Comparing Dynamic Scheduling Algorithms for Multi-Mode RCPSP/max under Uncertainty

A Comparative Analysis on the Proactive, Reactive, and STNU algorithms with Generalised Time-Lags and No-Wait Constraints

Jeffrey Meerovici[1]

Supervisor(s): Mathijs de Weerdt[1], Kim van den Houten[1], Léon Planken[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 22, 2025

Name of the student: Jeffrey Meerovici
Final project course: CSE3000 Research Project
Thesis committee: Mathijs de Weerdt, Kim van den Houten, Léon Planken, Jasmijn Baaijens

Abstract

This study investigates the performance of three dynamic scheduling approaches—proactive, reactive, and STNU-based—for solving the Multi-Mode Resource-Constrained Project Scheduling Problem with maximal time-lags and no-wait constraints (MMRCPSP/max) in uncertain environments. The performance of the approaches is validated on three key performance measures: solution quality (makespan), offline computation time, and online computation time. Drawing from the current work on stochastic RCPSP/max, this research introduces a more realistic formulation of the problem with multi-mode execution, generalised time lags, and no-wait constraints, under the PyJobShop library and stochastic duration modelling. Experimental results, based on altered PSPLIB instances, show that the three algorithms yield similar feasibility rates. However, they show distinct trade-offs in terms of solution quality, time offline, and time online. The proactive algorithm requires the shortest offline and online computational times, but provides slightly worse makespans. The STNU-based algorithm produces the best average solution quality, with significantly higher offline time. The reactive algorithm offers competitive offline times and solution quality, but has the largest online computational time. The findings provide insight into the trade-offs in solution quality and computational time in dynamic project scheduling and offer insight for using the strategies in the real world.

1 Introduction

Constraint programming comprises a class of NP-hard algorithms that are key to solving industrial problems, particularly the resource-constrained project scheduling problem (RCPSP) [10][17]. RCPSP aims to minimise the makespan (total duration of a schedule) by scheduling task start times that respect precedence constraints (tasks that must be completed before others can begin) and resource availability while ensuring non-preemptiveness (once an activity starts, it must run to completion without interruption) [1].

RCPSP has been applied in various domains, such as make-to-order production scheduling, batch scheduling in the process industry, and assembly line balancing [2]. Using RCPSP, efficient and high-quality scheduling solutions can be found, leading to significant improvements in operational performance.

An extension to the RCPSP problem is the Multi-Mode RCPSP/max with generalised time-lag and no-wait constraints. The RCPSP/max variant extends the classical RCPSP by allowing precedence constraints between activities that include minimum and maximum time lags [13].

The Multi-Mode RCPSP extends the classical RCPSP by allowing tasks to be executed in multiple modes, each with different durations and resource requirements [18]. This added flexibility better reflects real-world scenarios (for example, increasing the number of workers can reduce task duration), but also significantly increases the search space, making it more challenging for algorithms to find optimal solutions.

Generalised time-lags introduce additional constraints between task pairs by specifying minimum and maximum time differences between their start and/or finish times. Although this added control increases scheduling flexibility, it also imposes more constraints on problem instances [14].

The no-wait constraint is a special case of generalised time-lag constraints, where the time lag between the end of one task and the start of the next is zero, hence enforcing that a task must start immediately after the preceding task finishes. The no-wait constraint can be modelled by creating an end-to-start constraint from task A to task B of 0 lag and creating a start-to-end constraint from task B to task A of 0 lag.

In real-world applications, external factors often introduce uncertainty, leading to variability in the duration of tasks. Dynamic constraint programming is an extension of constraint programming that accounts for this uncertainty by modelling durations as stochastic instead of deterministic [4].

Currently, two main approaches address stochastic scheduling problems: proactive and reactive. The proactive approach creates an offline solution that anticipates uncertainty using the upper bounds of the duration. The reactive approach begins with a solution and modifies it dynamically during execution in response to uncertainty [3]. The two approaches represent opposite ends of a spectrum, while most real-life approaches follow a hybrid implementation [6]. This paper explores which of the two approaches is best for solving an instance of a Multi-Mode RCPSP/max with generalised time lags/no-wait constraints. It also compares a hybrid approach that uses a Simple Temporal Network with Uncertainty (STNU).

The metrics used to compare these algorithms in this paper were solution quality, offline time, and online time. The solution quality is measured by comparing the makespan. The time offline refers to all computations performed before the execution of the schedule begins, including schedule generation and any preprocessing steps. The time online is the time spent adapting the schedule in response to randomness during execution.

This paper builds on the work of van den Houten et al. [6] by developing a model that incorporates multi-mode features, generalised time lags, and a no-wait constraint, thus creating instances that more closely reflect real-world conditions and offering a fundamental contribution to the field.

Section 2 of this paper provides a formal description of the MMRCPSP/max problem. Section 3 discusses related works to this paper. Section 4 outlines the methodology of the investigation. Section 5 provides the setup and results of the experiment. Section 6 provides an overview of the ethical issues and reproducibility of the experiment. Section 7 provides a comparison with known results and a broader context for the investigation. Finally, Section 8 provides a conclusion and future works that can improve upon this investigation.

2 Formal problem description

The MMRCPSP/max is described as follows [15]:

1. Given a set of activities $V = \{0, 1, \ldots, n+1\}$, with 0 and $n+1$ as dummy start/end nodes. Each activity has a set of modes, each mode m containing a stochastic duration d_{jm_j} and number of resources used r_{jm_j}.
2. Given a set of resources $\mathcal{R}$ with capacities R_k
3. Given a set of generalised precedence constraints $E \subseteq V \times V$, each with:
   - A relation type $\tau_{ij} \in \{\mathrm{SS}, \mathrm{SF}, \mathrm{FS}, \mathrm{FF}\}$, where S and F denote the start and finish times of the respective tasks.
   - A time lag $l_{ij} \in \mathbb{Z}$
4. A set $\mathcal{N} \subseteq V \times V$ of no-wait constraints

The objective is to find the start times $S_i \in \mathbb{N}$ and mode assignments $m_i \in M_i$ for all $i \in V$ that minimise the makespan of the schedule such that the following constraints are satisfied:

1. Generalized temporal constraints:

$$\begin{cases} S_j \geq S_i + l_{ij} & \text{if } \tau_{ij} = \mathrm{SS} \\ S_j + d_{jm_j} \geq S_i + l_{ij} & \text{if } \tau_{ij} = \mathrm{SF} \\ S_j \geq S_i + d_{im_i} + l_{ij} & \text{if } \tau_{ij} = \mathrm{FS} \\ S_j + d_{jm_j} \geq S_i + d_{im_i} + l_{ij} & \text{if } \tau_{ij} = \mathrm{FF} \end{cases} \quad \forall (i,j) \in E$$

2. No-wait constraints:

$$S_j = S_i + d_{im_i} \quad \forall (i,j) \in \mathcal{N}$$

3. Resource constraints (for all time units t):

$$\sum_{i \in V} \delta_{im_i}(t) \cdot r_{im_ik} \leq R_k \quad \forall k \in \mathcal{R}_r, \forall t \in \mathbb{N}$$

where $\delta_{im}(t) = 1$ if $S_i \leq t < S_i + d_{im}$, otherwise 0.

Figure 1 illustrates an instance of an RCPSP/max problem. The precedence graph represents temporal constraints between task start times; for example, task *a* must start at least 2 seconds before task *b* begins. The accompanying Gantt chart shows a feasible schedule that satisfies all constraints, with the number of resources used at each time point.

3 Related works

The multi-mode resource-constrained project scheduling problem (MMRCPSP) has been studied many times, including the work of Tao and Dong [18]. Similarly, the resource-constrained project scheduling problem with maximal time lags (RCPSP/max) has also been widely investigated, demonstrated, for example, in the study by Oddi and Rasconi [13].

In Lin et al. [11], a robust proactive algorithm is developed that uses the maximum possible duration of each task to create the schedule.

The dynamic controllability of an STNU implies the existence of a strategy that can assign execution times to all controllable events in a way that guarantees that all temporal constraints are met, independent of duration [12]. The algorithm developed to check for dynamic controllability in Morris [12] takes $O(N^3)$, which also transforms the STNU into an extended STNU (ESTNU) by adding wait edges to the STNU.

In this paper, we used the RTE* algorithm developed by Hunsberger and Posenato [7]. This algorithm transforms an ESTNU into a schedule in polynomial time.

In Deblaere et al. [5], the stochastic MMRCPSP problem was studied. However, their investigation was limited to focusing on the reactive approach, while exploring changes in duration and resource availability.

