Proactive and Reactive Constraint Programming for Stochastic Project Scheduling with Maximal Time-Lags

Kim van den Houten [1], Léon Planken [1], Esteban Freydell [2], David M.J. Tax [1], Mathijs de Weerdt [1]

[1]Delft University of Technology, Delft, The Netherlands.
[2]dsm-firmenich, Delft, The Netherlands.
k.c.vandenhouten@tudelft.nl, l.r.planken@tudelft.nl, esteban.freydell@dsm-firmenich.com, d.m.j.tax@tudelft.nl, m.m.deweerdt@tudelft.nl

Abstract

This study 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 reinvoked interest in evaluating the advantages and drawbacks of various proactive and reactive scheduling methods. 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.

Code —
github.com/kimvandenhouten/AAAI25_SRCPSPmax
Extended version — www.arxiv.org/abs/2409.09107

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 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 2024) 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). Additionally, a recent comparison between solvers for CP and MIP demonstrated CP Optimizer's superiority over CPLEX across a broad set of benchmark scheduling problems (Naderi, Ruiz, and Roshanaei 2023). These advances, however, have not yet been explored for SRCPSP/max, despite potential applications. CP can be used for finding robust proactive schedules or within a reactive approach with rescheduling during execution, which has been viewed as too computationally heavy (Van de Vonder, Demeulemeester, and Herroelen 2007).

A proper benchmarking paper in which a statistical analysis is performed on 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 maximal time lags and stochastic durations, methods can fail by violating 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) even criticize their own evaluation metric and indicate that future work should

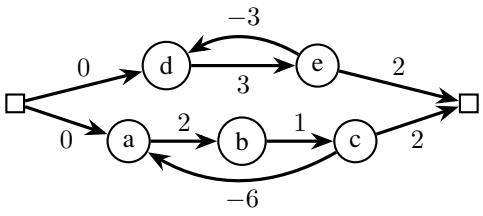

Figure 1: Example project graph (Schutt et al. 2013).

provide a more extensive comparison. We argue that statistical tests dealing with infeasibilities correctly are needed to compare different methods properly.

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 2024). A new reactive approach with complete rescheduling, which has not been investigated before as far as we know, is included in the comparison too. We target the existing research gap due to the lack of a clear and informative comparison among various methods, examining aspects such as infeasibility, solution quality, and computation time (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.

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 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 2.

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. The Stochastic RCPSP/max (SRCPSP/max) is an extension in which the durations follow a stochastic distribution (Fu et al. 2012). We define the problem as follows: the activity durations are independent random variables indicated by d_j representing the duration of activity j. We assume that each d_j follows a discrete uniform distribution, $d_j \sim \text{DiscreteUniform}(lb_j, ub_j)$ where lb_j, ub_j denote the minimum and maximum possible durations for activity j. Each duration becomes available when an activity finishes.

In general, comparing methods for this problem is not straightforward as while comparing method A and B, it could be that only method A obtains a feasible solution. Looking at double hits (instances solved by both methods) is a reasonable choice, but neglects that potentially one of the two methods solved more problem instances than the other. Earlier work on this stochastic variant introduces the α-robust makespan (Fu et al. 2012), that is the expected makespan value for a scheduling strategy for which the probability that the schedule is feasible is at least $1 - \alpha$, as a metric for comparing different methods but also indicated the limitations of this metric.

3 Background and Related Work

In this section, we discuss existing approaches for SRCPSP/max. Furthermore, we introduce all concepts that are needed to understand our scheduling methods that are presented in Section 4. We introduce proactive techniques based on Sample Average Approximation in Section 3.1. We explain partial order schedules and temporal networks in Section 3.2. Finally, Section 3.3 gives an overview of related work on benchmarking scheduling methods for SRCPSP/max.

3.1 Proactive Scheduling

Proactive scheduling methods aim to find a robust schedule offline by taking information about the uncertainty into account (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 $\alpha\%$ 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 nowadays CP is the state of the art for deterministic project scheduling (Naderi, Ruiz, and Roshanaei 2023), we re-investigate SAA approaches for SRCPSP/max with CP and compare this CP-based proactive approach to reactive approaches.

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 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 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 et al. 2012).

