4.1). A feasible solution to this problem is $\{s_a = 1, s_b = 3, s_c = 4, s_d = 0, s_e = 3\}$, for which the Gantt chart is visualized in Figure 4.2.

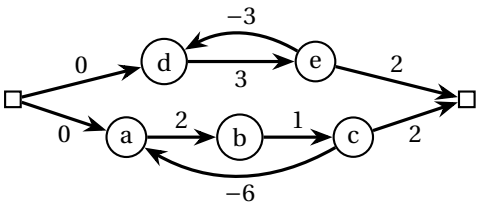


Figure 4.1 *Example project graph (Schutt* et al. *2013).*

A special property of RCPSP/max is the following:

Proposition 1. Suppose we are given a problem instance 1 with durations d^1, resource requirements r^1, and capacity c^1, and let s^1 be a feasible schedule for this instance. Suppose now that we transform instance 1 into instance 2, where all parameters stay equal except that one or more of the activity durations d^2 are shorter than the durations d^1, so $\forall\ j \in J : d^1_j \leq d^2_j$. Then, s^1 is also feasible for instance 2.

Proof. Schedule s^1 is still precedence feasible because the start times did not change and the precedence constraints are defined from start to start (they are deterministic). Schedule s^1 is resource feasible for d^1. Since d^2 is strictly smaller than d^1, the resource usage over time can only be smaller than for instance 1, and thus it will not exceed the capacity, and schedule s^1 is also resource feasible for d^2. □

This property can be helpful when reusing start times for a mutated instance, for example, because of stochastic activity durations.

Problem statement

The *Stochastic RCPSP/max* (SRCPSP/max) is an extension of the deterministic RCPSP/max in which activity durations are uncertain (Fu, Lau, *et al.* 2012). Formally, each activity $j \in J$ has a duration d_j modeled as an independent random variable. The actual duration of an activity is revealed upon its completion. *The optimization goal considered in this chapter is to develop a method that minimizes the sample average makespan across a test set of realizations of the activity durations.*

Comparing solution methods for this problem is inherently challenging. A direct comparison between two methods, say A and B, is complicated by the possibility that one method may produce feasible schedules for instances that the other cannot. Restricting the comparison to *double hits*—instances solved by both methods—provides some insight but ignores the relative coverage of each method across the full test set. To address this, previous work introduces the α-robust makespan (Fu, Lau, *et al.* 2012), defined as the expected makespan of a schedule that is feasible with probability at least $1-\alpha$. While useful, this metric has limitations, particularly when infeasible schedules are considered true failures. Accordingly, in this chapter, our focus is on developing a method that optimizes the expected makespan while considering infeasible scenarios as failures.

4.3. Background and related work

In this section, we discuss existing approaches for SRCPSP/max. Furthermore, we introduce all the concepts needed to understand the scheduling methods presented in Section 4.4. We introduce proactive techniques based on Sample Average Approximation in Section 4.3.1. We explain partial order schedules and temporal networks in Section 4.3.2. Finally, Section 4.3.3 gives an overview of related work on benchmarking scheduling methods for SRCPSP/max.

4.3.1. Proactive scheduling

Proactive scheduling methods aim to find a robust schedule offline by accounting for uncertainty (Herroelen and Leus 2002). In this section, we explain a core technique used

in proactive methods: Sample Average Approximation (SAA). Furthermore, we discuss the state-of-the-art proactive methods for SRCPSP/max.

Sample average approximation

A common method for handling discrete optimization under uncertainty is the Sample Average Approximation (SAA) approach (Kleywegt, Shapiro, and Homem-de-Mello 2002). Samples are drawn from stochastic distributions and added as scenarios to a stochastic programming formulation. The solver then seeks a solution feasible for all scenarios while optimizing the average objective. However, adding more samples to the SAA increases the number of constraints and variables, significantly raising the solution time.

SORU and SORU-H

The most recent proactive method on SRCPSP/max is proposed by Varakantham, Fu, and Lau (2016) and is recognized as the state of the art. The authors present the algorithm SORU, an SAA approach to the scheduling problem that relies on Mixed Integer Programming (MIP) and aims to minimize the α-robust makespan. It can be summarized as follows: (i) a selection of samples is used to set up the SAA; (ii) the model seeks a start time vector s such that the minimal and maximal time lags of precedence constraints are satisfied; (iii) it allows for α% of the scenarios to be resource infeasible; (iv) it minimizes the sample average makespan.

Since SORU is computationally expensive, the authors propose a heuristic version, dubbed SORU-H. Instead of a set of samples, one summarizing sample is used, which represents a quantile of the distribution. Note that because of Proposition 1, this heuristic approximates the α-robust makespan. At the same time, it is much cheaper to compute because typically the runtime increases for a larger sample size in SAA, as shown by Varakantham, Fu, and Lau (2016).

Since CP is promising for deterministic project scheduling (Naderi, Ruiz, and Roshanaei 2023), we reinvestigate SAA approaches for SRCPSP/max with CP and compare the CP-based proactive approach with reactive approaches.

4.3.2. Partial order scheduling

In this section, we discuss partial order scheduling approaches, which can be seen as a reactive-proactive hybrid. Partial order schedules have been used in the majority of the contributions to SRCPSP/max.

Constructing ordering constraints

A *partial order schedule* (POS) can be seen as a collection of schedules that ensure resource feasibility, but maintain temporal flexibility. A POS is defined as a graph where nodes represent activities, and edges temporal constraints between them. There are several methods to derive a POS. In the original paper by Policella, Smith, *et al.* (2004), two approaches are outlined to construct the ordering constraints between activities, either analyzing the resource profile to avoid all possible resource conflicts, based on *Minimal Critical Sets* (MCS) (Lgelmund and Radermacher 1983), or using a single-point solution together with a chaining procedure to construct *resource chains* (see Figure 4.2). They

find that using the single-point solution together with chaining seems both simple and most effective. Subsequent work explored alternative MCS-based approaches (Lombardi and Milano 2009; Lombardi, Milano, and Benini 2013) and chaining heuristics (Fu, Lau, *et al.* 2012).

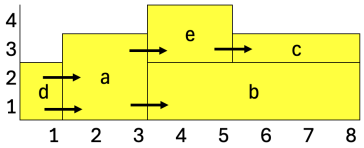


Figure 4.2 *Example Gantt chart (figure adjusted from (Fu, Lau,* et al. *2012)). The X-axis is time, and Y-axis shows resource demand.* ***The arrows*** *indicate an example of generating a POS with chaining, starting from a fixed solution schedule.*

Temporal networks

Partial order schedules are often complemented by a temporal model to reason about temporal constraints. Most POS approaches rely on a Simple Temporal Network (STN) (Dechter, Meiri, and Pearl 1991), which is a graph consisting of time points (nodes) and temporal difference constraints (edges). An STN can be formally expressed as $(\mathscr{T},\mathscr{C})$, where $\mathscr{T}$ represents a set of real-valued variables (time-points) and $\mathscr{C}$ represents a set of temporal constraints (Dechter, Meiri, and Pearl 1991), also referred to as ordinary links. An STN can be represented as a *distance graph.* An ordinary link indicates a relation between two time points A and B. In the distance graph, each ordinary link

$$A \xrightarrow{[x,y]} B$$

is replaced by two edges

$$A \xrightarrow{y} B \qquad \text{and} \qquad A \xleftarrow{-x} B.$$

This ordinary link implies that $x \leq B - A \leq y$. The edge $A \xrightarrow{y} B$ thus represents an upper bound edge, whereas $A \xrightarrow{-x} B$ represents a lower bound edge.

The Simple Temporal Network with Uncertainty (STNU) extends the STN by introducing *contingent links.* In STNUs, contingent links connect an activation point (controllable event) to a contingent point (uncontrollable event). An STNU thus is a triple, $\mathscr{S} = (\mathscr{T},\mathscr{C},\mathscr{L})$ (Morris, Muscettola, and Vidal 2001), where:

- $(\mathscr{T},\mathscr{C})$ is an STN,
- $\mathscr{L}$ 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, Muscettola, and Vidal 2001). The duration is bounded: $C - A \in [x,y]$ and uncontrollable.

Similarly, there is an analogous representation for an STNU called the *labeled distance graph* (Morris, Muscettola, and Vidal 2001). For a contingent link in the labeled distance

graph, we have for each contingent link:

$$A \overset{[x,y]}{\Longrightarrow} C,$$

we have the two edges

$$A \xrightarrow{y} C \qquad \text{and} \qquad A \xleftarrow{-x} C,$$

but we also have two additional edges of the form

$$A \xrightarrow{c:x} C \qquad \text{and} \qquad A \xleftarrow{C:-y} C.$$

These are called *labeled edges* because of the additional "c :" and "C :" annotations indicating the contingent timepoint C with which they are associated. We refer to $A \xleftarrow{C:-y} C$ and $A \xrightarrow{c:x} C$ as *upper-case* and *lower-case* edges, respectively.For STNUs, important are so-called wait constraints. These wait constraints are represented as $A \xrightarrow{<B,t>} C$, and they correspond to a single edge $A \xleftarrow{B:-t} C$. Such a wait edge indicates that while B is not executed, C must wait at least t after A (Hunsberger and Posenato 2024a; Morris 2014).

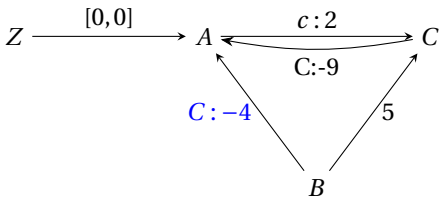


Figure 4.3 *Example STNU with wait constraint: while C unexecuted, B must wait at least 4 after A.*

Dynamic controllability

An STNU is *dynamically controllable* (DC) if there is a strategy to determine execution times for all controllable (non-contingent) time points that ensures all temporal constraints are met, regardless of the outcomes of contingent links. An STNU is dynamically controllable if and only if it does not have a semi-reducible negative cycle. These cycles can be detected using the DC-checking algorithm (Morris 2014), which systematically applies a set of reduction rules to the labeled distance graph to identify negative cycles. These reduction rules are used for backward distance calculation (i.e., distance to rather than distances from). Every time a new negative edge is discovered, a recursive call is started, as shown in Algorithm 4. While scanning for negative cycles, the DC-checking algorithm generates new edges while checking whether the network is DC (Morris 2014), including wait edges. This extended version is referred to as an Extended STNU (ESTNU) and returned by Algorithm 4 if the network is DC. The ESTNU is defined as $\mathscr{S} = (\mathscr{T}, \mathscr{C}, \mathscr{L}, \mathscr{L}_w)$, with time-points $\mathscr{T} = \mathscr{T}_x \cup \mathscr{T}_c$ where $\mathscr{T}_x$ are the executable timepoints , $\mathscr{T}_c$ the contingent timepoints, $\mathscr{C}$ the set of temporal constraints, $\mathscr{L}$ the set of contingent links and $\mathscr{L}_w$ the set of wait edges. This ESTNU provides input to the real-time execution algorithm described in the next section.

Algorithm 4 Determine Dynamic Controllability (Morris 2014)

```
procedure DETERMINEDC+(graph)                    ▷ The input is the labeled distance graph.
    for each negative node n do
        if DCBACKPROP(n) = false then
            return false
    return graph

procedure DCBACKPROP(source, graph)
    if ancestor call with same source then
        return false
    if prior terminated call with source then
        return true
    distance(source) = 0
    for each node x other than source do
        distance(x) = ∞
    PriorityQueue queue = empty priority queue
    for each e1 ∈ INEDGES(source) do
        Node n1 = START(e1)
        DISTANCE(n1) = WEIGHT(e1)
        insert n1 into queue
    while queue not empty do
        pop node u from queue
        if DISTANCE(u) ≥ 0 then
            create edge e' = (u, source)
            WEIGHT(e') = DISTANCE(u)
            add e' to graph
            continue
        if DISTANCE(u) < 0 then                  ▷ This step stores wait edges.
            create edge e' = (u, source)
            WEIGHT(e') = DISTANCE(u)
            add e' to graph
        if u is negative node then
            if DCBACKPROP(u) = false then
                return false
        for each e ∈ INEDGES(u) do
            if WEIGHT(e) < 0 then
                continue
            if e is unsuitable then
                continue
            node v = start(e)
            new = DISTANCE(u) + WEIGHT(e)
            if new < DISTANCE(v) then
                DISTANCE(v) = new
                insert v into queue
    return true
```

Suppose the example network of Figure 4.3 without the wait edge $A \xleftarrow{C:-4} B$. If we apply DC-checking to this network, Algorithm 4 would derive the wait edge during execution by starting backward propagation from the negative node A (which is a negative node because it has an incoming edge $A \xleftarrow{C:-9} C$, then following the path $A \leftarrow C \leftarrow B$ introduces the new negative edge $A \xleftarrow{C:-4} B$ following the reduction rules from Morris (2014). Since no negative cycles exist in this network, the network is dynamically controllable.

Real-time execution algorithm

Algorithm 5 RTE*: Real-Time Execution for ESTNUs (Hunsberger and Posenato 2024a)

Require: $\mathscr{S} = (\mathscr{T}_x \cup \mathscr{T}_c, \mathscr{C}, \mathscr{L}, \mathscr{C}_w)$
Ensure: $f : (\mathscr{T}_x \cup \mathscr{T}_c) \to \mathbb{R}$ or fail
1: $D = \text{RTE}^*_{init}(\mathscr{T}_x, \mathscr{T}_c)$ ▷ *Initialization*
2: **while** $D.U_x \cup D.U_c \neq \emptyset$ **do**
3: $\quad \Delta \leftarrow \text{RTE}^*_{genD}(D)$ ▷ *Generate execution decision*
4: $\quad$ **if** Δ = fail **then**
5: $\quad\quad$ **return** fail
6: $\quad (\rho, \tau) = \text{OBSERVE}_c(\mathscr{S}, D, \Delta)$
7: $\quad D \leftarrow \text{RTE}^*_{update}(D, \Delta, (\rho, \tau))$
8: $\quad$ **if** D = fail **then**
9: $\quad\quad$ **return** fail
10: **return** $D.f$

An ESTNU provides a start point for a Real-Time Execution Algorithm (Hunsberger and Posenato 2024a). Such an execution algorithm is the online component that transforms an STNU into a schedule (i.e., an execution time for each node), given the observations for the contingent nodes. The algorithm specifically tailored to ESTNUs is the RTE* algorithm (Hunsberger and Posenato 2024a; Posenato 2022). The high-level pseudocode of the RTE* algorithm is provided in Algorithm 5. On each iteration, the algorithm generates an execution decision and observes whether any contingent timepoint is executed. The RTE* algorithm returns a function $f : T \to \mathbb{R}$, which is a set of assigned values to the timepoints. The algorithm updates all relevant data in each iteration in a data structure `RTEdata`, which has the following fields:

- U_x (the unexecuted executable timepoints),
- U_c (the unexecuted contingent timepoints),
- $\texttt{Enabs}_x$ (the enabled executable timepoints),
- `now` (the current time).
- f (a set of variable assignments),
- for each executable timepoint $X \in \mathscr{T}_x$, $\texttt{TW}(X) = [\texttt{lb}(X), \texttt{ub}(X)]$ (time window for X),

- `AcWts`(X) (the activated waits for X, which refers to all wait edges associated with a contingent timepoint C that become active once the activation timepoint of the corresponding contingent link has been executed.).

A new `RTEdata` instance D is initialized using the `RTE`$^*{}_{init}$ algorithm (see Algorithm 6). For ESTNUs, an executable timepoint X is enabled only if all of its outgoing negative edges point to timepoints that have already been executed.

Algorithm 6 $RTE*_{init}$ Initialization (Hunsberger and Posenato 2024a)

Require: $\mathscr{T}_x$ executable timepoints, $\mathscr{T}_c$ contingent timepoints
Ensure: initialized `RTEdata` structure $D = new$(`RTEdata`)
1: $D.U_x = \mathscr{T}_x$
2: $D.U_c = \mathscr{T}_c$
3: D.`now` $= 0$
4: D.`Enabs`$_x = \{X \in \mathscr{T}_x | X$ has no outgoing negative edges$\}$
5: **for** each $X \in \mathscr{T}_x$ **do**
6: $\quad D$.`TW`$(X) = [0, \infty)$
7: $\quad D$.`AcWts`$(X) = \emptyset$
8: initialize enabled sets
9: **return** D

At every iteration, a new execution decision Δ is generated following Algorithm 7 and the decision is either `wait` or of the form (t, V), which means "if no contingent timepoints execute before time t, then execute the timepoints in the set V".

Algorithm 7 `RTE`$^*{}_{genD}$: Generate Execution Decision (Hunsberger and Posenato 2024a)

Require: D an `RTEdata` structure
Ensure: Execution decision: `wait` or (t, V); or fail
1: **if** D.`Enabs`$_x = \emptyset$ **then** ▷ *Check if the set of enable timepoints is empty.*
2: $\quad$ **return** Wait
3: **for** each $X \in D$.`Enabs`$_x$ **do**
4: $\quad$ `wt`$(X) = \max\{w \mid \exists(w, \_) \in D$.`AcWts`$(X)\}$ ▷ *Store maximum wait time for X*
5: $\quad$ `glb`$(X) = \max\{D.lb(X), wt(X)\}$ ▷ *Store greatest lower bound for X*
6: $t_L = \min\{$`glb`$(X) \mid X \in D$.`Enabs`$_x\}$ ▷ *Earliest possible next execution*
7: $t_U = \min\{D.ub(X) \mid X \in D$.`Enabs`$_x\}$ ▷ *Latest possible next execution*
8: **if** $[t_L, t_U] \cap [D$.`now`$, \infty) = \emptyset$ **then**
9: $\quad$ **return** fail
10: Select any $V \in D$.`Enabs`$_x$ for which $[$`glb`(V), `ub`$(V)] \cap [D$.`now`$, t_U] \neq \emptyset$
11: Select any $t \in [$`glb`(V), `ub`$(V)] \cap [D$.`now`$, t_U]$
12: **return** (t, V)

Then, the decision Δ is used to update the `RTEdata`, following Algorithm 8, if only contingent timepoints are executed then the `HCE` algorithm is used for updating (see Algorithm 9). The `HXE` algorithm is used for updating non-contingent timepoints (see Algorithm 10).

Algorithm 8 RTE$^*{}_{update}$: update information in D (Hunsberger and Posenato 2024a).

Require: S (an ESTNU), D (an RTE data structure), Δ (an RTED: WAIT or (t, V)), (ρ, τ) an observation with $\rho \in \mathbb{R}$ and $\tau \subseteq D.U_c$
Ensure: Updated D or FAIL
1: **if** $\rho = \infty$ **then** ▷ *Case 0: Failure (waiting forever)*
2: **return** FAIL
3: **if** Δ = WAIT or ($\Delta = (t, V)$ and $\rho < t$) **then** ▷ *Case 1: Only contingent timepoints executed*
4: HCE(S, D, ρ, τ)
5: **else**
6: HXE(S, D, t, V) ▷ *Case 2: Executable timepoint V executes at t*
7: **if** $\tau \neq \emptyset$ **then** ▷ *Case 3: Contingent timepoints also execute at t*
8: HCE(S, D, t, τ)
9: D.now $\leftarrow \rho$
10: **return** D

Algorithm 9 HCE: Handle contingent executions (Hunsberger and Posenato 2024a).

Require: S, an ESTNU; D, an RTEdata; $\rho \in \mathbb{R}$, an execution time; $\tau \subseteq U_c$, contingent timepoints to execute at ρ
Ensure: D updated
1: **for** $C \in \tau$ **do**
2: Add (C, ρ) to $D.f$
3: Remove C from $D.U_c$
4: Update time windows for neighbors of C
5: Remove C-waits from all D.AcWts sets
6: Update $D.\text{Enabs}_x$ due to incoming negative edges to C or any deleted C-waits

Algorithm 10 HXE: Handle a non-contingent execution (Hunsberger and Posenato 2024a).

Require: S, an ESTNU; D, an RTEdata structure; $t \in \mathbb{R}$; $V \in U_x$
Ensure: D updated
1: Add (V, t) to $D.f$
2: Remove V from $D.U_x$
3: Update time windows for neighbors of V
4: Update $D.\text{Enabs}_x$ due to any negative incoming edges to V
5: **if** V is activation TP for some CTP C **then**
6: **for** $(Y, C : -w, V) \in \mathscr{E}_w$ **do**
7: Insert $(t + w, C)$ into D.AcWts(Y)

Two example runs of RTE* applied to the ESTNU of Figure 4.3 are summarized in Tables 4.1 and 4.2. In the run shown in Table 4.1, C executes at time 2 while the wait constraint is still active (the wait constraint becomes active as soon as A is executed). As a result, B is executed earlier than initially dictated by the wait constraint. In the run shown

in Table 4.2, C does not execute before time 4. Therefore, B is executed at time 4, and it is later observed that C executes at time 8.

Table 4.1 *RTE* applied to ESTNU of Figure 4.3, when contingent link takes value 2 (Hunsberger and Posenato 2024a).*

Time	State	Action / Observation
$T=0$	$Z=0,\ A=0$	If nothing happens before $t=4$, execute B at $t=4$
$T=2$	$Z=0,\ A=0,\ C=2$	Observe that C executes at $t=2$
$T=2$	$Z=0,\ A=0,\ C=2,\ B=2$	Execute B at $t=2$

Table 4.2 *RTE* applied to ESTNU of Figure 4.3, when contingent link takes value 8 (Hunsberger and Posenato 2024a).*

Time	State	Action / Observation
$T=0$	$Z=0,\ A=0$	If nothing happens before $t=4$, execute B at $t=4$
$T=4$	$Z=0,\ A=0,\ B=4$	Execute B at $t=4$
$T=8$	$Z=0,\ A=0,\ B=4,\ C=8$	Observe that C executes at $t=8$

The literature on STNU theory is extensive, and we have aimed to limit the scope of the preceding paragraphs to the foundations necessary for the STNU-based approach presented in Section 4.4.3. For further details, we find the papers by Hunsberger and Posenato (2024a), Morris (2014), and Morris (2016) and the online toolbox by Posenato (2022) particularly informative.

Related work on STNUs for SRCPSP/max

As far as we know, the DC-checking and RTE* algorithms have not been applied to SRCPSP/max, despite their efficiency. The works by Lombardi and Milano (2009) and Lombardi, Milano, and Benini (2013) on POS are the only ones we know of that use STNUs as their temporal model. They introduce nodes for the start and end of each activity, with a duration constraint between them, and connect the respective start nodes with edges for the precedence constraints. Lombardi and Milano (2009) and Lombardi, Milano, and Benini (2013) used constraint propagation for DC-checking, but do not use RTE*. Other works on POS for SRCPSP/max that do not use STNUs and DC-checking risk violations of minimal or maximal time-lags during execution (Fu, Varakantham, and Lau 2016; Policella, Cesta, *et al.* 2007). Thus, we conclude that there is a research gap in applying the developments in STNU literature to SRCPSP/max.

4.3.3. Benchmarking approaches

Benchmarking procedures for comparing scheduling methods are inconsistent in the literature, with varying problem sets and comparison methods. Some studies used industrial scheduling instances (Lombardi and Milano 2009; Lombardi, Milano, and Benini 2013), while the majority (Fu, Lau, *et al.* 2012; Fu, Varakantham, and Lau 2016; Policella, Smith, *et al.* 2004; Varakantham, Fu, and Lau 2016) relied on PSPlib (Kolisch and Sprecher 1996)

instances, which are deterministic and transformed into stochastic versions with noise. Research typically focused on instances with 10, 20, and 30 activities (j10-j30). Different studies assessed varying metrics, such as schedule flexibility and robustness (Policella, Cesta, *et al.* 2007; Policella, Smith, *et al.* 2004), solver performance (Lombardi and Milano 2009; Lombardi, Milano, and Benini 2013), or the α-robust makespan (Fu, Lau, *et al.* 2012; Fu, Varakantham, and Lau 2016; Varakantham, Fu, and Lau 2016). However, no comprehensive benchmarking paper exists that evaluates both solution quality and computation time while also correctly accounting for infeasibilities. The main challenge is that not all instances can be solved by all methods, making it difficult to directly compare the distributions of solution quality or other metrics. We take inspiration from (Long and Fox 2003), who compare different planners that can fail, providing a framework for comparing solution quality and speed while also correctly considering these failures.

4.4. Scheduling methods

This section outlines the proposed methods. We explain how to use CP for these scheduling problems, which have so far been dominated by MIP approaches. Note that in our initial experiments, we include a comparison between a CP and an MIP approach for RCPSP/max (CPOptimizer and CPLEX; (IBM 2024)) demonstrating that CP outperforms MIP for this problem, which is in line with the literature.

First, we present a deterministic CP model for RCPSP/max in Section 4.4.1, which we compare to a deterministic MIP model in Section 4.4.2. The new, stochastic methods for SRCPSP/max are proposed in Section 4.4.3. We explain the statistical tests for performing pairwise comparisons of these new methods that lead to partial orderings based on solution quality and computation time in Section 4.4.4.

4.4.1. Constraint programming for RCPSP/max

For the deterministic RCPSP/max, we use the modern interval constraints from the IBM CP optimizer (IBM 2024). We use IBM's syntax and modify the RCPSP example from (Laborie *et al.* 2018) to RCPSP/max. We use the following nomenclature:

Sets

J : set of tasks, indexed by j,
R : set of resources, indexed by r,
T : set of discrete time points, indexed by t.

Parameters

$$
\begin{aligned}
d_j &: \text{duration of task } j, \quad \forall j \in J, \\
c_r &: \text{capacity of resource } r, \quad \forall r \in R, \\
r_{r,j} &: \text{demand of task } j \text{ on resource } r, \quad \forall r \in R, j \in J, \\
\mathsf{TemporalConstraints} &: \text{ set of precedence relationships } (i, \mathsf{lag}, j).
\end{aligned}
$$

Decision variables

$$
\begin{aligned}
x_j &: \text{IntervalVar}(J, d_j);\ \forall j \in J, \\
\mathsf{Makespan} &: \text{makespan (objective to minimize)}.
\end{aligned}
$$

Model The CP model is:

$$
\begin{aligned}
&\text{Min Makespan} \quad \text{subject to} \\
&\qquad \text{EndOf}(x_j) \le \mathsf{Makespan}; && \forall j \in J \\
&\qquad \text{StartOf}(x_i) + \mathsf{lag} \le \text{StartOf}(x_j); && \forall (i, \mathsf{lag}, j) \in \mathsf{TemporalConstraints} \\
&\qquad \sum_{j \in J} \text{Pulse}(x_j, l_{r,j}) \le c_r; && \forall r \in R
\end{aligned}
$$

The set of temporal constraints TemporalConstraints provides both minimal time lags and maximal time lags that are the temporal differences between start times of activities i and j if i is a predecessor of j with a temporal delay equal to lag which can be both negative or positive. We introduce the decision variable x_j as the interval variable for activity $j \in J$. The Pulse function generates a cumulative expression over a given interval x_j with a certain value. For a task j, this value is its resource usage $l_{r,j}$. The aggregated pulse values are constrained so that their sum does not exceed the total available resource capacity c_r.

4.4.2. Mixed Integer Programming for RCPSP/max

The MIP model that we use in the comparison is the following:

Decision variables

$$
\begin{aligned}
x_{j,t} &: \begin{cases} 1 & \text{if task } j \text{ starts at time } t, \\ 0 & \text{otherwise,} \end{cases} && \forall j \in J, t \in T, \\
\mathsf{Makespan} &: \text{makespan (objective to minimize)}.
\end{aligned}
$$

Model The MIP model is given by:

$$
\begin{aligned}
&\text{Min Makespan} \quad \text{subject to} \\
&\sum_{t \in T} t \cdot x_{j,t} + d_j, \leq \text{Makespan}; && \forall j \in J \\
&\sum_{t \in T} t \cdot x_{i,t} + \text{lag} \leq \sum_{t \in T} t \cdot x_{j,t}; && \forall (i, \text{lag}, j) \in \text{TemporalConstraints} \\
&\sum_{j \in J} \sum_{\tau = t - d_j + 1}^{t} l_{r,j} \cdot x_{j,\tau} \leq b_r; && \forall r \in R, \forall t \in T \\
&\sum_{t \in T} t \cdot x_{i,t} \geq 0; && \forall j \in J \\
&\sum_{t \in T} x_{j,t} = 1; && \forall j \in J
\end{aligned}
$$

4

4.4.3. New methods for SRCPSP/max

This section introduces three new methods for SRCPSP/max. The first method is a CP-based version of a proactive model. Then, we present a novel, fully reactive scheduling approach employing the deterministic model for RCPSP/max. Finally, we propose an STNU-based approach using CP and POS. We refer to these approaches as proactive, reactive, and stnu, respectively.

Proactive method

We outline how to use a scenario-based CP model (instead of MIP) for SRCPSP/max which we call $\text{proactive}_{\text{SAA}}$.

For this SAA method, we can reuse the deterministic RCPSP/max CP model, introduced in Section 4.4.1, but we introduce scenarios. This model is inspired by the MIP version by Varakantham, Fu, and Lau (2016). A special variant is the SAA with only one sample, for which a γ-quantile can be used which we call $\text{proactive}_{\gamma}$. If a feasible schedule can be found for the γ-quantile, this schedule will also be feasible for all duration realizations on the left-hand side of the γ-quantile because of Proposition 1.

We provide the SAA model below. We use the same nomenclature as for the deterministic model, but we introduce the notion of scenarios $\omega \in \Omega$, and find a start time schedule $s_j \ \forall j \in J$ that is feasible for all scenarios it has seen if one exists:

Decision variables

$$
\begin{aligned}
x_j^{\omega} &: \text{IntervalVar}(J, d_j); \ \forall j \in J, \ \forall \omega \in \Omega, \\
\text{Makespan}(\omega) &: \text{makespan for each scenario.}
\end{aligned}
$$

Model

$$\text{Min} \frac{1}{|\Omega|} \sum_{\omega \in \Omega} \text{Makespan}(\omega) \quad \text{subject to}$$

$$\begin{aligned}
\text{EndOf}(x_j^\omega) \leq \text{Makespan}(\omega); \quad & \forall j \in J,\ \forall \omega \in \Omega \\
\text{StartOf}(x_i^\omega) + \text{lag} \leq \text{StartOf}(x_j^\omega); \quad & \forall (i, \text{lag}, j) \in \text{TemporalConstraints} \\
& \forall \omega \in \Omega \\
\sum_{j \in J} \text{Pulse}(x_j^\omega, l_{r,j}) \leq c_r; \quad & \forall r \in R\ \forall \omega \in \Omega \\
s_j = \text{StartOf}(x_j^\omega); \quad & \forall j \in J\ \forall \omega \in \Omega
\end{aligned}$$

Reactive method

We now present a CP-based fully reactive approach for SRCPSP/max that we refer to as reactive.

Fully reactive approaches, involving complete rescheduling by solving a deterministic RCPSP/max, have been considered impractical due to high computational demands and low schedule stability (Van de Vonder, Demeulemeester, and Herroelen 2007). However, advances in CP for scheduling (Laborie *et al.* 2018; Naderi, Ruiz, and Roshanaei 2023) mitigate these issues. In the industrial setting where we are applying our work, decision-makers often reschedule their entire future plans when changes occur. Thus, including this reactive approach in our comparison is valuable. To our knowledge, such an approach has not been evaluated before. The outline is:

- Start by making an initial schedule with an estimation of the activity durations $\hat{d}$. We can see $\hat{d}$ as a hyperparameter for how conservative the estimation is. For example, using the mean of the distribution could lead to better makespans, but the risk of becoming infeasible for larger duration realizations, while taking the upper bound of the distribution could lead to a very high makespan.
- At every decision moment (when an activity finishes) resolve the deterministic RCPSP/max while fixing all variables until the current time to reschedule with new information, we again use the estimation $\hat{d}$ for the activities that did not finish yet. Resolving is needed when the finish time of an activity deviates from the estimated finish time. We warm start the solver with the previous solution.

STNU-based method

Finally, we present a partial order schedule approach using CP and STNU algorithms (Hunsberger and Posenato 2024a) which we call `stnu`. This approach is inspired by many earlier works (Fu, Varakantham, and Lau 2016; Lombardi and Milano 2009; Policella, Cesta, *et al.* 2007). We use a fixed-solution approach for constructing the ordering constraints. The outline of the approach is:

1. We make a fixed-point schedule by solving the deterministic RCPSP/max with an estimation of the activity durations $\hat{d}$ and using the chaining procedure, which was explained in Section 4.3.2.

2. We construct the STNU as follows:

   a) For each activity two nodes are created, representing its start and end.

   b) Contingent links are included between the start of the activity and the end of the activity with $[LB, UB]$, where LB and UB are the lower and upper bounds of the duration of that activity.

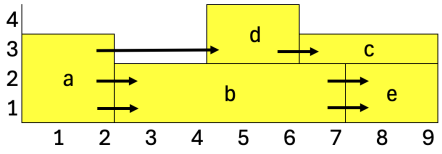
   c) The minimal and maximal time lags are modeled using edges $(start_B, -min, start_A)$ and $(start_A, max, start_B)$, where A and B are the preceding and succeeding activity, respectively. These edges correspond to the edges in the precedence graph of the instance, see Figure 4.1.

   d) The resource chains that construct the POS are added as additional edges as $(start_B, 0, end_A)$ if activity A precedes activity B. Each arrow in Figure 4.2 would lead to a resource chain edge.

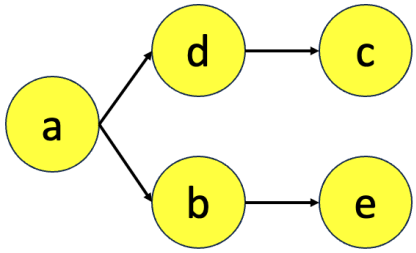
The resulting STNU is tested for dynamic controllability (DC), and if the network is DC its extended form (ESTNU) is given to the RTE* algorithm. Since makespan minimization is of interest, we slightly adjust the algorithm from (Hunsberger and Posenato 2024a): instead of selecting an arbitrary executable time point and an arbitrary allowed execution time, we always choose the earliest possible time point at the earliest possible execution time.

In the next paragraphs, we include a step-by-step example of the STNU-based method. We showcase what the temporal network for the given problem instance looks like and how we can determine whether it is DC. We reuse the example instance (Figure 4.1) with introduced uncertainty in the durations. Suppose that activity d follows a uniform discrete distribution with $P(d = 1) = 0.5$ and $P(d = 2) = 0.5$.

Step 1: Get fixed-point schedule by solving the deterministic RCPSP/max. Suppose we use $\hat{d} = \{\hat{d}_a = 2, \hat{d}_b = 5, \hat{d}_c = 3, \hat{d}_d = 2, \hat{d}_e = 2\}$, and obtain the following fixed-point solution:



(a) *Fixed-point solution with resource chains.*



(b) *Partial order schedule based on resource chains.*

Figure 4.4 *Partial order schedules.*

Step 2: construct the STNU . See in Figures 4.5 - 4.7 how an STNU is constructed step by step.

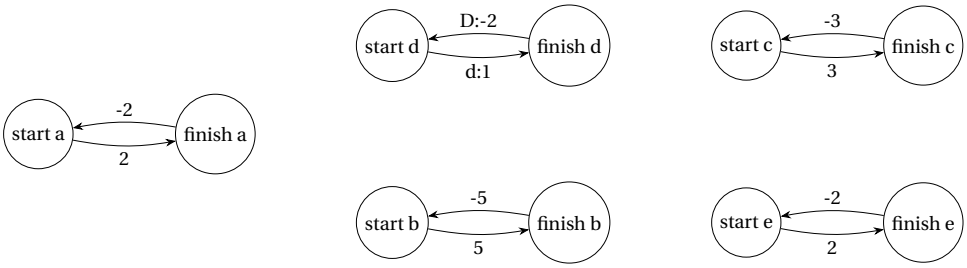


Figure 4.5 *Step 2a: For each activity, two nodes are created, representing its start and end, and Step 2b: We connect the start and end of each activity with a tight link (activities a, b, c, e) or a contingent link (activity d).*

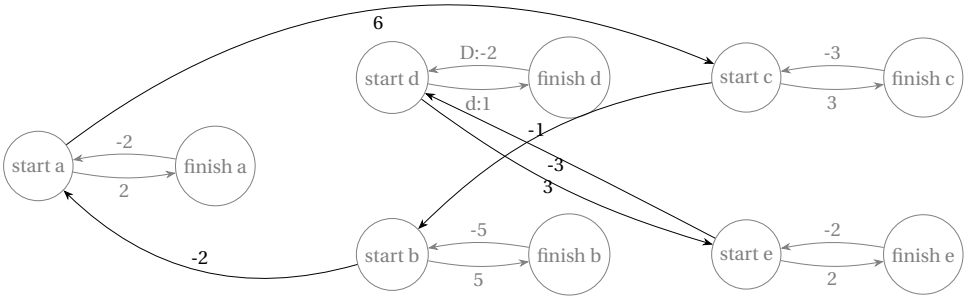


Figure 4.6 *Step 2c: The minimal and maximal time lags are modeled. These edges correspond to the precedence graph of Figure 4.1.*

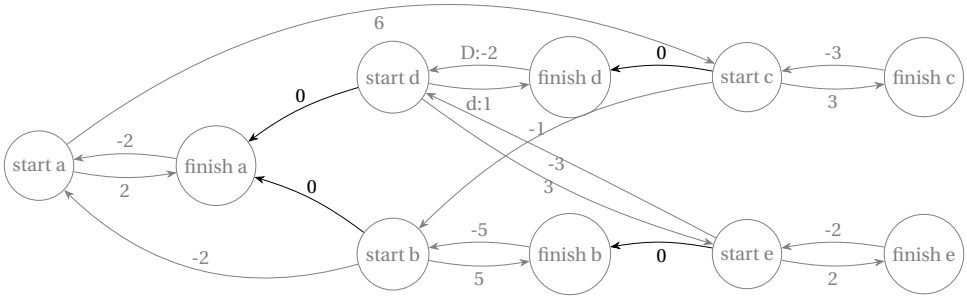


Figure 4.7 *Step 2d: The resource chains that construct the POS are added as additional edges as* $(start_B, 0, end_A)$ *if activity A precedes activity B. Each arrow in Figure 4.4 would lead to a resource chain edge.*

Step 3: Check whether the network is dynamically controllable . The resulting network in Figure 4.7 does not contain any negative cycles, and therefore it is dynamically controllable (Morris 2014). Note that for this specific example, if we had used the fixed-point solution and resource chains from Figure 4.2, we would have found a negative cycle in the resulting STNU and, therefore, no dynamic controllability; see Figure 4.8.