This study builds on the work of van den Houten et al. [6], which addresses dynamic scheduling for the RCPSP/max problem. In van den Houten et al. [6], different scheduling strategies are investigated for the stochastic RCPSP/max problem. They investigated four different algorithms: a proactive$_{0.9}$, reactive, STNU and proactive$_{SAA}$ approach. This paper extends their approach by focusing on the MMRCPSP/max, incorporating multi-mode execution and maximal time-lag constraints into a dynamic scheduling setting.

From our understanding, while the stochastic RCPSP/max problem was investigated by van den Houten et al. [6], and the stochastic MMRCPSP was investigated by Deblaere et al. [5], the stochastic MMRCPSP/max with generalised time-lags and no-wait constraints has not. Moreover, to our knowledge, only van den Houten et al. [6] have investigated using the STNU approach to investigate stochastic scheduling. Therefore, this paper aims to eliminate this knowledge gap by investigating which of the reactive, proactive, and STNU algorithms performs best when using the metrics of solution quality, time offline, and time online.

4 Methodology

This section outlines the methodology of the investigation. The library used to create a constraint programming model for MMRCPSP/max is given in 4.1. The selection of mode is shown in Section 4.2. The reactive, proactive and STNU approaches are discussed in 4.3. Finally, the method of comparison of the results is discussed in 4.4.

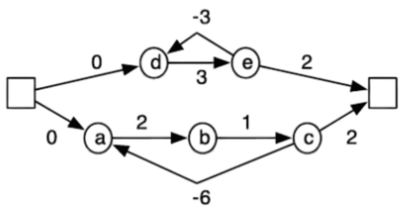
4.1 Constraint programming model for an MMRCPSP/max

This investigation uses the PyJobShop library, a Python-based modelling tool designed for job-shop and project scheduling problems [9].

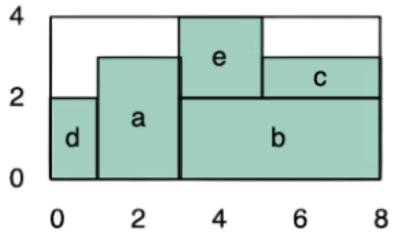
PyJobShop provides support for MMRCPSP through its Model class. The Model class allows for modelling the problem, with each task having multiple execution modes.

PyJobShop also supports integration with generalised time-lags. The Model class contains the following methods for adding temporal constraints between tasks:

- `add_start_before_start(Task1, Task2, lag)`
- `add_start_before_end(Task1, Task2, lag)`
- `add_end_before_start(Task1, Task2, lag)`
- `add_end_before_end(Task1, Task2, lag)`



(a) Precedence constraint graph



(b) Gantt chart

Figure 1: Precedence graph and Gantt chart of a solution to a small instance of the RCPSP/max problem with a single resource. [16]. The X-axis of the Gantt chart is time, and the Y-axis is the resource demand.

These methods allow for modelling both generalised time-lags and, as an extension, the no-wait constraint.

PyJobShop also integrates well with Constraint Programming solvers, making it suitable for handling maximal time-lags and mode-dependent resource constraints.

4.2 Mode of the task

In this investigation, the mode of a task is selected only during the offline phase, rather than during the offline and online phases. This simplifies the complexity of the problem for the reactive and STNU online phases at the expense of feasibility.

4.3 Algorithms investigated

The algorithms investigated in this paper are the proactive-reactive approach and the STNU approach.

Proactive-reactive model

The proactive-reactive model consists of two phases: A proactive phase and a reactive phase. The proactive phase generates a baseline schedule using a selected quantile of the duration bounds. The schedule is run with the proactive baseline using the real durations. If a duration is greater than its predicted duration, the schedule is considered proactively unfeasible.

The reactive phase builds on the same baseline. Whenever a task is completed at a different time from its predicted completion time, the model triggers a rescheduling procedure. When this happens, the scheduling problem is redefined: The start times of already completed tasks are fixed to their actual execution times, and unscheduled tasks are restricted to start no earlier than the current time. This process continues iteratively until all tasks are scheduled or infeasibility is detected.

STNU model

The STNU model is constructed using a quantile in the same way as the proactive approach. Based on this baseline, an STNU is generated and tested for dynamic controllability. If the network is dynamically controllable, it is considered feasible, and a solution is found using the RTE* algorithm. If not, the instance is unfeasible for the STNU algorithm.

4.4 Comparison of algorithms

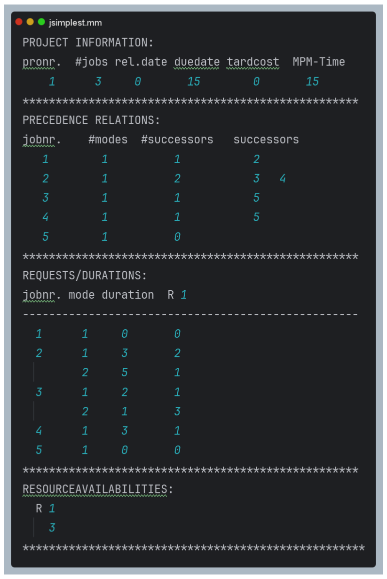
The algorithms are compared using the metrics of time offline, time online, and the quality of the solution (measured using the makespan). However, some instances are unfeasible and limit the direct comparison between the methods. For example, if an instance has a large makespan compared to the others and is found unfeasible in the reactive approach but feasible in the robust proactive approach, it would only hinder the average of the robust proactive approach if ignored or extremely punish the average of the reactive approach. To solve this problem, we used a partial ordering strategy using the Wilcoxon, proportion, and magnitude tests, as seen in van den Houten et al. [6]. The Wilcoxon test shows whether one algorithm consistently outperforms another. The proportion test shows how often an algorithm outperforms another, this being weaker than the Wilcoxon test. The magnitude test shows the magnitude by which one algorithm outperforms another, though it is limited as it cannot handle unfeasible values.

5 Experimental Setup and Results

This section outlines the experiment's setup and the investigation's results, including the generation of instances. The implementation of the stochastic duration is shown in 5.1. Instance generation is shown in 5.2. The experimental process is given in 5.3. The results are then presented and analysed in 5.4.

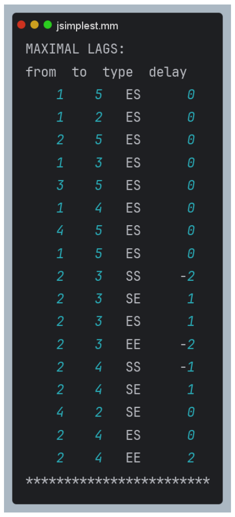
5.1 Duration of the tasks

The real duration of a task is determined by introducing variability to its original duration. This paper will follow the method proposed by van den Houten et al. [6]. The real duration is simulated by computing lower and upper bounds using a noise factor provided as a parameter. The lower bound is calculated as the original duration minus the noise factor multiplied by the square root of the duration, with a minimum value of 1. The upper bound is the original duration plus the same noise-based adjustment. Both bounds are rounded to integers, and the final duration is sampled from a discrete uniform distribution within this range. This discrete uniform sampling method was selected in contrast to other distributions such as normal, as the STNU approach needs a clearly defined lower and upper bound while only allowing for integer values. To adapt this implementation to an MMRCPSP instance, the real duration of each mode is simulated.



```
jsimplest.mm
PROJECT INFORMATION:
pronr.  #jobs rel.date duedate tardcost  MPM-Time
    1      3     0       15        0       15
***************************************************
PRECEDENCE RELATIONS:
jobnr.    #modes  #successors   successors
   1        1         1           2
   2        1         2           3   4
   3        1         1           5
   4        1         1           5
   5        1         0
***************************************************
REQUESTS/DURATIONS:
jobnr. mode duration  R 1
---------------------------------------------------
  1      1     0       0
  2      1     3       2
         2     5       1
  3      1     2       1
         2     1       3
  4      1     3       1
  5      1     0       0
***************************************************
RESOURCEAVAILABILITIES:
  R 1
    3
****************************************************
```

(a) Instance of a simple MMRCPSP problem



```
jsimplest.mm
MAXIMAL LAGS:
from  to  type  delay
     1    5   ES      0
     1    2   ES      0
     2    5   ES      0
     1    3   ES      0
     3    5   ES      0
     1    4   ES      0
     4    5   ES      0
     1    5   ES      0
     2    3   SS     -2
     2    3   SE      1
     2    3   ES      1
     2    3   EE     -2
     2    4   SS     -1
     2    4   SE      1
     4    2   SE      0
     2    4   ES      0
     2    4   EE      2
************************
```

(b) The generated generalised time-lags for the MMRCPSP problem

Figure 2: Simple instance of a Multi-Mode RCPSP/max with generalised time lags: 3 real tasks, a super-source, super-sink, and 1 resource (ID 1, capacity 3).