Temporal Networks Partial order schedules are often complemented with a temporal model to reason over the temporal constraints. Most POS approaches rely on a Simple Temporal Network (STN), which is a graph consisting of time points (nodes) and temporal difference constraints (edges). The Simple Temporal Network with Uncertainty (STNU) extends the STN by introducing *contingent links*. The duration of these contingent constraints can only be observed, while for regular constraints it can be determined during execution. The works by Lombardi and Milano (2009) and (Lombardi, Milano, and Benini 2013) on POS are the only ones that we know of using STNUs as their temporal model. They introduce nodes for the start and the end of each activity with duration constraints between them, connecting respective start nodes with edges for the precedence constraints.

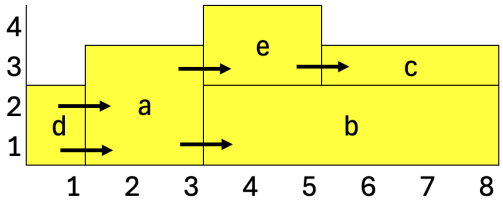

Figure 2: Example Gantt chart (figure adjusted from Fu et al. 2012). The X-axis is time, and Y-axis shows resource demand. **The arrows** indicate an example of generating a POS with chaining, starting from a fixed solution schedule.

Execution Strategies An STNU is *dynamically controllable* (DC) if there is a strategy to determine execution times for all controllable (non-contingent) time points which ensures that all temporal constraints are met, regardless of the outcomes of the contingent links. Some DC-checking algorithms generate new so-called wait edges to make the network DC (Morris 2014).

The DC STNU with wait edges is referred to as an Extended STNU (ESTNU) and is input to a Real-Time Execution Algorithm. Such an 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. An algorithm specifically tailored to ESTNUs is RTE* (Hunsberger and Posenato 2024; Posenato 2022).

As far as we know, the DC-checking and RTE* algorithms have not been applied to SRCPSP/max, despite their efficiency. (Lombardi and Milano 2009; 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 (Policella et al. 2007; Fu, Varakantham, and Lau 2016). Thus, we conclude that there is a research gap in applying the developments in STNU literature to SRCPSP/max.

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 (Policella et al. 2004; Fu et al. 2012; Varakantham, Fu, and Lau 2016; Fu, Varakantham, 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 et al. 2004, 2007), solver performance (Lombardi and Milano 2009; Lombardi, Milano, and Benini 2013), or the α-robust makespan (Fu et al. 2012; Varakantham, Fu, and Lau 2016; Fu, Varakantham, 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 Scheduling Methods

This section outlines the proposed methods. We explain how to use CP for these scheduling problems so far dominated by MIP approaches. Note that in the Technical Appendix (Van den Houten et al. 2024), we include a comparison between a CP and a MIP approach for RCPSP/max (CPOptimizer and CPLEX; IBM (2024)) demonstrating that CP outperforms MIP, which is in line with the literature.

First, we present a deterministic CP model for RCPSP/max in Section 4.1. The new, stochastic methods for SRCPSP/max are proposed in Section 4.2. 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 4.3.

4.1 Constraint Programming for RCPSP/max

The CP model is:

$$\text{Min } \mathit{Makespan} \quad \text{subject to}$$
$$\max\{\text{end}(x_j)\} \leq \mathit{Makespan}; \forall j \in J$$
$$\text{start}(x_i) \geq min_{j,i} + \text{start}(x_j); \forall j \in J \; \forall i \in S_j$$
$$\text{start}(x_i) \leq max_{j,i} + \text{start}(x_j); \forall j \in J \; \forall i \in S_j$$
$$\sum_{j \in J} \text{Pulse}(x_j, r_{r,j}) \leq c_r; \forall r \in R$$
$$x_j : \text{IntervalVar}(J, d_j); \forall j \in J$$

For the deterministic RCPSP/max, we use the modern interval constraints from the IBM CP optimizer (IBM 2024). In the equations above, we use IBM's syntax and modify the RCPSP example from Laborie et al. (2018) to RCPSP/max. We use the earlier introduced nomenclature together with the minimal time lags $min_{j,i}$ and maximal time lags $max_{j,i}$ that are the temporal differences between start times of activities j and i if i is a successor of j. 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 $r_{r,j}$. The aggregated pulse values are constrained so that their sum does not exceed the total available resource capacity c_r.