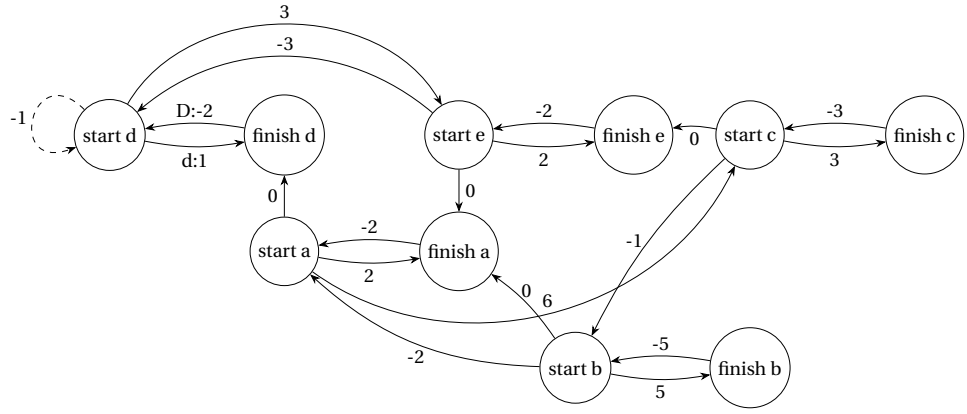


Figure 4.8 *This figure shows an example of a non-DC network (the resource chains from Figure 4.2), which is due to a negative cycle. The negative cycle can be detected by backward propagation (using the algorithm from Morris (2014)) and is the path:* $start_D \leftarrow end_D \leftarrow start_A \leftarrow start_E \leftarrow end_A \leftarrow start_D$ *with length* -1.

Step 4: Run `RTE*`**.** `RTE*`, presented in Algorithm 5 (Hunsberger and Posenato 2024a; Posenato 2022), provides the online scheduling strategy that transforms the network into a schedule (i.e., it assigns execution times to each node). `RTE*` adapts the execution times to the outcome of the contingent time points, resulting in schedule $\{s_a = 0, s_b = 2, s_c = 5, s_d = 4, s_e = 7\}$ if $d_d = 1$ and $\{s_a = 0, s_b = 2, s_c = 6, s_d = 4, s_e = 7\}$ if $d_d = 2$.

4.4.4. Statistical tests for pairwise comparison

A strategy for benchmarking is to provide partial orderings of the scheduling methods for the different metrics of interest (e.g., solution quality, runtime offline, runtime online). A partial ordering can be obtained by executing pairwise comparisons of the methods per problem size and per metric, taking inspiration from (Long and Fox 2003).

The Wilcoxon Matched-Pairs Rank-Sum Test (the version by Cureton (1967)) looks at the ranking of absolute differences and gives insight into which of a pair of methods has consistently better performance than another method. Infeasible cases can be handled by assigning infinitely bad time and solution quality to these cases, leading to an absolute difference of ∞ or $-\infty$ that will be pushed to the highest and lowest rankings.

An alternative to the Wilcoxon test, also used by Long and Fox (2003), is the proportion test (see Test 4 in the book by Kanji (2006)). This test is weaker than the Wilcoxon, but provides at least information about significance in the proportion of wins when the Wilcoxon test shows no significant difference.

The magnitude test provides more insight into the magnitude differences of the performance metrics. This test is also known as the pairwise t-test on two related samples of scores (see Test 10 in the book by Kanji (2006)). This test can only be performed on double hits because infinitely bad computation time or solution quality will disturb the test.

We provide a detailed explanation of all of the above statistical tests in Appendix 4.B

4.5. Experimental evaluation

The goal of the evaluation is to analyze the relative performance of the different proposed scheduling methods.

4.5.1. Data generation

We use the j10, j20, j30, ubo50, and ubo100 sets from the PSPlib (Kolisch and Sprecher 1996). Instead of using all instances from j10 to j30, we select 50 per set and extend previous work (Fu, Varakantham, and Lau 2016; Varakantham, Fu, and Lau 2016) by including 50 instances each from the ubo50 and ubo100 sets. Following prior research, we use deterministic durations to generate stochastic distributions with different noise levels, thereby converting deterministic instances into stochastic ones. We assume d_j follows a discrete uniform distribution based on the deterministic processing times in the PSPlib data sets p_j,

$$d_j \sim \text{DiscreteUniform}(lb_j, ub_j),$$

where lb_j and ub_j denote the minimum and maximum possible durations for activity j. Specifically, we define the lower bound as $lb_j = \max(1, \text{round}(p_j - \epsilon \cdot \sqrt{p_j}))$ and the upper bound as $ub_j = \text{round}(p_j + \epsilon \cdot \sqrt{p_j})$, where we vary the noise level $\epsilon = \{1, 2\}$. The source and sink nodes always have a deterministic duration of zero. Each evaluation of a method corresponds to one sample from the distribution, and we sample 10 times for each instance. We excluded the instance samples for which it is impossible to find a feasible schedule from the test set (i.e., the deterministic problem with perfect information is infeasible for the given activity duration sample).

4.5.2. Initial experiment: CP vs MIP

First, we obtain a comparison of a CP-based formulation and an MIP-based formulation for the deterministic RCPSP/max problem. We find that CP outperforms MIP on the selection of instances.

Table 4.3 *Runtime CP vs MIP, timelimit set to 600 seconds.*

Instance folder	# Instances	Average time CP	Average of time MIP
j10	50	0,02	0,41
j20	50	24,34	17,04
j30	50	1,28	68,58
ubo50	50	75,09	210,96
ubo100	50	135,30	510,75

Table 4.4 *Count hit timelimit (10 minutes) CP vs MIP*

Instance folder	# Instances	# Count CP	# Count MIP
j10	50	0	0
j20	50	2	0
j30	50	0	5
ubo50	50	6	14
ubo100	50	11	39

4.5.3. Tuning of the methods

In this section, we highlight the most important observations and outcomes of our tuning results; for further details, see Appendix 4.A.

All CP models are solved with the IBM CP solver (IBM 2024) with default settings.

Most of the deterministic RCPSP/max the j10-j30 instances can be solved within 60 seconds. For, ubo50 and ubo100 instances, we fixed the time limit to 600 seconds. We tune the classical SAA with multiple scenarios $\mathsf{proactive}_{\mathsf{SAA}}$ and a heuristic approach $\mathsf{proactive}_{\gamma}$. We used $\gamma = 0.9$ for $\mathsf{proactive}_{\gamma}$. For $\mathsf{proactive}_{\mathsf{SAA}}$, we sampled quantiles at $\gamma \in [0.25, 0.5, 0.75, 0.9]$ and set a time limit of 1800 seconds. The algorithm reactive uses offline the $\mathsf{proactive}_{\gamma}$ algorithm, for which we fixed $\gamma = 0.9$. We investigated the effect of the time limit for rescheduling and set this hyperparameter to 2 seconds. The algorithm stnu consists of an offline procedure, which comprises building up the network and checking dynamic controllability; if it's not DC, the method fails. We observed that the choice of the γ-quantile for the fixed-point schedule significantly affects the ratio of DC networks. We select $\gamma = 1$ for our final method stnu, because this setting results in much more DC networks than lower quantiles.

4.5.4. Results

We include an analysis of the feasibility ratios of the different methods. A selection of the results of the statistical tests are shown in Table 4.6 (the remaining results are included

in the Appendix 4.C). We present summarized partial orderings on 1) solution quality, 2) time offline, 3) time online.

Feasibility ratio

Table 4.5 *Feasibility Ratios for* $\epsilon = 1$ *and* $\epsilon = 2$.

Set	$\epsilon = 1$				$\epsilon = 2$			
	stnu	pro_{SAA}	$\text{pro}_{0.9}$	react	stnu	pro_{SAA}	$\text{pro}_{0.9}$	react
j10	0.65	0.85	0.85	0.85	0.63	0.63	0.64	0.63
j20	0.65	0.76	0.76	0.76	0.63	0.54	0.53	0.53
j30	0.78	0.89	0.89	0.89	0.63	0.51	0.48	0.49
ubo50	0.77	0.86	0.86	0.86	0.67	0.41	0.42	0.41
ubo100	0.84	0.91	0.91	0.88	0.79	0.35	0.36	0.33

First, we analyze the feasibility ratio obtained by the different methods in Table 4.5. For $\epsilon = 1$, we observe the feasibility ratio obtained by $\text{proactive}_{\text{SAA}}$, $\text{proactive}_{0.9}$, reactive are the same for instance sets j10 - ubo50, and only for ubo100 reactive is a little bit lower than the $\text{proactive}_{\text{SAA}}$, $\text{proactive}_{0.9}$. The stnu method has the lowest feasibility rate, but remarkably, the difference becomes smaller when the size of the problem grows (the difference is the smallest for ubo100).

We observe a different pattern for $\epsilon = 2$, with a higher noise factor. The highest feasibility ratios are obtained by stnu. This could be explained by the fact that the stnu method uses the information about the distribution and is, therefore, better at handling the larger variances in activity duration.

Results statistical tests

Table 4.6 shows a subset of the pairwise test results for the metrics solution quality, offline time, and online time. In Appendix 4.C, we provide all test results per instance set / noise level ϵ setting. The test results can be used to make partial orderings of the methods, distinguishing between a strong partial ordering (Wilcoxon test) and a weak ordering (proportion test). Besides that, the results of the magnitude test give insight in the magnitude differences of each performance metric based on the double hits. For example, in Table 4.6, the normalized average makespan is 0.958 for stnu and 1.015 for reactive, with a significant advantage for stnu on double hits.

Partial ordering visualization

In Figures 4.11–4.10, an arrow $A \rightarrow B$ indicates that in the majority of the settings either A is consistently better than B (Wilcoxon) and/or A is better than B a significant number of times (proportion). Due to space constraints and for clarity, we have chosen to display only the most common pattern per metric rather than all partial orderings per instance set and noise level. Consequently, the distinction between strong and weak partial orderings is omitted in these figures. We refer to Appendix 4.C for the partial orderings per setting, including the distinction between a strong and weak partial ordering.

Table 4.6 *Summary of all pairwise test results for ubo50, $\epsilon = 1$, covering solution quality and runtime. Metrics include Wilcoxon, proportion, and magnitude tests (Section 4.4.4). Results: [pairs] z-value (p-value) (* for $p < 0.05$), proportion (p-value), and t-stat (p-value) with normalized averages. Additional results for other ϵ, j10–30, ubo50, and ub0100 are in Appendix 4.C. Column headers list the compared methods, excluding zero-difference pairs.*

Test	Legend	stnu-react	stnu-pro$_{SAA}$	stnu-pro$_{0.9}$	react-pro$_{SAA}$	react-pro$_{0.9}$	pro$_{SAA}$-pro$_{0.9}$
Wilc. Quality	[n] z (p)	[370] -6.75 (*)	[370] -6.8 (*)	[370] -6.86 (*)	[370] -2.76 (*)	[370] -8.53 (*)	[370] -7.09 (*)
Prop. Quality	[n] prop (p)	[277] 0.78 (*)	[276] 0.78 (*)	[278] 0.78(*)	[61] 0.67 (*)	[73] 1.0 (*)	[65] 0.94 (*)
Magn. Quality	[n] t (p) norm. avg. norm. avg.	[330] -11.36 (*) stnu: 0.985 react: 1.015	[330] -11.39 (*) stnu: 0.985 pro$_{SAA}$: 1.015	[330] -11.35 (*) stnu: 0.984 pro$_{0.9}$: 1.016	[370] -2.86 (*) react: 1.0 pro$_{SAA}$: 1.0	[370] -6.73 (*) react: 0.999 pro$_{0.9}$: 1.001	[370] -6.01 (*) pro$_{SAA}$: 0.999 pro$_{0.9}$: 1.001

Test	Legend	react-stnu	react-pro$_{SAA}$	pro$_{0.9}$-stnu	pro$_{0.9}$-pro$_{SAA}$	stnu-pro$_{SAA}$
Wilc. Offline	[n] z (p)	[370] -16.67 (*)	[370] -16.67 (*)	[370] -16.67 (*)	[370] -16.67 (*)	[370] -7.26 (*)
Prop. Offline	[n] prop (p)	[370] 1.0 (*)	[370] 1.0 (*)	[370] 1.0 (*)	[370] 1.0 (*)	[370] 0.62 (*)
Magn. Offline	[n] t (p) norm. avg. norm. avg.	[330] -36.15 (*) react: 0.36 stnu: 1.64	[370] -72.56 (*) react: 0.24 pro$_{SAA}$: 1.76	[330] -36.15 (*) pro$_{0.9}$: 0.36 stnu: 1.64	[370] -72.56 (*) pro$_{0.9}$: 0.24 pro$_{SAA}$: 1.76	[330] -5.64 (*) stnu: 0.82 pro$_{SAA}$: 1.18

Test	Legend	pro$_{0.9}$-stnu	pro$_{SAA}$-stnu	pro$_{0.9}$-react	pro$_{SAA}$-react	stnu-react
Wilc. Online	[n] z (p)	[370] -16.67 (*)	[370] -16.67 (*)	[370] -16.67 (*)	[370] -16.67 (*)	[370] -9.85 (*)
Prop. Online	[n] prop (p)	[370] 1.0 (*)	[370] 1.0 (*)	[370] 1.0 (*)	[370] 1.0 (*)	[370] 0.89 (*)
Magn. Online	[n] t (p) norm. avg. norm. avg.	[330] -3.95e3 (*) pro$_{0.9}$: 0.01 stnu: 1.99	[330] -3.98e3 (*) pro$_{SAA}$: 0.01 stnu: 1.99	[370] -3.07e4 (*) pro$_{0.9}$: 0.0 react: 2.0	[370] -3.14e4 (*) pro$_{SAA}$: 0.0 react: 2.0	[330] -142.76 (*) stnu: 0.15 react: 1.85

4.5.5. Analysis

In this section, we summarize the main findings of our analysis on comparing the different scheduling methods based on computation time offline, online and solution quality.

Computation time

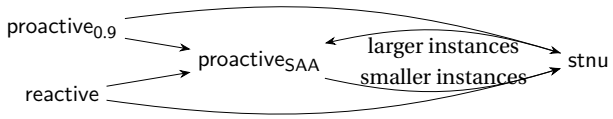


Figure 4.9 *Summarizing illustration of the partial ordering of the different methods for time offline.*

Figure 4.9 shows that the proactive$_{0.9}$ and reactive have the lowest relative **offline runtime** and we found no significant difference between the two. The ordering of stnu and proactive$_{SAA}$ depends on the problem size (the ordering flips for larger instances). These relative orderings are confirmed with the magnitude test on double hits. However, we assign infinitely bad offline computation time to infeasible solutions. When executing the Wilcoxon test we include all instances for which at least one of the two methods generated a feasible solution. For that reason, a flip in the partial ordering occurs for $\epsilon = 2$, ubo50

and ubo100: stnu shows the best performance, and proactive$_{0.9}$ outperforms reactive according to the Wilcoxon tests. This was mainly due to the higher feasibility ratio for the better methods (see Table 4.5) as the results from the magnitude test on double hits contradicted Wilcoxon in these cases (we did not visualize this pattern in the main paper, but it is included in Appendix 4.C).

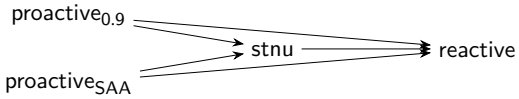


Figure 4.10 *Summarizing illustration of the partial ordering of the different methods for time online.*

We observe the general partial ordering for **online runtime** in Figure 4.10. The superiority of proactive$_{0.9}$ and proactive$_{SAA}$ are expected, as these methods only require a feasibility check online. The faster online time of the stnu compared to the reactive can be explained by the fact that the stnu employs a polynomial real-time execution algorithm, while reactive calls a deterministic CP solver multiple times. These results are confirmed with a magnitude test on double hits. Again, there are a few settings ($\epsilon = 2$, ubo50 and ubo100) in which the magnitude and the Wilcoxon test contradict each other: stnu outperforms the proactive methods based on the Wilcoxon test due to higher feasibility ratios, while the magnitude test on double hits shows better online runtime for proactive$_{0.9}$ and proactive$_{SAA}$.

Solution quality (makespan)

Figure 4.11 shows the visualization of the partial ordering for **solution quality** (makespan). The results are consistent for the different instance sets and noise levels. The stnu shows to be the outperforming method based on solution quality. The reactive approach outperforms the proactive methods. Furthermore, proactive$_{SAA}$ outperforms in many cases proactive$_{0.9}$, although for larger instances and a higher noise level a significant difference is not present. In earlier work, proactive approaches were considered state-of-the-art, but in our analysis, we found better makespan results for the STNU-based approach. We found that for each pair for which an arrow is visualized in Figure 4.11 also a significant magnitude difference was found on the double hits.

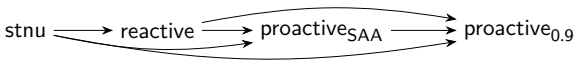


Figure 4.11 *Summarizing illustration of the partial ordering of the different methods for solution quality.*

In Table 4.7 and Table 4.8, a comparison of the magnitude differences in solution quality (makespan) on the double hits is provided. Here, the normalized averages indicate the extent of this difference. We see that the stnu method yields the lowest normalized average make-span across all pairwise comparisons on the double hits. In the comparison

of reactive and proactive we see lower normalized averages for the reactive approach. Furthermore, we observe that for $\epsilon = 2$, the magnitude differences are larger across the different pairs.

Table 4.7 *Magnitude test on solution quality for noise factor $\epsilon = 1$. Using a pairwise t-test, including all instances for which both methods found a feasible solution. Each cell shows on the first row [nr pairs] t-stat (p-value) with (*) for $p < 0.05$ and on the second row the normalized average of method 1 and on the third row the normalized average of method 2.*

j10	stnu-react	stnu-pro$_{SAA}$	stnu-pro$_{0.9}$	react-pro$_{SAA}$	react-pro$_{0.9}$	pro$_{SAA}$-pro$_{0.9}$
[n] t (p)	[260] -9.367 (*)	[260] -13.762 (*)	[260] -13.703 (*)	[340] -10.114 (*)	[340] -10.542 (*)	[340] -5.637 (*)
norm. avg.	stnu: 0.974	stnu: 0.962	stnu: 0.962	react: 0.986	react: 0.985	pro$_{SAA}$: 0.999
norm. avg.	react: 1.026	pro$_{SAA}$: 1.038	pro$_{0.9}$: 1.038	pro$_{SAA}$: 1.014	pro$_{0.9}$: 1.015	pro$_{0.9}$: 1.001
j20	stnu-react	stnu-pro$_{SAA}$	stnu-pro$_{0.9}$	react-pro$_{SAA}$	react-pro$_{0.9}$	pro$_{SAA}$-pro$_{0.9}$
[n] t (p)	[230] -7.62 (*)	[230] -8.608 (*)	[230] -8.992 (*)	[270] -6.868 (*)	[270] -7.713 (*)	[270] -5.82 (*)
norm. avg.	stnu: 0.982	stnu: 0.978	stnu: 0.977	react: 0.995	react: 0.993	pro$_{SAA}$: 0.998
norm. avg.	react: 1.018	pro$_{SAA}$: 1.022	pro$_{0.9}$: 1.023	pro$_{SAA}$: 1.005	pro$_{0.9}$: 1.007	pro$_{0.9}$: 1.002
j30	stnu-react	stnu-pro$_{SAA}$	stnu-pro$_{0.9}$	react-pro$_{SAA}$	react-pro$_{0.9}$	pro$_{SAA}$-pro$_{0.9}$
[n] t (p)	[280] -4.072 (*)	[280] -5.946 (*)	[280] -7.06 (*)	[320] -5.698 (*)	[320] -9.422 (*)	[320] -6.226 (*)
norm. avg.	stnu: 0.993	stnu: 0.99	stnu: 0.988	react: 0.997	react: 0.995	pro$_{SAA}$: 0.998
norm. avg.	react: 1.007	pro$_{SAA}$: 1.01	pro$_{0.9}$: 1.012	pro$_{SAA}$: 1.003	pro$_{0.9}$: 1.005	pro$_{0.9}$: 1.002
ubo50	stnu-react	stnu-pro$_{SAA}$	stnu-pro$_{0.9}$	react-pro$_{SAA}$	react-pro$_{0.9}$	pro$_{SAA}$-pro$_{0.9}$
[n] t (p)	[330] -11.359 (*)	[330] -11.391 (*)	[330] -11.346 (*)	[370] -2.858 (*)	[370] -6.734 (*)	[370] -6.014 (*)
norm. avg.	stnu: 0.985	stnu: 0.985	stnu: 0.984	react: 1.0	react: 0.999	pro$_{SAA}$: 0.999
norm. avg.	react: 1.015	pro$_{SAA}$: 1.015	pro$_{0.9}$: 1.016	pro$_{SAA}$: 1.0	pro$_{0.9}$: 1.001	pro$_{0.9}$: 1.001
ubo100	stnu-react	stnu-pro$_{SAA}$	stnu-pro$_{0.9}$	react-pro$_{SAA}$	react-pro$_{0.9}$	pro$_{SAA}$-pro$_{0.9}$
[n] t (p)	[358] -6.316 (*)	[370] -6.813 (*)	[370] -6.817 (*)	[388] -2.833 (*)	[388] -7.738 (*)	[400] -0.514 (0.608)
norm. avg.	stnu: 0.991	stnu: 0.99	stnu: 0.99	react: 0.999	react: 0.999	pro$_{SAA}$: 1.0
norm. avg.	react: 1.009	pro$_{SAA}$: 1.01	pro$_{0.9}$: 1.01	pro$_{SAA}$: 1.001	pro$_{0.9}$: 1.001	pro$_{0.9}$: 1.0

Table 4.8 *Magnitude test on solution quality for noise factor $\epsilon = 2$. Using a pairwise t-test, including all instances for which both methods found a feasible solution. Each cell shows on the first row [nr pairs] t-stat (p-value) with (*) for $p < 0.05$ and on the second row the normalized average of method 1 and on the third row the normalized average of method 2.*

j10	stnu-react	stnu-pro$_{SAA}$	stnu-pro$_{0.9}$	react-pro$_{SAA}$	react-pro$_{0.9}$	pro$_{SAA}$-pro$_{0.9}$
[n] t (p)	[205] -9.128 (*)	[204] -16.872 (*)	[206] -17.395 (*)	[230] -10.858 (*)	[233] -11.811 (*)	[231] -4.856 (*)
norm. avg.	stnu: 0.963	stnu: 0.929	stnu: 0.925	react: 0.966	react: 0.964	pro$_{SAA}$: 0.997
norm. avg.	react: 1.037	pro$_{SAA}$: 1.071	pro$_{0.9}$: 1.075	pro$_{SAA}$: 1.034	pro$_{0.9}$: 1.036	pro$_{0.9}$: 1.003
j20	stnu-react	stnu-pro$_{SAA}$	stnu-pro$_{0.9}$	react-pro$_{SAA}$	react-pro$_{0.9}$	pro$_{SAA}$-pro$_{0.9}$
[n] t (p)	[170] -10.855 (*)	[172] -11.237 (*)	[170] -11.89 (*)	[174] -4.95 (*)	[176] -6.22 (*)	[173] -3.656 (*)
norm. avg.	stnu: 0.972	stnu: 0.968	stnu: 0.964	react: 0.991	react: 0.989	pro$_{SAA}$: 0.997
norm. avg.	react: 1.028	pro$_{SAA}$: 1.032	pro$_{0.9}$: 1.036	pro$_{SAA}$: 1.009	pro$_{0.9}$: 1.011	pro$_{0.9}$: 1.003
j30	stnu-react	stnu-pro$_{SAA}$	stnu-pro$_{0.9}$	react-pro$_{SAA}$	react-pro$_{0.9}$	pro$_{SAA}$-pro$_{0.9}$
[n] t (p)	[140] -4.151 (*)	[148] -6.017 (*)	[138] -6.149 (*)	[152] -4.199 (*)	[160] -6.746 (*)	[150] -5.404 (*)
norm. avg.	stnu: 0.99	stnu: 0.986	stnu: 0.985	react: 0.994	react: 0.991	pro$_{SAA}$: 0.996
norm. avg.	react: 1.01	pro$_{SAA}$: 1.014	pro$_{0.9}$: 1.015	pro$_{SAA}$: 1.006	pro$_{0.9}$: 1.009	pro$_{0.9}$: 1.004
ubo50	stnu-react	stnu-pro$_{SAA}$	stnu-pro$_{0.9}$	react-pro$_{SAA}$	react-pro$_{0.9}$	pro$_{SAA}$-pro$_{0.9}$
[n] t (p)	[141] -7.974 (*)	[146] -8.139 (*)	[148] -8.125 (*)	[145] -3.581 (*)	[155] -4.646 (*)	[150] -2.823 (*)
norm. avg.	stnu: 0.984	stnu: 0.981	stnu: 0.979	react: 0.998	react: 0.997	pro$_{SAA}$: 0.999
norm. avg.	react: 1.016	pro$_{SAA}$: 1.019	pro$_{0.9}$: 1.021	pro$_{SAA}$: 1.002	pro$_{0.9}$: 1.003	pro$_{0.9}$: 1.001
ubo100	stnu-react	stnu-pro$_{SAA}$	stnu-pro$_{0.9}$	react-pro$_{SAA}$	react-pro$_{0.9}$	pro$_{SAA}$-pro$_{0.9}$
[n] t (p)	[94] -0.305 (0.761)	[114] -0.296 (0.767)	[114] -0.93 (0.355)	[76] -1.573 (0.12)	[99] -4.187 (*)	[94] 0.59 (0.556)
norm. avg.	stnu: 0.999	stnu: 0.999	stnu: 0.998	react: 0.999	react: 0.999	pro$_{SAA}$: 1.0
norm. avg.	react: 1.001	pro$_{SAA}$: 1.001	pro$_{0.9}$: 1.002	pro$_{SAA}$: 1.001	pro$_{0.9}$: 1.001	pro$_{0.9}$: 1.0

4.6. Conclusion and future work

This study introduces new scheduling methods for SRCPSP/max and statistically benchmarks them, addressing the existing research gap of a lacking benchmarking paper.

Until now, proactive (SORU-H) methods were considered the best method for SRCPSP/max, although partial order schedules have shown potential in earlier research. We found that proactive methods can be improved with online rescheduling, resulting in better solution quality for the method reactive compared to proactive approaches $\text{proactive}_{\text{SAA}}$ and $\text{proactive}_{\gamma}$. We find that the algorithm stnu, which uses partial-order schedules, outperforms the other methods in terms of solution quality in our evaluation. Although in general, $\text{proactive}_{\gamma}$ and reactive have better offline computation time than stnu, and $\text{proactive}_{\text{SAA}}$ and $\text{proactive}_{\gamma}$ have better online computation time than stnu, the stnu also showed good relative runtime results due to the polynomial time STNU-related algorithms.

A limitation of the STNU-based method is the assumption of bounded uncertainty intervals for the contingent links. A related limitation of the approach described in this chapter is that if the network is not DC, the STNU method fails. In future work, we want to explore approaches that combine partial-order schedules with Probabilistic Simple Temporal Networks (Hunsberger and Posenato 2024b) to incorporate more realistic assumptions about the underlying distributions.

In future work, the same approach could be used to evaluate other scheduling problems, and we can gain more insight into how these methods perform on both well-known problems from the literature and in practical situations. Temporal constraints defined from end to start can be an interesting problem domain for future work. The stnu method can be extended in multiple directions, e.g., by rearranging the job order online or by using probabilistic STNs, both of which are interesting for future work.

Furthermore, the set of methods could be broadened to include sequential approaches (Powell 2022) and machine learning-based methods, such as the graph neural network in (Teichteil-Königsbuch *et al.* 2023), for SRCPSP.

References

Bartusch, M., R. H. Möhring, and F. J. Radermacher (1988). "Scheduling project networks with resource constraints and time windows". In: *Annals of Operations Research* 16, pp. 199–240.

Cureton, E. E. (1967). "The normal approximation to the signed-rank sampling distribution when zero differences are present". In: *Journal of the American Statistical Association* 62.319, pp. 1068–1069.

Dechter, R., I. Meiri, and J. Pearl (1991). "Temporal constraint networks". In: *Artificial Intelligence* 49.1-3, pp. 61–95.

Fu, N., H. Lau, P. Varakantham, and F. Xiao (2012). "Robust local search for solving RCPSP/max with durational uncertainty". In: *Journal of Artificial Intelligence Research (JAIR)* 43, pp. 43–86.

Fu, N., P. Varakantham, and H. Lau (2016). "Robust partial order schedules for RCPSP/max with durational uncertainty". In: *Proceedings of the International Conference on Automated Planning and Scheduling* 26, pp. 124–130.

Herroelen, W. and R. Leus (2002). "Project scheduling under uncertainty: Survey and research potentials". In: *European Journal of Operational Research* 165, pp. 289–306.

Hunsberger, L. and R. Posenato (2024a). "Foundations of dispatchability for simple temporal networks with uncertainty". In: *Proceedings of 16th International Conference Agents and Artificial Intelligence 2024*. Vol. 2, pp. 253–263.

– (2024b). "Robust execution of probabilistic STNs". In: *31st International Symposium on Temporal Representation and Reasoning (TIME 2024)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, pp. 12–1.

IBM (2024). *IBM ILOG CPLEX Optimization Studio*. IBM. Armonk, NY. URL: https://www.ibm.com/products/ilog-cplex-optimization-studio.

Kanji, G. K. (2006). *100 statistical tests*. Sage Publications.

Kleywegt, A. J., A. Shapiro, and T. Homem-de-Mello (2002). "The sample average approximation method for stochastic discrete optimization". In: *SIAM Journal on optimization* 12.2, pp. 479–502.

Kolisch, R. and A. Sprecher (1996). "PSPLIB - a project scheduling problem library". In: *European Journal of Operational Research*, pp. 205–216.

Laborie, P. (2015). *Modeling and solving scheduling problems with CP optimizer*.

Laborie, P., J. Rogerie, P. Shaw, and P. Vilím (2018). "IBM ILOG CP optimizer for scheduling: 20+ years of scheduling with constraints at IBM/ILOG". In: *Constraints* 23.

Lgelmund, G. and F. J. Radermacher (1983). "Algorithmic approaches to preselective strategies for stochastic scheduling problems". In: *Networks* 13.1, pp. 29–48.

Lombardi, M. and M. Milano (2009). "A precedence constraint posting approach for the rcpsp with time lags and variable durations". In: *International Conference on Principles and Practice of Constraint Programming*. Springer, pp. 569–583.

Lombardi, M., M. Milano, and L. Benini (2013). "Robust scheduling of task graphs under execution time uncertainty". In: *Computers, IEEE Transactions on* 62, pp. 98–111.

Long, D. and M. Fox (2003). "The 3rd international planning competition: results and analysis". In: *Journal of Artificial Intelligence Research* 20, pp. 1–59.

Morris, P. (2014). “Dynamic controllability and dispatchability relationships”. In: *Integration of AI and OR Techniques in Constraint Programming: 11th International Conference, CPAIOR 2014, Cork, Ireland, May 19-23, 2014. Proceedings 11*. Springer, pp. 464–479.
– (2016). “The mathematics of dispatchability revisited”. In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 26, pp. 244–252.
Morris, P., N. Muscettola, and T. Vidal (2001). *Dynamic control of plans with temporal uncertainty*.
Naderi, B., R. Ruiz, and V. Roshanaei (2023). “Mixed-integer programming vs. constraint programming for shop scheduling problems: new results and outlook”. In: *INFORMS Journal on Computing* 35.
Policella, N., A. Cesta, A. Oddi, and S. Smith (2007). “From precedence constraint posting to partial order schedules: A CSP approach to Robust Scheduling”. In: *AI Communications* 20, pp. 163–180.
Policella, N., S. F. Smith, A. Cesta, and A. Oddi (2004). “Generating robust schedules through temporal flexibility.” In: *ICAPS*. Vol. 4, pp. 209–218.
Posenato, R. (2022). “CSTNU Tool: A Java library for checking temporal networks”. In: *SoftwareX* 17, p. 100905.
Powell, W. B. (2022). *Reinforcement learning and stochastic optimization: a unified framework for sequential decisions*. John Wiley & Sons, Inc.
Pratt, J. W. (1959). “Remarks on zeros and ties in the Wilcoxon signed rank procedures”. In: *Journal of the American Statistical Association* 54.287, pp. 655–667.
Schutt, A., T. Feydy, P. J. Stuckey, and M. G. Wallace (2013). “Solving RCPSP/max by lazy clause generation”. In: *Journal of scheduling* 16, pp. 273–289.
Teichteil-Königsbuch, F., G. Povéda, G. G. de Garibay Barba, T. Luchterhand, and S. Thiébaux (2023). “Fast and robust resource-constrained scheduling with graph neural networks”. In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 33. 1, pp. 623–633.
Van de Vonder, S., E. Demeulemeester, and W. Herroelen (2007). “A classification of predictive-reactive project scheduling procedures”. In: *Journal of Scheduling* 10, pp. 195–207.
Varakantham, P., N. Fu, and H. Lau (2016). “A proactive sampling approach to project scheduling under uncertainty”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 30.
Virtanen, P., R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors (2020). “SciPy 1.0: fundamental slgorithms for scientific computing in Python”. In: *Nature Methods* 17, pp. 261–272.

Appendix

4.A. Tuning

This section describes the tuning process for the different scheduling methods. First, we investigate the effect of the time limit on solving the deterministic CP. Then, for the proactive approach, we tuned the time limit and the sample selection for the Sample Average Approximation (SAA). For reactive, we tuned the quantile approximation and the time limit for rescheduling.

4.A.1. Time limit for CP on deterministic instances

To understand how the time limit may affect the results, we first consider the deterministic instances of RCPSP/max. (Varakantham, Fu, and Lau 2016) reported on their time budget that *"*SORU *was able to obtain solutions within 5 minutes for every one instance in J10 and 2 hrs for the J20 instances. However, for J30, we were unable to obtain optimal solutions for certain instances at the 3-hr cut-off. On the other hand,* SORU-H *was able to generate solutions for J10 instances within half of a second, J20 instances within 10 seconds, and J30 instances within 10 minutes on average."*

In our research, we extend the instance sets with ubo50 and ubo100, for which the time limit can become more crucial. For each set (j10-ubo100), we fixed the first 50 instances from the PSPlib (Kolisch and Sprecher 1996) (PSP1 - PSP50). For j10-j30, the IBM CP Optimizer could solve all instances (PSP1-PSP50) within 60 seconds to optimality or prove infeasibility.

For the ubo50 instances PSP1 to PSP10 and the ubo100 instances PSP1 to PSP10, the effect of the time limits of 60 seconds, 600 seconds, and 3600 seconds respectively, is presented in Table 4.9 and 4.10. We observe that the solver status does not change when increasing the time limit to 600 seconds, and only 1 instance flips from feasible to infeasible when increasing the time limit to 3600 seconds. The makespan improves significantly only for the ubo100 instances at higher time limits. In the remaining experiments, we fixed the time limit to 60 seconds for solving deterministic CPs.

4.A.2. Proactive approach

The sample selection process is expected to influence the performance of the proactive method. We used a subset of the j10 and ubo50 instances (for both sets PSP_1-PSP_20, and used 10 duration samples per instance. We fixed the time limit to 60 seconds and compared the feasibility ratios in Table 4.11. We found that the higher feasibility ratios were obtained using two settings for the proactive method, being 1 sample with $\gamma = 0.9$, to which we will refer with $\text{proactive}_{0.9}$, and the setting with the smart samples to which we will refer with $\text{proactive}_{\text{SAA}}$ in the remaining of our experiments. For the SAA with four

Table 4.9 *Count per solver status IBM CP solver after different time limits on deterministic instances.*

Instance set	Time limit	Feasible	Infeasible	Optimal	Unkown
ubo50	60s	3	3	2	2
ubo50	600s	3	3	2	2
ubo50	3600s	2	3	3	2
ubo100	60s	4	5	0	1
ubo100	600s	4	5	0	1
ubo100	3600s	4	5	0	1

Table 4.10 *Average makespan (filtered feasible and optimal solutions on ubo50 and ubo100 sets.*

Instance set / time limit	60 seconds	600 seconds	3600 seconds
ubo50	204	200	200
ubo100	421	410	410

samples (smart samples), we investigated the effect of the time limit on the makespan in Table 4.12. We observed that the makespan would still improve while making the step from 600 seconds to 3600 seconds. We decided to use a 1800-second time limit instead of 3600 in the remaining experiments to conduct more.

Table 4.11 *Feasibility ratio for different γ-quantiles in proactive approach. Each cell counts hits out of 200 experiments. * Uses a combination of smart samples that are the quantiles with $\gamma \in [0.25, 0.5, 0.75, 0.9]$*

1 sample	γ =**0.5**	γ =**0.75**	γ =**0.9**	γ =**1**
j10	0.11	0.53	0.94	0.75
ubo50	0	0.01	0.92	0.80
multiple samples	n=5	n=10	n=25	n=50
j10	0.63	0.78	0.94	0.94
ubo50	0.12	0.48	0.89	0.91
smart samples*	n=4			
j10	0.94			
ubo50	0.92			

4.A.3. Reactive approach

First, we observed the effect of the duration estimations on the performance of reactive. We used a subset of the j10 and ubo50 instances (for both sets PSP_1-PSP_20, and used 10 duration samples per instance. We fixed the time limit for the initial schedule to 60 seconds and for the rescheduling to 2 seconds.

Table 4.12 *Average makespan obtained with* $\text{proactive}_{\text{SAA}}$ *with four scenarios for different time limits (filtered feasible and optimal solutions on ubo50 and ubo100 set.*

Instance set / time limit	60 seconds	600 seconds	3600 seconds
ubo50	233	232	231
ubo100	485	480	472

Table 4.13 *Feasibility ratio for different γ-quantiles in reactive approach. Each cell counts hits out of 200 experiments.*

Instance set	γ =0.5	γ =0.75	γ =0.9	γ =1
j10	0.11	0.53	0.94	0.75
ubo50	0	0.01	0.92	0.80

Remarkably, we observe similar feasibility ratios to the proactive approach, indicating that for feasibility, the initial schedule is quite important. For the final evaluation, we fixed $\gamma = 0.9$ for reactive, and will analyze how the solution quality improves with the rescheduling procedure compared to a standard proactive approach.

Next, we examine the effect of the rescheduling time limit. We used the same subset of the j10 and ubo50 instances (for both sets PSP_1-PSP_20, used 10 duration samples per instance, and runtime limits of 1, 2, 10 and 30 seconds. The results (in Table 4.13) are almost similar for the different time limits, and this might be because the solver finishes already before the time limit, and the increase in time online has mainly to do with the number of solver calls. In the experimental evaluation, we therefore fixed the rescheduling time limit to 2 seconds, which we expected to be sufficient for larger or slightly more complicated problems.