5.2 Instance generation

The instances used in this experiment were obtained from the PSPLIB database at the Technical University of Munich [8]. Since the original instances are for the standard MMRCPSP, they were transformed into an MMRCPSP/max instance by introducing generalised time lags. This transformation was achieved by adding temporal lags between different types of relationships (start-to-start, start-to-end, end-to-start, and end-to-end) between tasks with a predefined precedence relationship (predecessor–successor). The precedence relationship between tasks was considered instead of applying full randomness to preserve the structure of the MMRCPSP instances and try to maintain feasible schedules. The lag values were randomly selected within an arbitrary range. The no-wait constraint was added between tasks with a start-to-end relationship with zero lag. This process was achieved by adding a reversed direction end-to-start relationship with zero lag. To reduce the number of constraints and allow the possibility of open-bounded start-to-end relationships with zero lag, the no-wait constraint was limited to a maximum of two per instance. Figure 2 shows a simple instance for an MMRCPSP/max and the respective lags between the different tasks. In this example, task 4 has a no-wait constraint with task 2 (task 4 must start as soon as task 2 finishes).

5.3 Experimental process

Each instance is run 10 times for each noise factor. Running multiple instances allows us to evaluate the instance with different real durations per task and evaluate different schedules that satisfy the instance's constraints. Due to time constraints, this investigation will be limited to 100 instances per configuration, with a configuration having a number of tasks (10 or 20) and noise factor (1 or 2), totalling 400 instances and 4000 runs.

Number	Noise factor 1			Noise factor 2		
of tasks	pro	react	STNU	pro	react	STNU
10	0.245	0.256	0.213	0.208	0.229	0.201
20	0.041	0.038	0.021	0.044	0.042	0.036

Table 1: Feasibility rates for the proactive, reactive and STNU methods with 10 and 20 tasks using a noise factor of 1 and 2.

PyJobShop stores the mode of tasks within its task object. This allows us to precompute and store the simulated duration of the different modes in an array of the same size as the total number of modes of the instance. Using an array, we can simulate real durations for all modes and efficiently access them during runtime. The real durations of the modes are calculated using the method explained in Section 5.1.

First, the proactive-reactive algorithm is run, followed by the STNU algorithm. For every run of an instance, the makespan, time online, and time offline of the three algorithms are saved. The results are then compared using the methods described in Section 4.4.

5.4 Results

Feasibility rate

Table 1 shows the feasibility rate of the three algorithms using noise factors 1 and 2. Table 2 shows the feasibility rate for

Number	Noise factor 1			Noise factor 2		
of tasks	pro	react	STNU	pro	react	STNU
10	1.000	1.000	0.113	1.000	1.000	0.073
20	1.000	1.000	0.015	1.000	1.000	0.008

Table 2: Feasibility rates for the proactive, reactive and STNU methods with 10 and 20 tasks using a noise factor of 1 and 2 when not using start-to-end and end-to-end constraints.

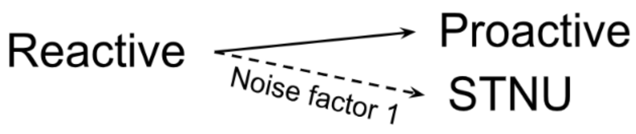


Figure 3: Partial ordering of the methods using all constraints, based on solution quality. Solid arrows indicate a significant difference using the Wilcoxon test, and dashed arrows indicate a significant difference using the proportion test when the Wilcoxon test did not find a significant difference. The reactive approach dominates the proactive approach, and dominates the STNU approach for the proportion test when the noise factor is 1.

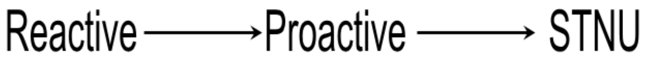


Figure 4: Partial ordering of the methods not using start-to-end and end-to-end constraints, based on solution quality. Solid arrows indicate a significant difference using the Wilcoxon test. The reactive approach yields the highest quality, followed by the proactive approach, and finally, the STNU approach.

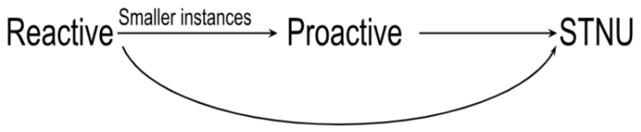


Figure 5: Partial ordering of the methods using all constraints, based on time offline. Solid arrows indicate a significant difference using the Wilcoxon test. For smaller instances, the reactive approach yields the smallest time offline, followed by the proactive, and last the STNU approach. For larger instances, the proactive and reactive approaches yield the smallest time offline, with the STNU approach yielding the largest.

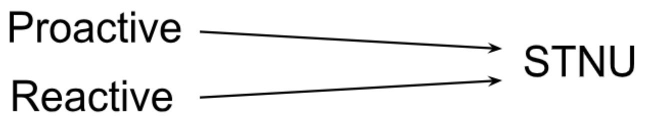


Figure 6: Partial ordering of the methods not using start-to-end and end-to-end constraints, based on time offline. Solid arrows indicate a significant difference using the Wilcoxon test. The proactive and reactive approaches yield the lowest offline time, followed by the STNU approach.

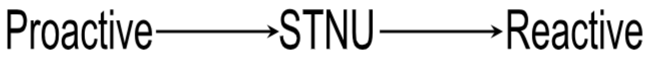


Figure 7: Partial ordering of the methods using all constraints, based on time online. Solid arrows indicate a significant difference using the Wilcoxon test. The proactive approach yields the lowest time online, followed by the STNU approach, and finally, the reactive approach.

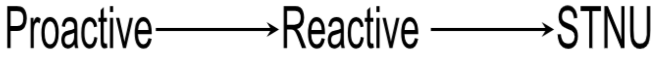


Figure 8: Partial ordering of the methods not using start-to-end and end-to-end constraints, based on time online. Solid arrows indicate a significant difference using the Wilcoxon test. The proactive approach yields the fastest time online. followed by the reactive approach, with the STNU approach being the slowest.

the same instances, but ignoring the precedence constraints: start-to-end and end-to-end.

Table 1 shows that the feasibility rates of the three algorithms are low when all constraints are maintained. For 10 tasks, the reactive approach shows the highest overall feasibility, followed by the proactive approach, with the STNU approach having the lowest feasibility. For 20 tasks, the proactive approach shows the highest overall feasibility, followed by the reactive approach, with the STNU approach again having the lowest feasibility.

The noise factor also affects the feasibility, with a noise factor of 1 showing a higher feasibility rate for smaller instances, while a noise factor of 2 shows greater feasibility in larger instances. However, the increases and decreases are minimal.

The magnitude tests in Appendix A show that, in most instances where the proactive approach found a solution, the reactive approach also did. In contrast, for most instances where the STNU algorithm found a solution, the other two algorithms did not, and vice versa.

Table 2 shows that, when the constraints affected by the end time of the second task are removed, the feasibility rate of the proactive and reactive algorithms increases to a rate of 1 for all instances and noise factors. In contrast, the feasibility of the STNU algorithm decreased to values of 0.113 and 0.073 for noise factors 1 and 2 with a task size of 10, and 0.015 and 0.008 for noise factors 1 and 2 with a task size of 20. It is important to note that this is a relaxation of the problem as it not only removes these relationships between tasks, but, in doing so, it removes the no-wait constraints as well.

Analysis of the Wilcoxon, proportion and Magnitude test Table 3 compares the three algorithms using the three different tests for instances of 10 tasks and a noise factor of 1. Appendix A shows the complete set of results for noise factors 1 and 2, and for the relaxation of the problem.