4.2 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 4.1, 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 schedule $s_j \; \forall j \in J$ that is feasible for all scenarios it has seen if one exists:

$$\text{Min} \frac{1}{|\Omega|} \sum_{\omega \in \Omega} \mathit{Makespan}(\omega) \quad \text{subject to}$$
$$\text{Max}_{\text{j}}(\text{end}(\text{x}_{\text{j}}^{\omega})) \leq \mathit{Makespan}(\omega); \forall \omega \in \Omega$$
$$\text{start}(x_i^{\omega}) \geq min_{j,i} + \text{start}(x_j^{\omega}); \forall j \in J \; \forall i \in S_j \; \forall \omega \in \Omega$$
$$\text{start}(x_i^{\omega}) \leq max_{j,i} + \text{start}(x_j^{\omega}); \forall j \in J \; \forall i \in S_j \; \forall \omega \in \Omega$$
$$\sum_{j \in J} \text{Pulse}(\text{x}_{\text{j}}^{\omega}, \text{r}_{\text{r,j}}) \leq \text{c}_{\text{r}}; \forall r \in R \; \forall \omega \in \Omega$$
$$x_j^{\omega} : \text{IntervalVar}(\text{J}, \text{y}_{\text{j}}^{\omega}); \forall j \in J \; \forall \omega \in \Omega$$
$$s_j = \text{start}(x_j^{\omega}); \forall j \in J \; \forall \omega \in \Omega$$

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 2024) which we call stnu. This approach is inspired by many earlier works (Policella et al. 2007; Lombardi and Milano 2009; Fu, Varakantham, and Lau 2016). 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 $\widehat{d}$ and using the chaining procedure, which was explained in Section 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 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 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 (2024): 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 Technical Appendix we include a step-by-step example of the STNU-based method.

4.3 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 test to the Wilcoxon test that is 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 our Technical Appendix.

5 Experimental Evaluation

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

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 (Varakantham, Fu, and Lau 2016; Fu, Varakantham, and Lau 2016) by including 50 instances each from the ubo50 and ubo100 sets. Following prior research, we use deterministic durations to set up stochastic distributions with different noise levels, converting deterministic instances into stochastic ones. We use uniform discrete distributions for durations based on the deterministic processing times d_j. Specifically, we define the lower bound as $lb_j = \max(1, \text{round}(d_j - \epsilon \cdot \sqrt{d_j}))$ and the upper bound as $ub_j = \text{round}(d_j + \epsilon \cdot \sqrt{d_j})$, 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 (i.e. the deterministic problem with perfect information is infeasible for the given activity duration sample).

5.2 Tuning of the Methods

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

All CP models are solved with the IBM CP solver (IBM 2024) with default settings. Most of the deter-

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)	[330] -11.36 (*)	[330] -11.39 (*)	[330] -11.35 (*)	[370] -2.86 (*)	[370] -6.73 (*)	[370] -6.01 (*)
	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

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)	[330] -36.15 (*)	[370] -72.56 (*)	[330] -36.15 (*)	[370] -72.56 (*)	[330] -5.64 (*)
	norm. avg.	react: 0.36	react: 0.24	pro$_{0.9}$: 0.36	pro$_{0.9}$: 0.24	stnu: 0.82
	norm. avg.	stnu: 1.64	pro$_{SAA}$: 1.76	stnu: 1.64	pro$_{SAA}$: 1.76	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)	[330] -3.95e3 (*)	[330] -3.98e3 (*)	[370] -3.07e4 (*)	[370] -3.14e4 (*)	[330] -142.76 (*)
	norm. avg.	pro$_{0.9}$: 0.01	pro$_{SAA}$: 0.01	pro$_{0.9}$: 0.0	pro$_{SAA}$: 0.0	stnu: 0.15
	norm. avg.	stnu: 1.99	stnu: 1.99	react: 2.0	react: 2.0	react: 1.85

Table 1: Pairwise test results for ubo50, $\epsilon = 1$, covering solution quality and runtime. Metrics include Wilcoxon, proportion, and magnitude tests (Section 4.3). 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 the Technical Appendix. Column headers list the compared methods, excluding zero-difference pairs.

ministic 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 proactive$_{SAA}$ and a heuristic approach proactive$_{\gamma}$. We used $\gamma = 0.9$ for proactive$_{\gamma}$. For proactive$_{SAA}$, 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 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 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.