Table 4.14 *Average makespan and time online for double hits on different time limits for rescheduling. Each cell averages over double hits out of 200 experiments.*

Instance set	Time limit rescheduling	1s	2s	10s	30s
j10	makespan	38	38	38	39
j10	time online	0.03	0.04	0.04	0.04
ubo50	makespan	171	171	171	172
ubo 50	time online	15.4	15.1	15.1	15.4

4.A.4. Hyperparameters selection

This subsection presents the hyperparameters in Table 4.15 that are used in the final experiments that are also presented in the main body of Chapter 4.

Table 4.15 *Hyperparameter settings.*

Hyperparameter / algorithm	$\text{proactive}_{\text{SAA}}$	$\text{proactive}_{0.9}$	stnu	reactive
γ	[0.25, 0.5, 0.75, 0.7]	0.9	1	0.9
Time limit CP	1800s	600s	600s	600s and 2s
Solver	IBM CP	IBM CP	IBM CP	IBM CP

4.B. Statistical tests

4.B.1. Wilcoxon Test

The Wilcoxon test that we use follows (Cureton 1967) and is described below:

- Collect a set of matched pairs (the results from two different methods on one instance sample).
- Compute the difference between the two test results for each pair.

  An important remark is that because of the discrete objective values (makespan), we can obtain a zero difference when there is a tie, we use Pratt's procedure for handling ties (Pratt 1959), which includes zero-differences in the ranking process, but drops the ranks of the zeros afterward.
- Order the pairs according to the absolute values of the differences.
- Assign ranks to the pairs based on these absolute values.
- Sum the positive ranks (T_{pos}) and the negative (T_{neg}) ranks separately.
- Take the smaller of the two $T = min(T_{pos}, T_{neg})$.
- If the two methods have no consistent difference in their relative performances, then the rank sums should be approximately equal. This is tested with a normal approximation for the Wilcoxon statistic, which is outlined by (Cureton 1967). (Cureton 1967) propose a corrected normal approximation, which is needed because of the usage of the Pratt procedure for handling zero differences.
- The normalized Z-statistic is given by the formula: $z = (T - d)/se$, where d is the continuity correction from (Cureton 1967), and se is the standard error.

All of the above can be executed using the Python package SciPy (Virtanen *et al.* 2020) built-in method $scipy.stats.wilcoxon$ that is called with parameters $method$="approx", "$zero\_method$="pratt", and $correction$=True.

4.B.2. Z-test for proportion (binomial distribution)

We use (Kanji 2006) as a reference for the Z-test for Proportion. It is important to mention that this test is approximate as it assumes that the number of observations justifies a normal approximation for the binomial. (In contradiction to the SciPy package and its built-in method $scipy.stats.binomtest$ containing an exact test).

The proportion test investigates whether the is a significant difference between the assumed proportion of wins p_0 and the observed proportion of wins p. In our analysis, two methods are compared, and the number of wins for each method is counted based on one metric.

The procedure for the proportion test is as follows:

- Collect a set of matched pairs (the results from two different methods on one atomic instance form a pair).
- For each pair, determine which method wins, and count the wins for both methods. Exclude all ties.
- Calculate the ratio of wins for one of the two methods.
- Test where this ratio differs significantly from $p_0 = 0.5$ (equal probability of winning).
  - The test statistic is $Z = \frac{|p-p_0|-\frac{1}{2n}}{\{\frac{p_0(1-p_0)}{n}^{1/2}\}}$
  - The term $\frac{1}{2n}$ in the numerator is a discontinuity correction.
  - For a two-sided test with a significance level $\alpha = 0.05$ the acceptance region for the null hypothesis is $-1.96 < Z < 1.96$.

.

4.B.3. Magnitude test

The magnitude test we use is a t-test for two population means, or the method of paired comparisons such as Test 10 in the book by (Kanji 2006). The test is whether there is a significant difference between the two population means. The procedure for this paired comparison t-test is as follows:

- Collect a set of matched pairs (so the results from two different methods on one atomic instance form a pair).
- Normalise the performances for each pair by computing the mean value of the pair and dividing the two items in the pair by the pairs' mean such that all normalized observations will be between 0 and 2, and 1 indicates a tie.
- Compute the differences d_1 between the two test results for each pair i.
- Compute the variances of the differences with the following formula: $s^2 = \sum_{i=1}^{n} \frac{(d_i-\bar{d})^2}{n-1}$.
- Compute the means of both methods $\bar{x_1}$ and $\bar{x_2}$.
- Compute the t-statistic using the formula: $t = \frac{\bar{x_1}-\bar{x_2}}{s/n^{1/2}}$
- Test significance by checking whether t lies within the acceptance region for which the values are given by the Student's t-distribution (two-sided) with $n-1$ degrees of freedom.

After the normalization step, it is possible to execute the test with the Scipy package and, specifically, its built-in method $scipy.stats.ttest\_rel$.

4.C. Results

Please find the results of the statistical test in this section. In Section 4.C.1, we present the visualizations of the partial orderings. In Section 4.C.2, we present the results of the magnitude tests on the double hits that can give some insight into the magnitude differences of the performance metrics between the different methods.

The test results from the Wilcoxon test and the proportion tests are used to make figures for the partial orderings, and the complete tables with the results can be found in the following tables at the end of this document.

- Table 4.20 for noise level $\epsilon = 1$ and Table 4.21 for noise level $\epsilon = 2$: including the results of the Wilcoxon and proportion tests on solution quality.
- Table 4.22 for noise level $\epsilon = 1$ and Table 4.23 for noise level $\epsilon = 2$: including the results of the Wilcoxon and proportion tests on time offline.
- Table 4.24 for noise level $\epsilon = 1$ and Table 4.25 for noise level $\epsilon = 2$: including the results of the Wilcoxon and proportion tests on online computation time.

4.C.1. Partial orderings

Partial Ordering Solution Quality

Next, we analyze the test results based on the Wilcoxon test, proportion test, and magnitude test for solution quality. We translate the test results into partial orderings that are visualized with solid arrows indicating a significant difference based on the Wilcoxon test and dashed arrows indicating a significant difference based on the proportion test for the pairs where the Wilcoxon test did not find significant results. Furthermore, for all pairs where we find a significant difference according to Wilcoxon, or the proportion test, we analyze whether we can find a significant magnitude difference on the double hits.

- First, we discuss partial orderings for the noise level $\epsilon = 1$, which are visualized in Figure 4.12. For $\epsilon = 1$, the partial ordering shows the same pattern across the different instance sets. However, there are a few exceptions:
  - In some cases we can only show a dashed arrow, indicating a weaker partial ordering, that is a significant proportional difference (proportion of wins) instead of a significant difference obtained by Wilcoxon.
  - Only for ubo50, all arrows are solid, indicating that for all pairs, we found a significant difference according to the Wilcoxon test.
  - For ubo100, there is no significant difference between $\mathsf{proactive}_{\mathsf{SAA}}$ and $\mathsf{proactive}_{0.9}$. Possibly, for the larger instances, the SAA becomes more difficult to solve, for which reason it is not better than the heuristic γ-quantile procedure anymore.
- Then, we analyze the partial orderings for the noise level $\epsilon = 2$, in Figure 4.13. Here, there are a few things to observe:
  - We observe a shift in the partial orderings for $\epsilon = 2$.
  - For j10-j20, the partial ordering is similar to $\epsilon = 1$, although for $\epsilon = 2$, all arrows are solid (indicating a relation obtained by the Wilcoxon test).

- For $j30$, there is no significant difference between $\text{proactive}_{0.9}$ and proactive_{SAA}, neither with Wilcoxon and neither proportionally.
- The partial orderings for $j30$ and $ubo50$ are similar.
- For ubo100, we observe the stnu is significantly better than the three other methods, although no significant difference can be found between the three other methods.

- The main conclusion is that, in all situations, the stnu method outperforms the others in terms of solution quality. In general, the reactive shows outperforming the proactivemethods. Furthermore, proactive_{SAA} outperforms in many cases $\text{proactive}_{0.9}$, although for larger instances and a higher noise level, this difference is not present anymore.

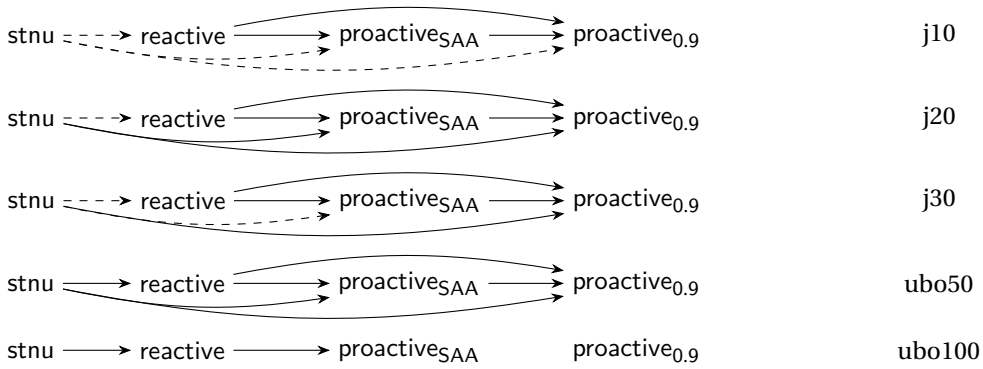


Figure 4.12 *Partial ordering on solution quality for $\epsilon = 1$. The solid arrow indicates a significant difference obtained by the Wilcoxon test, and the dashed arrow shows only a proportional difference.*

Partial ordering time offline

- We expected the $\text{proactive}_{0.9}$ and reactive to be the fastest offline, followed by the proactive_{SAA} and stnu, where the partial ordering between the latter two depends on the problem size. When two methods have (almost) only ties (such as $\text{proactive}_{0.9}$ and reactive), not all tests can be executed, and a *nan* will appear.
- Importantly, the tests are designed in such a way that infeasibilities are also weighted in significance testing. When a method results in an infeasible solution, infinite offline time is assigned to this experiment. For that reason, it can still occur that we find a difference between reactive and $\text{proactive}_{0.9}$: although they have the same offline procedure, one of the two can still fail, which results in infinite offline time.
- In general, we give preference to the test results of the Wilcoxon test because this test shows a stronger relation than the proportional test, but to fill in the

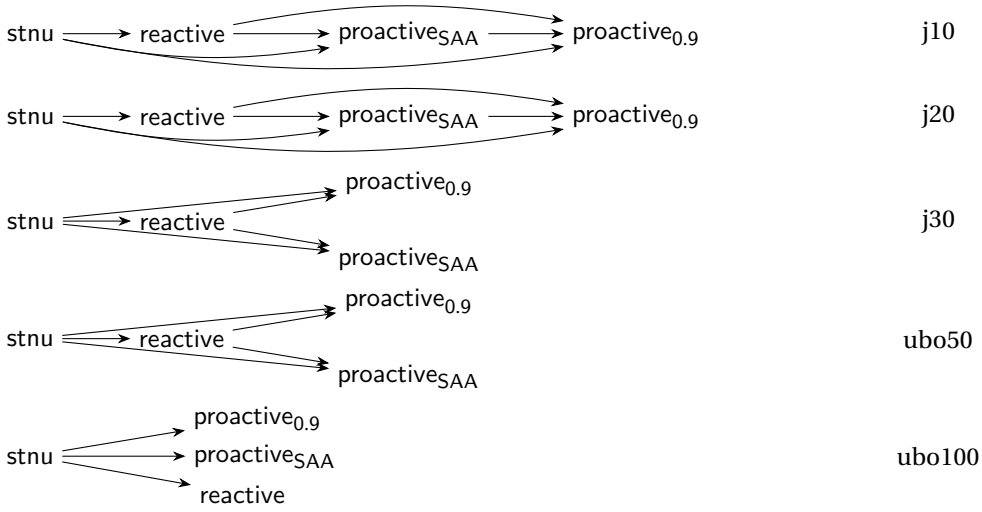


Figure 4.13 *Partial ordering on solution quality for $\epsilon = 2$. Solid arrow indicates a significant difference obtained by the Wilcoxon test, and dashed arrow indicates only a proportional difference.*

missing information, we use the proportion test to test if there is a significant difference in the proportion of wins. We noted, however, that the test outcomes of the two tests can sometimes be contradictory. The Wilcoxon test penalizes infeasibilities more severely, while the proportion test only considers the number of wins. Consequently, it is possible that method 1 produces more infeasible solutions but achieves better metrics, whereas method 2 generates fewer infeasible solutions but worse metrics, resulting in the Wilcoxon test on instances with at least one feasible solution indicating that method 2 performs better (infeasible solutions are penalized more heavily). The proportion test, applied to instances with at least one feasible solution, shows that method 1 performs better. The magnitude test on double hits shows that Method 1 performs better, as it has higher metric values.

- Now, we observe the different partial orderings obtained for $\epsilon = 1$:
  - In general, the time spent offline for the proactive$_{0.9}$ and the reactive is exactly the same, therefore, in most cases, we also did not find a significant difference between the two. However, especially for the ubo100 instances, we found that proactive$_{0.9}$ sometimes had more feasible solutions, leading to a partial ordering proactive$_{0.9}$ → reactive.
  - We observe that for j10 and j20 proactive$_{SAA}$ is consistently faster offline than stnu, but this flips for j30 - ubo100, where the stnu becomes faster. This can be explained by the fact that the solve time of the SAA can grow exponentially, whereas the DC-checking algorithm is polynomial-time.
- For noise level $\epsilon = 2$, we observe in Figure 4.15:

- For j10 and j20, the partial ordering is almost the same as for the $\epsilon = 1$ level, although the flipping already occurs at j20, where the stnu becomes faster than the $\text{proactive}_{\text{SAA}}$.
- Surprisingly, the pattern changes for the ubo50 and ubo100 instances. We observe that the best performance on time offline is obtained by the stnu according to the Wilcoxon. We found that this is mainly caused by the much higher feasibility ratio of the stnu method.
- Furthermore, we find that the $\text{proactive}_{0.9}$ becomes better than reactive because of the higher feasibility ratio again (as the two methods have the same offline procedure).
- Again, for the larger instances sets (j20-ubo100), the $\text{proactive}_{\text{SAA}}$ has the worst relative computation time offline.

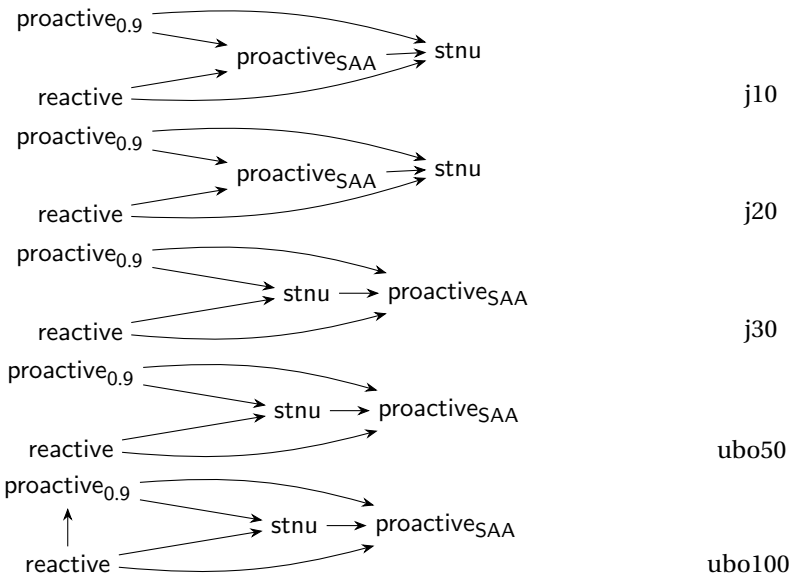


Figure 4.14 *Partial ordering time offline $\epsilon = 1$. Solid arrow indicating a significant difference obtained by the Wilcoxon test, and dashed arrow only a proportional difference.*

Partial ordering time online

We observe the results from the tests on online computation time:

- For $\epsilon = 1$ (Figure 4.17), we find the same partial ordering for each instance set, which is also in line with our expectations:
  - There is no difference between $\text{proactive}_{\text{SAA}}$ and $\text{proactive}_{0.9}$ as both only require a feasibility check only.

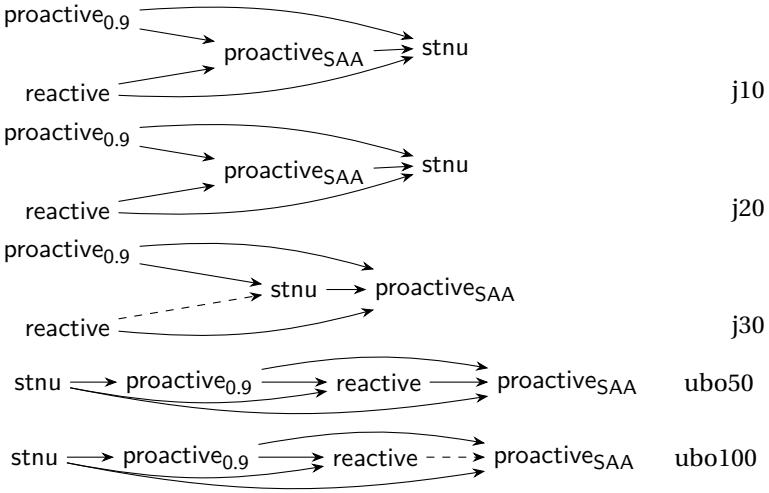


Figure 4.15 *Partial ordering time offline $\epsilon = 2$. The solid arrow indicates a significant difference obtained by the Wilcoxon test, and the dashed arrow shows only a proportional difference.*

  - The stnu real-time execution algorithm turns out to be more efficient online than the reactive method, which comprises multiple solver calls; this is also expected.

- For $\epsilon = 2$ (Figure 4.17), we find the same pattern as for $\epsilon = 1$; for problem sets j10-j30. However, we find that for ubo50 and ubo100, the stnu starts to outperform other methods, explained by the higher feasibility ratio of the stnu.

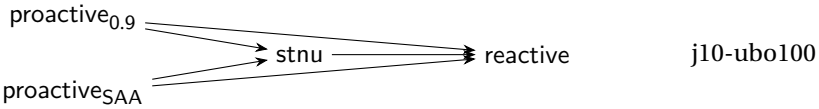


Figure 4.16 *Partial ordering time online $\epsilon = 1$, j10, j20, j30, ubo50, and ubo100. Solid arrow indicates a significant difference obtained by the Wilcoxon test, and dashed arrow indicates only a proportional difference.*

4.C.2. Magnitude tests

In general, we observed that for pairs of methods for which we found a significant partial ordering, we also found a significant difference in the magnitudes of the metrics. The test results for solution quality are presented in Chapter 4. Here, we present the results of the maginuted tests on time offline and time online.

The remarkable findings from the magnitude tests are highlighted for each performance metric in the following subsections.

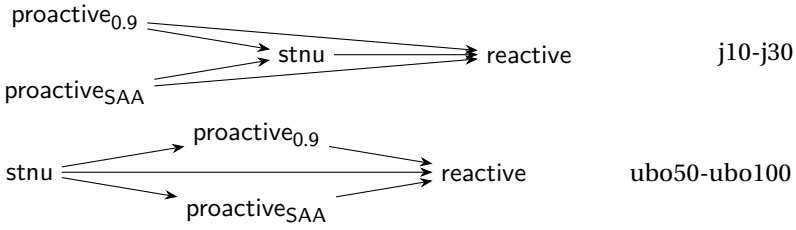


Figure 4.17 *Partial ordering time online $\epsilon = 2$, ubo50, and ubo100. Solid arrow indicating a significant difference obtained by the Wilcoxon test, and dashed arrow only a proportional difference.*

Magnitude Tests Time Offline

For time offline, $\epsilon = 2$, ubo50 and ubo100, the Wilcoxon test found that the stnu is better than e.g. proactive$_{0.9}$ and reactive while looking at the magnitude results on double hits (this is an important difference in test procedure!), we find that the proactive$_{0.9}$ and reactive are much faster offline than stnu, which is also expected. Then, we observe no difference between proactive$_{0.9}$ and reactive, which can be explained by the fact that both have the same offline procedure, and we select only double hits for the test.

Table 4.16 *Magnitude test on time offline for noise factor $\epsilon = 1$. Using a pairwise t-test, including all instances for which both methods found a feasible solution. Each cell shows on the first row [nr pairs] t-stat (p-value) with (*) for $p < 0.05$ and on the second row the normalized average of method 1 and on the third row the normalized average of method 2.*

j10	pro$_{0.9}$-react	react-stnu	react-pro$_{SAA}$	pro$_{0.9}$-stnu	pro$_{0.9}$-pro$_{SAA}$	pro$_{SAA}$-stnu
[n] t (p)	[340] nan (nan)	[260] -365.854 (*)	[340] -77.22 (*)	[260] -365.854 (*)	[340] -77.22 (*)	[260] -19.399 (*)
norm. avg.	pro$_{0.9}$: 1.0	react: 0.14	react: 0.32	pro$_{0.9}$: 0.14	pro$_{0.9}$: 0.32	pro$_{SAA}$: 0.65
norm. avg.	react: 1.0	stnu: 1.86	pro$_{SAA}$: 1.68	stnu: 1.86	pro$_{SAA}$: 1.68	stnu: 1.35
j20	pro$_{0.9}$-react	react-stnu	react-pro$_{SAA}$	pro$_{0.9}$-stnu	pro$_{0.9}$-pro$_{SAA}$	pro$_{SAA}$-stnu
[n] t (p)	[270] nan (nan)	[230] -45.769 (*)	[270] -52.707 (*)	[230] -45.769 (*)	[270] -52.707 (*)	[230] -3.062 (*)
norm. avg.	pro$_{0.9}$: 1.0	react: 0.22	react: 0.28	pro$_{0.9}$: 0.22	pro$_{0.9}$: 0.28	pro$_{SAA}$: 0.91
norm. avg.	react: 1.0	stnu: 1.78	pro$_{SAA}$: 1.72	stnu: 1.78	pro$_{SAA}$: 1.72	stnu: 1.09
j30	pro$_{0.9}$-react	react-stnu	react-pro$_{SAA}$	pro$_{0.9}$-stnu	pro$_{0.9}$-pro$_{SAA}$	stnu-pro$_{SAA}$
[n] t (p)	[320] nan (nan)	[280] -38.703 (*)	[320] -53.836 (*)	[280] -38.703 (*)	[320] -53.836 (*)	[280] 0.5 (0.618)
norm. avg.	pro$_{0.9}$: 1.0	react: 0.3	react: 0.32	pro$_{0.9}$: 0.3	pro$_{0.9}$: 0.32	stnu: 1.02
norm. avg.	react: 1.0	stnu: 1.7	pro$_{SAA}$: 1.68	stnu: 1.7	pro$_{SAA}$: 1.68	pro$_{SAA}$: 0.98
ubo50	pro$_{0.9}$-react	react-stnu	react-pro$_{SAA}$	pro$_{0.9}$-stnu	pro$_{0.9}$-pro$_{SAA}$	stnu-pro$_{SAA}$
[n] t (p)	[370] nan (nan)	[330] -36.15 (*)	[370] -72.563 (*)	[330] -36.15 (*)	[370] -72.563 (*)	[330] -5.636 (*)
norm. avg.	pro$_{0.9}$: 1.0	react: 0.36	react: 0.24	pro$_{0.9}$: 0.36	pro$_{0.9}$: 0.24	stnu: 0.82
norm. avg.	react: 1.0	stnu: 1.64	pro$_{SAA}$: 1.76	stnu: 1.64	pro$_{SAA}$: 1.76	pro$_{SAA}$: 1.18
ubo100	pro$_{0.9}$-react	react-stnu	react-pro$_{SAA}$	pro$_{0.9}$-stnu	pro$_{0.9}$-pro$_{SAA}$	stnu-pro$_{SAA}$
[n] t (p)	[388] nan (nan)	[358] -27.209 (*)	[388] -70.349 (*)	[370] -25.293 (*)	[400] -70.926 (*)	[370] -6.391 (*)
norm. avg.	pro$_{0.9}$: 1.0	react: 0.43	react: 0.27	pro$_{0.9}$: 0.45	pro$_{0.9}$: 0.27	stnu: 0.79
norm. avg.	react: 1.0	stnu: 1.57	pro$_{SAA}$: 1.73	stnu: 1.55	pro$_{SAA}$: 1.73	pro$_{SAA}$: 1.21

Magnitude Tests Time Online

Looking at the magnitude test of time online with $\epsilon = 2$, according to Wilcoxon, stnu is outperforming proactive$_{SAA}$ and proactive$_{0.9}$ for ubo50 and ubo100. However, this is

Table 4.17 *Magnitude test on time offline for noise factor $\epsilon = 2$. Using a pairwise t-test, including all instances for which both methods found a feasible solution. Each cell shows on the first row [nr pairs] t-stat (p-value) with (*) for $p < 0.05$ and on the second row the normalized average of method 1 and on the third row the normalized average of method 2.*

j10	$pro_{0.9}$-react	react-stnu	react-pro_{SAA}	$pro_{0.9}$-stnu	$pro_{0.9}$-pro_{SAA}	pro_{SAA}-stnu
[n] t (p)	[233] nan (nan)	[205] -229.689 (*)	[230] -59.39 (*)	[206] -230.812 (*)	[231] -59.404 (*)	[204] -17.066 (*)
norm. avg.	$pro_{0.9}$: 1.0	react: 0.15	react: 0.35	$pro_{0.9}$: 0.15	$pro_{0.9}$: 0.35	pro_{SAA}: 0.61
norm. avg.	react: 1.0	stnu: 1.85	pro_{SAA}: 1.65	stnu: 1.85	pro_{SAA}: 1.65	stnu: 1.39
j20	$pro_{0.9}$-react	react-stnu	react-pro_{SAA}	$pro_{0.9}$-stnu	$pro_{0.9}$-pro_{SAA}	stnu-pro_{SAA}
[n] t (p)	[176] nan (nan)	[170] -82.607 (*)	[174] -63.732 (*)	[170] -82.607 (*)	[173] -63.382 (*)	[172] 0.379 (0.705)
norm. avg.	$pro_{0.9}$: 1.0	react: 0.21	react: 0.23	$pro_{0.9}$: 0.21	$pro_{0.9}$: 0.23	stnu: 1.01
norm. avg.	react: 1.0	stnu: 1.79	pro_{SAA}: 1.77	stnu: 1.79	pro_{SAA}: 1.77	pro_{SAA}: 0.99
j30	$pro_{0.9}$-react	react-stnu	react-pro_{SAA}	$pro_{0.9}$-stnu	$pro_{0.9}$-pro_{SAA}	stnu-pro_{SAA}
[n] t (p)	[160] nan (nan)	[140] -33.765 (*)	[152] -40.336 (*)	[138] -33.253 (*)	[150] -39.904 (*)	[148] -2.168 (*)
norm. avg.	$pro_{0.9}$: 1.0	react: 0.34	react: 0.33	$pro_{0.9}$: 0.34	$pro_{0.9}$: 0.33	stnu: 0.9
norm. avg.	react: 1.0	stnu: 1.66	pro_{SAA}: 1.67	stnu: 1.66	pro_{SAA}: 1.67	pro_{SAA}: 1.1
ubo50	stnu-$pro_{0.9}$	stnu-react	stnu-pro_{SAA}	$pro_{0.9}$-react	$pro_{0.9}$-pro_{SAA}	react-pro_{SAA}
[n] t (p)	[148] 26.653 (*)	[141] 27.168 (*)	[146] -0.952 (0.343)	[155] nan (nan)	[150] -45.457 (*)	[145] -44.29 (*)
norm. avg.	stnu: 1.7	stnu: 1.72	stnu: 0.95	$pro_{0.9}$: 1.0	$pro_{0.9}$: 0.24	react: 0.24
norm. avg.	$pro_{0.9}$: 0.3	react: 0.28	pro_{SAA}: 1.05	react: 1.0	pro_{SAA}: 1.76	pro_{SAA}: 1.76
ubo100	stnu-$pro_{0.9}$	stnu-react	stnu-pro_{SAA}	$pro_{0.9}$-react	$pro_{0.9}$-pro_{SAA}	react-pro_{SAA}
[n] t (p)	[114] 14.765 (*)	[94] 15.183 (*)	[114] -3.125 (*)	[99] nan (nan)	[94] -42.151 (*)	[76] -42.944 (*)
norm. avg.	stnu: 1.55	stnu: 1.61	stnu: 0.83	$pro_{0.9}$: 1.0	$pro_{0.9}$: 0.21	react: 0.18
norm. avg.	$pro_{0.9}$: 0.45	react: 0.39	pro_{SAA}: 1.17	react: 1.0	pro_{SAA}: 1.79	pro_{SAA}: 1.82

again mainly caused by the higher ratio of feasible solutions, as when we analyze the magnitude difference the proactive methods are much faster online looking at the double hits (which is logical as this comprises only of feasibility checking).

Table 4.18 *Magnitude t-test on time online for noise factor $\epsilon = 1$ (double hits). Each cell shows on the first row [nr pairs] t-stat (p-value) with (*) for $p < 0.05$ and on the second row the normalized average of method 1 and on the third row the normalized average of method 2.*

j10	$pro_{0.9}$-pro_{SAA}	$pro_{0.9}$-stnu	pro_{SAA}-stnu	$pro_{0.9}$-react	pro_{SAA}-react	stnu-react
[n] t (p)	[340] 0.679 (0.498)	[260] -109.014 (*)	[260] -106.917 (*)	[340] -4393.475 (*)	[340] -4277.995 (*)	[260] -1151.565 (*)
norm. avg.	$pro_{0.9}$: 1.02	$pro_{0.9}$: 0.05	pro_{SAA}: 0.05	$pro_{0.9}$: 0.0	pro_{SAA}: 0.0	stnu: 0.04
norm. avg.	pro_{SAA}: 0.98	stnu: 1.95	stnu: 1.95	react: 2.0	react: 2.0	react: 1.96
j20	$pro_{0.9}$-pro_{SAA}	$pro_{0.9}$-stnu	pro_{SAA}-stnu	$pro_{0.9}$-react	pro_{SAA}-react	stnu-react
[n] t (p)	[270] -0.663 (0.508)	[230] -285.563 (*)	[230] -282.464 (*)	[270] -7311.952 (*)	[270] -7470.984 (*)	[230] -605.38 (*)
norm. avg.	$pro_{0.9}$: 0.97	$pro_{0.9}$: 0.03	pro_{SAA}: 0.03	$pro_{0.9}$: 0.0	pro_{SAA}: 0.0	stnu: 0.07
norm. avg.	pro_{SAA}: 1.03	stnu: 1.97	stnu: 1.97	react: 2.0	react: 2.0	react: 1.93
j30	$pro_{0.9}$-pro_{SAA}	$pro_{0.9}$-stnu	pro_{SAA}-stnu	$pro_{0.9}$-react	pro_{SAA}-react	stnu-react
[n] t (p)	[320] -0.156 (0.876)	[280] -789.246 (*)	[280] -786.726 (*)	[320] -14638.229 (*)	[320] -14425.009 (*)	[280] -338.664 (*)
norm. avg.	$pro_{0.9}$: 0.99	$pro_{0.9}$: 0.02	pro_{SAA}: 0.02	$pro_{0.9}$: 0.0	pro_{SAA}: 0.0	stnu: 0.09
norm. avg.	pro_{SAA}: 1.01	stnu: 1.98	stnu: 1.98	react: 2.0	react: 2.0	react: 1.91
ubo50	pro_{SAA}-$pro_{0.9}$	$pro_{0.9}$-stnu	pro_{SAA}-stnu	$pro_{0.9}$-react	pro_{SAA}-react	stnu-react
[n] t (p)	[370] -0.795 (0.427)	[330] -3951.584 (*)	[330] -3976.441 (*)	[370] -30723.506 (*)	[370] -31447.766 (*)	[330] -142.758 (*)
norm. avg.	pro_{SAA}: 0.97	$pro_{0.9}$: 0.01	pro_{SAA}: 0.01	$pro_{0.9}$: 0.0	pro_{SAA}: 0.0	stnu: 0.15
norm. avg.	$pro_{0.9}$: 1.03	stnu: 1.99	stnu: 1.99	react: 2.0	react: 2.0	react: 1.85
ubo100	pro_{SAA}-$pro_{0.9}$	$pro_{0.9}$-stnu	pro_{SAA}-stnu	$pro_{0.9}$-react	pro_{SAA}-react	stnu-react
[n] t (p)	[400] -0.065 (0.948)	[370] -56517.571 (*)	[370] -48810.689 (*)	[388] -83435.86 (*)	[388] -78013.468 (*)	[358] -60.202 (*)
norm. avg.	pro_{SAA}: 1.0	$pro_{0.9}$: 0.0	pro_{SAA}: 0.0	$pro_{0.9}$: 0.0	pro_{SAA}: 0.0	stnu: 0.23
norm. avg.	$pro_{0.9}$: 1.0	stnu: 2.0	stnu: 2.0	react: 2.0	react: 2.0	react: 1.77

Table 4.19 *Magnitude t-test on time online for noise factor $\epsilon = 2$ (double hits). Each cell shows on the first row [nr pairs] t-stat (p-value) with (*) for $p < 0.05$ and on the second row the normalized average of method 1 and on the third row the normalized average of method 2.*

j10	$pro_{0.9}$-pro_{SAA}	$pro_{0.9}$-stnu	pro_{SAA}-stnu	$pro_{0.9}$-react	pro_{SAA}-react	stnu-react
[n] t (p)	[231] 0.391 (0.697)	[206] -92.546 (*)	[204] -91.532 (*)	[233] -4426.055 (*)	[230] -4653.225 (*)	[205] -1399.699 (*)
norm. avg.	$pro_{0.9}$: 1.01	$pro_{0.9}$: 0.06	pro_{SAA}: 0.06	$pro_{0.9}$: 0.0	pro_{SAA}: 0.0	stnu: 0.03
norm. avg.	pro_{SAA}: 0.99	stnu: 1.94	stnu: 1.94	react: 2.0	react: 2.0	react: 1.97
j20	$pro_{0.9}$-pro_{SAA}	$pro_{0.9}$-stnu	pro_{SAA}-stnu	$pro_{0.9}$-react	pro_{SAA}-react	stnu-react
[n] t (p)	[173] 0.77 (0.443)	[170] -241.897 (*)	[172] -262.612 (*)	[176] -7787.819 (*)	[174] -8817.279 (*)	[170] -935.848 (*)
norm. avg.	$pro_{0.9}$: 1.03	$pro_{0.9}$: 0.03	pro_{SAA}: 0.03	$pro_{0.9}$: 0.0	pro_{SAA}: 0.0	stnu: 0.06
norm. avg.	pro_{SAA}: 0.97	stnu: 1.97	stnu: 1.97	react: 2.0	react: 2.0	react: 1.94
j30	pro_{SAA}-$pro_{0.9}$	$pro_{0.9}$-stnu	pro_{SAA}-stnu	$pro_{0.9}$-react	pro_{SAA}-react	stnu-react
[n] t (p)	[150] -0.264 (0.792)	[138] -514.302 (*)	[148] -547.413 (*)	[160] -14437.976 (*)	[152] -14871.238 (*)	[140] -397.488 (*)
norm. avg.	pro_{SAA}: 0.99	$pro_{0.9}$: 0.02	pro_{SAA}: 0.02	$pro_{0.9}$: 0.0	pro_{SAA}: 0.0	stnu: 0.07
norm. avg.	$pro_{0.9}$: 1.01	stnu: 1.98	stnu: 1.98	react: 2.0	react: 2.0	react: 1.93
ubo50	$pro_{0.9}$-pro_{SAA}	stnu-$pro_{0.9}$	stnu-pro_{SAA}	$pro_{0.9}$-react	pro_{SAA}-react	stnu-react
[n] t (p)	[150] -0.707 (0.481)	[148] 2600.902 (*)	[146] 2585.661 (*)	[155] -25795.773 (*)	[145] -23210.69 (*)	[141] -121.392 (*)
norm. avg.	$pro_{0.9}$: 0.96	stnu: 1.99	stnu: 1.99	$pro_{0.9}$: 0.0	pro_{SAA}: 0.0	stnu: 0.14
norm. avg.	pro_{SAA}: 1.04	$pro_{0.9}$: 0.01	pro_{SAA}: 0.01	react: 2.0	react: 2.0	react: 1.86
ubo100	pro_{SAA}-$pro_{0.9}$	stnu-$pro_{0.9}$	stnu-pro_{SAA}	$pro_{0.9}$-react	pro_{SAA}-react	stnu-react
[n] t (p)	[94] 0.141 (0.888)	[114] 21743.706 (*)	[114] 21911.639 (*)	[99] -46429.42 (*)	[76] -40238.643 (*)	[94] -41.405 (*)
norm. avg.	pro_{SAA}: 1.0	stnu: 2.0	stnu: 2.0	$pro_{0.9}$: 0.0	pro_{SAA}: 0.0	stnu: 0.21
norm. avg.	$pro_{0.9}$: 1.0	$pro_{0.9}$: 0.0	pro_{SAA}: 0.0	react: 2.0	react: 2.0	react: 1.79

4.C.3. Summary

In the main chapter, we employed the statistical analysis from this document to provide a summarizing partial ordering per metric, including a brief summary of the main findings. We decide to present the most occurring patterns and in the text, highlight any important exceptions.