As explained in Section 4.4, both the Wilcoxon and the proportion tests compare all instances between algorithms. In contrast, the magnitude test only compares instances where

		Schedule Quality (Makespan)	
j10	*reactive*-*STNU*	*proactive robust*-*STNU*	**reactive**-*proactive robust*
Wilc. Quality	[381] -0.815 (0.415)	[363] -1.729 (0.084)	[225] -12.724 (*)
Prop. Quality	[375] 0.544 (0.098)	[362] 0.494 (0.875)	[208] 0.995 (*)
	reactive-*STNU*	*STNU*-*proactive robust*	**reactive**-*proactive robust*
Magn. Quality	[40] -1.692 (0.099) *reactive*: 0.98 *STNU*: 1.02	[38] 1.578 (0.123) *STNU*: 0.979 *proactive robust*: 1.021	[203] -18.533 (*) *reactive*: 0.957 *proactive robust*: 1.043
		Offline Time	
j10	**reactive**-*proactive robust*	**reactive**-*STNU*	**proactive robust**-*STNU*
Wilc. Quality	[225] -4.252 (*)	[381] -8.195 (*)	[363] -7.096 (*)
Prop. Quality	[22] 0.955 (*)	[381] 0.588 (*)	[363] 0.562 (*)
	reactive-*proactive robust*	**reactive**-*STNU*	**proactive robust**-*STNU*
Magn. Quality	[203] nan (nan) reactive: 1.0 proactive robust: 1.0	[40] -28.903 (*) reactive: 0.26 STNU: 1.74	[38] -27.566 (*) proactive robust: 0.27 STNU: 1.73
		Online Time	
j10	**proactive robust**-*reactive*	**STNU**-*reactive*	**proactive robust**-*STNU*
Wilc. Quality	[225] -8.407 (*)	[381] -5.581 (*)	[363] -7.133 (*)
Prop. Quality	[225] -8.407 (*)	[381] -5.581 (*)	[363] -7.133 (*)
	proactive robust-*reactive*	**STNU**-*reactive*	**proactive robust**-*STNU*
Magn. Quality	[203] -8177.742 (*) proactive robust: 0.01 reactive: 1.99	[40] -99.327 (*) STNU: 0.18 reactive: 1.82	[38] -469.392 (*) proactive robust: 0.07 STNU: 1.93

Table 3: pairwise test results for 10 jobs with noise factor 2. The metrics covered are Wilcoxon, proportion, and magnitude tests. Results:[pairs]z-value(p-value)(* for $p < 0.05$), proportion (p-value), and t-stat (p-value) with normalised averages. Column headers list the compared methods. The order of the columns displays which algorithm (bolded) is found to be better in that category. If no algorithm is bold, neither outperforms the other. This table format is based on the results from van den Houten et al. [6]

both algorithms have found a solution. Hence, the Wilcoxon and proportion tests punish unfeasibility, while the magnitude test does not. The outcome of the Wilcoxon and proportion tests allows us to create a partial ordering of the methods, with the Wilcoxon test representing strong connections and the proportion test creating weak connections when the Wilcoxon test had $p > 0.05$. The magnitude test shows the magnitude difference between the algorithms when both instances find a solution.

Figure 3 shows the visualisation of the partial ordering of the quality of the solution for the three algorithms using all the constraints. The Wilcoxon test found that the reactive algorithm outperformed the proactive algorithm consistently. The proportion test found that the reactive algorithm outperforms the STNU algorithm when the noise factor is 1.

Figure 4 shows the visualisation of the partial ordering for the quality of the solution for the three algorithms without using start-to-end and end-to-end constraints. In this problem relaxation, the Wilcoxon test showed that the reactive approach dominated the proactive approach in all instances, with the STNU approach finding the worst solutions.

The magnitude test shows that, in instances where two algorithms found a solution, the reactive approach outperforms the proactive approach. The reactive approach also outperforms the STNU approach for noise factor 1, while being mostly equal for noise factor 2.

When relaxing the problem, the STNU algorithm does outperform the reactive and proactive algorithms in the magnitude test. This is consistent for both noise factors.

Figure 5 shows the visualisation of the partial ordering of the offline time for the three algorithms using all the constraints. For smaller instances, the reactive approach yields the lowest offline time, followed by the proactive approach, with the STNU algorithm being the slowest. For larger instances, the proactive and reactive algorithms are equal, with the STNU approach being the slowest.

Figure 6 shows the visualisation of the partial ordering of the offline time for the three algorithms when relaxing the problem by not using start-to-end and end-to-end constraints. The relaxed problem behaves similarly to the fully constrained problem for offline time, as the proactive and reactive algorithms outperform the STNU while performing equally with each other. This is consistent for 10 and 20 tasks.

The magnitude test for the offline time shows that both the reactive and proactive approaches perform the same, with the STNU approach performing the worst. This is consistent for all results. Hence, when all algorithms find a solution, the proactive and reactive approaches are equal in time offline, with the STNU being the worst of the three by a wide margin.

Figure 7 shows a visualisation of the partial ordering of the time online for the three algorithms. The proactive approach is yet again the best-performing algorithm when comparing online time, followed by the STNU approach, and the reactive approach being the worst-performing algorithm.

Figure 8 shows the partial order of the online time for the three algorithms when relaxing the problem by not using start-to-end and end-to-end constraints. In these instances, the proactive algorithm still outperforms, but the reactive algorithm dominates the STNU algorithm.

The magnitude test shows that, for the online time, when two algorithms find a solution, the proactive algorithm far outperforms the other two, followed by the STNU algorithm, with the reactive algorithm performing the worst by a considerable margin.

6 Responsible Research

6.1 Harms and biases

This paper reduces potential harms and biases using two main measures. First, the problem instances are adapted from the publicly available PSPLIB benchmark suite developed by the Technical University of Munich, ensuring a standard and widely accepted baseline. To introduce generalised time lags, random lags are created within a predefined range, reducing the risk of human-induced bias. Second, each experiment is conducted on 100 distinct instances, with each instance running 10 times. This repetition helps to reduce the effect of outliers in both instances and randomness.

6.2 Ethical discussion

This paper aims to find which of the predefined approaches is best for an MMRCPSP/max problem. Hence, the results of this paper can be used to try to solve instances of this problem in different fields, such as construction, logistics, or healthcare. Although these methods try to improve schedule efficiency, applying these methods without a clear understanding of their limitations can lead to suboptimal or harmful results.

In addition, this investigation focuses on simulated uncertainty using independent and identically distributed random variables. These variables are unlikely to reflect the random properties of real-world tasks, which limits the real-world applicability of this investigation.

Finally, this investigation deals with the usage of NP-hard problems, which require high computational power, leading to ethical concerns regarding energy consumption and environmental sustainability.

6.3 Reproducibility

This paper aims to ensure the reproducibility of the research. To that end, all relevant program files are available in a dedicated GitHub repository (https://github.com/kimvandenhouten/PyJobShopSTNUs). These include the source code, a README file with instructions for running the program, the requirements, and the instance files for the MMRCPSP/max problem. The code uses a fixed random seed to guarantee consistent results regardless of randomness.

7 Discussion

This section discusses the results by focusing on the larger picture and comparing it with baselines.

7.1 Comparison with the results of stochastic RCPSP/max

The main difference between the experimental setup of van den Houten et al. [6] and our investigation are the change from an RCPSP/max to an MMRCPSP/max with generalised time-lags and no-wait constraints, our use of the proactive robust algorithm instead of the proactive$_{0.9}$ algorithm, and our exclusion of the proactive$_{SAA}$ algorithm.

For the relaxed variant, when comparing the Wilcoxon and proportion tests, the results do differ. The proactive and reactive algorithms dominate the STNU algorithm in all metrics. The main reason for this discrepancy is the low feasibility rate of the STNU algorithm compared to the proactive and reactive algorithms, as these tests place a significant emphasis on feasibility. This explanation is supported by the magnitude tests, as it follows the same pattern as the results from van den Houten et al. [6], since the magnitude test only compares instances where both algorithms found a solution.

However, for the unrelaxed variant, the key difference from van den Houten et al. [6] is that the makespan of the STNU approach is equal to or greater than that of the reactive approach in all tests.

7.2 Feasibility rate of the reactive and proactive approaches

The low feasibility rate of the proactive and reactive algorithms is due to the time-lags using the end-time of the successor task, as seen in start-to-end and end-to-end relationships. In the reactive approach, this issue arises from its robust scheduling strategy. As the reactive approach fixes the starting time of tasks that have already started while rescheduling based on the worst-case scenario, a task whose real duration is lower than the predicted one can break the start-to-end and end-to-end constraints.

The low feasibility of the proactive approach is caused by the inability to anticipate the end duration of the successor tasks. While adding buffers based on maximum duration allows us to ensure that a successor task can always satisfy the start-to-start and end-to-start relationships, it cannot plan for a task ending earlier, hence creating issues with start-to-end and end-to-end relationships. This is especially seen with the no-wait constraint, as ensuring that a task starts as soon as another ends can only happen if the duration of a task ends as predicted by the maximum duration.