5.3 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 1 (the remaining results are included in the Technical Appendix). We present summarized partial orderings on 1) solution quality, 2) time offline, 3) time online.

Feasibility Ratio We first analyze the obtained feasibility ratios in Table 2. For $\epsilon = 1$, the feasibility ratios of proactive$_{SAA}$, proactive$_{0.9}$, and reactive are similar for instance sets j10-ubo50, with reactive slightly lower for ubo100. The stnu method has the lowest feasibility rate, but the difference narrows as problem size increases. For $\epsilon = 2$, the stnu achieves the highest feasibility ratios due to better handling of larger variances in durations.

Set	$\epsilon = 1$				$\epsilon = 2$			
	stnu	pro$_{SAA}$	pro$_{0.9}$	react	stnu	pro$_{SAA}$	pro$_{0.9}$	react
10	0.65	0.85	0.85	0.85	0.63	0.63	0.64	0.63
20	0.65	0.76	0.76	0.76	0.63	0.54	0.53	0.53
30	0.78	0.89	0.89	0.89	0.63	0.51	0.48	0.49
50	0.77	0.86	0.86	0.86	0.67	0.41	0.42	0.41
100	0.84	0.91	0.91	0.88	0.79	0.35	0.36	0.33

Table 2: Feasibility Ratios for $\epsilon = 1$ and $\epsilon = 2$. Set 10, 20 and 30 refer to j10, j20 and j30, and set 50 and 100 refer to ubo50 and ubo100.

Results Statistical Tests Table 1 shows a subset of the pairwise test results for the metrics solution quality, offline time, and online time. In our Appendix, we provide all test results per instance set / noise level ϵ setting. The outcomes of 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 1, the normalized

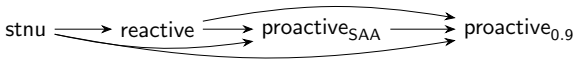

Figure 3: Summarizing illustration of the partial ordering of the different methods for solution quality.

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 3–5, 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 the Technical Appendix for the partial orderings per setting including the distinction between a strong and weak partial ordering.

5.4 Analysis

Figure 3 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 3 also a significant magnitude difference was found on the double hits.

Figure 4 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 2) 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 we included it in the Appendix).

We observe the general partial ordering for **online runtime** in Figure 5. 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,

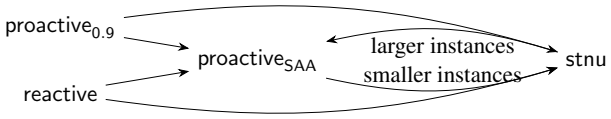

Figure 4: Summarizing illustration of the partial ordering of the different methods for time offline.

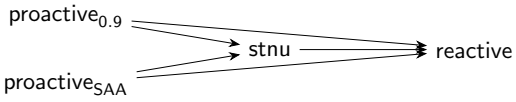

Figure 5: Summarizing illustration of the partial ordering of the different methods for time online.

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$_{SAA}$.

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 proactive$_{SAA}$ and proactive$_{\gamma}$. We find that the algorithm stnu that uses partial order schedules outperforms the other methods on solution quality in our evaluation. Although in general, proactive$_{\gamma}$ and reactive have better offline computation time than stnu, and proactive$_{SAA}$ and 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.

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 that are defined from end-to-start can be an interesting problem domain for future work. The stnu method can extended in multiple directions, e.g. rearranging the order of the jobs online or using probabilistic STNs are both interesting for future work.

Furthermore, the set of methods could even be broadened by including sequential approaches (Powell 2022) or machine learning-based methods like the graph neural network in (Teichteil-Königsbuch et al. 2023) for SRCPSP.

Acknowledgements

This work 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).