4.D. Results tables partial orderings

Table 4.20 *Pairwise comparison on schedule quality (makespan) for noise factor $\epsilon = 1$. Using a Wilcoxon test and a proportion test. Including all instances for which at least one of the two methods found a feasible solution. Note that the ordering matters: the first method shown is the better of the two in each pair, method 1 - method 2. Each cell shows on the first row [nr pairs] the z-value (p-value) of the Wilcoxon test with (*) for $p < 0.05$. Each cell shows on the second row [nr pairs] z-value (p-value) with (*) for $p < 0.05$ for the proportion test.*

j10	stnu-react	stnu-pro_{SAA}	stnu-$pro_{0.9}$	react-pro_{SAA}	react-$pro_{0.9}$	pro_{SAA}-$pro_{0.9}$
Wilcoxon: [n] z (p)	[340] -1.659 (0.097)	[340] -0.067 (0.947)	[340] -0.082 (0.935)	[340] -10.571 (*)	[340] -10.918 (*)	[340] -5.756 (*)
proportion: [n] z (p)	[308] 0.591 (*)	[310] 0.632 (*)	[310] 0.632 (*)	[117] 0.991 (*)	[121] 1.0 (*)	[37] 0.973 (*)
j20	stnu-react	stnu-pro_{SAA}	stnu-$pro_{0.9}$	react-pro_{SAA}	react-$pro_{0.9}$	pro_{SAA}-$pro_{0.9}$
Wilcoxon: [n] z (p)	[270] -1.523 (0.128)	[270] -2.401 (*)	[270] -2.679 (*)	[270] -5.936 (*)	[270] -7.852 (*)	[270] -6.144 (*)
proportion: [n] z (p)	[213] 0.596 (*)	[215] 0.633 (*)	[213] 0.643 (*)	[75] 0.827 (*)	[62] 1.0 (*)	[59] 0.898 (*)
j30	stnu-react	stnu-pro_{SAA}	stnu-$pro_{0.9}$	react-pro_{SAA}	react-$pro_{0.9}$	pro_{SAA}-$pro_{0.9}$
Wilcoxon: [n] z (p)	[320] -0.892 (0.372)	[320] -1.218 (0.223)	[320] -2.157 (*)	[320] -5.54 (*)	[320] -9.395 (*)	[320] -5.299 (*)
proportion: [n] z (p)	[214] 0.579 (*)	[214] 0.575 (*)	[217] 0.618 (*)	[103] 0.757 (*)	[89] 1.0 (*)	[79] 0.797 (*)
ubo50	stnu-react	stnu-pro_{SAA}	stnu-$pro_{0.9}$	react-pro_{SAA}	react-$pro_{0.9}$	pro_{SAA}-$pro_{0.9}$
Wilcoxon: [n] z (p)	[370] -6.748 (*)	[370] -6.8 (*)	[370] -6.859 (*)	[370] -2.76 (*)	[370] -8.531 (*)	[370] -7.085 (*)
proportion: [n] z (p)	[277] 0.776 (*)	[276] 0.779 (*)	[278] 0.781 (*)	[61] 0.672 (*)	[73] 1.0 (*)	[65] 0.938 (*)
ubo100	stnu-react	stnu-pro_{SAA}	stnu-$pro_{0.9}$	react-pro_{SAA}	react-$pro_{0.9}$	pro_{SAA}-$pro_{0.9}$
Wilcoxon: [n] z (p)	[400] -2.12 (*)	[400] -1.76 (0.078)	[400] -1.842 (0.065)	[400] -0.442 (0.659)	[400] -6.534 (*)	[400] -0.2 (0.842)
proportion: [n] z (p)	[312] 0.519 (0.533)	[314] 0.51 (0.778)	[311] 0.514 (0.65)	[147] 0.51 (0.869)	[88] 0.864 (*)	[151] 0.503 (1.0)

Table 4.21 *Pairwise comparison on schedule quality (makespan) for noise factor $\epsilon = 2$. Using a Wilcoxon test and a proportion test. Including all instances for which at least one of the two methods found a feasible solution. Note that the ordering matters: the first method shown is the better of the two in each pair, method 1 - method 2. Each cell shows on the first row [nr pairs] the z-value (p-value) of the Wilcoxon test with (*) for $p < 0.05$. Each cell shows on the second row [nr pairs] z-value (p-value) with (*) for $p < 0.05$ for the proportion test.*

j10	stnu-react	stnu-pro$_{SAA}$	stnu-pro$_{0.9}$	react-pro$_{SAA}$	react-pro$_{0.9}$	pro$_{SAA}$-pro$_{0.9}$
Wilcoxon: [n] z (p)	[258] -5.194 (*)	[258] -7.825 (*)	[258] -7.822 (*)	[235] -10.082 (*)	[234] -11.046 (*)	[235] -4.058 (*)
proportion: [n] z (p)	[237] 0.696 (*)	[247] 0.802 (*)	[248] 0.802 (*)	[146] 0.911 (*)	[134] 0.993 (*)	[37] 0.838 (*)
j20	stnu-react	stnu-pro$_{SAA}$	stnu-pro$_{0.9}$	react-pro$_{SAA}$	react-pro$_{0.9}$	pro$_{SAA}$-pro$_{0.9}$
Wilcoxon: [n] z (p)	[217] -9.825 (*)	[217] -10.194 (*)	[216] -10.57 (*)	[182] -4.58 (*)	[177] -7.243 (*)	[182] -3.241 (*)
proportion: [n] z (p)	[185] 0.854 (*)	[188] 0.872 (*)	[188] 0.888 (*)	[79] 0.759 (*)	[53] 1.0 (*)	[60] 0.7 (*)
j30	stnu-react	stnu-pro$_{SAA}$	stnu-pro$_{0.9}$	react-pro$_{SAA}$	react-pro$_{0.9}$	pro$_{SAA}$-pro$_{0.9}$
Wilcoxon: [n] z (p)	[233] -6.986 (*)	[234] -7.175 (*)	[232] -7.993 (*)	[183] -1.355 (0.175)	[163] -7.557 (*)	[182] -5.689 (*)
proportion: [n] z (p)	[191] 0.749 (*)	[192] 0.776 (*)	[196] 0.796 (*)	[94] 0.574 (0.18)	[58] 1.0 (*)	[96] 0.802 (*)
ubo50	stnu-react	stnu-pro$_{SAA}$	stnu-pro$_{0.9}$	react-pro$_{SAA}$	react-pro$_{0.9}$	pro$_{SAA}$-pro$_{0.9}$
Wilcoxon: [n] z (p)	[284] -11.76 (*)	[285] -11.73 (*)	[284] -11.799 (*)	[171] -1.973 (*)	[162] -3.74 (*)	[173] -0.938 (0.348)
proportion: [n] z (p)	[236] 0.907 (*)	[238] 0.908 (*)	[236] 0.911 (*)	[50] 0.66 (*)	[39] 0.821 (*)	[36] 0.583 (0.405)
ubo100	stnu-react	stnu-pro$_{SAA}$	stnu-pro$_{0.9}$	react-pro$_{SAA}$	react-pro$_{0.9}$	pro$_{SAA}$-pro$_{0.9}$
Wilcoxon: [n] z (p)	[285] -12.139 (*)	[284] -11.55 (*)	[288] -11.26 (*)	[141] -1.83 (0.067)	[122] -1.017 (0.309)	[146] -0.083 (0.934)
proportion: [n] z (p)	[261] 0.851 (*)	[254] 0.799 (*)	[265] 0.796 (*)	[82] 0.415 (0.151)	[44] 0.477 (0.88)	[82] 0.5 (0.912)

Table 4.22 *Pairwise comparison on time offline for noise factor $\epsilon = 1$. Using a Wilcoxon test and a proportion test. Including all instances for which at least one of the two methods found a feasible solution. Note that the ordering matters: the first method shown is the better of the two in each pair, method 1 - method 2, according to Wilcoxon. Each cell shows on the first row [nr pairs] the z-value (p-value) of the Wilcoxon test with (*) for $p < 0.05$. Each cell shows on the second row [nr pairs] the ratio of wins (p-value) with (*) for $p < 0.05$.*

j10	pro$_{0.9}$-react	react-stnu	react-pro$_{SAA}$	pro$_{0.9}$-stnu	pro$_{0.9}$-pro$_{SAA}$	pro$_{SAA}$-stnu
Wilcoxon: [n] z (p)	[340] nan (nan)	[340] -15.982 (*)	[340] -15.982 (*)	[340] -15.982 (*)	[340] -15.982 (*)	[340] -13.986 (*)
proportion: [n] z (p)	[nan] nan (nan)	[340] 1.0 (*)	[340] 1.0 (*)	[340] 1.0 (*)	[340] 1.0 (*)	[340] 0.882 (*)
j20	pro$_{0.9}$-react	react-stnu	react-pro$_{SAA}$	pro$_{0.9}$-stnu	pro$_{0.9}$-pro$_{SAA}$	pro$_{SAA}$-stnu
Wilcoxon: [n] z (p)	[270] nan (nan)	[270] -14.125 (*)	[270] -14.245 (*)	[270] -14.125 (*)	[270] -14.245 (*)	[270] -3.927 (*)
proportion: [n] z (p)	[nan] nan (nan)	[270] 0.963 (*)	[270] 1.0 (*)	[270] 0.963 (*)	[270] 1.0 (*)	[270] 0.63 (*)
j30	pro$_{0.9}$-react	react-stnu	react-pro$_{SAA}$	pro$_{0.9}$-stnu	pro$_{0.9}$-pro$_{SAA}$	stnu-pro$_{SAA}$
Wilcoxon: [n] z (p)	[320] nan (nan)	[320] -13.963 (*)	[320] -15.506 (*)	[320] -13.963 (*)	[320] -15.506 (*)	[320] -1.63 (0.103)
proportion: [n] z (p)	[nan] nan (nan)	[320] 0.969 (*)	[320] 1.0 (*)	[320] 0.969 (*)	[320] 1.0 (*)	[320] 0.5 (0.955)
ubo50	pro$_{0.9}$-react	react-stnu	react-pro$_{SAA}$	pro$_{0.9}$-stnu	pro$_{0.9}$-pro$_{SAA}$	stnu-pro$_{SAA}$
Wilcoxon: [n] z (p)	[370] nan (nan)	[370] -16.671 (*)	[370] -16.671 (*)	[370] -16.671 (*)	[370] -16.671 (*)	[370] -7.261 (*)
proportion: [n] z (p)	[nan] nan (nan)	[370] 1.0 (*)	[370] 1.0 (*)	[370] 1.0 (*)	[370] 1.0 (*)	[370] 0.622 (*)
ubo100	pro$_{0.9}$-react	react-stnu	react-pro$_{SAA}$	pro$_{0.9}$-stnu	pro$_{0.9}$-pro$_{SAA}$	stnu-pro$_{SAA}$
Wilcoxon: [n] z (p)	[400] -3.463 (*)	[400] -14.988 (*)	[400] -15.286 (*)	[400] -15.969 (*)	[400] -17.332 (*)	[400] -6.406 (*)
proportion: [n] z (p)	[12] 1.0 (*)	[400] 0.962 (*)	[400] 0.97 (*)	[400] 0.975 (*)	[400] 1.0 (*)	[400] 0.6 (*)

Table 4.23 *Pairwise comparison on time offline for noise factor $\epsilon = 2$. Using a Wilcoxon test and a proportion test. Including all instances for which at least one of the two methods found a feasible solution. Note that the ordering matters: the first method showed is the better of the two in each pair method 1 - method 2 according to Wilcoxon. Each cell shows on the first row [nr pairs] the z-value (p-value) of the Wilcoxon test with (*) for $p < 0.05$. Each cell shows on the second row [nr pairs] the ratio of wins (p-value) with (*) for $p < 0.05$.*

j10	$pro_{0.9}$-react	react-stnu	react-pro_{SAA}	$pro_{0.9}$-stnu	$pro_{0.9}$-pro_{SAA}	pro_{SAA}-stnu
Wilcoxon: [n] z (p)	[234] -0.996 (0.319)	[258] -9.382 (*)	[235] -12.848 (*)	[258] -9.554 (*)	[235] -13.069 (*)	[258] -6.664 (*)
proportion: [n] z (p)	[nan] nan (nan)	[258] 0.903 (*)	[235] 0.991 (*)	[258] 0.907 (*)	[235] 0.996 (*)	[258] 0.764 (*)
j20	$pro_{0.9}$-react	react-stnu	react-pro_{SAA}	$pro_{0.9}$-stnu	$pro_{0.9}$-pro_{SAA}	stnu-pro_{SAA}
Wilcoxon: [n] z (p)	[177] -0.994 (0.32)	[217] -4.543 (*)	[182] -10.458 (*)	[216] -4.456 (*)	[182] -10.213 (*)	[217] -5.281 (*)
proportion: [n] z (p)	[nan] nan (nan)	[217] 0.816 (*)	[182] 0.973 (*)	[216] 0.815 (*)	[182] 0.967 (*)	[217] 0.604 (*)
j30	$pro_{0.9}$-react	react-stnu	react-pro_{SAA}	$pro_{0.9}$-stnu	$pro_{0.9}$-pro_{SAA}	stnu-pro_{SAA}
Wilcoxon: [n] z (p)	[163] -1.728 (0.084)	[233] -0.083 (0.934)	[183] -7.137 (*)	[232] -0.267 (0.789)	[182] -6.667 (*)	[234] -7.974 (*)
proportion: [n] z (p)	[3] 0.0 (0.248)	[233] 0.7 (*)	[183] 0.891 (*)	[232] 0.69 (*)	[182] 0.879 (*)	[234] 0.726 (*)
ubo50	stnu-$pro_{0.9}$	stnu-react	stnu-pro_{SAA}	$pro_{0.9}$-react	$pro_{0.9}$-pro_{SAA}	react-pro_{SAA}
Wilcoxon: [n] z (p)	[284] -4.827 (*)	[284] -5.557 (*)	[285] -10.588 (*)	[162] -2.643 (*)	[173] -8.741 (*)	[171] -7.484 (*)
proportion: [n] z (p)	[284] 0.454 (0.138)	[284] 0.479 (0.514)	[285] 0.712 (*)	[7] 1.0 (*)	[173] 0.936 (*)	[171] 0.906 (*)
ubo100	stnu-$pro_{0.9}$	stnu-react	stnu-pro_{SAA}	$pro_{0.9}$-react	$pro_{0.9}$-pro_{SAA}	react-pro_{SAA}
Wilcoxon: [n] z (p)	[288] -8.783 (*)	[285] -10.58 (*)	[284] -13.068 (*)	[122] -4.786 (*)	[146] -5.008 (*)	[141] -1.441 (0.15)
proportion: [n] z (p)	[288] 0.576 (*)	[285] 0.653 (*)	[284] 0.82 (*)	[23] 1.0 (*)	[146] 0.836 (*)	[141] 0.702 (*)

Table 4.24 *Pairwise comparison on time online for noise factor $\epsilon = 1$. Using a Wilcoxon test and a proportion test. Including all instances for which at least one of the two methods found a feasible solution. Note that the ordering matters: the first method showed is the better of the two in each pair method 1 - method 2. Each cell shows on the first row [nr pairs] the z-value (p-value) of the Wilcoxon test with (*) for $p < 0.05$. Each cell shows on the second row [nr pairs] proportion (p-value) with (*) for $p < 0.05$.*

j10	$pro_{0.9}$-pro_{SAA}	$pro_{0.9}$-stnu	pro_{SAA}-stnu	$pro_{0.9}$-react	pro_{SAA}-react	stnu-react
Wilcoxon: [n] z (p)	[340] -0.368 (0.713)	[340] -15.993 (*)	[340] -15.997 (*)	[340] -15.98 (*)	[340] -15.98 (*)	[340] -2.726 (*)
proportion: [n] z (p)	[85] 0.471 (0.664)	[340] 1.0 (*)	[340] 1.0 (*)	[340] 1.0 (*)	[340] 1.0 (*)	[340] 0.765 (*)
j20	$pro_{0.9}$-pro_{SAA}	$pro_{0.9}$-stnu	pro_{SAA}-stnu	$pro_{0.9}$-react	pro_{SAA}-react	stnu-react
Wilcoxon: [n] z (p)	[270] -1.17 (0.242)	[270] -14.246 (*)	[270] -14.245 (*)	[270] -14.243 (*)	[270] -14.243 (*)	[270] -6.441 (*)
proportion: [n] z (p)	[133] 0.556 (0.225)	[270] 1.0 (*)	[270] 1.0 (*)	[270] 1.0 (*)	[270] 1.0 (*)	[270] 0.852 (*)
j30	$pro_{0.9}$-pro_{SAA}	$pro_{0.9}$-stnu	pro_{SAA}-stnu	$pro_{0.9}$-react	pro_{SAA}-react	stnu-react
Wilcoxon: [n] z (p)	[320] -0.653 (0.514)	[320] -15.505 (*)	[320] -15.505 (*)	[320] -15.504 (*)	[320] -15.504 (*)	[320] -8.247 (*)
proportion: [n] z (p)	[191] 0.524 (0.563)	[320] 1.0 (*)	[320] 1.0 (*)	[320] 1.0 (*)	[320] 1.0 (*)	[320] 0.875 (*)
ubo50	pro_{SAA}-$pro_{0.9}$	$pro_{0.9}$-stnu	pro_{SAA}-stnu	$pro_{0.9}$-react	pro_{SAA}-react	stnu-react
Wilcoxon: [n] z (p)	[370] -0.771 (0.44)	[370] -16.67 (*)	[370] -16.67 (*)	[370] -16.67 (*)	[370] -16.67 (*)	[370] -9.859 (*)
proportion: [n] z (p)	[299] 0.522 (0.488)	[370] 1.0 (*)	[370] 1.0 (*)	[370] 1.0 (*)	[370] 1.0 (*)	[370] 0.892 (*)
ubo100	pro_{SAA}-$pro_{0.9}$	$pro_{0.9}$-stnu	pro_{SAA}-stnu	$pro_{0.9}$-react	pro_{SAA}-react	stnu-react
Wilcoxon: [n] z (p)	[400] -0.719 (0.472)	[400] -17.331 (*)	[400] -17.331 (*)	[400] -17.331 (*)	[400] -17.331 (*)	[400] -12.488 (*)
proportion: [n] z (p)	[391] 0.453 (0.069)	[400] 1.0 (*)	[400] 1.0 (*)	[400] 1.0 (*)	[400] 1.0 (*)	[400] 0.925 (*)

Table 4.25 *Pairwise comparison on time online for noise factor $\epsilon = 2$. Using a Wilcoxon test and a proportion test. Including all instances for which at least one of the two methods found a feasible solution. Note that the ordering matters: the first method showed is the better of the two in each pair method 1 - method 2. Each cell shows on the first row [nr pairs] the z-value (p-value) of the Wilcoxon test with (*) for $p < 0.05$. Each cell shows on the second row [nr pairs] proportion (p-value) with (*) for $p < 0.05$.*

j10	$pro_{0.9}$-pro_{SAA}	$pro_{0.9}$-stnu	pro_{SAA}-stnu	$pro_{0.9}$-react	pro_{SAA}-react	stnu-react
Wilcoxon: [n] z (p)	[235] -0.384 (0.701)	[258] -9.554 (*)	[258] -9.212 (*)	[234] -13.262 (*)	[235] -12.623 (*)	[258] -8.801 (*)
proportion: [n] z (p)	[66] 0.47 (0.712)	[258] 0.907 (*)	[258] 0.899 (*)	[234] 1.0 (*)	[235] 0.987 (*)	[258] 0.891 (*)
j20	$pro_{0.9}$-pro_{SAA}	$pro_{0.9}$-stnu	pro_{SAA}-stnu	$pro_{0.9}$-react	pro_{SAA}-react	stnu-react
Wilcoxon: [n] z (p)	[182] -1.204 (0.229)	[216] -4.456 (*)	[217] -4.913 (*)	[177] -11.279 (*)	[182] -10.957 (*)	[217] -11.457 (*)
proportion: [n] z (p)	[79] 0.43 (0.261)	[216] 0.815 (*)	[217] 0.825 (*)	[177] 0.994 (*)	[182] 0.984 (*)	[217] 0.968 (*)
j30	pro_{SAA}-$pro_{0.9}$	$pro_{0.9}$-stnu	pro_{SAA}-stnu	$pro_{0.9}$-react	pro_{SAA}-react	stnu-react
Wilcoxon: [n] z (p)	[182] -2.014 (*)	[232] -0.928 (0.353)	[234] -2.526 (*)	[163] -10.268 (*)	[183] -9.31 (*)	[233] -9.839 (*)
proportion: [n] z (p)	[119] 0.58 (0.099)	[232] 0.69 (*)	[234] 0.735 (*)	[163] 0.982 (*)	[183] 0.94 (*)	[233] 0.901 (*)
ubo50	$pro_{0.9}$-pro_{SAA}	stnu-$pro_{0.9}$	stnu-pro_{SAA}	$pro_{0.9}$-react	pro_{SAA}-react	stnu-react
Wilcoxon: [n] z (p)	[173] -0.722 (0.47)	[284] -3.843 (*)	[285] -3.933 (*)	[162] -11.04 (*)	[171] -9.02 (*)	[284] -13.107 (*)
proportion: [n] z (p)	[147] 0.524 (0.621)	[284] 0.43 (*)	[285] 0.435 (*)	[162] 1.0 (*)	[171] 0.942 (*)	[284] 0.951 (*)
ubo100	pro_{SAA}-$pro_{0.9}$	stnu-$pro_{0.9}$	stnu-pro_{SAA}	$pro_{0.9}$-react	pro_{SAA}-react	stnu-react
Wilcoxon: [n] z (p)	[146] -0.482 (0.63)	[288] -8.467 (*)	[284] -9.06 (*)	[122] -9.585 (*)	[141] -6.136 (*)	[285] -14.285 (*)
proportion: [n] z (p)	[144] 0.458 (0.359)	[288] 0.576 (*)	[284] 0.585 (*)	[122] 1.0 (*)	[141] 0.837 (*)	[285] 0.982 (*)


LONGTIME NO SEE!
MUCH TOO LONG! HOW ARE YOU?

5

Rolling-horizon production sequence optimization

In this chapter, we study a highly complex scheduling problem that requires the generation and optimization of production schedules for a multi-product biomanufacturing system with continuous and batch processes. There are two main objectives here; makespan and lateness, which are combined into a cost function that is a weighted sum. An additional complexity comes from long horizons considered (up to a full year), yielding problem instances with more than 200 jobs, each consisting of multiple tasks that must be executed in the factory. We aim to answer the question: ***"How can optimization approaches be designed to effectively plan production sequences in complex biomanufacturing environments over long planning horizons?"*** *We investigate the integration of discrete-event simulation, used to evaluate production sequences, with search algorithms that optimize these sequences. We study whether a rolling-horizon principle is more efficient than a global strategy. We evaluate how cost function weights for makespan and lateness should be set in a rolling-horizon approach where deadlines are used for subproblem definition. We show that the rolling-horizon strategy outperforms a global search, evaluated on problem instances of a real biomanufacturing system, and we show that this result generalizes to problem instances of a synthetic factory.*

This chapter is based on the article: *Van Den Houten, K., De Weerdt, M., Tax, D. M., Freydell, E., Christoupoulou, E., & Nati, A. (2023, December). Rolling-Horizon Simulation Optimization For A Multi-Objective Biomanufacturing Scheduling Problem. In 2023 Winter Simulation Conference (WSC) (pp. 1912-1923). IEEE.*

5.1. Introduction

Efficient bioprocess industries can play a crucial role in feeding the world's population sustainably. Production sites for the process industries are often multipurpose, highly flexible systems. The number of products that use (partially) the same machines in their production processes has increased under market pressure. Given long-horizon demand spanning a full production year, factory operators want to optimally use their resources while ensuring customer order deadlines are met. Using these deadlines as hard constraints makes the scheduling too restrictive; instead, a lateness objective can be defined, which sums the lateness of all customer orders. The efficiency of a schedule can be evaluated by measuring the makespan. Intuitively, makespan minimization also reduces the lateness of customer orders. On the other hand, not considering deadlines can lead to schedules with lower makespans. To illustrate this, ignoring deadlines allows clustering products so that shared resources are used more efficiently. For example, the same products for different customers can sometimes be produced in a single batch, or when subsequent batches are for the same or similar products, machine set-up and cleaning times can be shorter. The trade-off between the lateness and makespan objectives makes the optimization, therefore, challenging.

Schedulers in these industries face highly complex scheduling problems that are, in general, NP-hard (G. P. Georgiadis, Elekidis, and M. C. Georgiadis 2019a; Pinedo 2016). Therefore, developing efficient algorithms to optimize scheduling decisions in large-scale bioprocessing industries is challenging. Several studies are dedicated to exact optimization methods, such as linear programming, for this type of scheduling problem, which are summarized by G. P. Georgiadis, Elekidis, and M. C. Georgiadis (2019a). The main conclusion is that, for the optimization of large-scale, long-horizon scheduling problems, the only successes are achieved with the help of decomposition, heuristics, and simplifications, because linear programming models are otherwise intractable. Especially in biomanufacturing, as performed by dsm-firmenich (a global company in Nutrition, Health, and Bioscience), details of operational constraints are extremely important for generating feasible production schedules. Oversimplification should preferably be avoided in the solution strategy. Ideally, we have algorithms that can produce optimal solutions for very detailed models of such scheduling problems, but for problems of this complexity (NP-hard) and size, this may never be possible (Garey and Johnson 1990).

Simulation-based methods are more suitable than linear programming for modeling interdependent processes, constraints, and interactions in manufacturing systems. We observe that detailed Discrete Event Simulators (DES) are already used in practice for scenario analysis and manual scheduling. Currently, dsm-firmenich schedulers aim for a good production sequence, which is then later augmented by a highly detailed DES that contains heuristic rules for machine selection and timing to evaluate different production sequences. Mimicking detailed processes of a factory can be done with help of sophisticated simulation software, while including all these details in a linear programming model results in intractable models. These DES are often very specific to the manufacturing system they represent. Algorithmic techniques that support scheduling decisions are more generic. One prime example of this is simheuristics (Juan, Faulin, *et al.* 2015), which are defined as metaheuristics combined with a simulation tool for cost function evaluation. Only a few studies have been dedicated to the design of such

simheuristics integrated with industry-proven DES for long-horizon scheduling of large-scale biomanufacturing systems.

In this research, we focus on developing simheuristics for sequencing customer orders. The employed DES translates a production sequence into a feasible schedule, and evaluates the corresponding objectives. We investigate which optimization strategies are efficient for scheduling with long production horizons in the (bio)process industries. Inspired by manually constructed schedules, in which deadlines are used to construct a reasonable production sequence, we wonder how much better results could be obtained by a global simheuristic or by a rolling-horizon simheuristic that uses deadlines to reduce the search space. We aim to gain insight into how this search space reduction affects solution quality and runtime. Given a weighted sum of the makespan and lateness as a cost function for optimization, the research examines whether the same weights should guide the subproblem optimizations within the rolling-horizon simheuristic.

We observe that using product deadlines to construct a rolling-horizon simheuristic improves the algorithm's cost and computation time compared to a global strategy. Surprisingly, the rolling-horizon strategy performs even better (by a significant margin) than a global simheuristic that begins with a production sequence sorted by deadline, when both simheuristics are given an equal budget. Although we expected it would be better to increase the makespan weight in the rolling-horizon simheuristic, we observe that the differences in the objectives obtained are minor compared to using the true weights in the subproblem search.

The formal problem description is described in Section 5.2, which is inspired by an industrial use case, provided by dsm-firmenich. Then, in Section 5.3, we give an overview of related work. The employed DES are discussed in Section 5.5, which comprises the industry-proven model of the dsm-firmenich factory (Rockwell Arena), and a model for a synthetic factory implemented in SimPy, which we make publicly available. The latter is developed to show that our algorithms work independently of the DES and can be generalized to other factories. Additionally, we discuss optimization strategies in Section 5.6, including the rolling-horizon. For the experiments and results, we refer to Section 5.7 and Section 5.8.

5.2. Formal problem statement

In this chapter, we consider a combinatorial optimization problem of the form

$$\min_{\pi \in S} f(\pi, x), \tag{5.1}$$

where $\pi \in S$ denotes a solution in the discrete solution space S, x represents the problem parameters, and $f(\cdot)$ is the objective function. We assume that $f(\cdot)$ is a performance metric that can only be evaluated via simulation, given x and a solution π. Throughout this chapter, π is restricted to be a sequence.

5.3. Related work

Scheduling decisions in the process industries include batching, sequencing, routing, machine assignment, and timing. Over the past decades, generic optimization-driven

methods for these problems have been studied (Kondili, Pantelides, and Sargent 1993; Pantelides 1993), and a comprehensive overview is provided by G. P. Georgiadis, Elekidis, and M. C. Georgiadis (2019a). In the process industries, it is often the case that the system cannot be modelled with jobs and operations, as is usually done in discrete manufacturing (Pinedo 2016). This is because operations such as batch mixing and splitting can occur, meaning a product batch may not retain its identity throughout the process. Other complicating factors include processing times that depend on batch size and the presence of continuous and semi-continuous processes. Due to the large scale, long-term, and industry-specific operational constraints, the potential of exact optimization methods is limited (Chica *et al.* 2020). Some examples where either Mixed Integer Linear Programming (MILP) or Constraint Programming (CP) models are employed for short-horizon industrial problems can be found in (Awad *et al.* 2022; G. P. Georgiadis, Ziogou, *et al.* 2019b). It was noted that, even for short-horizon problems, simplifications are needed in linear programming (G. P. Georgiadis, Elekidis, and M. C. Georgiadis 2019a), and manufacturers may not accept the proposed schedules due to infeasibility caused by insufficient detail (Klanke and Engell 2022).

Simulation optimization (SO) is considered a suitable alternative for exact optimization, since industry-strength simulation models could be integrated with optimization algorithms that interface with the simulation model as an external component. Simulation is powerful for evaluating stochastic components but can also be used to model complex systems with interdependent processes and constraints. More specifically, the value of simheuristics for realistic combinatorial optimization problems has been widely recognized (Chica *et al.* 2020; Juan, Faulin, *et al.* 2015; Juan, Li, *et al.* 2022). A recent tutorial on connecting Python to DES software for the development of simheuristics was given by Peyman and Dehghanimohammadabadi (2021). A simulation tool can be used to translate input scheduling decisions π into an actual schedule (a Gantt chart) and to estimate objective values $f(\pi, x)$. Simheuristic applications in scheduling include various job-shop problems combined with Monte Carlo Simulation (Gonzalez-Neira *et al.* 2017; Hatami *et al.* 2018; Juan, Barrios, *et al.* 2014). Research on simheuristics for scheduling in process industries is limited, especially for multi-objective, long-horizon problems. (Piana and Engell 2010) developed a simheuristic method for a chemical engineering plant, with makespan as the objective. (Klanke, Bleidorn, *et al.* 2021) proposed an evolutionary algorithm (EA) combined with a DES tool to optimize an industrial formulation plant. They evaluated their method on medium-term problem instances.

For the design of simheuristics, it is advised that the choice of a simheuristic framework align with the complexity of the simulation tool, meaning that complex simulation tools often work better with simple simheuristics, and vice versa (Chica *et al.* 2020). Iterated Greedy (IG) algorithms have proven successful for sequencing scheduling problems, such as the permutation flowshop scheduling problem (PFSP) (Ruiz and Stützle 2007) and its hybrid version (HFSP) (Öztop *et al.* 2018).

When focusing specifically on problems with long horizons, rolling-horizon algorithms can be helpful. With this principle, a sequence of subproblems is solved, each of which can be controlled in size. Such strategies have been applied in several studies (Glomb, Liers, and Rösel 2022; Ovacik and Uzsoy 1994). A rolling-horizon principle applied to a batch multi-product production system is provided by Wu *et al.* (2021), although the

studied system is significantly smaller than the factory of interest. Earlier research has shown that such principles combined with exact methods can even yield near-optimal solutions (Glomb, Liers, and Rösel 2022).

We recognize a clear gap in the literature regarding the scheduling of bioprocess factories. On the one hand, we observe that linear programming approaches have been successful on small, short-horizon, and/or simplistic problems. On the other hand, simulation-based methods for large-scale industrial problems focusing on long horizons are rare. Rolling-horizon principles have been applied to batch manufacturing, but, as far as we know, it is not clear whether a rolling-horizon method would yield better results in the same time as a global search in such a large-scale, complex system as studied in this research. In particular, we would like to understand whether the objectives in such a multi-objective problem should be weighted differently in a rolling-horizon setting compared to the long-horizon objective.

5.4. The factory scheduling problem

The scheduling problem can be described using the standardized framework provided by G. P. Georgiadis, Elekidis, and M. C. Georgiadis (2019a), complemented by additional specifications. We observe a multi-purpose, multi-product network facility, where different bio-based products are produced according to unique recipes, and the plant routings are product-specific and flexible. Full-year demand is provided, yielding more than 200 standard-quantity orders per year. Different terminology for these orders can be used (orders/batches/products/jobs), but we formally refer to them as jobs $\mathscr{J}$ in the remainder of this paper. Each job consists of multiple unit operations executed either subsequently or (partially) in parallel. An additional complexity is the mix of batch and semi-continuous processes.

Currently, dsm-firmenich schedulers focus on sequencing the different products based on the full-year production plan. Decisions regarding batch sizing are made in advance. Soft deadlines for the different jobs are given per month. A schedule assigns resources to tasks and sequences and schedules the tasks; a feasible schedule satisfies all predefined operational constraints. The scheduling objectives are the schedule's total time (makespan) and the total lateness of customer orders, which are combined into a cost function. Factory managers define the cost function as a weighted sum of the two objectives, with weight $\lambda \in [0,1]$ for the makespan objective and $1-\lambda$ for the lateness objective. For the factory of interest, the decision-makers set $\lambda^* = 0.5$. Given that c_j is the completion time of job j, and d_j is the deadline of job j, the cost function is defined as:

$$f_{\lambda^*} = \lambda^* \max_{j\in\mathscr{J}} c_j + (1-\lambda^*) \sum_{j\in\mathscr{J}} \max(0, c_j - d_j) = 0.5 \max_{j\in\mathscr{J}} c_j + 0.5 \sum_{j\in\mathscr{J}} \max(0, c_j - d_j) \quad (5.2)$$

The facility that we study consists of thirteen resource groups, with one up to eleven machines per group, from which capacities can differ within one group (for an overview, see Table 5.1). Processing times are often quoted as rates (dependent on batch sizes) and are typically long (> 100 hours), which makes the use of time-indexed models problematic. The availability of raw material and man-hours is left out of scope. At this manufacturing

site, 55 different end products can be produced. Additional complexities of the system are:

- compatibility constraints,
- sequence-dependent cleaning times and pre- and post-processes,
- scheduled maintenance,
- changing batch weights/sizes (e.g., due to filtering steps),
- cooling restrictions.

Table 5.1 *Resource groups.*

Resource group	Number of machines
Fermenter 1 - 5	5
Harvesting tanks A	4
Harvesting tanks B	8
Filters A	3
Filters B	1
Buffer tanks	6
Filters C	5
Filters D	6
Stabilization tanks	11

Given this large-scale, NP-hard scheduling problem with a comprehensive set of detailed operational constraints, we wonder which optimization strategy is best suited to generating and improving production schedules. We emphasize that oversimplification is undesirable because it can lead to schedules that factory managers will reject. We are particularly interested in how to handle long-horizon problem instances up to a full production year because 1) this is highly relevant for our industrial partner, and 2) there is a clear gap in the literature, in which the majority studied short to medium term horizons.

5.5. Discrete-event simulation

Given our interest in scheduling a highly detailed process, simulation optimization (SO) emerges as a suitable approach for modelling all operational constraints and interdependent processes (Juan, Faulin, *et al.* 2015; Klanke and Engell 2022). As discussed earlier, finding optimal solutions in time for this problem is not expected due to its size and complexity. To address this challenge, we employ two deterministic discrete-event simulators (DES) that map a production sequence π, representing the order of products $\mathscr{J}$, to a feasible schedule. The simulation model transforms π into feasible start times $s_j(\pi)$ and completion times $c_j(\pi)$, which are then used to evaluate the objective function, yielding the following equivalent formulation:

$$f_\lambda(\pi, x) = \lambda \max_{j \in \mathcal{J}} c_j(\pi) + (1-\lambda) \sum_{j \in \mathcal{J}} \max\big(0, c_j(\pi) - d_j\big). \tag{5.3}$$

Here, x summarizes the input parameters of the simulation model that are not explicitly shown; in particular, the deadlines d_j are components of x. For simplicity and readability reasons, we sometimes also use f_λ or $f_\lambda(\pi)$ throughout this paper.

The simulation model mimics the flow of batches through the factory while adhering to all scheduling rules specified by the product-specific recipes. The simulation incorporates heuristic rules to determine machine assignments and the precise timing of operations. Therefore, it should be noted that some sub-optimality cannot be avoided. Nonetheless, this approach facilitates the use of simheuristics that solely optimize the production sequence without having to consider all the complex interactions modeled in the DES.

Currently, dsm-firmenich schedulers use a DES that is implemented in Rockwell Arena software (see Drevna and Kasales (1994)). This tool is validated and is now integrated into a simheuristic framework. To test whether our proposed methods can be generalized, we create a synthetic factory and develop a DES using the open-source Python package SimPy (see Matloff (2008)). Both models take as main input a production sequence, and output a feasible schedule, which can be visualized as a Gantt chart, such as is shown in Figure 5.1, and Figure 5.2.

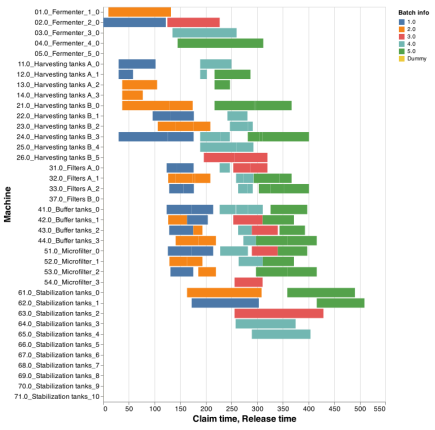


Figure 5.1 *Example Gantt dsm-firmenich Factory*

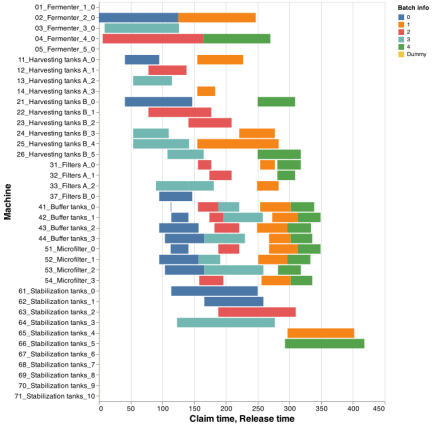


Figure 5.2 *Example Gantt Synthetic Factory*

1. dsm-firmenich factory (Arena) The Arena model is a highly detailed DES, which was developed by Systems Navigator consultants. Communication between Arena and Python is realized through text files and Arena replications. The model reflects the dsm-firmenich factory, including all product flows, mandatory cleaning times, scheduled maintenance, cooling constraints, and measures the required objectives. Given a production sequence, it keeps track of resource utilization at discrete time steps, including batch weights, and

information about intermediate and semi-finished products. The simulation tool also deterministically translates the input sequence into the objectives.

2. Synthetic factory (SimPy) We developed a synthetic factory with the help of the open-source Python library SimPy (Matloff 2008). For the SimPy simulator, we copied the combinatorial size of the real factory, i.e., we used a similar number of resource groups, resources, and unique products. Some characteristics of the real factory are omitted, such as the exact flow of liquid volumes, maintenance, and changeover times. The simulation tool is accessible via `https://github.com/kimvandenhouten/SimPyManufacturing`. This tool is currently used in a deterministic mode.