7.3 Explaining the results with theory

Makespan

The proactive robust approach used in this paper creates a schedule using the worst-case duration. The schedule created is then unchanged. However, the reactive and STNU approaches are more flexible in their scheduling, as both approaches can react to the stochastic duration dynamically. Hence, it goes with the theory that the proactive robust approach creates the largest makespan when all approaches find a solution.

Time offline

All three algorithms need to create a baseline during their offline time. The STNU approach has to transform the baseline into an STNU and then transform the STNU into an ESTNU. As reactive and proactive approaches only need to create a baseline, while the STNU approach has to create and then transform its baseline, it explains why the magnitude test shows that both the reactive and proactive algorithms have no difference in offline time, while the STNU approach is considerably slower.

Time online

The proactive algorithm creates a schedule that tries to account for uncertainty by using the worst-case scenario of durations during the time offline. Hence, its online time is negligible as no rescheduling is needed.

The reactive algorithm has to constantly create schedules to account for uncertainty. Creating each new schedule is an NP-hard problem [10], and requires a lot of computational power.

For the STNU approach, the ESTNU created during offline time needs to be solved. The RTE* algorithm can solve an ESTNU in polynomial time [7], thus being faster than the reactive approach.

Our results agree with the theory, as the magnitude test shows that, when all algorithms find a solution, the reactive approach is the slowest. followed by the STNU approach, with the proactive robust being the fastest for this metric.

7.4 Real world applications

It is necessary to point out that, when the problem is not relaxed, the feasibility rate that these algorithms offer is a great deterrent for using them in real-life scheduling, even more so in projects where a working schedule is of extreme importance. However, for relaxed instances, both the proactive and reactive algorithms could have multiple uses outside academia.

The proactive algorithm

The proactive algorithm has the lowest online and offline times. Its main drawback is its makespan as it is higher than the other two algorithms. Due to its low creation time, the proactive algorithm seems ideal for the creation of quick solutions to real-life scheduling problems when time is of the essence.

The STNU algorithm

The main drawback of the STNU algorithm is its slow offline time. Its main advantages are that the STNU algorithm can find solutions with a good makespan while having a relatively short online time. This algorithm seems ideal when the offline time to prepare a schedule is not strict. However, this algorithm has an even worse feasibility rate when relaxing the problem, hence limiting its real-world usage even more.

The reactive algorithm

The reactive algorithm shows great ability to minimise the makespan, while conserving a low offline time. Its main drawback is its online time, as it is NP-hard, hence not very scalable. The reactive algorithm excels in instances where the online time is not as strict, or not many tasks have randomness, as it limits the amount of reshuffles to the schedule.

8 Conclusions and Future Work

This section serves as a conclusion to the investigation, while considering future works that can be expanded from this investigation.

8.1 Conclusion

This paper has investigated three different algorithms and compared them to analyse their performance using three distinct evaluation metrics: Quality of solution, time offline, and time online. The paper has found that the proactive algorithm serves as a faster option for solving instances of MMRCPSP/max with generalised time-lags and no-wait constraints, but both the reactive and STNU algorithms offer solutions of greater quality. This study has also found that, when relaxing the problem by removing start-to-end and end-to-end constraints, in terms of feasibility, both the proactive and reactive algorithms heavily outperform the STNU algorithm, which limits the real usage of the STNU algorithm outside of academia for generalised time-lags.

8.2 Future works

This paper specifically investigates the MMRCPSP/max problem with generalised time-lags and no-wait constraints while also taking liberties with the formulation of the problem. The following investigations, hence, can build upon this work.

Precedence constraints between modes

Currently, the precedence constraints are added between tasks. In the real world, there are instances where, depending on the mode selected, the precedence constraint is different between two tasks. Hence, an investigation focusing on precedence constraints between modes would expand on this paper.

Limiting mode selection between tasks

In this paper, the modes have no relation to each other. However, in the real world, some task modes depend on the previously selected mode of their predecessor [18]. For example, if task A has two modes and task B has two modes, the mode selected from B might depend on the mode selected from A. This follows from most real-world multi-mode applications, as some tasks cannot be performed if a specific mode is not selected.

Mode being changed dynamically in the experiment

In both the reactive and STNU approaches, the mode is not allowed to change during the online phase. It can be investigated how the quality of the solution and the feasibility of the instance change when the mode is allowed to be selected dynamically.

The noise factor to be individual per task instead of all having an arbitrary noise factor

In this investigation, the upper and lower bounds of the duration for all tasks use the same noise factor. In the real world, different tasks have different duration variance. Hence, an investigation where the noise factor is independent per task could expand on the real-world modelling of instances.

References

[1] Christian Artigues, S. Demassey, E. Néon, and F. Sourd. Resource-constrained project scheduling: Models, algorithms, extensions and applications. *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications*, pages 2–3, 01 2010.

[2] Christian Artigues, Sönke Hartmann, and Mario Vanhoucke. Fifty years of research on resource-constrained project scheduling explored from different perspectives. *European Journal of Operational Research*, 2025.

[3] Tarek Chaari, Sondes Chaabane, Nassima Aissani, and Damien Trentesaux. Scheduling under uncertainty: Survey and research directions. In *2014 International Conference on Advanced Logistics and Transport (ICALT)*, pages 229–234, 2014.

[4] A. Davenport and J. Beck. A survey of techniques for scheduling with uncertainty. 01 2000.

[5] Filip Deblaere, Erik Demeulemeester, and Willy Herroelen. Reactive scheduling in the multi-mode rcpsp. *Computers & Operations Research*, 38(1):63–74, 2011. Project Management and Scheduling.

[6] Kim van den Houten, Léon Planken, Esteban Freydell, David M. J. Tax, and Mathijs de Weerdt. Proactive and Reactive Constraint Programming for Stochastic Project Scheduling with Maximal Time-Lags. 2024.

[7] Luke Hunsberger and Roberto Posenato. Foundations of dispatchability for simple temporal networks with uncertainty. In *Proceedings of the 16th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART*, pages 253–263. INSTICC, SciTePress, 2024.

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

[9] Leon Lan and Joost Berkhout. Pyjobshop: Solving scheduling problems with constraint programming in python, 2025.

[10] C. Le Pape. Constraint-based scheduling: principles and application. *COLLOQUIUM DIGEST- IEE*, 197:5, 1996. 5.

[11] Xiaoxia Lin, Stacy L. Janak, and Christodoulos A. Floudas. A new robust optimization approach for scheduling under uncertainty:: I. bounded uncertainty. *Computers & Chemical Engineering*, 28(6):1069–1085, 2004. FOCAPO 2003 Special issue.

[12] Paul Morris. Dynamic controllability and dispatchability relationships. In Helmut Simonis, editor, *Integration of AI and OR Techniques in Constraint Programming*, pages 464–479, Cham, 2014. Springer International Publishing.

[13] Angelo Oddi and Riccardo Rasconi. Solving resource-constrained project scheduling problems with time-windows using iterative improvement algorithms. *Proceedings of the International Conference on Automated Planning and Scheduling*, 19:378–381, 10 2009.

[14] Andreas Schutt, Thibaut Feydy, Peter J. Stuckey, and Mark G. Wallace. Solving the resource constrained project scheduling problem with generalized precedences by lazy clause generation, 2010.

[15] Andreas Schutt, Thibaut Feydy, Peter J. Stuckey, and Mark G. Wallace. Solving the resource constrained project scheduling problem with generalized precedences by lazy clause generation. *CoRR*, abs/1009.0347, 2010.

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

[17] Reza Shahabi-Shahmiri, Reza Tavakkoli-Moghaddam, Alexandre Dolgui, Seyed-Ali Mirnezami, Mohammad Ghasemi, and Mahsa Ahmadi. Preemptive and non-preemptive multi-skill multi-mode resource-constrained project scheduling problems considering sustainability and energy consumption: A comprehensive mathematical model. *Journal of Environmental Management*, 367:121986, 2024.

[18] Sha Tao and Zhijie Sasha Dong. Multi-mode resource-constrained project scheduling problem with alternative project structures. *Computers & Industrial Engineering*, 125:333–347, 2018.