References

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

Cureton, E. E. 1967. The normal approximation to the signed-rank sampling distribution when zero differences are present. *Journal of the American Statistical Association*, 62(319): 1068–1069.

Fu, N.; Lau, H.; Varakantham, P.; and Xiao, F. 2012. Robust Local Search for Solving RCPSP/max with Durational Uncertainty. *Journal of Artificial Intelligence Research (JAIR)*, 43: 43–86.

Fu, N.; Varakantham, P.; and Lau, H. 2016. Robust Partial Order Schedules for RCPSP/max with Durational Uncertainty. *Proceedings of the International Conference on Automated Planning and Scheduling*, 26: 124–130.

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

Hunsberger, L.; and Posenato, R. 2024. Foundations of Dispatchability for Simple Temporal Networks with Uncertainty. In *Proceedings of 16th International Conference Agents and Artificial Intelligence 2024*, volume 2, 253–263.

IBM. 2024. *IBM ILOG CPLEX Optimization Studio*. IBM, Armonk, NY.

Kanji, G. K. 2006. 100 statistical tests.

Kleywegt, A. J.; Shapiro, A.; and Homem-de Mello, T. 2002. The sample average approximation method for stochastic discrete optimization. *SIAM Journal on optimization*, 12(2): 479–502.

Kolisch, R.; and Sprecher, A. 1996. PSPLIB - a project scheduling problem library. *European Journal of Operational Research*, 205–216.

Laborie, P. 2015. Modeling and Solving Scheduling Problems with CP Optimizer.

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

Lgelmund, G.; and Radermacher, F. J. 1983. Algorithmic approaches to preselective strategies for stochastic scheduling problems. *Networks*, 13(1): 29–48.

Lombardi, M.; and Milano, M. 2009. A Precedence Constraint Posting Approach for the RCPSP with Time Lags and Variable Durations. 569–583. ISBN 978-3-642-04243-0.

Lombardi, M.; Milano, M.; and Benini, L. 2013. Robust Scheduling of Task Graphs under Execution Time Uncertainty. *Computers, IEEE Transactions on*, 62: 98–111.

Long, D.; and Fox, M. 2003. The 3rd international planning competition: Results and analysis. *Journal of Artificial Intelligence Research*, 20: 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*, 464–479. Springer.

Naderi, B.; Ruiz, R.; and Roshanaei, V. 2023. Mixed-Integer Programming vs. Constraint Programming for Shop Scheduling Problems: New Results and Outlook. *INFORMS Journal on Computing*, 35.

Policella, N.; Cesta, A.; Oddi, A.; and Smith, S. 2007. From precedence constraint posting to partial order schedules: A CSP approach to Robust Scheduling. *AI Communications*, 20: 163–180.

Policella, N.; Smith, S. F.; Cesta, A.; and Oddi, A. 2004. Generating Robust Schedules through Temporal Flexibility. In *ICAPS*, volume 4, 209–218.

Posenato, R. 2022. CSTNU Tool: A Java library for checking temporal networks. *SoftwareX*, 17: 100905.

Powell, W. B. 2022. *Reinforcement Learning and Stochastic Optimization: A Unified Framework for Sequential Decisions*. John Wiley & Sons, Inc.

Schutt, A.; Feydy, T.; Stuckey, P. J.; and Wallace, M. G. 2013. Solving RCPSP/max by lazy clause generation. *Journal of scheduling*, 16: 273–289.

Teichteil-Königsbuch, F.; Povéda, G.; de Garibay Barba, G. G.; Luchterhand, T.; and Thiébaux, S. 2023. Fast and robust resource-constrained scheduling with graph neural networks. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 33, 623–633.

Van de Vonder, S.; Demeulemeester, E.; and Herroelen, W. 2007. A classification of predictive-reactive project scheduling procedures. *Journal of Scheduling*, 10: 195–207.

Van den Houten, K.; Planken, L.; Freydell, E.; Tax, D. M.; and De Weerdt, M. 2024. Proactive and Reactive Constraint Programming for Stochastic Project Scheduling with Maximal Time-Lags. *arXiv:2409.09107*.

Varakantham, P.; Fu, N.; and Lau, H. 2016. A Proactive Sampling Approach to Project Scheduling under Uncertainty. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30.