5.6. Optimization strategy

This section presents two simheuristic strategies for tackling Equation (5.1), where the search space is the permutation space S_n, which represents all possible permutations of the unordered set $[n]$. We propose a global search approach and a rolling-horizon approach. We investigate which strategy is more effective for the considered long-horizon problem instances. The budget refers to the number of cost function evaluations or different sequences evaluated using the DES in a single algorithm run. To compare the different optimization strategies, we fix the number of cost function evaluations. First, we introduce the search algorithms used in Section 5.6.1, then we elaborate on the global search simheuristic in Section 5.6.2 and the rolling-horizon approach in Section 5.6.3.

5.6.1. Search algorithms

We employ relatively simple search algorithms, following key advice from the literature for simheuristics integrated with complex DES (Chica *et al.* 2020). Therefore, we use a local search with random swaps, as shown in Algorithm 11. The initialisation is either random or sorted by deadline, and we hypothesize that the latter is helpful for the lateness objective. We also test an iterated greedy algorithm, shown in Algorithm 12, which has been successfully applied to sequencing scheduling problems (Ruiz and Stützle 2007).

Algorithm 11 Local search

requires f: objective function, n: length of sequence, $budget$: total number of objective evaluations
π = random sequence of length n ▷ *Initialize, alternatively an initial solution can be provided.*
$\pi_b = \pi$ ▷ *Store the best solution so far*
while number of objective evaluations <budget **do**
    $\pi' = \text{SWAP}(\pi)$ ▷ *Create new candidate solution by random swap*
    if $f(\pi', x) < f(\pi, x)$ **then** ▷ *Update current solution*
        $\pi = \pi'$
    if $f(\pi, x) < f(\pi_b)$ **then** ▷ *Update best solution*
        $\pi_b = \pi$
return π_b

Algorithm 12 Iterated Greedy (Ruiz and Stützle 2007)

requires f: objective , n: length of sequence, budget: total number of objective evaluations, a: parameter for local search
π = random sequence of length n ▷ *Initialize, alternatively an initial solution can be provided.*
π = IteratedImprovementInsertion(π)
$\pi_b = \pi$
while number of objective evaluations < budget **do**
    π' = destruct version π by removing a items at random
    for i=1, ..., a **do**
        π' = best_insert(π')
    π' = IteratedImprovementInsertion(π')
    if $f(\pi') < f(\pi)$ **then** ▷ *Update current solution*
        $\pi = \pi'$
        if $f(\pi) < f(\pi_b)$ **then** ▷ *Update best solution*
            $\pi_b = \pi$
return π_b

Algorithm 13 Iterated Improvement Insertion (Ruiz and Stützle 2007)

requires π, f objective function
improve = True
while improve=True **do**
    improve = False
    for i=1,...,n **do**
        remove an item k from π
        π' = best_insert(π)
        if $f(\pi') < f(\pi)$ **then**
            $\pi = \pi'$
            improve = True
return x

5.6.2. Global search

The `global search` strategy utilizes the entire search space S_n and aims to minimize f_{λ^*} (see Equation (5.2) or Equation (5.3)). A fixed evaluation budget is given as a hyperparameter and tuned to the instance size. Due to the randomness in the search algorithms, we perform multiple restarts and report the minimum costs obtained. Furthermore, we compare all strategies against a random search baseline.

5.6.3. Rolling-horizon

The `rolling-horizon` strategy aims for an efficient reduction of the search space S_n, and is summarized in Algorithm 14. Rolling-horizon algorithms have been successfully

employed for large-scale problems (Ovacik and Uzsoy 1994). Our proposed strategy utilizes product deadlines to divide the global problem into subproblems. We aim to understand how the consequent reduction of the search space affects solution quality. We aim to investigate the performance difference between the rolling-horizon and global search strategies. The first step is to sort the production sequence by deadline. Subsequently, parts of the sequence are iteratively optimized using the local search method with random swaps. The hyperparameter k determines the size of each subproblem, and the hyperparameter m fixes the first m items of the just-optimized subsequence. Different (k, m) combinations control how much the search space is reduced. During subproblem evaluation, all previously solved and fixed items of the production sequence are included. This means that the length of the sequence evaluated with the DES increases throughout the algorithm. The optimization task remains to minimize f_{λ^*}, where the decision-makers set $\lambda^* = 0.5$. However, the search algorithms within the rolling-horizon strategy use f_λ with $\lambda \in [0, 1]$ as a hyperparameter. This investigation aims to determine whether the rolling-horizon strategy can achieve better sequences (i.e., lower f_{λ^*} values) while tuning λ. We again perform multiple random restarts and report the minimum costs obtained and the total runtime for each restart.

Algorithm 14 Rolling-horizon strategy

Require: `search`: search algorithm for sequence optimization, $\mathsf{budget}^{\mathsf{total}}$: total number of cost function evaluations, f_{λ^*}: true cost function, f_λ: cost function used for subproblems, demandList: list of jobs $\mathscr{J}$ including deadlines d_j, n: length of demand list, k: number of products considered per search, m: number of products fixed after each search

initialize:
$\pi^{production} \leftarrow$ SORT_BY_DEADLINE(demandList) ▷ *Provide initial solution.*
numSearches $\leftarrow$ ROUND(n/m) ▷ *Determine total number of runs.*
$\mathsf{budget}^{\mathsf{search}} \leftarrow$ ROUND($\mathsf{budget}^{\mathsf{total}}$/numSearches) ▷ *Determine the budget per search.*
$\pi^{fixed} \leftarrow []$ ▷ *Initialize the fixed part as an empty list.*
for i in `range`(0, numSearches): **do**
  $\pi \leftarrow \pi^{production}[i \cdot m : i \cdot m + k]$ ▷ *Slice from production sequence.*
  $\pi \leftarrow$ COMBINE(π^{fixed}, π) ▷ *Combine fixed part of the sequence with a new slice.*
  $\pi^{optimized} \leftarrow$ SEARCH(budgetsearch, f_λ,π)
  $\pi^{production}[i \cdot m : i \cdot m + k] \leftarrow \pi^{optimized}$ ▷ *Update the optimized part.*
  $\pi^{fixed} \leftarrow \pi^{production}[0 : (i+1) \cdot m]$ ▷ *Update the fixed part.*
return $\pi^{production}$, $f_{\lambda^*}(\pi^{production})$ ▷ *Return final production sequence and objective value.*

5.7. Experiments

This section provides details on our experimental setup, including the hyperparameter configurations for the algorithms. We begin by explaining the tuning process, which involves adjusting the budget and random restarts for the algorithms. Specifically, for the rolling-horizon strategy, we focus on tuning the weight parameter λ. Given that the rolling-

horizon simheuristic implicitly considers the lateness objective, we hypothesize that it could be beneficial to increase the makespan weight λ during the rolling-horizon. We also explain how we evaluate the different search algorithms, first testing them on short-horizon problems to identify the most promising search strategies. Subsequently, we conduct experiments comparing the effectiveness of global and rolling-horizon strategies for long-horizon instances and optimize the tuning of the rolling-horizon approach.

5.7.1. Evaluation

We evaluate our algorithms on two factories introduced in Section 5.5. All experiments involving the Arena model are done on a Dell Latitude E7450 with Intel Core i7 5600U. The SimPy experiments are run on a virtual server with an Intel(R) Xeon(R) Gold 6148 CPU with 2.39 GHz processors and 16.0 GB of RAM. We generated test instances for both problems that are representative of the issues our industrial partner faces. In the dsm-firmenich factory, on average, 20 products per month are produced. We used historical data to generate problem instances (specific parameter combinations for the scheduling problem) of varying sizes with product mixes representative of the real factory.

First, we evaluate different simheuristics on small instances, with a horizon of one, or two months. We test the different algorithms with different initialisations: one initialised randomly, and one initialised using a heuristic rule: sorted by deadline. We analyze which search strategy minimizes the cost function for the short horizon. The outcome is used to determine which search strategies to use within the rolling-horizon algorithm. Then, we test the simheuristics for the long-horizon problem instances with horizons of six months or a full year. Finally, we include an evaluation of one particular problem instance, for which we know the sequence selected by experts in the plant.

5.7.2. Hyperparameter settings

The settings for budget and random restarts are tuned on a subset of problem instances. For the budget, we use $\mathsf{budget} = (\mathsf{size}/20) \cdot 200$, and we decide to use a multi-start with 3 different random restarts for all algorithms, yielding per problem instance a total budget of $\mathsf{budget}^{\mathsf{total}} = 3 \cdot (\mathsf{size}/20) \cdot 200$. For the rolling-horizon (k, m), and λ are tuned. The tuning for (k, m) resulted in the setting $(k, m) = (40, 10)$. While tuning the setting λ, we observed that in calibration across different problem instances, different λ settings performed best. Since it was not always the case that the best performing setting was equal to λ^*, we decided to include two different rolling-horizon settings in our final experimentation. We both use $\lambda = \lambda^* = 0.5$ (the one that is similar to the weights in the costs function), and $\lambda = 0.9$ (which performed promisingly according to the tuning).

5.8. Results

To read the results in Table 5.2, and Table 5.3, we introduce some abbreviations. We use `A`, and `S` to refer to the Arena, and the SymPy models. We refer to local search with `LS`, random search with `RS`, iterated greedy with `IG`, and rolling-horizon with `RH`. Furthermore, `i=r` refers to a random initialisation, and `i=s` is an initialisation sorted by deadline. The different instances are encoded with `size_id`. The presented costs are the best values

obtained across the different restarts, and the runtime is the total time required to finish the algorithm.

5.8.1. Short horizon problem instances

Our objective is to determine the most effective search strategy in terms of computation time and costs for the given problem instances. To accomplish this, we compare the performance of three search strategies: random search, local search, and the iterated greedy approach. We specifically focus on short-horizon problem instances spanning one to two production months, as presented in Table 5.2. These experimental results determine the search strategy that will be used in the rolling-horizon simheuristic.

We observe that, for short-horizon problem instances, the local search strategy with random initialization outperforms both random search and the iterated greedy algorithm on instances of size 20. For instances of size 40, the local search strategy with sorted initialization outperforms all other methods. The differences in runtime are negligible.

5.8.2. Long horizon problem instances

Based on the outcomes from the short-horizon problem instances summarized in Table 5.3, we observe that local search is the most effective approach. However, whether a random or sorted initialization is preferable remains unclear. We focus on comparing the effectiveness of employing a global simheuristic versus a rolling-horizon simheuristic for the long-horizon problem instances. Additionally, we explore the potential benefits of adjusting the weights in the cost function used in the rolling-horizon approach.

We observe a clear pattern in the results. In the majority of instances, local search with sorted initialization outperforms local search with random initialization. However, a rolling-horizon strategy performs better than the global search method across all test instances. The tuned setting, $\lambda = 0.9$, only yielded better performance for some instances. Overall, our results demonstrate that the rolling-horizon strategy is significantly faster than the global search strategy across all test instances. This is because the DES evaluates shorter sequences at the beginning of the algorithm, thereby improving efficiency.

5.8.3. Evaluation on real production plan

As a final evaluation, we test how much we can improve a real schedule created by experts for the horizon July - December 2022, consisting of 122 products. We obtain the best solution with the rolling-horizon strategy. The total cost reduction is 45%. Looking at the two objectives seperately, this schedule improved the makespan with 4%, and total lateness with 58%.

5.9. Conclusion and discussion

In this chapter, we studied a real-world biomanufacturing scheduling problem that is very computationally challenging due to the high system flexibility, two objectives, and a long horizon. We investigated whether a rolling-horizon strategy is better at tackling such long-horizon problem instances than a global search approach. Additionally, we analyze whether adjusting objective weights in a rolling-horizon simheuristic yields lower-cost

Table 5.2 *Results short horizon problem instances.*

		Costs					Runtime (s)				
DES	**I**	$IG_{i=r}$	$IG_{i=s}$	$LS_{i=r}$	$LS_{i=s}$	RS	$IG_{i=r}$	$IG_{i=s}$	$LS_{i=r}$	$LS_{i=s}$	RS
A	20_1	1035	966	**614**	643	875	209	**206**	220	219	215
A	20_2	1605	1197	**1141**	**1141**	1513	199	204	202	**197**	220
A	20_3	1084	1360	891	**820**	1111	192	**190**	196	196	203
A	20_4	763	644	**578**	593	766	**185**	185	192	193	193
A	20_5	1228	1043	**945**	977	1210	**194**	**194**	201	201	201
A	20_6	1132	1128	**871**	904	1260	197	**196**	203	202	202
A	20_7	1180	1202	**878**	935	1138	**195**	**195**	203	204	203
A	20_8	1029	1316	972	**884**	1206	**188**	189	194	194	195
A	20_9	1274	1472	**830**	977	1357	**194**	195	202	203	201
A	40_1	6314	3410	3186	**2721**	6830	507	482	**459**	**459**	460
A	40_2	8014	3481	3012	**2876**	8083	481	480	**473**	476	**473**
A	40_3	7387	4816	3994	**3891**	8168	467	514	469	467	**466**
A	40_4	8834	6003	5056	**4813**	8903	509	490	**465**	469	467
A	40_5	6929	3474	3253	**2589**	7145	509	514	476	474	**473**
A	40_6	6655	3604	2479	**2447**	5852	490	484	467	473	**466**
A	40_7	8624	7400	5082	**4767**	8612	514	521	**475**	**475**	477
A	40_8	6814	5663	2897	**2811**	7004	497	501	**462**	481	464
A	40_9	6407	4806	**2151**	2608	5860	502	510	463	464	**462**
S	20_1	877	912	**817**	820	829	52	51	**39**	**39**	**39**
S	20_2	547	**467**	**467**	**467**	468	54	51	38	38	**37**
S	20_3	1491	1403	**1303**	1306	1321	51	53	**39**	41	**39**
S	20_4	735	724	**702**	**702**	708	51	54	40	**39**	**39**
S	20_5	972	**857**	**857**	**857**	863	36	40	29	**30**	**30**
S	20_6	624	628	589	587	593	54	54	**39**	**39**	**39**
S	20_7	976	**882**	**882**	**882**	890	54	59	**42**	44	**42**
S	20_8	689	**659**	**659**	**659**	660	48	47	36	**35**	**35**
S	20_9	1132	1130	**1086**	**1086**	1092	61	61	45	47	**45**
S	40_1	3991	2413	**2300**	2323	3068	235	237	181	**180**	**180**
S	40_2	3700	2210	1920	**1901**	2869	266	259	**192**	193	196
S	40_3	2851	1063	992	**984**	1689	254	256	**188**	191	189
S	40_4	2918	1759	1563	**1557**	2178	257	249	**188**	193	196
S	40_5	5519	3731	3420	**3405**	4184	240	231	178	**176**	181
S	40_6	3144	1547	**1402**	1404	2162	247	244	**183**	184	185
S	40_7	3781	2412	**2259**	2277	3108	250	245	191	187	**185**
S	40_8	4492	2386	2323	**2314**	3347	230	231	**171**	175	175
S	40_9	3090	2126	1860	**1848**	2824	250	256	190	**186**	189

solutions. Our objective was to develop a solution strategy that avoids oversimplifying the system, a common requirement for exact methods that can result in infeasible solutions violating essential constraints, which are subsequently rejected by plant managers.

A simheuristic framework integrated with an industry-proven DES proved effective for generating feasible schedules in a reasonable amount of time. We showed that a rolling-horizon principle yields better solutions than a global optimization strategy within a limited budget and significantly reduces computation time. Furthermore, we demonstrated the generalizability of our methods using the synthetic factory. We discovered that adjusting the cost function weights in the rolling-horizon strategy, particularly by increasing the makespan weight, improved performance for certain problem instances.

As a valuable contribution, we achieved cost reduction in an actual schedule imple-

Table 5.3 *Results long horizon instances.*

		Costs					Runtime (m)				
DES	**I**	LS	LS	RS	RH	RH	LS	LS	RS	RH	RH
		$i=r$	$i=s$		λ_1=0.5	λ_1=0.9	$i=r$	$i=s$		λ_1=0.5	λ_1=0.9
A	120_1	26157	20540	77153	**11903**	14069	43	43	42	38	**37**
A	120_2	30402	23217	74626	**14958**	15740	41	44	41	38	**36**
A	120_3	32640	23125	82278	**15765**	17604	40	40	40	36	**35**
A	120_4	25123	16379	77207	**11733**	11484	40	39	40	32	**30**
A	120_5	32416	23968	84219	**20328**	18215	40	41	40	32	**31**
A	120_6	30917	23509	84376	**17455**	18585	40	40	41	**32**	**32**
A	120_7	26769	21854	82850	14244	**11947**	40	40	40	**32**	33
A	120_8	17555	13239	62741	**5496**	6967	39	39	39	**31**	**31**
A	120_9	20671	12481	69812	**5932**	7650	40	39	39	32	**31**
A	240_1	158342	97156	380652	**43090**	49758	120	118	117	87	**86**
A	240_2	123525	72521	348745	**34820**	38030	110	120	126	**84**	**84**
A	240_3	120497	89206	343697	53132	**38557**	111	113	111	86	**83**
A	240_4	121902	77800	333146	**43307**	46398	121	118	119	**84**	85
A	240_5	113269	78149	350476	48115	**42695**	122	122	119	**84**	**84**
A	240_6	110896	71205	361004	**26128**	26901	124	137	129	84	**81**
A	240_7	124405	85067	369330	61705	**57794**	135	145	139	**81**	**81**
A	240_8	118431	86317	352642	**52791**	57862	151	126	159	**81**	**81**
A	240_9	111132	86317	345495	**43259**	44054	119	108	112	80	84
S	120_1	5272	3982	23986	**4029**	4074	57	56	57	**39**	40
S	120_2	4705	3640	24463	3675	**3670**	57	57	57	41	**40**
S	120_3	2590	2336	19617	2318	**2305**	48	49	50	36	**35**
S	120_4	8536	8105	25313	8208	**8135**	54	54	55	41	**39**
S	120_5	3900	3282	21076	3316	**3264**	44	45	45	34	**33**
S	120_6	4521	3308	24599	**3318**	3345	51	51	52	**37**	**37**
S	120_7	7113	5398	27043	**5343**	5395	52	51	52	**38**	**38**
S	120_8	3521	3004	21048	2984	**2979**	47	47	48	**34**	**34**
S	120_9	5799	4691	22149	4783	**4678**	54	53	54	**39**	**39**
S	240_1	31855	23246	124680	23019	**22873**	308	309	315	**173**	**173**
S	240_2	23204	16701	115122	**16127**	16337	298	294	300	**169**	176
S	240_3	11236	5596	99495	**5552**	5565	307	312	296	**170**	172
S	240_4	8373	5488	86714	**5289**	5300	274	294	299	**158**	**158**
S	240_5	10386	5412	96277	**5375**	5405	326	327	328	172	**171**
S	240_6	8987	6114	92082	**6093**	6098	313	315	317	**163**	166
S	240_7	16620	11549	105234	11346	**11334**	329	327	330	178	**174**
S	240_8	12416	4728	99579	4692	**4691**	337	343	341	**184**	**184**
S	240_9	18326	12600	109805	**12424**	12757	331	328	332	**174**	177

mented at the dsm-firmenich factory during the period of July-December 2022. For future research, we aim to better understand which cases benefit from adjusting cost function weights within the context of the rolling-horizon strategy. Future work will involve improving the simheuristics by using more informed metaheuristics, expanding simulations to incorporate uncertainty, and including additional scheduling decisions beyond sequencing to reduce the optimality gap. Despite these potential future improvements, this study lays the foundation for the development of intelligent simulation-based algorithms for highly complex manufacturing systems in the process industries.

To help further research in this domain, we have made the SimPy DES of the synthetic factory publicly available. We hope this will stimulate the research community to explore

this area and contribute insights that can lead to the creation of a diverse set of benchmark factories for evaluation and algorithm development.

References

Awad, M., K. Mulrennan, J. Donovan, R. Macpherson, and D. Tormey (2022). "A constraint programming model for makespan minimisation in batch manufacturing pharmaceutical facilities". In: *Computers and Chemical Engineering* 156, p. 107565.

Chica, M., A. A. Juan Pérez, O. Cordon, and D. Kelton (2020). "Why simheuristics? Benefits, limitations, and best practices when combining metaheuristics with simulation". In: *Statistics and Operations Research Transactions* 44.2, pp. 311–334.

Drevna, M. and C. Kasales (1994). "Introduction to Arena". In: *Proceedings of the 1994 Winter Simulation Conference*. Ed. by J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila. Institute of Electrical and Electronics Engineers, Inc. Piscataway, New Jersey, pp. 431–436.

Garey, M. R. and D. S. Johnson (1990). *Computers and intractability; a guide to the theory of NP-completeness*. 1st. USA: W. H. Freeman Co.

Georgiadis, G. P., A. P. Elekidis, and M. C. Georgiadis (2019a). "Optimization-based scheduling for the process industries: from theory to real-life industrial applications". In: *Processes* 7 (7), p. 438.

Georgiadis, G. P., C. Ziogou, G. Kopanos, B. M. Pampín, D. Cabo, M. Lopez, and M. C. Georgiadis (2019b). "On the optimization of production scheduling in industrial food processing facilities". In: *Computer Aided Chemical Engineering* 46, pp. 1297–1302.

Glomb, L., F. Liers, and F. Rösel (2022). "A rolling-horizon approach for multi-period optimization". In: *European Journal of Operational Research* 300 (1), pp. 189–206.

Gonzalez-Neira, E. M., D. Ferone, S. Hatami, and A. A. Juan (2017). "A Biased-Randomized Simheuristic for the Distributed Assembly Permutation Flowshop Problem with Stochastic Processing Times". In: *Simulation modelling practice and theory* 79, pp. 23–36.

Hatami, S., L. Calvet, V. Fernández-Viagas, J. M. Framiñán, and A. A. Juan (2018). "A Simheuristic Algorithm to Set Up Starting Times in the Stochastic Parallel Flowshop Problem". In: *Simulation modelling practice and theory* 86, pp. 55–71.

Juan, A. A., B. B. Barrios, E. Vallada, D. Riera, and J. Jorba (2014). "A simheuristic algorithm for solving the permutation flow shop problem with stochastic processing times". In: *Simulation Modelling Practice and Theory* 46, pp. 101–117.

Juan, A. A., J. Faulin, S. E. Grasman, M. Rabe, and G. Figueira (2015). "A review of simheuristics: Extending metaheuristics to deal with stochastic combinatorial optimization problems". In: *Operations Research Perspectives* 2, pp. 62–72.

Juan, A. A., Y. Li, M. Ammouriova, J. Panadero, and J. Faulin (2022). "Simheuristics: an introductory tutorial". In: *Proceedings of the 2022 Winter Simulation Conference*. Ed. by B. Feng, G. Pedrielli, Y. Peng, S. Shashaani, E. Song, C. Corlu, E. C. L.H. Lee, T. Roeder, and P. Lendermann. Institute of Electrical and Electronics Engineers, Inc. Piscataway, New Jersey, pp. 1325–1339.

Klanke, C., D. Bleidorn, C. Koslowski, C. Sonntag, and S. Engell (2021). "Simulation-based scheduling of a large-scale industrial formulation plant using a heuristics-assisted genetic algorithm". In: *Proceedings of the 2021 Genetic and Evolutionary Computation Conference Companion*. Ed. by F. Chicano. Association for Computing Machinery. New York, United States, pp. 1587–1595.

Klanke, C. and S. Engell (2022). “Scheduling and batching with evolutionary algorithms in simulation–optimization of an industrial formulation plant”. In: *Computers and Industrial Engineering* 174, p. 108760.

Kondili, E., C. Pantelides, and R. Sargent (1993). “A general algorithm for short-term scheduling of batch operations”. In: *Computers & Chemical Engineering* 17 (2), pp. 212–227.

Matloff, N. (2008). *Introduction to discrete-event simulation and the SimPy language*. https://web.cs.ucdavis.edu/~matloff/matloff/public_html/156/PLN/DESimIntro.pdf.

Ovacik, I. M. and R. Uzsoy (1994). “Rolling horizon algorithms for a single-machine dynamic scheduling problem with sequence-dependent setup times”. In: *International Journal of Production Research* 32 (6), pp. 1243–1263.

Öztop, H., M. F. Tasgetiren, D. T. Eliiyi, and Q.-K. Pan (2018). “Iterated greedy algorithms for the hybrid flowshop scheduling with total flow time minimization”. In: *Proceedings of the 2018 Genetic and Evolutionary Computation Conference*. Kyoto, Japan: Association for Computing Machinery, pp. 379–385.

Pantelides, C. (1993). “Unified frameworks for optimal process planning and scheduling”. In: *Proceedings of the Second International Conference on Foundations of Computer-Aided Process Operations*. 18-23 July, Crested Butte, United States, pp. 253–273.

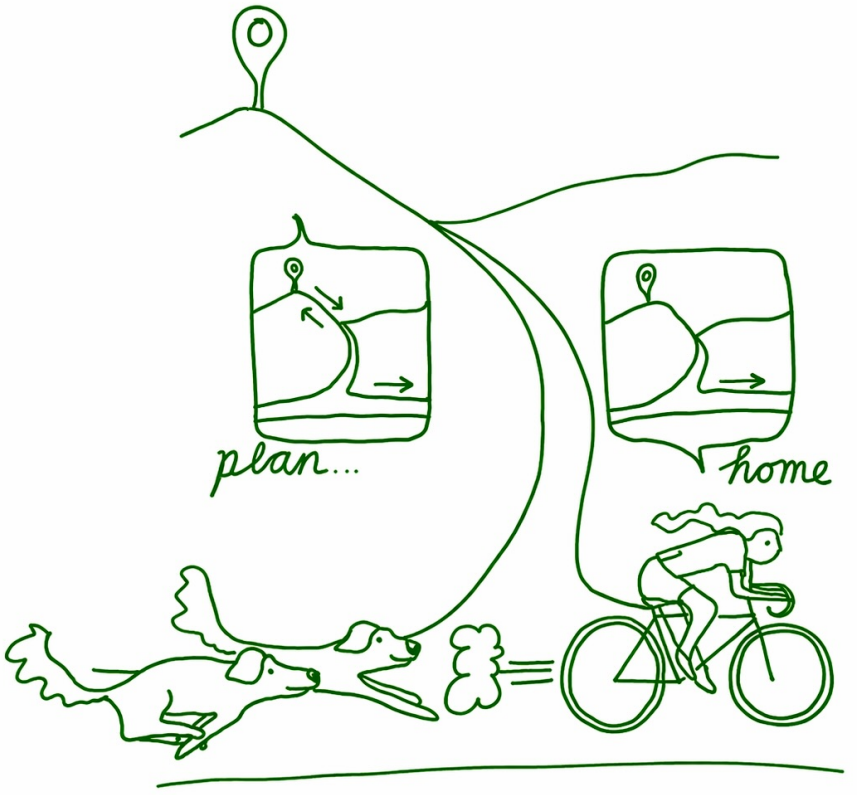
Peyman, M. and M. Dehghanimohammadabadi (2021). “A tutorial on how to connect Python with different simulation software to develop rich simheuristics”. In: *Proceedings of the 2021 Winter Simulation Conference*. Ed. by S. Kim, B. Feng, K. Smith, S. Masoud, Z. Zheng, C. Szabo, and M. Loper. Institute of Electrical and Electronics Engineers, Inc. Piscataway, New Jersey, pp. 1–12.

Piana, S. and S. Engell (2010). “Hybrid evolutionary optimization of the operation of pipeless plants”. In: *Journal of Heuristics* 16 (3), pp. 311–336.

Pinedo, M. (2016). *Scheduling: theory, algorithms, and systems*. 6th. Basel, Switzerland: Springer Nature.

Ruiz, R. and T. Stützle (2007). “A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem”. In: *European Journal of Operational Research* 177 (3), pp. 2033–2049.

Wu, O., G. D. Ave, I. Harjunkoski, and L. Imsland (2021). “A rolling horizon approach for scheduling of multiproduct batch production and maintenance using generalized disjunctive programming models”. In: *Computers and Chemical Engineering* 148, p. 107268.


plan...
home

6

Constraint programming for scheduling a fermentation plant

*In this chapter, we study "**How effective and scalable is constraint programming for solving real biomanufacturing scheduling instances?**" This study examines the scheduling challenges faced by a real-world biomanufacturing site. This fermentation factory is a multi-product, multi-stage facility where batches are scheduled to meet customer demand. Importantly, in the production process of these batches, they are split and mixed into subbatches. Between stages, the intermediate product's expiration should be avoided by maintaining a continuous flow without waiting in the factory, resulting in zero-wait policies. Sequence-dependent cleaning operations are also required between processing steps to avoid cross-contamination. Recent advances in constraint programming (CP) have demonstrated strong performance on standard scheduling problems. In this work, we investigate how effective state-of-the-art CP solvers are for solving real, large-scale biomanufacturing scheduling instances. Our empirical evaluation shows that CP Optimizer is more effective than Google OR Tools CP SAT. We show that a warm-start strategy based on domain knowledge improves the performance of the CP approach. Our empirical results show that relaxing the zero-wait constraints results in lower optimality gaps. Finally, we make our set of problem instances and CP model publicly available, thereby extending the scheduling literature with large-scale, realistic benchmarking instances obtained by our industrial collaboration.*

This chapter is submitted: *Van den Houten, K., Christoupoulou, E., Freydell, E., Tax, D.M.J., De Weerdt, M.M.. 2025. Constraint programming for scheduling a fermentation plant.*

6.1. Introduction

Biotechnology holds great promise for developing sustainable solutions to global food challenges. Optimizing the operations of large-scale biomanufacturing factories remains difficult due to their multi-stage processes and multi-product production, making it impossible to maintain the same schedule week after week. Ensuring that all production orders are delivered on time to meet demand is particularly challenging, given the complexity of scheduling and resource allocation in such facilities.

In biomanufacturing for the food industry, the complexity of scheduling production processes is further amplified by strict operational constraints. Typically, production orders translate into batches that must be scheduled accordingly. For example, in fermentation plants (Al-Maqtari, Waleed, and Mahdi 2019), each batch requires a sequence of unit operations, with fermentation—the stage in which living organisms grow an active intermediate product—being the most critical. These intermediate products are unstable and cannot wait long between stages due to their short shelf lives. Such restrictions translate to strict temporal constraints; for example, a harvesting operation must start shortly after fermentation is complete. Once a batch starts, the flow must go on! The required cleaning steps in between specific tasks to avoid contamination further complicate the process, which is very important, especially when the end products are food ingredients. Additionally, a major complicating factor is that frequently batches are split into subbatches and later mixed.

Although optimization methods (mostly Mixed-Integer Linear Programming (MILP)) have shown promise for solving scheduling problems, also in biomanufacturing, prior studies highlight the difficulty of developing generalizable and scalable optimization models that capture the detailed dynamics of biomanufacturing processes (G. P. Georgiadis, Elekidis, and M. C. Georgiadis 2019a; Harjunkoski *et al.* 2014). In particular, the survey by G. P. Georgiadis, Elekidis, and M. C. Georgiadis (2019a) identifies frequent mixing and splitting of material flows as a defining characteristic of such systems, and one that challenges the assumptions of conventional job-shop and project scheduling formulations that are common in scheduling literature (Kolisch and Sprecher 1996; Pinedo 2016). In contrast, material-based models (Kondili, Pantelides, and Sargent 1993; Pantelides 1993) are typically used when such features are present. According to (G. P. Georgiadis, Elekidis, and M. C. Georgiadis 2019a), most of these models have been tested only on small to medium-sized problems and are not capable of solving large-scale industrial problems.

Recent advances in Constraint Programming (CP) have sparked renewed interest in CP-based scheduling (Da Col and Teppan 2022; Laborie 2015; Lan and Berkhout 2025; Naderi, Ruiz, and Roshanaei 2023). These developments have demonstrated the ability of solvers such as IBM ILOG CP Optimizer (Laborie, Rogerie, *et al.* 2018) and Google OR-Tools CP-SAT (Perron and Furnon 2024) to find (near-)optimal solutions for instances involving more than one million operations (Da Col and Teppan 2022). However, the problem instances used in these benchmark studies are based on synthetic data and include different types of scheduling constraints in isolation. Furthermore, all these studies adopt job-shop scheduling frameworks, for which, as noted by Harjunkoski *et al.* (2014) and (G. P. Georgiadis, Elekidis, and M. C. Georgiadis 2019a), it is unknown how they can be applied to production environments with mixing and splitting. Consequently, only a few CP formulations for biomanufacturing have been proposed (Awad *et al.* 2022; Escobet

Canal *et al.* 2019; Novara, Novas, and Henning 2016; Zeballos, Novas, and Henning 2011), none of which account for mixing and splitting operations. We thus identify a research gap and aim to determine how well state-of-the-art CP solvers perform on large-scale biomanufacturing scheduling instances that include batch mixing and splitting.

In this work, we introduce and publicly release new problem instances that combine several challenging constraints, including zero-wait storage policies, sequence-dependent and independent setup times, flexible machines, complex unit-topology restrictions, and precedence constraints that naturally arise in production environments involving mixing and splitting operations, where batches are divided into subbatches. The splitting into subbatches results in complex precedence relations, and resources that must be claimed overlap, rather than always having a strict finish-to-start dependency structure. The instances use the data of a real biomanufacturing facility, which is an enzyme production site of dsm-firmenich, and combine constraints that go beyond the typical benchmark sets used in the scheduling literature (Lan and Berkhout 2025; Naderi, Ruiz, and Roshanaei 2023). We extend CP-formulations for job-shop scheduling problems (Lan and Berkhout 2025) to handle complex unit-topology restrictions.

Next, in our experimental study, we analyze the performance of CP solvers on these real instances. In this work, we advance the state of the art by evaluating performance across two CP solvers (with CP Optimizer consistently outperforming OR-Tools) and by contributing to a public repository of our benchmark instances using an extended version of the PyJobShop framework (Lan and Berkhout 2025). We extend this framework with a new constraint to represent topology dependencies, which are highly relevant and frequently encountered in batch scheduling problems. We propose a dedicated warm-start strategy to enhance computational efficiency. We analyze which constraint most significantly impacts scalability by studying the outcomes of problem variants that relax it.

We organize this paper as follows. We start with the formal problem statement in Section 6.2. Section 6.3 presents the state of the art. Section 6.4 explains the scheduling problem faced in the real factory. Section 6.5 presents the constraint programming solution. In Section 6.7, we report our experimental evaluation, in which we investigate our central research question: "What is the performance of CP when applied to realistic biomanufacturing scheduling problems?" Section 6.8 summarizes our main conclusions and reflections on this study.

6.2. Formal problem statement

In this chapter, we consider a combinatorial scheduling problem defined over a set of jobs $\mathscr{J}$. The goal is to determine a feasible schedule that optimizes a given objective function. Formally, the problem can be stated as

$$\min f(x, y) \text{ s.t. } x \in C(x, y), \tag{6.1}$$

where x represents the decision variables defining a schedule, $C(x, y)$ denotes the set of feasible solutions satisfying the scheduling problem constraints, y represents the problem parameters, and $f(\cdot)$ is the objective function.

The feasible set $C(x, y)$ is implicitly defined by a set of scheduling constraints, such as precedence relations, resource capacities, and timing constraints. A detailed constraint programming formulation is provided in Section 6.5.

6.3. Literature review

The literature on batch scheduling encompasses various problem features commonly encountered in biomanufacturing industries. We provide a brief overview of the primary problem characteristics considered in batch scheduling in Section 6.3.1. We will later use these problem features to compare our contribution with the state of the art (see Table 6.1).

Batch scheduling problems have traditionally been addressed using mixed-integer programming (MIP) formulations (G. P. Georgiadis, Elekidis, and M. C. Georgiadis 2019a). In their survey on MIP approaches for batch scheduling, (G. P. Georgiadis, Elekidis, and M. C. Georgiadis 2019a) distinguish between models developed for sequential facilities, where resource-constrained project scheduling and job shop scheduling formulations are more common, and network-based facilities, such as the State Task Network (STN) (Kondili, Pantelides, and Sargent 1993) and Resource Task Network (RTN) (Pantelides 1993) models. For both model types, they note that scalability remains limited and that the obtained solutions are often suboptimal for large-scale or industrial instances. An overview of MILP-based related work is provided in Appendix 6.A.

In comparison, there are far fewer contributions based on constraint programming (CP), although recent advances demonstrate that CP techniques outperform MILP for scheduling applications (Naderi, Ruiz, and Roshanaei 2023). Older literature (G. P. Georgiadis, Elekidis, and M. C. Georgiadis 2019a) even regards CP as less favorable for batch scheduling, noting that it does not provide optimality gaps. However, while this limitation applies to the general CP paradigm, modern solvers such as CP Optimizer include procedures that can provide lower bounds and optimality gaps in different ways, through (linear program) relaxations and destructive lower bounds, as explained by Vilím, Laborie, and Shaw (2015).

In Section 6.3.2, we review related work that applied constraint programming to batch manufacturing, and summarize the covered problem features in a comparative table.

6.3.1. Problem features

To describe the problem features typically encountered in batch scheduling, we draw on the overview provided by Awad *et al.* (2022), and construct Table 6.1. We repeat most of their explanations here, as this provides a valuable reference point and facilitates comparison with the state-of-the-art. We also utilize these problem features in a comparative table (Table 6.1) to highlight the features covered by related work.

To start with, an important distinction is made regarding batch facilities, specifically whether they are *network facilities* or *sequential facilities* (G. P. Georgiadis, Elekidis, and M. C. Georgiadis 2019a). A sequential process is one in which mixing or splitting of batches is not allowed. In contrast, a network facility allows a batch to be split into subbatches, which can then be mixed again.

Furthermore, in different settings, different types of equipment or resources are considered. We utilize the definitions provided by Lan and Berkhout (2025) and distinguish between machines, renewable resources, and non-renewable resources. The authors define a machine as a unique piece of equipment that can process only one task at a time; a renewable resource as having a fixed capacity at each time period; and a non-renewable resource as having a finite capacity that it can process over the planning horizon. In many cases, a machine can be seen as a special case of a renewable resource. Furthermore, the assignment of resources to tasks is either *fixed* or *flexible*, depending on whether there is a single possible assignment that accomplishes the task or whether a choice needs to be made. The task's processing time can also be *fixed* or *variable*, where the latter means it depends on the resources allocated to it. A variable processing time can also be used to model tasks whose duration depends on other scheduling decisions.

The nature of capacity restrictions and processing times also depends on whether a facility operates with *batch processes* or *continuous processes*, as also noted by Harjunkoski *et al.* (2014). A batch process has a fixed processing time and a batch size that drives capacity constraints, whereas a processing rate drives a continuous process.