A Full results

This appendix contains the full results of the experiment. It includes tables for noise factors 1 and 2, and instances without start-end and end-end relationships. For each experiment, there is a table for the Wilcoxon, proportion and magnitude test comparing the makespan, the time offline and the time online.

j10	**reactive**-*STNU*	*proactive robust*-*STNU*	**reactive**-*proactive robust*
Wilcoxon : [n] z (p)	[383] -0.964 (0.335)	[375] -0.601 (0.548)	[235] -12.624 (*)
proportion: [n] z (p)	[377] 0.562 (*)	[372] 0.522 (0.437)	[200] 0.99 (*)

	reactive-*STNU*	*STNU*-*proactive robust*	**reactive**-*proactive robust*
magnitude: [n] z (p)	[44] -2.052 (*)	[42] 1.583 (0.121)	[221] -18.225 (*)
	reactive: 0.986	*STNU*: 0.987	*reactive*: 0.973
	STNU: 1.014	*proactive robust*: 1.013	*proactive robust*: 1.027

j20	**reactive**-*STNU*	**proactive robust**-*STNU*	**reactive**-*proactive robust*
Wilcoxon : [n] z (p)	[50] -2.158 (*)	[52] -2.31 (*)	[37] -3.145 (*)
proportion: [n] z (p)	[50] 0.66 (*)	[52] 0.673 (*)	[35] 0.886 (*)

	reactive-*STNU*	*proactive robust*-*STNU*	**reactive**-*proactive robust*
magnitude: [n] z (p)	[1] nan (nan)	[1] nan (nan)	[31] -8.42 (*)
	reactive: 0.972	*proactive robust*: 0.981	*reactive*: 0.983
	STNU: 1.028	*STNU*: 1.019	*proactive robust*: 1.017

Table 4: Pairwise comparison on schedule quality (makespan) for noise factor c=1. Using a Wilcoxon test, proportion test and magnitude test. The Wilcoxon and proportion tests include all instances for which at least one of the two methods found a feasible solution. The magnitude test includes all instances for which both methods found a feasible solution. Note that the ordering matters: the first method (bolded) shown is the better of the two in each pair, method 1 - method 2. If no algorithm is bold, neither outperforms the other. Each cell shows on the first row [nr pairs] 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. Each cell shows on the third row [nr pairs] t-stat (p-value) with (*) for $p < 0.05$, on the fourth row the normalised average of method 1 and on the fifth row the normalised average of method 2.

j10	*reactive*-*STNU*	*proactive robust*-*STNU*	**reactive**-*proactive robust*
Wilcoxon : [n] z (p)	[381] -0.815 (0.415)	[363] -1.729 (0.084)	[225] -12.724 (*)
proportion: [n] z (p)	[375] 0.544 (0.098)	[362] 0.494 (0.875)	[208] 0.995 (*)

	reactive-*STNU*	*STNU*-*proactive robust*	**reactive**-*proactive robust*
magnitude: [n] z (p)	[40] -1.692 (0.099)	[38] 1.578 (0.123)	[203] -18.533 (*)
	reactive: 0.98	*STNU*: 0.979	*reactive*: 0.957
	STNU: 1.02	*proactive robust*: 1.021	*proactive robust*: 1.043

j20	*reactive*-*STNU*	*proactive robust*-*STNU*	**reactive**-*proactive robust*
Wilcoxon : [n] z (p)	[71] -1.398 (0.162)	[73] -1.213 (0.225)	[43] -4.116 (*)
proportion: [n] z (p)	[71] 0.549 (0.476)	[73] 0.548 (0.483)	[38] 0.921 (*)

	reactive-*STNU*	*STNU*-*proactive robust*	**reactive**-*proactive robust*
magnitude: [n] z (p)	[3] -0.08 (0.944)	[3] 0.346 (0.762)	[39] -7.373 (*)
	reactive: 0.998	*STNU*: 0.992	*reactive*: 0.975
	STNU: 1.002	*proactive robust*: 1.008	*proactive robust*: 1.025

Table 5: Pairwise comparison on schedule quality (makespan) for noise factor c=2. Using a Wilcoxon test, proportion test and magnitude test. The Wilcoxon and proportion tests include all instances for which at least one of the two methods found a feasible solution. The magnitude test includes all instances for which both methods found a feasible solution. Note that the ordering matters: the first method (bolded) shown is the better of the two in each pair, method 1 - method 2. If no algorithm is bold, neither outperforms the other. Each cell shows on the first row [nr pairs] 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. Each cell shows on the third row [nr pairs] t-stat (p-value) with (*) for $p < 0.05$, on the fourth row the normalised average of method 1 and on the fifth row the normalised average of method 2.

j10	**reactive**-$STNU$	**proactive robust**-$STNU$	**reactive**-$proactive\ robust$
Wilcoxon : [n] z (p)	[1000] -26.729 (*)	[1000] -26.697 (*)	[1000] -26.82 (*)
proportion: [n] z (p)	[970] 0.916 (*)	[992] 0.894 (*)	[835] 1.0 (*)
	STNU-$reactive$	**STNU**-$proactive\ robust$	**reactive**-$proactive\ robust$
magnitude: [n] z (p)	[113] 11.019 (*)	[113] 17.021 (*)	[1000] -38.131 (*)
	$STNU$: 0.966	$STNU$: 0.933	$reactive$: 0.969
	$reactive$: 1.034	$proactive\ robust$: 1.067	$proactive\ robust$: 1.031
j20	**reactive**-$STNU$	**proactive robust**-$STNU$	**reactive**-$proactive\ robust$
Wilcoxon : [n] z (p)	[1000] -27.383 (*)	[1000] -27.386 (*)	[1000] -26.595 (*)
proportion: [n] z (p)	[999] 0.986 (*)	[1000] 0.985 (*)	[808] 1.0 (*)
	STNU-$reactive$	**STNU**-$proactive\ robust$	**reactive**-$proactive\ robust$
magnitude: [n] z (p)	[15] 5.95 (*)	[15] 7.247 (*)	[1000] -37.031 (*)
	$STNU$: 0.967	$STNU$: 0.951	$reactive$: 0.983
	$reactive$: 1.033	$proactive\ robust$: 1.049	$proactive\ robust$: 1.017

Table 6: Pairwise comparison on schedule quality (makespan) for noise factor c=1 with no start-to-end and end-to-end relationships. Using a Wilcoxon test, proportion test and magnitude test. The Wilcoxon and proportion tests include all instances for which at least one of the two methods found a feasible solution. The magnitude test includes all instances for which both methods found a feasible solution. Note that the ordering matters: the first method (bolded) shown is the better of the two in each pair, method 1 - method 2. If no algorithm is bold, neither outperforms the other. Each cell shows on the first row [nr pairs] 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. Each cell shows on the third row [nr pairs] t-stat (p-value) with (*) for $p < 0.05$, on the fourth row the normalised average of method 1 and on the fifth row the normalised average of method 2.

j10	**reactive**-$STNU$	**proactive robust**-$STNU$	**reactive**-$proactive\ robust$
Wilcoxon : [n] z (p)	[1000] -27.125 (*)	[1000] -27.103 (*)	[1000] -27.278 (*)
proportion: [n] z (p)	[985] 0.946 (*)	[999] 0.928 (*)	[922] 1.0 (*)
	STNU-$reactive$	**STNU**-$proactive\ robust$	**reactive**-$proactive\ robust$
	[73] 7.751 (*)	[73] 15.486 (*)	[1000] -38.82 (*)
	$STNU$: 0.941	$STNU$: 0.888	$reactive$: 0.95
	$reactive$: 1.059	$proactive\ robust$: 1.112	$proactive\ robust$: 1.05
j20	**reactive**-$STNU$	**proactive robust**-$STNU$	**reactive**-$proactive\ robust$
Wilcoxon : [n] z (p)	[1000] -27.392 (*)	[1000] -27.393 (*)	[1000] -27.066 (*)
proportion: [n] z (p)	[1000] 0.993 (*)	[1000] 0.992 (*)	[879] 1.0 (*)
	STNU-$reactive$	**STNU**-$proactive\ robust$	**reactive**-$proactive\ robust$
magnitude: [n] z (p)	[8] 4.382 (*)	[8] 6.899 (*)	[1000] -36.781 (*)
	$STNU$: 0.948	$STNU$: 0.926	$reactive$: 0.974
	$reactive$: 1.052	$proactive\ robust$: 1.074	$proactive\ robust$: 1.026

Table 7: Pairwise comparison on schedule quality (makespan) for noise factor c=2 with no start-to-end and end-to-end relationships. Using a Wilcoxon test, proportion test and magnitude test. The Wilcoxon and proportion tests include all instances for which at least one of the two methods found a feasible solution. The magnitude test includes all instances for which both methods found a feasible solution. Note that the ordering matters: the first method (bolded) shown is the better of the two in each pair, method 1 - method 2. If no algorithm is bold, neither outperforms the other. Each cell shows on the first row [nr pairs] 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. Each cell shows on the third row [nr pairs] t-stat (p-value) with (*) for $p < 0.05$, on the fourth row the normalised average of method 1 and on the fifth row the normalised average of method 2.