In batch scheduling, different storage policies can be used between different stages of the production process. We distinguish *unlimited intermediate storage* (UIS), *no intermediate storage* (NIS), *finite intermediate storage* (FIS), and a *zero-wait* (ZW) policy. In batch scheduling, it is also common to require dedicated cleaning steps, which are often referred to as *setup times*. These setup times are *sequence-dependent* or *sequence-independent*, and also *unit-dependent* or *unit-independent*. Related to cleaning times, some factories operate in *campaign mode*, which means that different batches of the same product are processed sequentially, thereby reducing sequence-dependent cleaning times.

Additional constraints covered in the literature (Awad *et al.* 2022; Escobet Canal *et al.* 2019; Novara, Novas, and Henning 2016; Zeballos, Novas, and Henning 2011) include, for example, constraints on the *production sequence*, i.e., that two batches cannot be scheduled consecutively. Very common for batch scheduling are *topology* constraints, which put restrictions on the routing through a facility, i.e., if a task is processed by machine A, it can only flow to machine B. In contrast, if it has been processed by machine C, it can only be processed by machine D. *Transfer times* could indicate some additional time that is needed to transfer a batch from one machine to another. Other restrictions on scheduling may arise from scheduled *maintenance* on the resources or *work shifts*, which are necessary when human resources are required for specific tasks that cannot be performed during weekends, nights, lunch breaks, and/or holidays.

Furthermore, batches may be subject to *release dates* and *due dates*, indicating when they can be started or when they should be delivered. These due dates could be incorporated in the objective function, for example, when considering *earliness* or *tardiness*. The most common objective in batch scheduling and scheduling in general is the *makespan*, indicating the finish time of the latest task in the schedule. Of course, other *cost*-based objectives could also be defined.

The main scheduling decisions include *batching*, which defines how product demand is translated into batches to be scheduled; *sequencing*, which determines the order in which batches are processed; *timing*, which specifies the start and finish times of all tasks; and *machine assignment*, which allocates machines to individual tasks. In network-type

facilities (G. P. Georgiadis, Elekidis, and M. C. Georgiadis 2019a), decisions related to *mixing and splitting* may also be made, which describe how a batch is combined with or divided into subbatches.

We could also use Graham's notation $\alpha \mid \beta \mid \gamma$ to classify the scheduling problem model of the real fermentation plant, where the first field (α) specifies the machine environment, the second field (β) specifies the constraints, and the third field (γ) specifies the objective. Then we have FJ: flexible jobshop | s_{ijk}: setup times, b_i: buffers, $no-wait$: zero-wait constraints, $block$: jobs are blocked, $d_{ii'}^{kk'}$: general time-lags | C_{max}: makespan.

Table 6.1 *Features covered by literature on constraint programming for batch scheduling. In bold, we emphasized the features that are important in this contribution.*

Features			ZNH	NNH	EPQa	AMDa	OUR WORK
Year			2011	2016	2019	2022	2026
Journal			CCE	CCE	CCE	CCE	CCE
Mixing and splitting	**Network process**						x
	Sequential process		x	x	x	x	x
Storage policy	Unlimited intermediate storage		x	x			
	No intermediate storage		x				x
	Finite intermediate storage				x		
	Zero-wait		x				x
Set up features	Sequence independent		x	x			x
	Sequence dependent	Unit dependent			x	x	x
		Unit independent	x				x
Resources	Machines		x	x	x	x	x
	Renewable resources			x	x	x	
	Unrenewable resources						
Resource assignments	Fixed		x	x	x	x	x
	Flexible		x		x	x	x
Processing times	Fixed		x	x		x	x
	Variable				x	x	x
Additional constraints	Production sequence		x	x			
	Topology constraints		x	x	x		x
	Transfer times						
	Identical resources						x
	Maintenance						x
	Work shifts					x	x
	Release date		x		x	x	
	Due dates		x		x	x	
Objective function	Makespan		x	x		x	x
	Tardiness		x	x			
	Earliness		x				
	Cost			x	x		
Problem size	Medium				x		x
	Large		x	x		x	x
Campaign mode				x		x	
Decisions	Batching						
	Subbatching						
	Sequencing		x	x	x	x	x
	Machine assignment		x	x	x	x	x
	Timing		x	x	x	x	x
Solver	CP Optimizer		x	x	x	x	x
	OR-Tools						x
Number of instances			78	21	3	2	160[1]
Public repository							x

[1] More instances can be generated on request.

6.3.2. Constraint programming for batch scheduling

In this section, we review and summarize existing CP-based approaches for batch scheduling. (Zeballos, Novas, and Henning 2011) (ZNH) provided an extensive evaluation of a CP solution for short-term scheduling of a multi-product, multi-stage batch plant, featuring a rich set of industrial environment characteristics. They obtained promising computational results on a diverse set of instances for short-term horizons. Notably, they found that the CP Optimizer's default depth-first search performed poorly on scheduling problems. Since this paper dates back to 2011, we expect the results on scalability to be different in light of recent advances in CP (Laborie 2015; Lan and Berkhout 2025). (Novara, Novas, and Henning 2016) (NNH) presented a CP model for solving large-scale scheduling problems in multi-product, multi-stage batch plants. This model also incorporates several realistic problem features and considers a so-called campaign mode, commonly employed in batch manufacturing, in which multiple batches of the same product type are processed consecutively to reduce cleaning time. (Escobet Canal *et al.* 2019) (EPQa) investigated a hybrid constraint programming approach for the batch scheduling of a multi-product dairy process and demonstrated its performance across three scenarios. Additionally, (Awad *et al.* 2022) (AMDa) presented a constraint programming model for a real biopharmaceutical plant and tested it on two examples, successfully improving the makespan in both cases.

Despite these promising contributions, existing CP approaches for batch scheduling remain limited in both scope and experimental evaluation (see Table 6.1 for a comparison). Reviewing related work, we found only the works done by Zeballos, Novas, and Henning (2011) and (Novara, Novas, and Henning 2016) that include extensive experimental evaluations. Notably, none explicitly addressed the splitting of batches into subbatches. All these CP formulations are conceptually closer to a job shop model rather than to the MIP-based State Task Network (STN) or Resource Task Network (RTN) formulations typically used for network-based facilities (G. P. Georgiadis, Elekidis, and M. C. Georgiadis 2019a).

More broadly, in the field of CP for production scheduling, (Prata, Abreu, and Nagano 2024) highlighted the need for studies that integrate multiple realistic problem characteristics and constraints, such as sequence-dependent setup times, release dates, zero-wait conditions, and buffer-zero (machine blocking) constraints. Although highly standardized CP frameworks for scheduling have recently emerged (Lan and Berkhout 2025), they have not yet been applied to batch scheduling problems that involve these complex constraints.

6.4. Scheduling of a real biomanufacturing facility

The scheduling problem studied is based on a real enzyme production site. The factory can be described as a multi-purpose, multi-product facility, which is a production site where different bio-based products (enzymes) are produced according to unique recipes, and the routings through the plant are product-specific and flexible. Enzymes are widely used in the food industry. For a review of the fermentation processes involved in producing such products, we refer to (Al-Maqtari, Waleed, and Mahdi 2019). These enzymes are produced in fixed quantities, for which the term *batch* is used. Each batch

has a specific batch size and its own unique recipe, resulting in a distinct (and sometimes flexible) production process within the factory. Demand is typically specified for the next three months, which translates approximately to fifteen to twenty batches of a standard quantity per month. We assume that decisions regarding batch sizing are made in advance. The production of one batch requires a set of unit operations that are executed either subsequently or (partially) in parallel. Each unit operation requires one or multiple resources and has a specific processing time.

A schedule involves assigning resources to tasks, sequencing and timing them, and ensuring that the resulting schedule satisfies all predefined operational constraints. The facility we study consists of 13 resource groups, each with 1 to 11 machines, and capacities within a group can vary (for an overview, see Table 6.2). On this part of the manufacturing site, a variety of 168 unique recipes can be produced, stored, and later used for the production of finished goods.[2] Additional complexities of the system are:

- sequence-dependent cleaning times and pre- and post-processes,
- changing batch weights/sizes (e.g., due to filtering steps),
- flexible machine selection with size restrictions and forbidden assignments,
- batches can be split into subbatches,
- subbatches can be mixed again,
- maximal time-lags and zero-wait constraints.

Now, we discuss a few examples of batches and how their recipes translate into schedules (visualized as Gantt charts). Again, we refer to (Al-Maqtari, Waleed, and Mahdi 2019) for a more comprehensive review of fermentation processes for producing enzymes in the food industry.

The production process of one enzyme batch follows a predefined sequence of unit operations, as illustrated in Figure 6.1. In the factory, a combination of batch and continuous processes is employed. In our current formulation, however, we use batch sizes and processing rates up front to derive effective process times, and thus do not explicitly differentiate between batch and continuous processes in the model. This slight reduction in detail was not perceived as an issue when compared to a detailed simulation model. The process always begins with fermentation, followed by downstream processing tasks, of which harvesting is the first. Harvesting involves removing the fermented broth from the fermentation tank and separating the product.

After harvesting, various filtering steps begin, starting with the Filter A (or Filter B) operation, which requires access to dedicated Filter A (or Filter B) equipment. The recipe determines whether, after harvesting, Filter A or Filter B is required, which can vary per batch. The next steps are the Filter C and Filter D operations. In some cases, an additional stabilization step is performed, requiring the use of a stabilization tank to store active products before they are packed and shipped to customers.

[2] Some of these recipes differ only in minor aspects, such as whether a specific filter step is required or not.

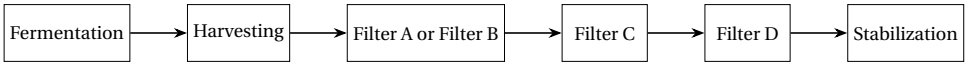


Figure 6.1 *Overview of the scheduling and execution workflow. We used generic labels for the different stages to avoid sharing sensitive factory data.*

Although the general structure is illustrated in Figure 6.1, the unit operations actually involve more substeps. For example, fermentation always involves a preparation and a post-fermentation step. Additionally, the harvesting step involves both a killing step for sterilizing the microorganisms used in fermentation and a broth preparation step, which, for some batches, co-occurs in one or multiple harvesting tanks. For some batches, the killing operation occurs in the fermenter rather than in the harvesting tank. A further distinction is made between harvesting in batch mode (where all the broth is harvested at once) and harvesting in fractions. For harvesting in fractions, harvesting tanks are required before fermentation is complete to initiate the fraction harvesting process. Harvesting in fractions always requires both harvesting tanks of type A and tanks of type B. The tanks of type A are first used to collect the first fractions, until a minimum amount is collected, making it possible to start transferring to the harvesting of type B. Batches with batch harvesting always require one or more harvesting tanks of type B, and only for some recipes, one or more harvesting tanks of type A during fermentation for additional processing steps, such as water pasteurization. This claiming of a harvesting tank of type A is also illustrated in Figure 6.2.

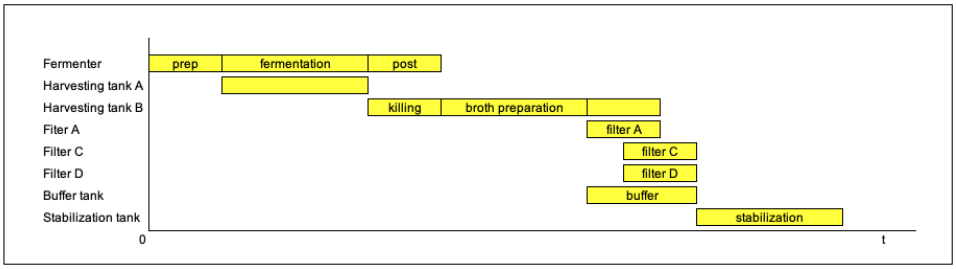


Figure 6.2 *Gantt chart of producing one enzyme batch (example recipe `enzyme 1`). The fermentation step first requires preparation and, afterward, a post-processing step. The harvesting step consists of a killing step and a broth preparation step, and takes place in harvesting tank B. Additionally, a type A harvesting tank is required during fermentation to pasteurize the water. The post-fermentation step can occur once fermentation is complete and can begin in parallel with the killing process. After the broth preparation, Filter A processes the enzyme. The harvesting tank B is still occupied during the Filter A step. After a fixed time-lag from the start of Filter A, Filter C&D can start (they are claimed in parallel because they are connected). From the start of Filter A till the end of Filter C&D, a buffer tank is claimed. The final step requires a stabilization tank.*

A simplified Gantt chart of such a process is shown in Figure 6.2. A key observation

from this visualization is that the allocation of tanks does not necessarily follow a strict finish-to-start dependency structure, as suggested in Figure 6.2. For instance, harvesting tanks must remain occupied during the execution of Filter A. This overlapping resource usage represents one of the key challenges in accurately modeling these production recipes, stemming from the continuous flow of intermediate products from tank to tank.

Additional complexity arises from batch mixing and splitting. At specific stages of the process, a batch may be divided into multiple subbatches that are later recombined to form the final product. This behavior creates significant difficulty for both modeling and scheduling enzyme production processes. Importantly, we assume that decisions on how the batch is split are made in advance (by the product recipe) and are thus not determined by the CP model.

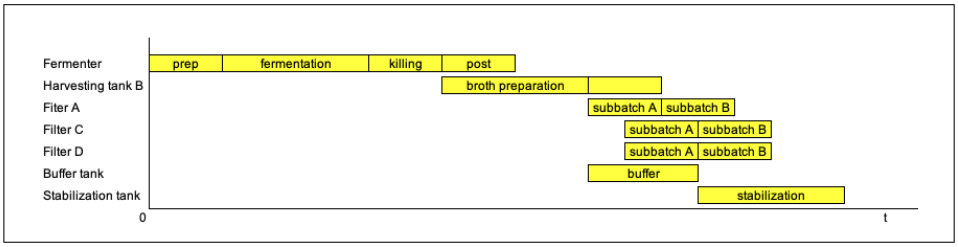


Figure 6.3 *A Gantt chart of the example recipe* `enzyme 2`*, with splitting at Filter C.*

We illustrate this phenomenon with two new enzyme examples in which mixing and splitting occur. In the first example, shown in Figure 6.3 (which visualizes `enzyme 2`), the batch is divided into two subbatches at the Filter A step. These subbatches continue through the process independently until Filter C&D, where they subsequently recombine in the stabilization tank.

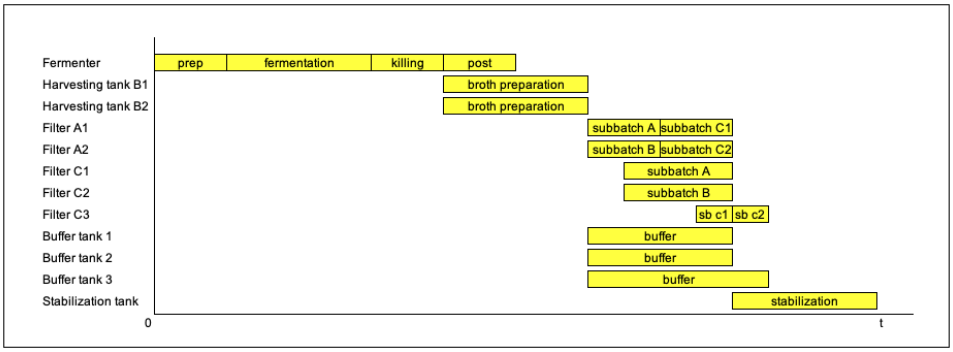


Figure 6.4 *A Gantt chart of the example recipe* `enzyme 3`*, with splitting at both harvesting and Filter C.*

A more complex recipe is illustrated in Figure 6.4, which visualizes `enzyme 3`. In this case, the process already involves a split at the harvesting stage, with the batch divided

between two harvesting tanks. At the filtering stage, however, the number of subbatches increases to three. Each harvesting tank connects to one Filter A unit. Consequently, at the Filter C stage (where processing occurs in three subbatches), subbatch C receives part of its material from Filter A1 and part from Filter A2.

6.5. From factory to constraint programming model

Recent studies have shown that Constraint Programming (CP) outperforms Mixed-Integer Linear Programming (MILP) on a set of benchmark scheduling problems involving complex task sequencing, such as job-shop scheduling (Naderi, Ruiz, and Roshanaei 2023). MILP is a mathematical optimization in which a set of linear constraints restricts a mix of integer and non-integer variables. Modern MILP solvers are typically based on the famous branch-and-bound algorithm (Land and Doig 1960), and use linear relaxations to guide the search and provide optimality guarantees. CP models consist of a set of variables with discrete domains and a set of (non-linear) constraints. CP solvers combine constraint propagation techniques and search techniques to find a feasible assignment of the variables (Rossi, Van Beek, and Walsh 2006). CP-based scheduling typically utilizes integer variables and global constraints, including disjunctive (Vilím 2007) and cumulative (Baptiste, Le Pape, and Nuijten 1999) constraints. Modern CP, such as CP Optimizer (IBM 2024), provides a natural way to model scheduling constraints using interval variables (Laborie 2015). An interval variable is a decision variable that includes a *start* and an *end* with a *length* equal to *end*−*start*. The interval can be fixed in advance or variable, and the variable interval can be optional, meaning the model can decide whether to include it. Sequence variables are also unique to modern CP (Laborie, Rogerie, *et al.* 2018). They are defined over a set of interval variables, and their value is a permutation of the present intervals in that sequence variable (Laborie, Rogerie, *et al.* 2018; Naderi, Ruiz, and Roshanaei 2023).

We use the CP Optimizer (IBM 2024) syntax to formulate our model, as it is the solver used in our experiments. While CP Optimizer offers numerous constructs, we focus here on those specifically used in our model. Functions such as StartOf, EndOf, and sizeOf access the start, end, and size of interval variables. The function presenceOf evaluates to one if the interval is present and zero otherwise. A crucial construct is the constraint NoOverlap, which prevents the simultaneous execution of tasks on the same resource. Previous is a constraint applicable to sequence variables and two task variables, enforcing that one task must appear at an earlier index than the other within that sequence.

CP Optimizer uses various algorithms to solve the scheduling constraints (Laborie, Rogerie, *et al.* 2018). The typical approach for handling resources and their corresponding constructs is the time-table filtering algorithm, for which IBM has its own implementation (Laborie, Rogerie, *et al.* 2018), but which is related to what is presented in (Gay, Hartert, and Schaus 2015). If that algorithm fails, the CP optimizer resorts to the algorithms presented in (Vilím 2007) for disjunctive resources and to (Vilím 2011) for cumulative resources. Furthermore, CP Optimizer uses a combination of large neighborhood search (Shaw 1998), failure-directed search (Vilím, Laborie, and Shaw 2015), simple temporal networks (Dechter, Meiri, and Pearl 1991), and temporal relaxations (Laborie and Rogerie 2016), as extensively described in (Laborie, Rogerie, *et al.* 2018).

Standardized frameworks such as PyJobShop (Lan and Berkhout 2025) provide a first step toward bridging the gap between research developments and practical implementation, offering a modular CP-based scheduling framework that leverages Google's OR-Tools and CP Optimizer as backend solvers. However, how effective CP is for solving realistic large-scale batch scheduling problems with a combination of realistic biomanufacturing constraints remains an open question. Thus, in this section, we present an extended version of the CP model provided by Lan and Berkhout (2025) to model a real biomanufacturing facility. We provide example instances in Section 6.5.2 to demonstrate how the abstract CP model can be translated into a concrete model using the recipe parameters of the example enzymes presented in Section 6.4.

The OR-Tools variant (Lan and Berkhout 2025) of the model is included in Appendix 6.B. For a more general overview on scheduling with constraint programming, we suggest (Baptiste, Le Pape, and Nuijten 2001).

6.5.1. A constraint programming model

In this section, we explain the constraint programming model used to model the scheduling operations in the real factory. We relate the more abstract concepts used in the model to the real processes and concepts in the factory.

The model is defined at an abstract level consisting of jobs, resources, tasks, and modes. Each job $j \in J$ represents one batch with a standard quantity of a specific enzyme. The tasks represent the different unit operations required to produce one batch; thus, the recipe of that batch determines which tasks are needed. In other words, a batch recipe translates to a job and a set of tasks in the model. For most batches, the tasks can be executed in different operating modes (M): i.e., it is a decision which resources (R) will execute the task, and in some cases, the duration can also be decided or depend on the selected resources.

Each job $j \in J$ has a set of related tasks $T_j \subseteq T$ required for producing that batch. We use these tasks to model core operations such as fermentation, as well as preparatory or post-processing steps. Additionally, we introduce auxiliary tasks to model the occupation of shared resources during the execution of other operations, such as the buffer tanks that must be occupied starting from the start of the Filter A operation until the end of the Filter C operation. For some tasks, the duration is fixed; for others, it is variable.

For specific tasks, multiple resource-processing time configurations may be possible; for example, either a single larger tank or two smaller tanks can be selected (see also Example 2). Another example: consider a scenario where the processing time for some filtering steps varies depending on the filter selected, because different filters have different filtering rates or filtering surfaces. We provide these alternative resource options to the constraint programming model, using the mode logic. Each task $t \in T$ can be executed in one of several modes $m \in M_t$, where each mode defines a specific processing configuration characterized by a duration p_m, a set of required resources $R_m \subseteq R$, and corresponding resource demands q_{mr}. Exactly one mode must be selected per task. We also use these modes to model constraints for the various tanks in the factory, which have specific sizes and capacity limitations. These constraints apply when a batch with a specific batch size requires one or multiple tanks. Example 2 illustrates the use of modes for different tank combinations with different sizes and capacity constraints.

Example 2. *Consider a batch with a harvesting weight of 100.000. Suppose we have four harvesting tanks in the factory: tank A with a capacity of 75,000 liters, tank B with a capacity of 75,000 liters, tank C with a capacity of 90,000 liters, and tank D with a capacity of 150,000 liters. The possible modes for this batch are the following combinations: A+B, D, A+C, and B+C. Importantly, we do not consider A+D as a possible combination here, as tank A would be redundant. Harvesting processing time is not affected by the selected tanks and remains constant across all modes.*

Decision Variables

The model is formalized through the following decision variables and constraints.

- ϕ_j: interval variable for each job $j \in J$
- τ_t: interval variable for each task $t \in T$
- μ_m: interval variable for each mode $m \in M$ with $\mu_{duration} = p_m$
- S_r: sequence variable for each machine $r \in R^{\text{machine}}$ over the interval variables μ_m corresponding to modes $m \in M$ that require machine r (i.e., $r \in R_m$)

Objective

We consider planning horizons of one to three months. For these horizons, we use the makespan as the objective. The end products of this facility are stored in stock for further downstream processing, so strict customer due dates are less relevant here. Schedulers therefore aim to minimize makespan, i.e., to finish all demand within the given horizon. The makespan objective can also be seen as a proxy for efficient resource utilization. Formally, we define:

$$\max_{t \in T} \tau_t^{\text{end}} \tag{6.2}$$

$$\text{StartOf}(\phi_j) = \min_{t \in T_j} \text{StartOf}(\tau_t) \qquad \forall j \in J \tag{1a}$$

$$\text{EndOf}(\phi_j) = \max_{t \in T_j} \text{EndOf}(\tau_j) \qquad \forall j \in J \tag{1b}$$

$$\sum_{m \in M_t} \text{presenceOf}(\mu_m) = 1 \qquad \forall t \in T \tag{1c}$$

$$\text{StartOf}(\mu_m) = \text{StartOf}(\tau_t) \qquad \forall t \in T, m \in M_t \tag{1d}$$

$$\text{EndOf}(\mu_m) = \text{EndOf}(\tau_t) \qquad \forall t \in T, m \in M_t \tag{1e}$$

$$\text{presenceOf}(\mu_m) \Rightarrow \text{lengthOf}(\mu_m) = \text{lengthOf}(\tau_t) \qquad \forall t \in T, m \in M_t \tag{1f}$$

$$\text{NoOverlap}(S_r, \text{Setup}_r) \qquad \forall r \in R^{\text{machine}} \tag{1g}$$

$$\text{StartOf}(\tau_i) + l \leq \text{StartOf}(\tau_k) \qquad \forall (i,k,l) \in C^{\text{StartBeforeStart}} \tag{1h}$$

$$\text{StartOf}(\tau_i) + l \leq \text{EndOf}(\tau_k) \qquad \forall (i,k,l) \in C^{\text{StartBeforeEnd}} \tag{1i}$$

$$\text{EndOf}(\tau_i) + l \leq \text{StartOf}(\tau_k) \qquad \forall (i,k,l) \in C^{\text{EndBeforeStart}} \tag{1j}$$

$$\text{EndOf}(\tau_i) + l \leq \text{EndOf}(\tau_k) \qquad \forall (i,k,l) \in C^{\text{EndBeforeEnd}} \tag{1k}$$

$$\text{presenceOf}(\mu_{m_i}) \leq \sum_{\substack{m_k \in M_k \\ R_{m_k} = R_{m_i}}} \text{presenceOf}(\mu_{m_k}) \qquad \forall (i,k) \in C^{\text{IdenticalResources}}, m_i \in M_i \tag{1l}$$

$$\text{presenceOf}(\mu_{m_i}) \leq \sum_{\substack{m_k \in M_k \\ R_{m_k} \cap R_{m_i} = \emptyset}} \text{presenceOf}(\mu_{m_k}) \qquad \forall (i,k) \in C^{\text{DifferentResources}}, m_i \in M_i \tag{1m}$$

$$\text{presenceOf}(\mu_u) \wedge \text{presenceOf}(\mu_v) \Rightarrow \text{Previous}(S_r, \mu_u, \mu_v) \qquad \forall (i,k) \in \mathscr{C}^{\text{Consecutive}}, r \in \mathscr{R}_{ik}^{\text{machine}}, u \in \mathscr{M}_i \cap \mathscr{M}_r^R, v \in \mathscr{M}_k \cap \mathscr{M}_r^R. \tag{1n}$$

Constraints (1a)-(1o) provide an overview of all relevant constraints used to model the manufacturing site using the CP optimizer syntax. In the following, we describe all constraints:

(1a) These constraints connect the start of the job intervals to the earliest start of the related task intervals. The job intervals refer to the different batches.

(1b) These constraints connect the end of the job intervals to the latest end of the related task intervals.

(1c) These constraints ensure that exactly one mode is selected for each task. So, suppose that a filtering task can either be executed with Filter A1 in 2 hours or with Filter A2 in 3 hours. This constraint ensures that only one of the two options is selected and embedded in the schedule.

(1d) These constraints connect the start of the mode intervals to the start of the related task intervals.

(1e) These constraints connect the start of the mode intervals to the start of the related task intervals.

(1f) These constraints connect the length of the selected mode interval to the length of the task interval.

(1g) These constraints model that mode interval variables that are using the same resource are not overlapping each other. This constraint also handles sequence-dependent setup times. These setup times occur, for example, between certain enzyme combinations that use the same Filter A after each other. They also apply to the required spacing between the post-operation of a preceding batch and the prep operation of a succeeding batch scheduled on the same fermenter; there must always be a fixed amount of slack between them. If $(t_u, t_v, r, l) \in C^{SetupTime}$, then the setup time $s_{tu,tv,r}$ is given by l, otherwise it is zero. We define $SetUp_r$ as the matrix of setup times that contains all setup times between intervals for each machine $r \in R^{\text{machine}}$ over the interval variables μ_m corresponding to modes $m \in M$ that require machine r (i.e., $r \in R_m$).

(1h) - (1k) These constraints address all the various temporal constraints that may arise. For example, looking at fermentation: the zero-wait between the preparation step and the main fermentation results in a constraint of the form (1j) with $l = 0$, and with i referring to the preparation and k to the fermentation itself, and a constraint of the form (1i) with $l = 0$ and i and k reversed, such that a distance of 0 is enforced between the finish of the preparation and the fermentation.

(1l) These constraints ensure that specific tasks are assigned to the same resource, i.e., they must use the same machine or resource mode. We use this constraint to model auxiliary equipment (such as buffer tanks) and to enforce the correct logic for claiming and releasing shared equipment in the facility. For example, the fermentation process involves four tasks that all require the same fermenter: the preparation step, the main fermentation, the post-processing step, and an auxiliary task representing fermenter occupation during harvesting. To reflect this, all four tasks must be assigned to the same fermenter.

(1m) These constraints ensure that specific tasks are **not** assigned to the same resource set.

(1n) This constraint does not allow a task to be scheduled in between two tasks if they are scheduled in sequence on mode m_i. This constraint applies in some scenarios to subbatches that belong to the same batch and that are scheduled sequentially on, for example, a Filter A. In this case, it is not allowed to assign Filter A to another batch in between.

The constraint programming model above uses the formulation presented by Lan and Berkhout (2025), which enables a flexible approach to combine various scheduling

models from the literature. We extended the PyJobShop framework with a new constraint necessary for our factory (and relevant to many other biomanufacturing applications), namely mode dependencies (1o). The mode dependencies are relevant for topology constraints, and they dictate that if a specific mode m is selected, then also one out of a set of modes ($\mathcal{M}_m^{dependent}$) must be selected, which results in the following formalization:

$$\text{presenceOf}(\mu_m) \leq \sum_{n \in \mathcal{M}_m^{dependent}} \text{presenceOf}(\mu_n) \quad \forall (m, \mathcal{M}_m^{dependent}) \in C^{\text{ModeDependencies}} \tag{1o}$$

Output

The output of the CP model is, for each task, a set of allocated resources, a start time, a finish time, and, next to that, the objective value, which is the finish time of the latest task (the makespan). This output can be visualized using a Gantt chart, as shown in Figures 6.2 to 6.4.

6.5.2. Model instantiation

In this section, we illustrate how to translate the example recipes from Section 6.4 into parameters for the constraint programming models. Table 6.2 lists the available factory resources.

Table 6.2 *Machine groups. Each group consists of multiple unique machines that differ in characteristics such as capacity or size. Additional constraints define which specific resources within a group can be assigned to each batch, as our scheduling problem corresponds to a flexible job shop setting in Graham's notation.*

Machine group	Number of machines
Fermenter 1 - 5	5
Harvesting tanks A	4
Harvesting tanks B	8
Filters A	3
Filters B	1
Buffer tanks	6
Filters C	5
Filters D	6
Stabilization tanks	11
Operator	1

Example `enzyme 1`: Single batch without splitting

As a first example, we consider the scheduling of a single batch. We provide an example model instantiation that can generate a schedule similar to the Gantt chart shown in

Figure 6.5, i.e., a batch that is not split into subbatches (as described in more detail in Section 6.4 and Figure 6.2).

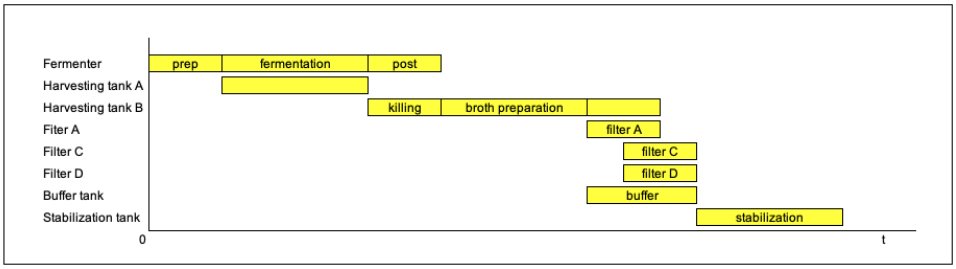


Figure 6.5 *Gantt chart example recipe `enzyme 1`.*

Table 6.3 summarizes all tasks for this batch, including their durations and resource requirements. In most cases, these resource requirements are flexible: the model must select from a set of alternative resources and assign one or more to each task.

Table 6.3 *Example `enzyme 1`: tasks related to the production of a simple batch. All tasks related to fermentation share the same resource (Fermenter 1). We provide a visualization in Figure 6.2.*

Stage	Task	Description	Dur. (h)	Resource
Fermentation	PrepFermentation	Preparation of the fermenter	2	Fermenter-1 & Operator
Fermentation	Fermentation	Main fermentation process	4	Fermenter-1
Fermentation	PostFermentation	Post-operations on the fermenter	2	Fermenter-1
Harvesting	Killing	Killing of the broth	2	Harvesting tank B
Harvesting	Preparation	Preparation of the broth	4	Harvesting tank B
Storage policy	Helper	Occupies harvesting tank B	Var.	Harvesting tank B
Downstream processing	Filter A	First filtering step	4	Filter A
Downstream processing	Filter C	First filtering step	4	Filter C
Downstream processing	Buffer	Buffer tank during DSP	Var.	Filter C
Stabilization	Stabilization	Final stabilization step	4	Stabilization tank

The temporal constraints are listed in Table 6.4 and visualized in Figure 6.6.

Table 6.4 *Example `enzyme 1`: Temporal constraints applied between tasks and their correspondence to model constraints.*

Task 1	Task 2	Constraint type	Delay
PrepFermentation	Fermentation	zero-wait	0
Fermentation	PostFermentation	zero-wait	0
Fermentation	Killing	zero-wait	0
Killing	Preparation	zero-wait	0
Preparation	Helper	zero-wait	0
Preparation	Filter A	zero-wait	0
Filter A	Helper	EndBeforeEnd	0
Filter A	Filter C	StartBeforeStart Min-lag & Max-lag	[2, 4]
Buffer	Filter A	StartBeforeStart	0
Filter C	Buffer	EndBeforeEnd	0
Buffer	Stabilization	zero-wait	0

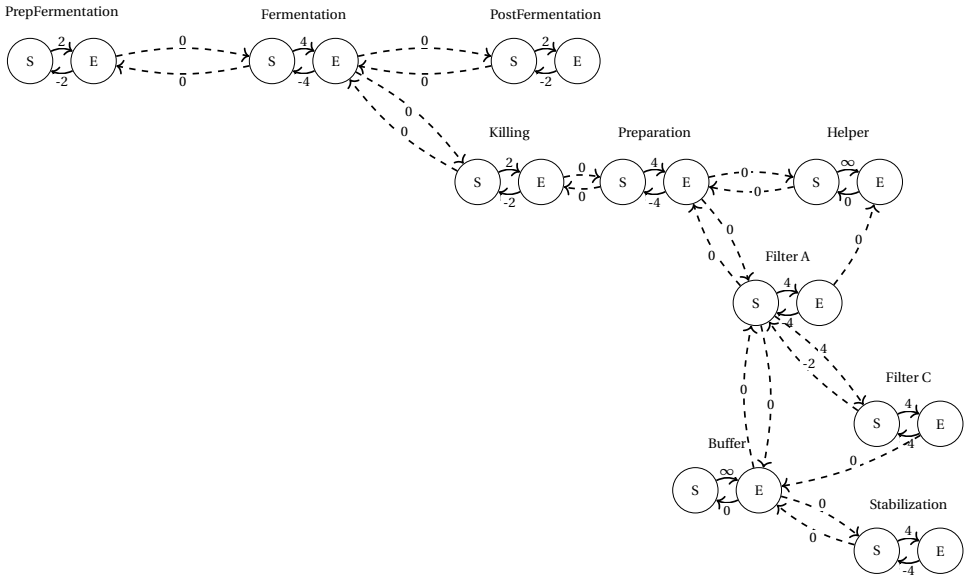


Figure 6.6 *Network representation of the precedence relations and durations. S indicates the start of a task and E the end of a task. $A \xrightarrow{y} B$ indicates that $B - A \leq y$ and the edge $A \xleftarrow{-x} B$ indicates $x \leq B - A$. Thus, the task PrepFermentation with duration 2 results in two arrows $S \xrightarrow{2} E$ and $S \xleftarrow{-2} E$. The two tasks with a variable duration are bounded by 0 and ∞. A zero-wait constraint between PrepFermentation and Fermentation results in $E_{Prep} \xrightarrow{0} S_{Ferm}$ and $E_{Prep} \xleftarrow{0} S_{Ferm}$. Then, an EndBeforeEnd constraint between Filter C and Buffer results in $E_{FilterC} \xrightarrow{0} E_{Buffer}$ and a StartBeforeStart constraint between Buffer and Filter A results in $S_{Buffer} \xrightarrow{0} S_{FilterA}$.*

Finally, specific tasks must be assigned to the same resource, such as the three fermentation steps, the harvesting steps, and the buffer tasks. The factory's storage policy is represented by both the zero-wait constraints and the buffer tasks. These buffer tasks are included to ensure that resources remain occupied beyond the actual process duration, in accordance with the storage policy. These additional constraints are listed in Table 6.4.

Table 6.5 *Example `enzyme 1`: Additional constraints applied between tasks and their correspondence to model constraints.*

Task 1	**Task 2**	**Constraint Type**
PrepFermentation	Fermentation	Identical Resource
Fermentation	PostFermentation	Identical Resource
Killing	Preparation	Identical Resource
Preparation	Helper	Identical Resource

Example enzyme 3: Splitting into subbatches

Next, we provide the parameters of the example of enzyme 3, an example that includes splitting into subbatches, as illustrated in Figure 6.7 (which processes are described in more detail in Section 6.4 and Figure 6.4). We skip enzyme 2, because enzyme 3 is an extension of it. For this example, Table 6.6 summarizes the set of tasks and their associated resource requirements.

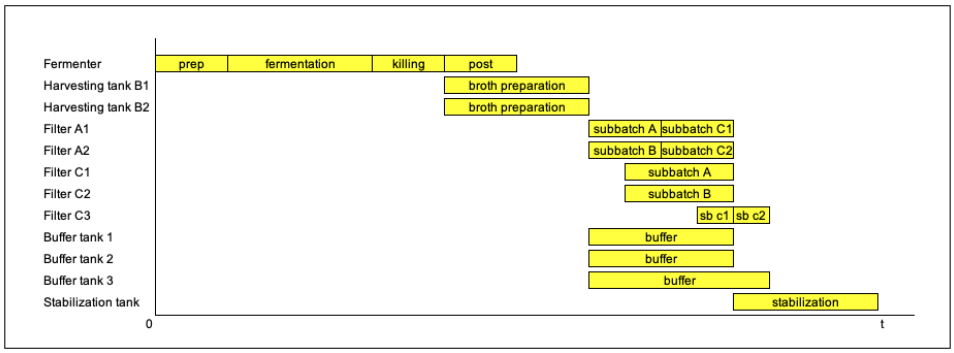


Figure 6.7 *A Gantt chart of the example recipe enzyme 3, with splitting at both harvesting and Filter C.*

Table 6.6 *Example enzyme 3: tasks representing a batch with splitting into subbatches. We provide a visualization in Figure 6.3. d refers to the duration in hours. Op. refers to the operator.*

Stage	Task	Description	*d*	Resource
Fermentation	PrepFermentation	Preparation of the fermenter	2	Fermenter-1 & Op.
Fermentation	Fermentation	Main fermentation process	4	Fermenter-1
Fermentation	PostFermentation	Post-operations on the fermenter	2	Fermenter-1
Harvesting	Killing group 1	Killing of the broth	2	Harvesting tank B
Harvesting	Preparation group 1	Preparation of the broth	4	Harvesting tank B
Harvesting	Killing group 2	Killing of the broth	2	Harvesting tank B
Harvesting	Preparation group 2	Preparation of the broth	4	Harvesting tank B
Storage policy	Helper group 1	Occupies harvesting tank B	Var.	Harvesting tank B
Storage policy	Helper group 2	Occupies harvesting tank B	Var.	Harvesting tank B
Downstream processing	Filter A subbatch A	First filtering step	4	Filter A
Downstream processing	Filter C subbatch A	First filtering step	4	Filter C
Downstream processing	Buffer subbatch A	Buffer tank	Var.	Filter C
Downstream processing	Filter A subbatch B	First filtering step	4	Filter A
Downstream processing	Filter C subbatch B	First filtering step	4	Filter C
Downstream processing	Buffer subbatch B	Buffer tank	Var.	Filter C
Downstream processing	Filter A subbatch C1	First filtering step	4	Filter A
Downstream processing	Filter C subbatch C1	Second filtering step	2	Filter C
Downstream processing	Buffer subbatch C1	Buffer tank	Var.	Filter C
Downstream processing	Filter A subbatch C2	First filtering step	4	Filter A
Downstream processing	Filter C subbatch C2	Second filtering step	2	Filter C
Downstream processing	Buffer subbatch C2	Buffer tank	Var.	Filter C
Stabilization	Stabilization	Final stabilization step	6	Stabilization tank

The temporal and additional resource constraints are given in Table 6.7 and Table 6.8. In this configuration, the process is divided into two harvesting groups and three downstream subbatches. The model must therefore include additional tasks and constraints linking these subbatches. Each harvesting tank B connects to a specific Filter A, and the subbatches originating from the same harvesting tank are processed sequentially on that filter. Between two consecutive subbatches, no other task may be scheduled, which is enforced using the consecutive constraint (1n).

Table 6.7 *Example `enzyme 3`: Temporal constraints applied between tasks and their correspondence to model constraints of example Figure 6.4.*

Task 1	**Task 2**	**Constraint type**	**Delay**
PrepFermentation	Fermentation	zero-wait	0
Fermentation	PostFermentation	zero-wait	0
Fermentation	Killing	zero-wait	0
Killing	Preparation group 1	zero-wait	0
Killing	Preparation group 2	zero-wait	0
Preparation group 1	Helper group 1	zero-wait	0
Preparation group 2	Helper group 2	zero-wait	0
Preparation group 1	Filter A subbatch A	zero-wait	0
Preparation group 2	Filter A subbatch B	zero-wait	0
Preparation group 1	Filter A subbatch C1	EndBeforeStart	0
Preparation group 2	Filter A subbatch C2	EndBeforeStart	0
Filter A subbatch A	Helper group 1	EndBeforeEnd	0
Filter A subbatch A	Filter C subbatch A	StartBeforeStart Min-lag & Max-lag	[2, 4]
Buffer subbatch A	Filter A subbatch A	StartBeforeStart	0
Filter A subbatch A	Buffer subbatch A	EndBeforeEnd	0
Filter A subbatch B	Helper group 2	EndBeforeEnd	0
Filter A subbatch B	Filter C subbatch B	StartBeforeStart Min-lag & Max-lag	[2, 4]
Buffer subbatch B	Filter A subbatch B	StartBeforeStart	0
Filter A subbatch B	Buffer subbatch B	EndBeforeEnd	0
Filter A subbatch C1	Helper group 1	EndBeforeEnd	0
Filter A subbatch C1	Filter C subbatch C1	StartBeforeStart Min-lag & Max-lag	[2, 4]
Buffer subbatch C1	Filter A subbatch C1	StartBeforeStart	0
Filter A subbatch C1	Buffer subbatch C1	EndBeforeEnd	0
Filter A subbatch C2	Helper group 2	EndBeforeEnd	0
Filter A subbatch C2	Filter C subbatch C2	StartBeforeStart Min-lag & Max-lag	[2, 4]
Buffer subbatch C2	Filter A subbatch C2	StartBeforeStart	0
Filter A subbatch C2	Buffer subbatch C2	EndBeforeEnd	0
Filter C subbatch A	Filter A subbatch C1	StartBeforeStart	0
Filter C subbatch B	Filter A subbatch C2	StartBeforeStart	0
Filter C subbatch C1	Filter A subbatch C2	zero-wait	0

6.5.3. Additional examples of constraints

Each enzyme recipe results in a distinct model instantiation, reflecting its specific process characteristics. These differences may involve not only processing times but also

Table 6.8 *Example enzyme 3: Additional constraints applied between tasks and their correspondence to model constraints of example Figure 6.4.*

Task 1	Task 2	Constraint Type
PrepFermentation	Fermentation	Identical Resource
Fermentation	PostFermentation	Identical Resource
Killing	Preparation	Identical Resource
Preparation	Helper	Identical Resource
Filter A subbatch A	Filter A subbatch C1	Identical Resource
Filter A subbatch B	Filter A subbatch C2	Identical Resource
Filter B subbatch C1	Filter B subbatch C2	Identical Resource
Filter A subbatch A	Filter A subbatch C2	Different resource
Filter A subbatch B	Filter A subbatch C1	Different resource
Filter A subbatch C1	Filter A subbatch C2	Different resource
Filter A subbatch A	Filter A subbatch C1	Consecutive
Filter A subbatch B	Filter A subbatch C2	Consecutive
Filter C subbatch C1	Filter C subbatch C2	Consecutive

structural variations in how tasks and resources interact.

For instance, in some enzyme processes, the Filter A step can be executed in parallel, allowing two Filter A units to operate simultaneously and thereby reducing the overall processing time. In such cases, two buffer tanks must also be connected to the filters, creating a mode dependency between the Filter A and buffer steps: if two filters are selected, two buffer tanks must likewise be used.

In other cases, the killing operation takes place directly in the fermenter rather than in the harvesting tanks. Some fermentations also occur in fractions, where parts of the batch proceed to harvesting and downstream processing earlier than others. In these situations, the fermentation tasks in the model remain the same, but additional harvesting tanks (type A) are required during fermentation. Specific enzymes additionally require a dedicated harvesting tank A during fermentation for handling water removal.

When scheduling multiple batches in parallel, additional inter-batch constraints apply. In particular, consecutive batches using the same fermenter must be separated by a minimum spread of three hours, represented as a sequence-dependent setup time between the post-fermentation of the preceding batch and the preparation step of the subsequent batch. Moreover, a contamination matrix specifies which product combinations require sequence-dependent setup times on the filter machines Filter A, Filter B, and Filter C.

A typical time window (e.g., 1–3 months) contains 20–60 jobs/batches, each following a different recipe. The combination of all constraints in such a period constitutes one whole problem instance. All data files and corresponding model instantiations (using the PyJobShop (Lan and Berkhout 2025) interface) are publicly available in our repository at `github.com/kimvandenhouten/pyjobshop_biomanufacturing`.

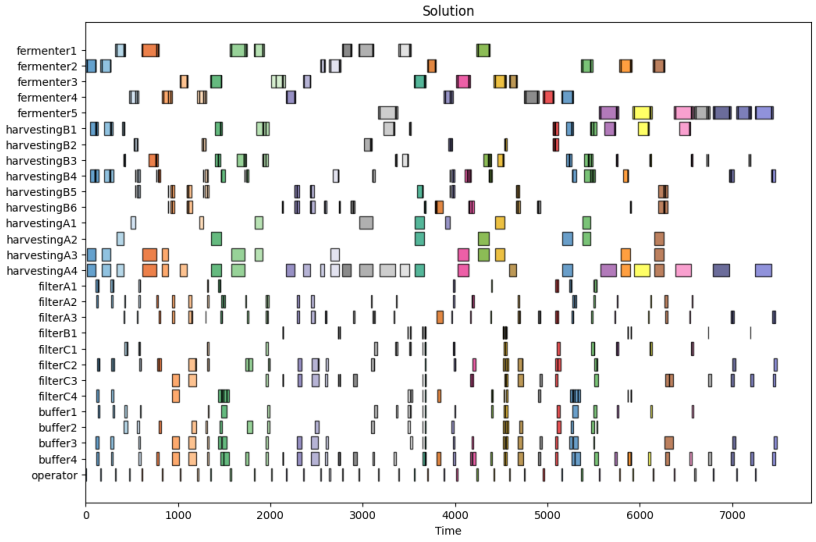


Figure 6.8 *Example of a schedule generated by the warm-start approach, where batches are executed sequentially without overlap (time in hours).*

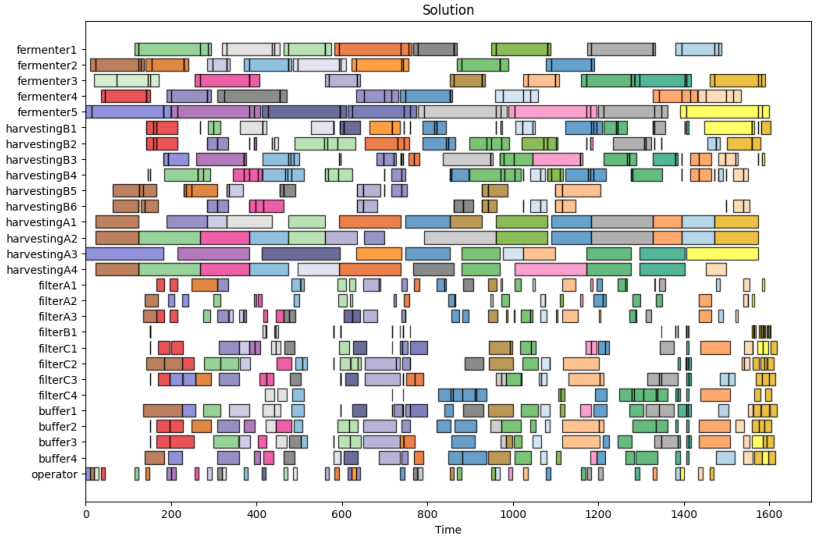


Figure 6.9 *Example of a schedule after additional computation time, showing improved overlap and reduced makespan (time in hours). Note the significantly shorter time (horizontal) axis compared to Figure 6.8.*

6.6. Warm-start strategy

The presence of no-wait constraints renders the feasibility problem of the resource-constrained scheduling problem NP-hard (Bartusch, Möhring, and Radermacher 1988), making it computationally challenging to directly construct a feasible schedule for multiple batches. However, based on domain knowledge, it is known that at least one recipe can be executed feasibly in isolation.

We exploit this insight by designing a warm-start strategy that incrementally builds a feasible solution. Specifically, we first solve the scheduling problem for a single product in an otherwise empty factory. Subsequently, additional products are introduced, each scheduled only after the completion of the previous one. This effectively decomposes the problem into a series of smaller, tractable subproblems.

As a result, the warm-start procedure produces a schedule in which batches do not overlap, leading to a sequence of enzyme productions with a relatively large makespan. While this solution is suboptimal in terms of makespan, it provides a feasible starting point for further optimization. An example of such a schedule is shown in Figure 6.8.

Starting from this initial solution, the CP solver can further improve the schedule by introducing batch overlaps where possible. An example of an improved schedule after additional computation time is shown in Figure 6.9.

6.7. Evaluation

The primary research question we aim to address with this evaluation is: What is the performance (solution quality and computation time) of state-of-the-art constraint programming solvers when applied to realistic biomanufacturing scheduling problems that incorporate multiple interacting constraints, including zero-wait storage policies, sequence-dependent and independent setup times, flexible machines, complex unit-topology restrictions, and non-trivial precedence constraints and overlapping resource usage coming from mixing and splitting?

6.7.1. Experimental set-up

First, we outline the experimental settings used for our evaluation. We utilize a set of benchmark instances explicitly generated for this purpose. The benchmark instances randomly combine batches from the list of recipes that can be produced in the real factory. Thus, the instances differ in terms of the number of batches and the product mix. For an overview of the number of jobs, tasks, and constraints per instance, see Appendix 6.C.

For the experiments, we utilize the DelftBlue cluster (Delft High Performance Computing Centre (DHPC) 2024). We run each instance on an Intel Xeon E5-6248R 24C 3.0 GHz processor, with eight cores allocated and 4GB of RAM per core. We use the OR-Tools CP-SAT solver (v9.15.6755) and the CP Optimizer (version 22.1.1.0) with default hyperparameters, configured to use 8 cores.

6.7.2. Results

In this section, we present the results of our experiments, which aim to investigate the performance of the CP approach for realistic biomanufacturing scheduling problems. We

first compare two state-of-the-art CP solvers (CP-Optimizer and OR-Tools) on our benchmark instances and show that CP-Optimizer consistently performs better. Additionally, we demonstrate that a warm-start strategy can substantially improve its performance. We then investigate the scalability of the CP approach and quantify how solution quality and computation time evolve as instance size grows. Finally, we examine the zero-wait constraints and find that relaxing them significantly improves solver performance, yielding lower optimality gaps and shorter computation times.

Initial results: warm-start

First, we investigated whether we could improve the CP solver's performance using the warm-start strategy described in Section 6.6. We evaluated the warm-start strategy on 30 problem instances of sizes 20, 40, 60 (# batches). We set a time limit of 600 seconds (excluding the warm start, which we argue could have been performed offline) and tested the warm-start strategy with both the CP Optimizer solver and the OR-Tools optimizer. Table 6.9 shows that applying a *warm-start strategy* substantially improves the scalability of the CP model, yielding many more feasible solutions.

A Wilcoxon (following (Cureton 1967)) confirmed that the solver combined with a warm-start consistently outperforms the solver without warm-start. For the Wilcoxon test, we used $\alpha = 0.05$, the Pratt procedure for handling ties, and a continuity correction because the objectives are discrete. The test was executed using the Python package SciPy (Virtanen *et al.* 2020) built-in method $scipy.stats.wilcoxon$ with parameters "$zero\_method$="pratt", $method$="approx" and $correction$=True. For CP Optimizer, the results show that warm-starting leads to a statistically significant improvement ($Z = -4.710$, $p = 2.475e-06$). Similarly, for OR-Tools, a significant improvement is observed ($Z = -4.639$, $p = 3.494e-06$).

Table 6.9 *Number (#) of feasible solutions found with and without warm-start.*

Solver	# Orders	# Without warm-start	# With warm-start
CPOptimizer	20	4	10
CPOptimizer	40	0	10
CPOptimizer	60	0	10
OR-Tools	20	5	10
OR-Tools	40	0	10
OR-Tools	60	0	10

Initial results: solver comparison

Second, we evaluated the performance difference between the CP Optimizer and the OR-Tools solver with warm-start enabled. We evaluated 45 test instances with varying problem sizes (i.e., number of batches). Each instance was solved under varying time limits, resulting in 135 experiments per solver.

For the smaller instances, there was little performance difference between the two solvers. However, as the problem size increases, CP Optimizer performs better in terms of the makespan obtained under a fixed time limit. We observed that, for a subset of

instances, the OR-Tools solver exceeded the memory limit we set. In Table 6.10, we summarize the number of instances where each solver achieves the best solution. In the appendix, we included the results per instance (see Appendix 6.D).

Table 6.10 *Solver comparison per time limit and problem size.*

Size	Metric	600s	3600s	10800s
10	CPOptimizer wins	0	0	0
	Ties	5	6	8
	OR-Tools wins	4	3	1
20	CPOptimizer wins	3	6	6
	Ties	0	0	0
	OR-Tools wins	6	3	3
40	CPOptimizer wins	9	9	9
	Ties	0	0	0
	OR-Tools wins	0	0	0
60	CPOptimizer wins	9	9	9
	Ties	0	0	0
	OR-Tools wins	0	0	0
120	CPOptimizer wins	9	9	9
	Ties	0	0	0
	OR-Tools wins	0	0	0

We applied a Wilcoxon matched-pairs rank-sum test (as described by Cureton (1967)) to the optimality gaps obtained to compare the different solvers. The test ranks absolute differences and provides insight into whether one solver consistently outperforms the other. The results indicate that CP Optimizer performs significantly better than OR-Tools ($Z = -8.511$, $p = 1.725e\text{-}17$). Based on these initial results, we adopt the warm-start strategy and CP Optimizer in all subsequent experiments.

What is the performance of the CP approach?

We evaluate the performance of the CP approach on test instances with 20, 40, and 60 batches, corresponding to approximately 1, 2, and 3 months of production, respectively. These planning horizons align with typical planning practice, as demand forecasts are usually available at this time scale. We solve each instance using the CP Optimizer and with the initial solution provided by our warm-start strategy.

Figure 6.10 summarizes the aggregated results. The results per instance are partly shown in Table 6.12 (10 per instance size). However, for the remaining results obtained with CP Optimizer, we refer the reader to 6.E (for readability). The first solution reported in this table refers to the warm-start solution and its corresponding makespan. The CPU time of the first solution thus corresponds to the time required to perform the warm start. Then, we report on the solutions found in 600 seconds (ten minutes), 3600 seconds (one hour), and 10800 seconds (three hours). Finally, we apply a Wilcoxon matched-pairs rank-sum test (as described by Cureton (1967)) to the optimality gaps obtained to compare different time limits. The results are presented in Table 6.11 and confirm that increasing the time

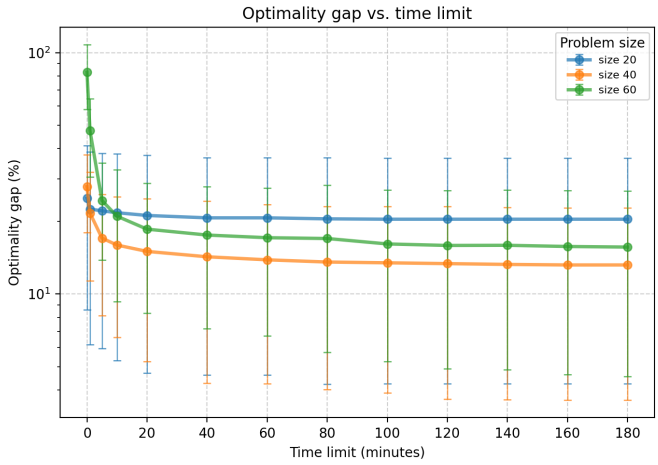


Figure 6.10 *Reported optimality gap of the CP solver as a function of problem size and time-limit, aggregated results of all instances. The bars visualize the standard deviations of the reported optimality gaps. We additionally ran the models for 10 hours and found that for sizes 40 and 60, this still yields significantly better optimality gaps, whereas for size 20, it yields no significant improvement.*

Table 6.11 *Results of the Wilcoxon test on different time limit settings. * indicates significantly better.*

# Orders	Time-limit pair for comparison	z-value	p-value
10	(600, 3600*)	-2.4147	0.01575
10	(3600, 10800)	-1.70009	0.08911
20	(600, 3600*)	-3.6766	0.000236
20	(3600, 10800*)	-2.6246	0.008676
40	(600, 3600*)	-4.7423	2.11e-06
40	(3600, 10800*)	-4.0138	5.98e-05
60	(600, 3600*)	-4.7719	1.83e-06
60	(3600, 10800*)	-4.7136	2.43e-06

limit generally results in smaller optimality gaps. The only exception occurs in the smallest instance (instances with size 10, which is reported in Table 6.18), where extending the time limit from 1 hour to 3 hours does not yield a statistically significant gain. Overall, our experiments confirm that larger time limits are beneficial for CP performance, particularly as instance size increases.

In Figure 6.10, we report the aggregated optimality gaps for all experiments (Table 6.12 and Appendix 6.E). Interestingly, the smallest instances yield the largest relative gaps. Although this may seem counterintuitive at first, our interpretation is that the larger instances also have longer scheduling horizons. In such cases, the effect of individual

Table 6.12 *Selection of the results obtained with the CP model for different time limits. We provide additional results in 6.E. The first reported solution is the one obtained using the warm-start strategy. LB is the lower bound reported by the solver.*

		First solution		Solution in 600 s			Solution in 3600 s			Solution in 10800 s		
		Makespan	CPU time	Makespan	Gap	LB	Makespan	Gap	LB	Makespan	Gap	LB
# Orders	Instance ID											
20	1	3823	4.72	737	0.8%	731	737	0.8%	731	737	0.8%	731
	2	3843	7.61	853	33.5%	639	833	30.4%	639	833	30.4%	639
	3	3856	6.84	770	8.5%	710	770	8.5%	710	767	8.0%	710
	4	3826	7.14	958	19.0%	805	958	19.0%	805	958	19.0%	805
	5	3554	7.60	710	6.4%	667	697	4.5%	667	697	4.5%	667
	6	3840	4.49	886	37.4%	645	872	35.2%	645	872	35.2%	645
	7	3886	4.39	985	44.9%	680	985	44.9%	680	985	44.9%	680
	8	3815	5.40	941	21.3%	776	925	19.2%	776	924	19.1%	776
	9	4129	4.42	970	40.8%	689	963	39.8%	689	958	39.0%	689
	10	3847	6.73	742	4.7%	709	732	3.2%	709	732	3.2%	709
40	1	7494	9.83	1650	6.3%	1552	1586	2.2%	1552	1573	1.4%	1552
	2	7764	13.13	1655	10.2%	1502	1650	9.9%	1502	1650	9.9%	1502
	3	7999	12.59	1630	5.0%	1552	1600	3.1%	1552	1600	3.1%	1552
	4	7600	9.09	1648	10.5%	1492	1629	9.2%	1492	1627	9.0%	1492
	5	8049	12.14	1872	33.1%	1406	1846	31.3%	1406	1843	31.1%	1406
	6	7820	8.44	1733	20.5%	1438	1678	16.7%	1438	1648	14.6%	1438
	7	7718	9.33	1783	21.1%	1472	1749	18.8%	1472	1749	18.8%	1472
	8	8068	10.11	1809	22.6%	1476	1792	21.4%	1476	1767	19.7%	1476
	9	7694	9.12	1672	22.8%	1362	1647	20.9%	1362	1647	20.9%	1362
	10	7653	14.07	1466	5.2%	1393	1445	3.7%	1393	1441	3.4%	1393
60	1	11631	22.64	2467	18.9%	2074	2462	18.7%	2074	2428	17.1%	2074
	2	11552	22.64	2158	5.0%	2056	2121	3.2%	2056	2096	1.9%	2056
	3	11488	21.87	2451	14.0%	2150	2314	7.6%	2150	2285	6.3%	2150
	4	11622	22.77	2501	9.8%	2278	2459	7.9%	2278	2405	5.6%	2278
	5	11646	26.00	2523	27.3%	1982	2385	20.3%	1982	2360	19.1%	1982
	6	12082	24.10	2665	27.0%	2098	2531	20.6%	2098	2504	19.4%	2098
	7	12322	21.38	3137	45.5%	2156	3047	41.3%	2156	3033	40.7%	2156
	8	11723	22.87	2689	28.7%	2090	2520	20.6%	2090	2520	20.6%	2090
	9	11810	22.75	2709	14.3%	2370	2677	13.0%	2370	2677	13.0%	2370
	10	11923	20.70	2555	19.4%	2140	2462	15.0%	2140	2407	12.5%	2140

suboptimal decisions is spread out over a longer time window, resulting in lower overall reported relative optimality gaps. Furthermore, we observe that the standard deviations are relatively large, indicating that the achieved optimality gap varies widely across instances.

Effect of the zero-wait constraints on solver performance

Analyzing the performance results (Table 6.12), we observed that many instances are not proven to be optimal, even with large time limits and a warm-starting strategy. These findings motivated a closer examination of what makes these instances so challenging. We hypothesize that zero-wait constraints make the CP approach less effective. This is consistent with prior theory: maximum time lags (of which zero-wait is a special case) already make feasibility in resource-constrained scheduling NP-hard (Bartusch, Möhring, and Radermacher 1988), and feasibility for zero-wait job-shop scheduling is significantly more challenging (Mascis and Pacciarelli 2002). Recent empirical evidence comparing MIP and CP also confirms that zero-wait tends to lead to longer computation times or larger optimality gaps (Naderi, Ruiz, and Roshanaei 2023).

Table 6.13 shows the results for the same set of instances as in Table 6.12, but with the zero-wait constraints removed. We observe that now one-third of the instances are solved to optimality (i.e., with a reported optimality gap of 0%), even though the time limit was only 600 seconds. In contrast, when zero-wait constraints were enforced,

Table 6.13 *Results on instances for which the zero-wait constraints are switched off. We used the same set of instances as in Table 6.12 and gave a time limit of 600 seconds.*

# Orders	Instance ID	Status	Warm-start makespan	Warm-start time	Objective	Runtime	Gap
20	1	Optimal	3591	4.25	728	0.69	0%
	2	Feasible	3583	7.77	727	600.01	23.6%
	3	Optimal	3550	5.78	710	32.01	0%
	4	Feasible	3639	4.7	880	600.03	9.7%
	5	Optimal	3308	6.84	664	0.73	0%
	6	Feasible	3587	5.19	765	600.01	18.6%
	7	Feasible	3618	5.88	864	600.01	27.0%
	8	Feasible	3502	7.48	825	600.02	6.3%
	9	Feasible	3828	4.92	855	600.05	32.8%
	10	Optimal	3562	5.59	682	0.74	0%
40	1	Optimal	7036	12.63	1552	76.22	0%
	2	Feasible	7231	11.59	1578	600.02	5.0%
	3	Optimal	7490	11.91	1552	24.37	0%
	4	Feasible	7199	12.49	1500	600.06	0.5%
	5	Feasible	7534	16.08	1720	600.06	22.3%
	6	Feasible	7244	12.1	1586	600.05	10.3%
	7	Feasible	7198	10.59	1643	600.07	11.6%
	8	Feasible	7499	13.27	1612	600.05	9.2%
	9	Feasible	7164	18.64	1501	600.02	10.2%
	10	Optimal	7108	11.15	1390	3.9%	0%
60	1	Feasible	10795	22.05	2252	600.1	8.6%
	2	Optimal	10698	23.03	2056	154.99	0%
	3	Optimal	10631	26.7	2150	298.97	0%
	4	Optimal	10918	22.96	2278	534.17	0%
	5	Feasible	10871	23.51	2158	600.06	8.9%
	6	Feasible	11268	21.97	2353	600.02	12.2%
	7	Feasible	11442	27.61	2919	600.03	35.4%
	8	Feasible	10896	26.46	2269	600.05	8.6%
	9	Feasible	11105	29.59	2405	600.04	1.5%
	10	Feasible	11062	20.72	2148	600.05	0.4%

none of the instances achieved a 0% gap, even after 3 hours. Finally, we perform a Wilcoxon test applied to the obtained optimality gaps. The test confirms that when the zero-wait constraints are disabled, the solver consistently achieves lower optimality gaps ($Z = -4.7719$ and $p = 1.83e-06$). The hypothesis that the presence of zero-wait constraints worsens CP performance can thus be confirmed by our experiments.

6.8. Conclusion and future work

In this work, we demonstrate that constraint programming, combined with a warm-start strategy, is effective for solving large-scale, real-world scheduling problems in a biomanufacturing facility and provide a set of realistic problem instances. In our experiments, the combination of industrial constraints, particularly zero-wait policies, appeared to reduce the effectiveness of state-of-the-art CP solvers such as OR-Tools. Notably, Google OR-Tools struggled considerably to find practical solutions for industrial-sized instances with the required planning horizons. Fortunately, we leveraged domain knowledge to de-

velop a warm-start strategy that significantly improved solution quality. Our warm-start strategy was successful, yet it highlights an opportunity for CP solvers to more effectively exploit problem structures directly.

We did not consider scheduled maintenance or operator shift patterns in this study, due to the lack of data; however, these extensions could be integrated relatively easily into the CP/PyJobShop framework. The availability of raw materials was also outside the scope, as dsm-firmenich indicated it is generally less of a bottleneck for operations. In addition to the modeling challenges described in Section 6.4, tanks have not only a fixed capacity but also a tank-specific minimum weight that must be collected in a tank before a specific processing step can start. In our current model, we use these minimum weights to derive the required time-lags (e.g., between fermentation and harvesting, or between filter A and filter C) rather than keeping track of the flow from tank to tank and triggering the next event once the minimum weight is reached, as could be done in a simulation model. Relatedly, the buffer tanks between filter A/B and filter C require careful synchronization because the different filters operate at different rates; therefore, the model must ensure that these buffers do not overflow. These details may become more important if our CP-derived schedules are used at the operational day-to-day level, rather than at the tactical planning level that determines which products to produce within a week or month. We observed that, for this time horizon, the impact on the makespan was negligible.

Furthermore, we did not account for variations in product quality, yield, or processing times. Variabilities in product quality and yield are partly inherent to the biological nature of the products. However, excessively long waiting times between stages can also contribute to product degradation (due to the shelf lives). An interesting direction for future research is to investigate the relationship between waiting time and product quality in the context of bio-based production. A better understanding of this relationship could enable its integration into the scheduling process. We can relax the zero-wait constraint slightly between specific stages to achieve more efficient schedules; however, to do so, we need to understand its impact on product quality.

Another natural next step is to account for variability in the processing times. In particular, when zero-wait constraints and/or maximum time lags are present, the combination with uncertain processing times can lead to infeasibilities during online schedule execution, sometimes in ways that cannot be easily repaired. Nevertheless, deterministic CP scheduling can be extended with techniques for handling uncertainty, as demonstrated by Van den Houten *et al.* (2025), for example, through simple temporal networks with uncertainty (Morris, Muscettola, and Vidal 2001). Future research could integrate these approaches to evaluate stochastic scheduling methods on realistic industrial instances.

We fitted a job-shop-style constraint-programming model to a facility in which batches are split into subbatches. By doing so, we showed that it is not always necessary to use material-based models to model such factories. Within the scope of this work, we assumed that all decisions regarding batch splitting and mixing are made before scheduling. An interesting direction for future work would be to integrate the choice of mixing and splitting directly into the optimization model, which could yield more efficient schedules but would also increase the number of variables and thereby the overall complexity of the optimization. A method to realize this within CP models has yet to be identified.

While our study focuses on a single real-world facility, the findings and methodology

are generalizable to other settings. For one, our model integrates multiple constraints that commonly arise in other scenarios, as highlighted by related work (Tables 6.1 and 6.15). Second, we adopt a highly standardized CP formulation—following (Lan and Berkhout 2025)—which should lower the barrier for future research to reuse and build on these modeling choices in other batch scheduling factories. By publicly releasing all instances and model instantiations, we aim to enable practitioners and researchers in the field of biomanufacturing to build upon our solutions. We encourage the broader CP community to study our instances, which are considerably more challenging than standard scheduling benchmarks.

Finally, although a validation study was conducted during the test phase of the CP model, the primary focus of this work was on analyzing the modeling approach and the resulting schedules, rather than on full-scale deployment of the CP solution. Of course, for making a real impact, this is a crucial next step that should be taken together with the planners and schedulers. The tools developed in this work can be used to support production planning by determining which batches to plan for the next months, given predicted customer demand, as well as to support schedulers at the tactical to operational level in optimizing their capacity usage. Future research should place greater emphasis on how such tools can be used flexibly in practice, capable of handling uncertainty and adapting to changes in production plans. To achieve this, it is crucial to incorporate the insights and expertise of practitioners who use these scheduling tools in real-world settings.

Appendix

6.A. Literature on mixed-integer linear programming solutions

In the main body of the paper, we present a table comparing our contribution with other known CP solutions to batch scheduling from the literature. We did not include the MILP contributions in the table for readability purposes. However, for completeness, we present these in Table 6.15. This overview is an extension and slight adjustment of the one presented by Novara, Novas, and Henning 2016.

Castro, Harjunkoski, and Grossmann 2011 (CHGb) propose a decomposition method that sequentially solves the problem batch by batch, fixing part of the decision variables after each iteration before moving on to the next batch. Our approach is related, but instead of fixing variables, we solve each batch separately to obtain a warm-start solution, which is then provided as an initial solution to the solver. Kopanos, Méndez, and Puigjaner 2010 (KMP) introduce a MIP-based strategy that first rapidly generates a feasible initial solution through an iterative insertion procedure, followed by an optimization step using a rescheduling procedure. Baumann and Trautmann 2014 (BT) present a two-stage approach involving the division of tasks into sub-batches, where the sub-batch configurations differ across stages but remain independent between batches.

Table 6.14 *Overview of abbreviations used in Table 6.15.*

Abbreviation	Reference
MCa	Marchetti and Cerdá 2009a
MCb	Marchetti and Cerdá 2009b
CHGa	Castro, Harjunkoski, and Grossmann 2009
CHGb	Castro, Harjunkoski, and Grossmann 2011
KMP	Kopanos, Méndez, and Puigjaner 2010
BT	Baumann and Trautmann 2014
PM	Prasad and Maravelias 2008
SM	Sundaramoorthy and Maravelias 2008
MMC	Marchetti, Mendez, and Cerda 2012
CCE	Computers & Chemical Engineering
IECR	Industrial & Engineering Chemistry Research
AIChE	Journal of the American Institue of Chemical Engineers
EURO	European Journal of Operational Research

Table 6.15 *Features covered by literature on mixed integer linear programming for batch scheduling.*

Problem features			MCa	MCb	CHGa	CHGb	KMP	BT	PM	SM	MMC
Journal			CCE	CES	IECR	AIChE	EURO	EURO	CCE	IECR	IECR
Year			2009	2009	2009	2011	2010	2014	2008	2008	2012
Mixing and splitting	Network process							x	x		
	Sequential process		x	x	x	x	x	x	x	x	x
Storage policy	Unlimited intermediate storage		x	x	x	x	x		x	x	x
	No intermediate storage								x	x	
	Finite intermediate storage							x	x		
	Zero wait						x	x	x		
Set up features	Sequence independent				x		x	x			
	Sequence dependent	Unit dependent	x	x		x	x	x	x	x	x
		Unit independent									
Resources	Machines		x	x	x	x	x	x	x	x	x
	Renewable resources		x	x							
	Unrenewable resources										
Processing times	Fixed		x	x	x	x	x		x		
	Variable							x		x	x
Additional constraints	Production sequence							x	x	x	
	Toplogy constraints		x	x	x	x		x	x	x	x
	Transfer times						x	x			
	Identical resources										
	Maintenance										
	Work shifts										
	Release dates		x	x			x		x	x	
	Due dates		x	x					x	x	x
Objective function	Makespan		x	x	x	x	x	x	x	x	x
	Tardiness		x				x	x	x	x	x
	Earliness						x	x	x	x	
	Cost						x	x	x		
Problem size	Medium		x	x				x	x	x	x
	Large				x	x	x	x			
Campaign mode	Considered										
Decisions	Batching								x	x	x
	Subbatching										
	Sequencing		x	x	x	x	x	x	x	x	x
	Machine assignment		x	x	x	x	x	x	x	x	x
	Timing		x	x	x	x	x	x	x	x	x
Solver	CPLEX		x	x	x	x	x	x	x	x	x
Number of instances			4	10	2	10	12	30	5	18	5
Instance data			x	x	x	x	x		x		x

6.B. OR-Tools model

In this section, we present the OR-Tools version of the CP model, which comes from Lan and Berkhout 2025.

Variables

Lan and Berkhout (2025) define the following interval variables:

- ϕ_j: interval variable for each job $j \in J$ with $\phi_j^{\text{present}} = 1$
- τ_t: interval variable for each task $t \in T$ with $\tau_t^{\text{present}} = 1$
- μ_m: interval variable for each mode $m \in M$ with $\mu_m^{\text{duration}} = p_m$

Objective

$$\min \max_{t \in T} \tau_t^{\text{end}} \tag{6.3}$$

Constraints

Linking jobs to tasks

$$\phi_j^{\text{start}} = \min_{t \in T_j} \tau_t^{\text{start}} \qquad \forall j \in J \tag{6.4}$$

$$\phi_j^{\text{end}} = \max_{t \in T_j} \tau_t^{\text{end}} \qquad \forall j \in J \tag{6.5}$$

Linking tasks to modes

$$\sum_{m \in M_t} \mu_m^{\text{present}} = 1 \qquad \forall t \in T \tag{6.6}$$

$$\mu_m^{\text{start}} = \tau_t^{\text{start}} \qquad \forall t \in T,\ m \in M_t \tag{6.7}$$

$$\mu_m^{\text{end}} = \tau_t^{\text{end}} \qquad \forall t \in T,\ m \in M_t \tag{6.8}$$

$$\mu_m^{\text{present}} \Rightarrow \mu_m^{\text{duration}} = \tau_t^{\text{duration}} \qquad \forall t \in T,\ m \in M_t \tag{6.9}$$

Resource constraints

$$\text{NoOverlap}(\{\mu_m : m \in M_r^R\}) \qquad \forall r \in R^{\text{machine}} \tag{6.10}$$

$$\text{Cumulative}(\{\mu_m : m \in M_r^R\}, \{q_{mr} : m \in M_r^R\}, Q_r) \qquad \forall r \in R^{\text{renewable}} \tag{6.11}$$

(6.12)

Timing constraints

$$\tau_i^{\text{start}} + l \le \tau_k^{\text{start}} \qquad \forall (i,k,l) \in \mathscr{C}^{\text{StartBeforeStart}} \tag{6.13}$$

$$\tau_i^{\text{start}} + l \le \tau_k^{\text{end}} \qquad \forall (i,k,l) \in \mathscr{C}^{\text{StartBeforeEnd}} \tag{6.14}$$

$$\tau_i^{\text{end}} + l \le \tau_k^{\text{start}} \qquad \forall (i,k,l) \in \mathscr{C}^{\text{EndBeforeStart}} \tag{6.15}$$

$$\tau_i^{\text{end}} + l \le \tau_k^{\text{end}} \qquad \forall (i,k,l) \in \mathscr{C}^{\text{EndBeforeEnd}} \tag{6.16}$$

Assignment constraints

$$\mu_{m_i}^{\text{present}} \le \sum_{\substack{m_k \in M_k, \\ s.t.\ R_{m_k} = R_{m_i}}} \mu_{m_k}^{\text{present}} \qquad \forall (i,k) \in C^{\text{IdenticalResources}}, m_i \in M_i \tag{6.17}$$

$$\mu_{m_i}^{\text{present}} \le \sum_{\substack{m_k \in M_k, \\ s.t.\ R_{m_k} \cap R_{m_i} = \emptyset}} \mu_{m_k}^{\text{present}} \qquad \forall (i,k) \in C^{\text{DifferentResources}}, m_i \in M_i \tag{6.18}$$

Sequencing constraints Lan and Berkhout 2025 define a sequence of intervals that can be modeled using a complete graph combined with the global `Circuit` constraint to enforce orderings. For each machine $r \in R_{\text{machine}}$, they define a complete directed graph with node set $V_r = M_r \cup \{0\}$, where M_r denotes the set of modes that require machine r,

and node 0 is a dummy node. The arc set includes all possible ordered pairs of nodes, and they use binary decision variables

$$B_r = \{b_{ruv} \in \{0,1\} : u, v \in V_r\},$$

where $b_{ruv} = 1$ if arc (u, v) is selected in the graph corresponding to machine r, and 0 otherwise. They use sequencing constraints as follows:

$$\text{Circuit}(B_r) \qquad \forall r \in R^{\text{machine}} b_{uv} \Rightarrow \mu_u^{\text{present}} \wedge \mu_v^{\text{present}} \quad \forall u, v \in M_r^R \tag{6.19}$$

$$b_{uv} \Rightarrow \neg\mu_u^{\text{present}} \qquad \forall u \in M_r^R \tag{6.20}$$

$$b_{00} \Rightarrow \neg\mu_u^{\text{present}} \qquad \forall u \in M_r^R \tag{6.21}$$

$$b_{uv} \Rightarrow \mu_u^{\text{end}} + s_{t_u,t_v,r} \leq \mu_v^{\text{start}} \qquad \forall u, v \in M_r^R \tag{6.22}$$

Consecutive constraints

$$\mu_u^{\text{present}} \wedge \mu_v^{\text{present}} \Rightarrow b_{uv} \qquad \forall (i,k) \in \mathscr{C}^{\text{Consecutive}} \tag{6.23}$$

Mode dependencies

$$\mu_m^{present} \leq \sum_{n \in \mathscr{M}_m^{dependent}} \mu_n^{present} \quad \forall (m, \mathscr{M}_m^{dependent}) \in \mathscr{C}^{\text{ModeDependencies}}.$$

6.C. Overview instances

The characteristics of our problem instances are summarized in Table 6.16. Constraints refer to the modular PyJobShop constraints Lan and Berkhout 2025. To give an example of how this translates to the number of variables and constraints with CP Optimizer and OR-Tools, for a problem instance with 5 products, the OR-Tools model contains 2494 variables, and it includes 2975 Boolean constraints, 469 interval constraints, 28 no-overlap constraints, 9 circuit constraints, and 11+27+2552+2479+210 linear constraints. The CP Optimizer model has 519 variables and 1995 constraints and is thus significantly smaller.