j10	**reactive**-*proactive robust*	**reactive**-*STNU*	**proactive robust**-*STNU*
Wilcoxon : [n] z (p)	[235] -2.645 (*)	[383] -8.691 (*)	[375] -8.206 (*)
proportion: [n] z (p)	[14] 0.857 (*)	[383] 0.608 (*)	[375] 0.595 (*)

	reactive-proactive robust	**reactive**-*STNU*	**proactive robust**-*STNU*
magnitude: [n] z (p)	[221] nan (nan)	[44] -30.736 (*)	[42] -29.306 (*)
	reactive: 1.0	reactive: 0.23	proactive robust: 0.24
	proactive robust: 1.0	STNU: 1.77	STNU: 1.76

j20	*reactive-proactive robust*	**reactive**-*STNU*	**proactive robust**-*STNU*
Wilcoxon : [n] z (p)	[37] -0.709 (0.478)	[50] -4.103 (*)	[52] -4.339 (*)
proportion: [n] z (p)	[6] 0.333 (0.683)	[50] 0.66 (*)	[52] 0.673 (*)

	reactive-proactive robust	*reactive-STNU*	*proactive robust-STNU*
magnitude: [n] z (p)	[31] nan (nan)	[1] nan (nan)	[1] nan (nan)
	reactive: 1.0	reactive: 0.94	proactive robust: 0.94
	proactive robust: 1.0	STNU: 1.06	STNU: 1.06

Table 8: Pairwise comparison on time offline for noise factor c=1. Using a Wilcoxon test, proportion test and magnitude test. The Wilcoxon and proportion tests include all instances for which at least one of the two methods found a feasible solution. The magnitude test includes all instances for which both methods found a feasible solution. Note that the ordering matters: the first method (bolded) shown is the better of the two in each pair, method 1 - method 2. If no algorithm is bold, neither outperforms the other. Each cell shows on the first row [nr pairs] 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. Each cell shows on the third row [nr pairs] t-stat (p-value) with (*) for $p < 0.05$, on the fourth row the normalised average of method 1 and on the fifth row the normalised average of method 2.

j10	**reactive**-*proactive robust*	**reactive**-*STNU*	**proactive robust**-*STNU*
Wilcoxon : [n] z (p)	[225] -4.252 (*)	[381] -8.195 (*)	[363] -7.096 (*)
proportion: [n] z (p)	[22] 0.955 (*)	[381] 0.588 (*)	[363] 0.562 (*)

	reactive-proactive robust	**reactive**-*STNU*	**proactive robust**-*STNU*
magnitude: [n] z (p)	[203] nan (nan)	[40] -28.903 (*)	[38] -27.566 (*)
	reactive: 1.0	reactive: 0.26	proactive robust: 0.27
	proactive robust: 1.0	STNU: 1.74	STNU: 1.73

j20	*reactive-proactive robust*	**reactive**-*STNU*	**proactive robust**-*STNU*
Wilcoxon : [n] z (p)	[43] -0.951 (0.341)	[71] -1.991 (*)	[73] -2.166 (*)
proportion: [n] z (p)	[4] 0.25 (0.617)	[71] 0.563 (0.342)	[73] 0.575 (0.242)

	reactive-proactive robust	**reactive**-*STNU*	**proactive robust**-*STNU*
magnitude: [n] z (p)	[39] nan (nan)	[3] -9.837 (*)	[3] -9.837 (*)
	reactive: 1.0	reactive: 0.26	proactive robust: 0.26
	proactive robust: 1.0	STNU: 1.74	STNU: 1.74

Table 9: Pairwise comparison on time offline for noise factor c=2. Using a Wilcoxon test, proportion test and magnitude test. The Wilcoxon and proportion tests include all instances for which at least one of the two methods found a feasible solution. The magnitude test includes all instances for which both methods found a feasible solution. Note that the ordering matters: the first method (bolded) shown is the better of the two in each pair, method 1 - method 2. If no algorithm is bold, neither outperforms the other. Each cell shows on the first row [nr pairs] 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. Each cell shows on the third row [nr pairs] t-stat (p-value) with (*) for $p < 0.05$, on the fourth row the normalised average of method 1 and on the fifth row the normalised average of method 2.

j10	*reactive-proactive robust*	**reactive**-$STNU$	**proactive robust**-$STNU$
Wilcoxon : [n] z (p)	[1000] nan (nan)	[1000] -27.393 (*)	[1000] -27.393 (*)
proportion: [n] z (p)	[nan] nan (nan)	[1000] 1.0 (*)	[1000] 1.0 (*)

	proactive robust-reactive	**reactive**-$STNU$	**proactive robust**-$STNU$
magnitude: [n] z (p)	[1000] nan (nan)	[113] -51.558 (*)	[113] -51.558 (*)
	reactive: 1.0	reactive: 0.26	proactive robust: 0.26
	proactive robust: 1.0	STNU: 1.74	STNU: 1.74

j20	*proactive robust-reactive*	**reactive**-$STNU$	**proactive robust**-$STNU$
Wilcoxon : [n] z (p)	[1000] nan (nan)	[1000] -27.393 (*)	[1000] -27.393 (*)
proportion: [n] z (p)	[nan] nan (nan)	[1000] 1.0 (*)	[1000] 1.0 (*)

	proactive robust-reactive	**reactive**-$STNU$	**proactive robust**-$STNU$
magnitude: [n] z (p)	[1000] nan (nan)	[15] -26.676 (*)	[15] -26.676 (*)
	reactive: 1.0	reactive: 0.25	proactive robust: 0.25
	proactive robust: 1.0	STNU: 1.75	STNU: 1.75

Table 10: Pairwise comparison on time offline for noise factor c=1 with no start-to-end and end-to-end relationships. Using a Wilcoxon test, proportion test and magnitude test. The Wilcoxon and proportion tests include all instances for which at least one of the two methods found a feasible solution. The magnitude test includes all instances for which both methods found a feasible solution. Note that the ordering matters: the first method (bolded) shown is the better of the two in each pair, method 1 - method 2. If no algorithm is bold, neither outperforms the other. Each cell shows on the first row [nr pairs] 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. Each cell shows on the third row [nr pairs] t-stat (p-value) with (*) for $p < 0.05$, on the fourth row the normalised average of method 1 and on the fifth row the normalised average of method 2.

j10	*reactive-proactive robust*	**reactive**-$STNU$	**proactive robust**-$STNU$
Wilcoxon : [n] z (p)	[1000] nan (nan)	[1000] -27.393 (*)	[1000] -27.393 (*)
proportion: [n] z (p)	[nan] nan (nan)	[1000] 1.0 (*)	[1000] 1.0 (*)

	proactive robust-reactive	**reactive**-$STNU$	**proactive robust**-$STNU$
magnitude: [n] z (p)	[1000] nan (nan)	[73] -26.028 (*)	[73] -26.028 (*)
	reactive: 1.0	reactive: 0.33	proactive robust: 0.33
	proactive robust: 1.0	STNU: 1.67	STNU: 1.67

j20	*proactive robust-reactive*	**reactive**-$STNU$	**proactive robust**-$STNU$
Wilcoxon : [n] z (p)	[1000] nan (nan)	[1000] -27.393 (*)	[1000] -27.393 (*)
proportion: [n] z (p)	[nan] nan (nan)	[1000] 1.0 (*)	[1000] 1.0 (*)

	proactive robust-reactive	**reactive**-$STNU$	**proactive robust**-$STNU$
magnitude: [n] z (p)	[1000] nan (nan)	[8] -7.728 (*)	[8] -7.728 (*)
	reactive: 1.0	reactive: 0.31	proactive robust: 0.31
	proactive robust: 1.0	STNU: 1.69	STNU: 1.69

Table 11: Pairwise comparison on time offline for noise factor c=2 with no start-to-end and end-to-end relationships. Using a Wilcoxon test, proportion test and magnitude test. The Wilcoxon and proportion tests include all instances for which at least one of the two methods found a feasible solution. The magnitude test includes all instances for which both methods found a feasible solution. Note that the ordering matters: the first method (bolded) shown is the better of the two in each pair, method 1 - method 2. If no algorithm is bold, neither outperforms the other. Each cell shows on the first row [nr pairs] 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. Each cell shows on the third row [nr pairs] t-stat (p-value) with (*) for $p < 0.05$, on the fourth row the normalised average of method 1 and on the fifth row the normalised average of method 2.