Table 6.16 *Average instance characteristics per job size.*

Jobs	Tasks	Modes	Constraints
5	104	361	550
10	215	750	1389
20	438	1521	3813
40	873	3036	11594
60	1310	4554	23435
120	2584	8930	82641

6.D. Solver comparison

In Table 6.10, we included the results per instance that were summarized in Section 6.7.2.

Table 6.17 *We report on the difference in the objective obtained by OR-Tools and the CP Optimizer. *The NaN status obtained by OR-Tools is due to a memory limit; even increasing the memory from 4GB to 15GB did not resolve this issue.*

	Time limit (seconds)		600		3600		10800
	Solver	CP Optimizer	OR-Tools	CP Optimizer	OR-Tools	CP Optimizer	OR-Tools
# Orders	**Instance ID**						
10	11	395	395	395	395	395	395
	12	481	481	481	481	481	481
	13	584	584	584	584	584	584
	15	429	424	429	424	429	424
	16	583	581	583	581	583	581
	17	495	464	467	464	467	464
	18	573	573	573	573	573	573
	19	401	401	401	401	401	401
	20	446	429	429	429	429	429
20	11	977	963	977	954	977	936
	12	848	840	848	NaN	848	NaN
	13	927	918	922	911	918	885
	15	758	794	758	NaN	751	NaN
	16	848	813	837	NaN	837	NaN
	17	923	963	918	NaN	918	NaN
	18	787	774	787	764	787	752
	19	890	885	890	NaN	890	NaN
	20	835	982	835	864	835	NaN
40	11	1805	1906	1774	1790	1763	1851
	12	1813	2808	1813	2350	1768	2243
	13	1739	2888	1737	2200	1713	1835
	15	1648	2362	1644	2181	1623	1829
	16	1627	2053	1571	1798	1571	1825
	17	1772	2352	1762	2074	1762	1968
	18	1757	3102	1680	2067	1643	NaN
	19	1501	2445	1501	2000	1501	1717
	20	1644	2171	1633	1937	1625	1868
60	11	2789	10410	2652	4602	2646	3590
	12	2638	8045	2582	4163	2582	3086
	13	2511	9173	2342	3681	2342	3525
	15	2571	9286	2522	3790	2506	4196
	16	2446	5702	2377	3304	2377	2939
	17	2649	9283	2605	4218	2581	3434
	18	2622	9888	2614	4119	2569	3462
	19	2307	8788	2233	6552	2173	2747
	20	2295	5809	2286	4053	2231	2687
120	11	6269	23965	4881	23639	4865	NaN
	12	7143	23600	5100	22775	4998	22888
	13	7969	23415	5415	22409	5320	13296
	15	6610	23600	5038	21755	4914	13124
	16	5864	22638	5023	19288	4815	15400
	17	6030	23175	4863	19168	4755	16719
	18	7499	23904	5212	18693	5002	14453
	19	6587	22620	5453	21141	5295	13153
	20	5900	22634	5065	21851	4976	14316

6.E. More results with CP Optimizer

In Tables 6.18 and 6.19, we provide additional results on further test instances, which were omitted from the main paper for readability. The results in these tables are also included in the aggregation used for Figure 6.10, together with the results from Table 6.12. All instances are publicly available in our repository.

Table 6.18 *Additional results obtained with CP Optimizer on problem instances of size 10 and 20. Reported are the warm-start makespan, the CPU time required for the warm start, and the reported objective and optimality gap after 10 minutes, 1 hour, and 3 hours.* ***Bolded*** *numbers are proven optimal objectives for the given instance.*

		First solution		Solution in 600 s		Solution in 3600 s		Solution in 10800 s	
		Makespan	CPU time	Makespan	Gap	Makespan	Gap	Makespan	Gap
# Orders	Instance ID								
10	21	1833	1.53	388	0.3%	388	0.3%	**388**	0.0%
	22	1890	1.26	487	42.0%	480	2.1%	**480**	0.0%
	23	2145	2.34	613	47.4%	613	47.4%	613	47.4%
	24	1748	2.20	363	1.7%	**357**	0.0%	**357**	0.0%
	25	2128	1.56	541	29.1%	541	29.1%	541	29.1%
	26	2195	2.12	521	19.2%	521	19.2%	521	19.2%
	27	2046	1.83	589	44.0%	578	41.3%	578	41.3%
	28	1950	1.84	440	0.0%	**440**	0.0%	**440**	0.0%
	29	2024	2.19	461	4.8%	461	4.8%	461	4.8%
	30	1945	2.21	424	28.1%	424	28.1%	424	28.1%
	32	2028	2.87	528	36.8%	528	36.8%	504	30.6%
	33	2016	1.89	510	40.5%	493	35.8%	493	35.8%
	34	2036	2.20	516	26.2%	516	26.2%	516	26.2%
	35	1834	3.14	429	28.1%	429	28.1%	429	28.1%
	36	1723	1.55	**387**	0.0%	**387**	0.0%	**387**	0.0%
	37	1839	1.28	**419**	0.0%	**419**	0.0%	**419**	0.0%
	38	1812	1.94	**429**	0.0%	**429**	0.0%	**429**	0.0%
	39	1978	2.19	543	48.0%	529	44.1%	529	44.1%
	40	1999	2.43	517	26.4%	517	0.0%	**517**	0.0%
	41	1930	2.53	427	19.6%	427	19.6%	427	19.6%
20	21	3814	4.64	741	23.1%	741	23.1%	741	23.1%
	22	4046	7.95	942	23.6%	936	22.8%	936	22.8%
	23	3871	3.10	865	27.4%	865	27.4%	837	23.3%
	24	3790	3.15	803	20.8%	801	20.5%	796	19.7%
	25	3797	2.79	728	17.8%	701	13.4%	701	13.4%
	26	4013	4.80	828	2.9%	828	2.9%	828	2.9%
	27	3937	6.18	975	29.0%	975	29.0%	975	29.0%
	28	3728	3.72	749	1.2%	749	1.2%	749	1.2%
	29	3924	3.99	841	1.1%	841	1.1%	841	1.1%
	30	4014	3.17	810	14.2%	759	7.1%	759	7.1%
	32	4170	2.85	929	25.7%	929	25.7%	929	25.7%
	33	4035	3.72	883	19.6%	883	19.6%	883	19.6%
	34	4051	4.33	867	33.2%	867	33.2%	867	33.2%
	35	4075	6.27	885	15.1%	864	12.4%	864	12.4%
	36	4079	4.15	750	6.7%	750	6.7%	750	6.7%
	37	4114	5.28	909	12.9%	909	12.9%	891	10.7%
	38	4031	3.91	858	6.6%	858	6.6%	840	4.3%
	39	3853	3.82	979	30.0%	965	28.2%	965	28.2%
	40	3905	3.21	774	9.2%	768	8.3%	768	8.3%
	41	3775	5.61	881	19.2%	851	15.2%	851	15.2%

Table 6.19 *Additional results obtained with CP Optimizer on problem instances of size 10 and 20. Reported are the warm-start makespan, the CPU time required for the warm start, and the reported objective and optimality gap after 10 minutes, 1 hour, and 3 hours.*

		First solution		Solution in 600 s		Solution in 3600 s		Solution in 10800 s	
		Makespan	**CPU time**	**Makespan**	Gap	**Makespan**	**Gap**	**Makespan**	**Gap**
# Orders	**Instance ID**								
40	21	7831	9.89	1625	15.4%	1598	13.5%	1598	13.5%
	22	7910	9.33	1718	19.5%	1681	16.9%	1679	16.8%
	23	8204	11.39	1722	9.3%	1706	8.3%	1706	8.3%
	24	7045	12.78	1511	9.3%	1439	4.1%	1438	4.1%
	25	7695	11.73	1506	3.4%	1475	1.2%	1475	1.2%
	26	7976	9.58	1873	32.6%	1861	31.7%	1861	31.7%
	27	7853	11.37	1562	15.3%	1529	12.8%	1491	10.0%
	28	7766	9.73	1735	13.3%	1650	7.7%	1627	6.2%
	29	7673	9.27	1528	20.6%	1524	20.3%	1507	18.9%
	30	7621	8.86	1576	24.4%	1550	22.3%	1534	21.1%
	32	7775	9.15	1584	25.7%	1584	25.7%	1584	25.7%
	33	7517	10.21	1503	1.4%	1503	1.4%	1503	1.4%
	34	7747	10.84	1642	19.7%	1625	18.4%	1625	18.4%
	35	7777	7.60	1639	23.8%	1637	23.6%	1637	23.6%
	36	7697	9.17	1593	24.4%	1569	22.5%	1513	18.1%
	37	8007	10.39	1704	21.0%	1702	20.9%	1702	20.9%
	38	7694	10.18	1625	5.7%	1607	4.5%	1603	4.2%
	39	7717	9.90	1766	29.6%	1730	26.9%	1712	25.6%
	40	8004	10.62	1771	32.6%	1718	28.6%	1709	27.9%
	41	7845	11.26	1692	17.5%	1669	15.9%	1663	15.5%
60	21	11737	17.72	2285	21.8%	2261	20.5%	2200	17.3%
	22	12067	20.61	2505	20.7%	2395	15.4%	2361	13.7%
	23	11998	24.11	2606	29.3%	2560	27.0%	2542	26.1%
	24	11406	21.75	2240	10.2%	2189	7.7%	2153	6.0%
	25	11198	20.98	2199	10.3%	2172	9.0%	2163	8.5%
	26	11710	26.53	2424	14.3%	2408	13.6%	2395	13.0%
	27	11254	20.55	2387	18.5%	2276	13.0%	2268	12.6%
	28	12037	20.20	2702	17.8%	2616	14.0%	2607	13.6%
	29	12195	19.15	2405	10.9%	2282	5.3%	2261	4.3%
	30	11926	19.88	2564	16.3%	2511	13.9%	2511	13.9%
	32	11812	20.13	2645	33.0%	2505	26.0%	2462	23.8%
	33	11386	23.33	2131	3.7%	2111	2.8%	2087	1.6%
	34	11749	22.10	2335	15.6%	2279	12.8%	2274	12.6%
	35	11300	22.24	2287	8.7%	2205	4.8%	2202	4.7%
	36	11715	20.70	2417	11.6%	2393	10.5%	2325	7.3%
	37	12213	18.92	2978	31.9%	2899	28.4%	2878	27.5%
	38	11463	19.38	2491	15.0%	2460	13.6%	2454	13.3%
	39	12321	19.28	2646	26.0%	2587	23.2%	2584	23.0%
	40	12193	20.92	2675	23.0%	2570	18.2%	2564	17.9%
	41	11872	22.42	2406	5.2%	2351	2.8%	2337	2.1%


PLAN A's SIMPLY DON't WORK OUT ALL THE TIME
SO BETTER GO WITH THE FLOW

7

Conclusion

The goal of this research was to explore how optimization algorithms can be designed to solve real scheduling problems, such as those faced in biomanufacturing. This thesis has demonstrated that progress in scheduling for biomanufacturing can be achieved both by rethinking how uncertainty is handled in scheduling and by developing optimization approaches that can be used in an industrial setting. The research questions posed in the introduction of this thesis are answered in the concluding section of each chapter. In Chapter 2 - Chapter 4, we explored methods for handling uncertainty in scheduling. Then, in Chapter 5 and Chapter 6, we focused on algorithms that directly support the tactical planning and scheduling operations of a real biomanufacturing facility.

In this final part of the thesis, a reflection on the main overall conclusions and a discussion are provided. We start with summarizing the key findings in Section 7.1. The limitations of this research are discussed in Section 7.2. Concluding this dissertation, we also see many potential directions for future work, on which we elaborate in Section 7.3.

7.1. Key findings

Deterministic surrogates

In this dissertation, we explored methods for handling uncertainty in scheduling while maintaining computational efficiency. We investigated the use of deterministic scenarios for solving stochastic problems. A central finding of this thesis is that deterministic solvers can be leveraged effectively for stochastic scheduling problems when used carefully, specifically when combined with simulation. In Chapter 2, we studied a learning-based approach that fits deterministic constraint programming models to represent stochastic problems with repair strategies. Rather than relying on traditional scenario-based stochastic programming, this approach fits a single scenario to construct a deterministic constraint programming model whose solutions perform well under uncertainty. This idea draws inspiration from decision-focused learning and is tailored to stochastic scheduling with repair. We found that this approach outperformed a fully deterministic and a scenario-based stochastic (constraint) programming approach on a set of benchmark instances, especially on larger problem instances and/or instances with a high repair penalty.

In Chapter 3, we extended this philosophy to large-scale problems where even solving a deterministic counterpart to optimality is unlikely within a fixed time budget. We showed that combining simulation with deterministic metaheuristics, in the form of simheuristics, can be made more effective by moving beyond a standard expected-value representation and alternating between quantile-based representations. This dynamic mechanism improved performance on a set of stochastic combinatorial optimization problems, highlighting that combining deterministic simplifications with adaptive strategies can be effective for finding better solutions within a fixed time budget.

Together, these contributions demonstrate how we can use deterministic solvers to find solutions to stochastic problems by carefully tuning deterministic representations and complementing them with simulation.

Strict temporal constraints: the flow must go on

A second key contribution is research into temporal uncertainty with strict temporal constraints, such as maximal time lags, an combination that has received limited attention but that is highly relevant for real-world biomanufacturing. These maximum time-lag constraints would, in a real factory, ensure that a product can wait only a limited time before proceeding to the next piece of equipment, because it is essential that the flow continues to avoid impacting product quality. The challenges in such a problem setting are fundamentally different from those in Chapter 2 and Chapter 3, where repair functions affect the objective function through a penalty term. In Chapter 4, we looked at these problems where uncertainties can cause violations of temporal constraints.

An effective approach to modelling such problems is by using simple temporal networks with uncertainty (STNUs). Our work was the first to combine constraint programming with the real-time execution algorithm RTE* (Hunsberger and Posenato 2024) to scheduling problems of this type. We demonstrated that with such an approach feasible online execution can be guaranteed while most of the computational effort is spent offline. Finally, we also contributed by means of an extensive comparison with fully proactive and fully reactive constraint programming and confirmed our hypothesis that combining constraint programming with temporal network methods offers strong advantages over the baseline methods considered.

Impact in biomanufacturing

Finally, this thesis demonstrated that combining discrete-event simulation with relatively simple search strategies can have a meaningful impact in real applications. In Chapter 5, we demonstrated that a rolling-horizon approach could substantially improve makespan and lateness objectives of the schedules of a real biomanufacturing facility, even when considering long planning horizons. This makes the approach suitable for supporting strategic-to-tactical production decisions. We could improve a schedule that was created by experts.

Furthermore, we demonstrated how constraint programming can be utilized to model the complex scheduling operations of a real factory, where multiple complicating constraints come together that have not been studied together in the literature. Earlier work questioned how to handle batch mixing and splitting. We modeled it by allowing single

batches to be split into subbatches and then mixed again, while maintaining computational efficiency. We see great potential in using constraint programming to support tactical and operational scheduling decisions in biomanufacturing.

Unlike standard benchmark problems, the realistic instances we presented in Chapter 6 combined multiple constraints simultaneously, including the no-wait condition, which significantly worsened solver performance. The findings from this chapter highlight that solving industrial-scale scheduling problems is not only practically impactful but also exposes new computational challenges that benchmark instances from the literature fail to capture.

7.2. Limitations

In this section, we first reflect on the concrete limitations of our work and offer suggestions for addressing them in future work.

Simplified scheduling problems

In Chapters 2–4, we developed novel approaches for handling uncertainty in scheduling, but these were tested on standard benchmark problems rather than full-scale factory instances. For example, in Section 4.4.1, the instances were inspired by the factory but omitted key constraints such as flexible machines, no-wait conditions, and setup times. Similarly, in Chapter 4, we used benchmark problems with maximal time lags, but only from the SRCPSPmax class. By contrast, in Chapter 6, we modeled instances that closely reflect the real factory, yet did not incorporate uncertainty. Also, the employed discrete-event simulator in Chapter 5 was used only in deterministic mode due to a lack of data. In this thesis, uncertainty was studied in simplified problems, while realistic factory models were considered only in deterministic settings. Future work should aim to bridge this gap by developing uncertainty-aware models tailored to industrial-scale instances.

A clear roadmap for future research or an industry-proof algorithm is to integrate the discrete-event simulation model presented in Chapter 5 with the constraint programming model described in Chapter 6. In such a framework, schedules generated by the CP model can be evaluated using the simulation model across a range of scenarios that capture stochastic variability in the system parameters. To maintain computational efficiency, this integrated approach can leverage the building blocks developed in this thesis to construct effective simheuristics. In particular, strategies such as the rolling-horizon approach (Chapter 5) and the adaptive simheuristic framework (Chapter 3) provide effective mechanisms for balancing solution quality and computational effort.

Definition of uncertainty

In Chapters 2–4, we made specific assumptions about how uncertainty is defined. Either by a fixed set of scenarios, or a (bounded) distribution. In practice, however, estimating uncertainty is challenging. For example, in the factory setting, historical data on processing times is limited in size. This raises a fundamental question about what it means to find an “optimal” solution to a stochastic combinatorial optimization problem, since the notion of optimality is highly sensitive to the assumptions made about the underlying

uncertainty. This limitation underscores the importance of developing approaches that remain robust under imperfect definitions of uncertainty.

It could also be questioned why we focused on the expected value of an objective; for some applications, an expected value at risk may be more suitable. This depends strongly on the level of risk practitioners are willing to accept. For example, achieving a KPI with 90% certainty may be more valuable than simply optimizing its expected value. In summary, because of the imperfect definitions of uncertainty, we argue that, in practice, it is most important to explain the potential impact of uncertainties on practitioners rather than claiming to know the optimal solution.

Mixed-integer linear programming

We observed that for the scheduling instances from the literature (that combine sequencing and assignment decisions), constraint programming scaled considerably better than mixed-integer linear programming. For this reason, we excluded mixed-integer linear programming from the main experiments presented in this thesis. This choice, however, introduces a limitation: we did not perform a comprehensive benchmarking of constraint programming against mixed-integer linear programming. Moreover, we did not include stochastic programming decomposition techniques for mixed-integer linear programming, which could have provided an additional point of comparison to our algorithms.

Gaps with the real factory

Several other aspects that could influence the scheduling operations in the real factory were left out of scope.

While making modeling decisions for the real factory, we noted a limitation of all scheduling models: it is difficult to capture a factory's objectives in a single function. In Chapter 5, we looked at a combination of makespan and deadlines, because we solved longer horizon problems, in which deadlines are present but not super critical. To illustrate this, it is acceptable to schedule a product for January in February, but not in December, which is reflected in our rolling-horizon approach. In Chapter 6, we examined shorter horizons and thus modeled only the makespan problem. In practice, planners and schedulers can adjust the weights of the objectives in our algorithms as desired or even incorporate hard deadlines into the optimization.

Furthermore, while modeling the factory, we made some simplifying assumptions. First, we did not include the mixing and splitting decisions as decision variables in our CP model in Chapter 6; instead, we assumed that the recipe determines how a batch is divided into subbatches and reflected this in our constraints. Including these decisions could improve the schedule's efficiency, though it might also reduce the scalability of the algorithms.

Additionally, we did not account for raw material availability, maintenance activities, machine breakdowns, or workforce constraints (e.g., tasks that cannot be performed during lunch breaks or at night). Even without these constraints, we demonstrated how challenging these instances are for state-of-the-art CP solvers (Chapter 6). However, to employ our tools in practice, these constraints must be addressed.

Finally, we focused on uncertainty in process durations; however, in the factory, other sources of uncertainty can also impact scheduling. In biomanufacturing, uncertainty in product yield and product quality is highly relevant, and the fact that we did not study this is a major limitation of our work. Finally, we considered the demand as given, which, based on interviews with planners, shows considerable variability; thus, future work could elaborate on this.

7.3. Future work

This section offers a broader reflection on potential directions for future research that build upon the findings of this thesis.

Instance space analysis scheduling under uncertainty

This thesis presented several procedures for handling uncertainty in scheduling. A promising direction for future research is to conduct an extensive instance-space analysis on a broader set of scheduling problems to understand under which scheduling conditions which methods work best. Such a study could lead to a better understanding of how scheduling constraints, objectives, instance size, and uncertain parameters affect the effectiveness of different stochastic scheduling methods.

Rescheduling

We provided methods for handling temporal uncertainty and demonstrated how real-time algorithms can adjust schedules. In practice, additional rescheduling or repair procedures may be required to effectively manage the inherent unpredictability and day-to-day fluctuations of real-world operations. We identify an open research direction in defining scheduling objectives while rescheduling. For instance, when disruptions such as batch failures occur or new customer orders arrive after the monthly schedule has been established, it is important to consider the appropriate objective of rescheduling. Should the goal be to re-optimize the schedule to minimize the makespan, or rather to maintain stability by limiting deviations from the existing plan? This choice of objective directly impacts the design of rescheduling algorithms and their performance evaluation. Addressing this topic requires close collaboration with practitioners to understand the trade-offs they face between optimality, stability, and operational feasibility. Explicitly defining and quantifying rescheduling objectives could open new research directions and motivate the development of algorithms that better reflect real-world scheduling needs.

Fundamental questions in temporal networks

Building on the foundations established in this thesis, several promising research directions emerge regarding the use of temporal networks for scheduling. While this work focused on simple temporal networks with uncertainty (STNUs), extending these concepts to probabilistic simple temporal networks (PSTNs) offers an opportunity to capture more realistic, data-driven uncertainty through unbounded probability distributions of process durations. This also requires collecting real-world data from the running factory

and carries the risk of bias. Thus, such an extension would introduce new challenges related to learning appropriate probability distributions from historical production data.

Moreover, the methods introduced in this thesis provide a basis for developing algorithms that automatically construct partial-order schedules and translate them into PSTNs, potentially following a simheuristic or decomposition-based constraint programming approach. These methods could iteratively add constraints to optimize the dynamic controllability of the resulting networks.

Finally, the real-time execution framework (RTE*) can be further developed and extended with repair-based strategies to handle infeasibilities during execution—an important step towards dynamic control in uncertain environments and closely connected to the broader rescheduling problem discussed earlier.

Biotechnological considerations

A key direction for future research is to build on this thesis by integrating biotechnological process knowledge more directly into scheduling optimization. For example, bioreactors are equipped with real-time sensors that monitor the fermentation process. Leveraging such data earlier in the process could enable delays in fermentation to be detected in real-time and incorporated directly into scheduling decisions.

Furthermore, we investigated the relationship between the no-wait constraints and computation time in CP-based scheduling. In practice, a relationship exists between waiting time and product quality: longer waits often lead to a decline in quality. Exploring this relationship and embedding it into scheduling models offers promising opportunities to design schedules that are responsive to biotechnological dynamics.

Impact in practice

Most importantly, future work should focus on translating the developed methods into tools that can have a tangible impact in real biomanufacturing environments. Achieving this will require close collaboration with practitioners. In addition, it is important to embed these methods within a broader operations management framework, clearly identifying which planning decisions they are intended to support and which objectives they aim to achieve. Such a perspective requires an understanding of how different planning decisions interact across organizational levels and time horizons. Inspiration for further discussion on this integration can be found in (Shapiro, Rangan, and Sviokla 1992). In particular, for optimization algorithms to be successful in practice, a coherent business control framework is required that clarifies how planning decisions across different organisational levels are connected and aligned with their respective planning horizons and who is the responsible decision-maker for each planning decision.

Within such a framework, the algorithms and tools developed in this research can effectively support decision-making at multiple levels of the planning process. We see clear potential for 1) strategic analysis, via what-if simulations for capacity and investment through simheuristic approaches such as explored in Chapter 3 and Chapter 5; 2) tactical planning, by linking CP-based scheduling models (Chapter 6) with demand and inventory information; 3) operational day-to-day scheduling, combining CP (Chapter 6) and simulation to generate robust schedules; and 4) real-time operation, by progressing

toward digital-shadow or digital-twin architectures that can incorporate real-time data and adaptive algorithms such as explored in Chapter 4.

A key opportunity lies in advancing our constraint programming (CP) model from Chapter 6 into an industrial tool that integrates with existing planning systems. To complement this, a simulation component—such as the discrete-event simulation tool developed by Systems Navigator—could be utilized to assess the robustness of CP-generated schedules under uncertainty, incorporating methods for machine selection, sequencing, and start-time assignment. Reliable data on historical process durations will be essential for such simulation-based evaluation.

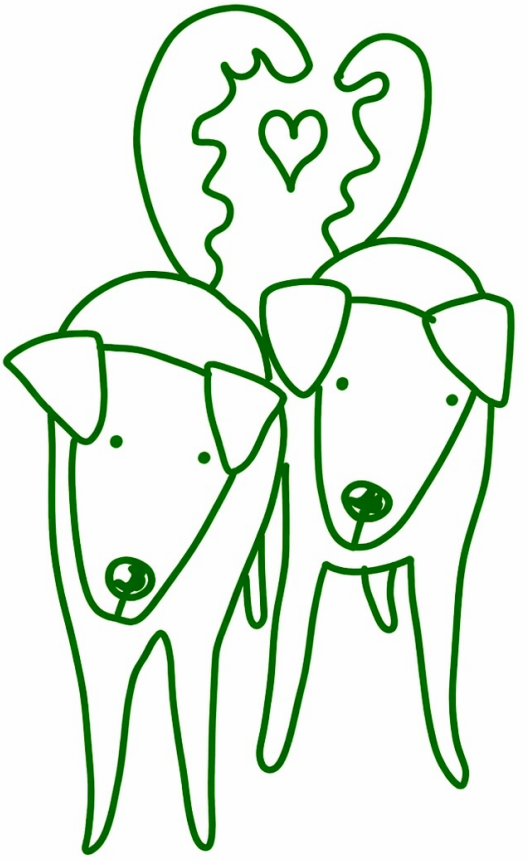
It is crucial that our algorithms are embedded in user-friendly tools that support meaningful human interaction. For instance, the tool could be used to provide scheduling suggestions and help the user make manual adjustments through an intuitive, visual interface.

Our vision is that these tools should simplify schedulers' lives by automating the complex, combinatorial computations that are difficult to perform manually, while ensuring the final decision remains in the user's hands. Therefore, such tools must be developed with human decision-making at their core. We hope that this work provides a valuable building block in that direction and helps to bridge the gap between academic research and industrial impact.

References

Hunsberger, L. and R. Posenato (2024). “Foundations of dispatchability for simple temporal networks with uncertainty”. In: *Proceedings of 16th International Conference Agents and Artificial Intelligence 2024*. Vol. 2, pp. 253–263.

Shapiro, B. P., V. K. Rangan, and J. J. Sviokla (1992). “Staple yourself to an order”. In: *Harvard Business Review* 70.4, pp. 113–122.



Dankwoord

Dit dankwoord schrijvend besef ik dat het echt zo ver is: de eindstreep is in zicht. Dit was mij nooit gelukt zonder de steun en het vertrouwen die ik heb gekregen van mijn promotoren gedurende het hele project, ook tijdens een wat moeizame opstartfase. Ook tijdens de laatste zware loodjes kon ik op jullie rekenen. Mathijs, bedankt voor je snelle reacties en het altijd willen meedenken, of het nou over de opzet van een introductie, of nadenken over een volgende stap in werk is. Ik ga het missen om je om advies te kunnen vragen—al sluit ik niet uit dat ik het toch nog af en toe zal doen. David, jouw interesse en gezelligheid als copromotor heb ik enorm gewaardeerd. Samen de materie in duiken om te kijken of we het écht begrepen of goed hadden opgeschreven vond ik altijd leuk. Jij kwam telkens met precies die vragen waar ik zelf nog niet op was gekomen. En dan, Esteban, wat was jij een fijne toevoeging aan mijn begeleidersteam. Jouw energie en enthousiasme zijn erg aanstekelijk en het was een feestje om met jou samen te mogen werken. Jouw vrolijke en persoonlijke noot bracht precies de warmte die ik nodig had.

Behalve met mijn directe begeleidersteam heb ik met heel veel mensen kunnen samenwerken. Léon, ik ben jou eeuwig dankbaar voor de rol van tolk tussen mij en de computer. Ik heb heel veel van je geleerd en je hebt mij op veel momenten uit de brand geholpen op het gebied van programmeren. Ook ben ik heel trots op ons gezamenlijke paper! Bovendien was het altijd heel gezellig om samen op kantoor te werken en de musical was natuurlijk een hoogtepunt!

Eva, I would not have made nearly as much progress in my PhD without you. I'd love to hear when you're nearby again for a coffee or lunch. My thanks also go to the rest of the Systems Navigator team, including Rienk and Alessandro, for your involvement in the capacity model and in the brainstorming sessions we organized with AI4b.io. Also, great thanks to Ana Catherina for brainstorming with me on planning topics, and to all dsm-firmenich colleagues working at the factory in Seclin.

I am also very grateful to all my AI4b.io colleagues for all the great sessions we've had together—you've proven that people in Delft really know how to enjoy a good drink. I will truly miss the beers and bitterballen at De Gist. Renger, thank you very much for your initiatives in this regard and for all the organization around AI4b.io. I'm sure we'll run into each other again on our road bikes—maybe even during a lap around Flakkee. Paul, Mahdi, Chengyao, Joery: it was wonderful to share this adventure with you, and I'm very curious to see where your paths will lead in the future. Also, Henk, thank you very much for being involved in the first phase of my PhD and keeping me sharp.

Dear members of my PhD committee, Prof. Henk Noorman, Dr. Esteban Freydell, Dr.

Michele Lombardi, Dr. Yingqian Zhang, and Dr. Marjan van den Akker, I would like to thank you for your willingness to serve on my PhD committee and for the feedback on my thesis. It helped me reach this final version.

Karin, dankjewel voor de prachtige illustraties. Ik word er heel vrolijk van en door het printen van het boekje met jouw leuke tekeningen voelt het eindelijk echt af!

Noah, dankjewel dat jij mijn fijne collega en kantoormaatje bent geweest de afgelopen jaren. Heerlijk om te kletsen, vooral als het over hardlopen gaat! Jouw optimisme en discipline zijn inspirerend! Eggie, dankjewel voor jouw luisterend oor, shiny persoonlijkheid en geweldige humor. Ik vind het nu de hoogste tijd om samen wat meer glitter en glamour op te zoeken dan we ooit in de schedulingliteratuur zullen vinden. Daniel, bedankt voor het warmhouden van het koffiezetapparaat. Ik vind het zeer bewonderenswaardig hoe iemand die zo van goede koffie houdt met zoveel enthousiasme naast het TU Delft-koffiezetapparaat kan staan. Ook zal ik de basicfitsessie met jou en Clinton niet snel vergeten! Maaike, Issa, Sterre, succes met de laatste loodjes van jullie PhD. Ik heb alle vertrouwen in jullie girlpower! Ook al mijn andere fijne collega's van de Algoritmiekafdeling hebben mijn tijd hier een stuk gezelliger gemaakt. De sociale commissie heb ik met veel plezier op mij genomen en ik bied mijn excuses aan voor de vele sportactiviteiten op het programma.

Ook in Amsterdam heb ik mij altijd welkom gevoeld tijdens mijn bezoekjes aan de VU. Bernd en Joost, ik wil jullie bedanken voor de gastvrijheid en de tijd die jullie in ons project samen hebben gestoken. Bernd, ik wil jou in het bijzonder bedanken voor jouw enthousiasme vanaf mijn masterscriptie en ik vind het ontzettend leuk dat wij contact hebben gehouden tijdens mijn PhD, met als hoogtepunt het feestje in Delft, inclusief stadstour. Ook Leon Lan, bedankt voor het sparren en de gezelligheid op de VU tijdens de laatste loodjes van ons promotietraject. Ik ben een groot fan van PyJobShop! Ik wens je alle succes met je eigen bedrijf en hoop je zeker nog te spreken!

Unforgettable is my trip to Bologna. I immediately felt part of the AI group and of Italian life. I will never forget the cappuccino at the bar and the lunches in the mensa, including broccoli. Gaetano and Federico, thank you for being such great colleagues and for including me in the great life of Bologna, including aperitivo—we got to know each other quickly! Michele, your passion for research is inspiring—I wish you every success and hope to see you again in Bologna. The wonderful bike rides I took on the weekends with Bene and Tesse are something I will never forget. Of course, the absolute highlight was the lunch at Mattia's family home, made by nonna—thank you so much for that!

En dan mijn paranimfen, dapper dat jullie deze rol op jullie hebben genomen! Lieve Katja, natuurlijk ben jij mijn paranimf! Ik kijk uit naar alle koffietjes met jou, Elo en Abdel die er nog gaan volgen door ons volwassen (?) leven, en ik vind het heel bijzonder om straks jullie getuige te zijn! En, Marelot, ben jij na het lezen van al mijn hoofdstukken al overtuigd dat je toch een PhD wil doen? Ik zal je zeker aanmoedigen en steunen! Bedankt voor jouw eeuwige enthousiasme, bijvoorbeeld om mee te gaan op tripjes, ook al blijken

mijn (route)planningen soms nog niet heel robuust geoptimaliseerd. Je bent een grote steun geweest, ook weer de afgelopen tijd, en daar ben ik heel dankbaar voor. Ik zou geen beter persoon kunnen bedenken om mijn paranimf te zijn. De foutjes die je in mijn proefschrift tegenkomt moet je maar voor je houden trouwens :)

Ik wil mijn lieve familie en vrienden bedanken voor alle gezelligheid onderweg. De bezoekjes aan Goeredeede houden mij ook professioneel scherp als opa mij weer eens vraagt wat ik nou eigenlijk voor werk doe en waar dat onderzoek dan wel over gaat. En onderweg terug van Delft blijft Den Haag een geliefde stop dankzij mijn lieve vriendinnen Cee, Laar en Dien (de wijn staat altijd koud!), mijn lieve broer en zuslief Job en Iris. Al mijn lieve vriendinnen uit Amsterdam, inmiddels deels weer uitgevlogen, bedankt voor de vele borrels, etentjes, koffietjes, hardlooprondjes, fietstochten, stapavondjes (steeds schaarser), creamiddagen (zelfs die!) en zelfs de eerste babyshowers en bruiloften. Wat een rijkdom!

Lieve papa en mama, bedankt dat ik altijd mag opbellen, langskomen, aanschuiven. Het betekent veel!

En natuurlijk jij, lieve Martijn, dankjewel voor al je geduld en het in mij geloven. Jij staat het allerdichtst bij mij van iedereen en daar ben ik heel erg blij mee.

Curriculum Vitæ

Kim Cecilia van den Houten

14-06-1997	Born in ‘s-Gravenhage, The Netherlands.

Education

2020–2021	M.Sc. Econometrics and Operations Research, Vrije Universiteit Amsterdam, The Netherlands.
2016-2019	B.Sc. Econometrics and Operations Research, University of Amsterdam, The Netherlands.
2015-2016	Liberal Arts and Sciences (Major: Mathematics and Computer Science), Amsterdam University College, The Netherlands.

Awards

2021	Extrie Thesis Award 2021 (Vrije Universiteit Amsterdam).

List of Publications

1. K. van Houten, M. de Vos, E. van der Sluis, T. Schneider, and N. van Dijk (2021). “Roosteren vanuit endoscopisch perspectief”. In: *STAtOR* 22.2, pp. 29–33

2. K. van den Houten, E. van Krieken, and B. Heidergott (2022). “Analysis of measure-valued derivatives in a reinforcement learning actor-critic framework”. In: *2022 Winter Simulation Conference (WSC)*, pp. 2736–2747

3. N. Schutte*, E.-A. Eigbe*, and K. van den Houten* (2022). “Dynamic scenario reduction for simulation-based optimization under uncertainty”. In: *Workshop at IJCAI*. Vienna, Austria

4. K. Van Den Houten, M. De Weerdt, D. M. Tax, E. Freydell, E. Christoupoulou, and A. Nati (2023). “Rolling-horizon simulation optimization for a multi-objective biomanufacturing scheduling problem”. In: *2023 Winter Simulation Conference (WSC)*. IEEE, pp. 1912–1923

5. K. van den Houten, D. M. Tax, E. Freydell, and M. de Weerdt (2024). “Learning from scenarios for repairable stochastic scheduling”. In: *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, pp. 234–242

6. K. Van den Houten, L. Planken, E. Freydell, D. M. Tax, and M. De Weerdt (2025). “Proactive and reactive constraint programming for stochastic project scheduling with maximal time-lags”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 39. 25, pp. 26534–26541