j10	**proactive robust**-$reactive$	**STNU**-$reactive$	**proactive robust**-$STNU$
Wilcoxon : [n] z (p)	[235] -10.673 (*)	[383] -4.842 (*)	[375] -8.207 (*)
proportion: [n] z (p)	[235] 0.949 (*)	[383] 0.507 (0.838)	[375] 0.595 (*)

	proactive robust-$reactive$	**STNU**-$reactive$	**proactive robust**-$STNU$
magnitude: [n] z (p)	[221] -5825.29 (*)	[44] -101.291 (*)	[42] -791.108 (*)
	proactive robust: 0.01	STNU: 0.19	proactive robust: 0.07
	reactive: 1.99	reactive: 1.81	STNU: 1.93

j20	**proactive robust**-$reactive$	$STNU$-$reactive$	**proactive robust**-$STNU$
Wilcoxon : [n] z (p)	[37] -4.322 (*)	[50] -0.748 (0.454)	[52] -4.727 (*)
proportion: [n] z (p)	[37] 0.946 (*)	[50] 0.36 (0.066)	[52] 0.673 (*)

	proactive robust-$reactive$	$STNU$-$reactive$	$proactive\ robust$-$STNU$
magnitude: [n] z (p)	[31] -9261.66 (*)	[1] nan (nan)	[1] nan (nan)
	proactive robust: 0.0	STNU: 0.27	proactive robust: 0.02
	reactive: 2.0	reactive: 1.73	STNU: 1.98

Table 12: Pairwise comparison on time online for noise factor c=1. Using a Wilcoxon test, proportion test and magnitude test. The Wilcoxon and proportion tests include all instances for which at least one of the two methods found a feasible solution. The magnitude test includes all instances for which both methods found a feasible solution. Note that the ordering matters: the first method (bolded) shown is the better of the two in each pair, method 1 - method 2. If no algorithm is bold, neither outperforms the other. Each cell shows on the first row [nr pairs] 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. Each cell shows on the third row [nr pairs] t-stat (p-value) with (*) for $p < 0.05$, on the fourth row the normalised average of method 1 and on the fifth row the normalised average of method 2.

j10	**proactive robust**-$reactive$	**STNU**-$reactive$	**proactive robust**-$STNU$
Wilcoxon : [n] z (p)	[225] -8.407 (*)	[381] -5.581 (*)	[363] -7.133 (*)
proportion: [n] z (p)	[225] -8.407 (*)	[381] -5.581 (*)	[363] -7.133 (*)

	proactive robust-$reactive$	**STNU**-$reactive$	**proactive robust**-$STNU$
magnitude: [n] z (p)	[203] -8177.742 (*)	[40] -99.327 (*)	[38] -469.392 (*)
	proactive robust: 0.01	STNU: 0.18	proactive robust: 0.07
	reactive: 1.99	reactive: 1.82	STNU: 1.93

j20	**proactive robust**-$reactive$	**STNU**-$reactive$	**proactive robust**-$STNU$
Wilcoxon : [n] z (p)	[43] -5.228 (*)	[71] -2.659 (*)	[73] -4.186 (*)
proportion: [n] z (p)	[43] 0.977 (*)	[71] 0.479 (0.812)	[73] 0.575 (0.242)

	proactive robust-$reactive$	**STNU**-$reactive$	**proactive robust**-$STNU$
magnitude: [n] z (p)	[39] -10424.541 (*)	[3] -23.526 (*)	[3] -265.878 (*)
	proactive robust: 0.0	STNU: 0.25	proactive robust: 0.03
	reactive: 2.0	reactive: 1.75	STNU: 1.97

Table 13: Pairwise comparison on time online for noise factor c=2. Using a Wilcoxon test, proportion test and magnitude test. The Wilcoxon and proportion tests include all instances for which at least one of the two methods found a feasible solution. The magnitude test includes all instances for which both methods found a feasible solution. Note that the ordering matters: the first method (bolded) shown is the better of the two in each pair, method 1 - method 2. If no algorithm is bold, neither outperforms the other. Each cell shows on the first row [nr pairs] 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. Each cell shows on the third row [nr pairs] t-stat (p-value) with (*) for $p < 0.05$, on the fourth row the normalised average of method 1 and on the fifth row the normalised average of method 2.

j10	**proactive robust**-$reactive$	**reactive**-$STNU$	**proactive robust**-$STNU$
Wilcoxon : [n] z (p)	[1000] -27.393 (*)	[1000] -26.688 (*)	[1000] -27.393 (*)
proportion: [n] z (p)	[1000] 1.0 (*)	[1000] 0.887 (*)	[1000] 1.0 (*)

	proactive robust-$reactive$	**STNU**-$reactive$	**proactive robust**-$STNU$
magnitude: [n] z (p)	[1000] -14995.664 (*)	[113] 207.391 (*)	[113] -1284.565 (*)
	proactive robust: 0.01	STNU: 0.18	proactive robust: 0.07
	reactive: 1.99	reactive: 1.82	STNU: 1.93

j20	**proactive robust**-$reactive$	**reactive**-$STNU$	**proactive robust**-$STNU$
Wilcoxon : [n] z (p)	[1000] -27.393 (*)	[1000] -27.38 (*)	[1000] -27.393 (*)
proportion: [n] z (p)	[1000] 1.0 (*)	[1000] 0.985 (*)	[1000] 1.0 (*)

	proactive robust-$reactive$	**STNU**-$reactive$	**proactive robust**-$STNU$
magnitude: [n] z (p)	[1000] -48515.729 (*)	[15] 74.127 (*)	[15] -668.258 (*)
	proactive robust: 0.0	STNU: 0.22	proactive robust: 0.03
	reactive: 2.0	reactive: 1.78	STNU: 1.97

Table 14: Pairwise comparison on time online for noise factor c=1 with no start-to-end and end-to-end relationships. Using a Wilcoxon test, proportion test and magnitude test. The Wilcoxon and proportion tests include all instances for which at least one of the two methods found a feasible solution. The magnitude test includes all instances for which both methods found a feasible solution. Note that the ordering matters: the first method (bolded) shown is the better of the two in each pair, method 1 - method 2. If no algorithm is bold, neither outperforms the other. Each cell shows on the first row [nr pairs] 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. Each cell shows on the third row [nr pairs] t-stat (p-value) with (*) for $p < 0.05$, on the fourth row the normalised average of method 1 and on the fifth row the normalised average of method 2.

j10	**proactive robust**-$reactive$	**reactive**-$STNU$	**proactive robust**-$STNU$
Wilcoxon : [n] z (p)	[1000] -27.393 (*)	[1000] -27.097 (*)	[1000] -27.393 (*)
proportion: [n] z (p)	[1000] 1.0 (*)	[1000] 0.927 (*)	[1000] 1.0 (*)

	proactive robust-$reactive$	**STNU**-$reactive$	**proactive robust**-$STNU$
magnitude: [n] z (p)	[1000] -17311.034 (*)	[73] 211.909 (*)	[73] -669.699 (*)
	proactive robust: 0.01	STNU: 0.16	proactive robust: 0.08
	reactive: 1.99	reactive: 1.84	STNU: 1.92

j20	**proactive robust**-$reactive$	**reactive**-$STNU$	**proactive robust**-$STNU$
Wilcoxon : [n] z (p)	[1000] -27.393 (*)	[1000] -27.389 (*)	[1000] -27.393 (*)
proportion: [n] z (p)	[1000] 1.0 (*)	[1000] 0.992 (*)	[1000] 1.0 (*)

	proactive robust-$reactive$	**STNU**-$reactive$	**proactive robust**-$STNU$
magnitude: [n] z (p)	[1000] -52164.235 (*)	[8] 46.044 (*)	[8] -472.131 (*)
	proactive robust: 0.0	STNU: 0.2	proactive robust: 0.03
	reactive: 2.0	reactive: 1.8	STNU: 1.97

Table 15: Pairwise comparison on time online for noise factor c=2 with no start-to-end and end-to-end relationships. Using a Wilcoxon test, proportion test and magnitude test. The Wilcoxon and proportion tests include all instances for which at least one of the two methods found a feasible solution. The magnitude test includes all instances for which both methods found a feasible solution. Note that the ordering matters: the first method (bolded) shown is the better of the two in each pair, method 1 - method 2. If no algorithm is bold, neither outperforms the other. Each cell shows on the first row [nr pairs] 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. Each cell shows on the third row [nr pairs] t-stat (p-value) with (*) for $p < 0.05$, on the fourth row the normalised average of method 1 and on the fifth row the normalised average of method 2.