CP for Scheduling under Uncertainty

A Comparative Study of STNUs against Proactive and Reactive Approaches

Mayte Steeghs[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 26, 2025

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

Abstract

This report investigates the effectiveness of Simple Temporal Networks with Uncertainty (STNUs) for solving the Stochastic Flexible Job-Shop Scheduling Problem with Sequence-Dependent Setup Times (SFJSP-SDST), comparing it against proactive and reactive Constraint Programming (CP) approaches. Using a benchmark dataset with varying noise levels, the study evaluates solution quality, feasibility, and computational cost. Results show that the reactive method achieves the lowest makespan due to its dynamic rescheduling capability but incurs high online computation time. The proactive method offers fast execution, while the STNU-based approach provides a dynamically controllable schedule, albeit with conservative makespans.

1 Introduction

Industry 4.0 "digital factories" demand agile shop-floor scheduling algorithms that can react in real time to stochastic processing times, a natural fact of real-life manufacturing processes. The Flexible Job-Shop Problem Scheduling with Sequence-Dependent Set-Up Times (FJSP-SDST) extends the classical job-shop by allowing multiple alternative machines per operation and by imposing set-up times that depend on the order of consecutive tasks on the same machine. This temporal constraint reflects the dependencies present in many high-tech manufacturing processes.

Job-shop scheduling problems (JSP) with sequence-dependent setup times are known to be strongly NP-hard combinatorial problems [9]. Previously, Mixed Integer Programming (MIP) methods such as BACCHUS [8] and SORU-H [7] were considered state-of-the-art [20] for deterministic scheduling. However, recent work by Naderi et al. [16] demonstrates that Constraint Programming (CP) outperforms MIP across a wide range of benchmarking scheduling problem instances. This opens a research gap for finding robust proactive schedules or reactive approaches with rescheduling during execution, which was previously considered too computationally heavy [20].

Deterministic FJSP-SDST instances remain NP-hard and have been tackled with MIP, CP, and meta-heuristics (e.g., Evolutionary Algorithms, Variable Neighborhood Search, and Simulated Annealing) [1]. However, these deterministic schedules lose feasibility or optimality when activity durations deviate from their nominal values.

More recently, AI-enhanced and learning-based methods have been explored to solve FJSP variants by learning heuristics or generating schedules from data [5]. While promising in scalability and adaptability, these methods often require extensive training data and lack rigorous feasibility guarantees under uncertainty. In contrast, graph-theoretic approaches such as Simple Temporal Networks with Uncertainty (STNUs) offer provable guarantees of feasibility under bounded duration uncertainty [17].

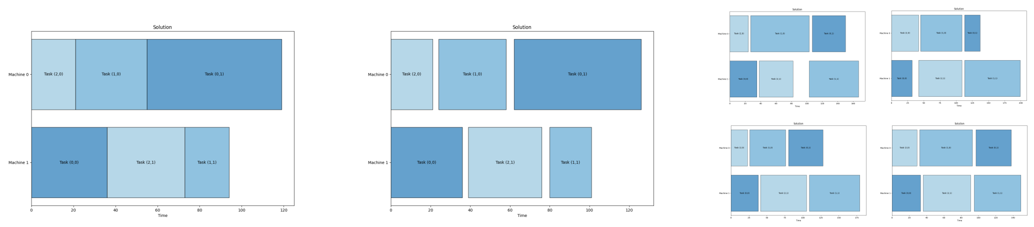
Recent work by van den Houten et al. [20] showed that representing a resource-feasible partial-order schedule (POS) as a STNU, and dispatching it with the real-time algorithm RTE* [10], can outperform proactive and reactive CP approaches on the stochastic RCPSP/max benchmark. Yet, the potential of STNUs for multi-machine, sequence-dependent problems such as FJSP-SDST has not been studied.

In this paper, we investigate whether an STNU-based method yields superior solution quality and runtime for the stochastic Flexible Job-Shop Scheduling Problem with sequence-dependent set-up times compared to proactive and reactive CP methods. Specifically, we address the following research questions

1. How does **uncertainty** affect performance and feasibility of solutions?
2. To what extent do **sequence-dependent set-up times** affect makespan, the feasibility ratio, and computational time (both offline and online)?
3. How robust are the different methodologies when **scaling** input problem size?

We hypothesize that the STNU-based method will achieve a lower expected makespan and higher feasibility ratio than both proactive and reactive CP approaches, owing to its use of dynamic controllability to absorb stochastic duration deviations. Moreover, we anticipate that the STNU framework will incur lower online computation time, especially as instance size grows, by avoiding repeated CP solves required by the reactive method.

2 The Scheduling Problem



(a) FJSP (b) FJSP with SDST (c) Stochastic FJSP with SDST

Figure 1: Scheduling Example Stochastic FJSP with SDST

The scheduling problem studied in this work is the Flexible Job Shop Problem (FJSP) with sequence-dependent setup times (SDST) and stochastic task durations. This section defines the key components of the problem formulation.

Flexible Job Shop Problem (FJSP)

Let J denote the set of jobs, T the set of tasks, and M the set of machines. A job $j \in J$ consists of a sequence of tasks denoted as $T_j \subseteq T$. Each task $t \in T$ must be assigned to exactly one machine $m \in M$. Each machine can process at most one task at a time. The tasks $t_1, t_2, \ldots, t_n \in T_j$ associated with job j must be scheduled in order, such that each task t_{i+1} cannot start until t_i has completed.

A schedule specifies, for each task $t \in T$: (i) its start and end times, and (ii) the machine to which it is assigned, while respecting all precedence and resource constraints. The objective is to minimize the maximum end time, also known as the *makespan*.

Sequence-Dependent Setup Times

For each machine $m \in M$, a sequence-dependent setup time $s_{t,t',m} \geq 0$ is defined for every ordered pair of tasks (t, t') scheduled consecutively on m. This represents the time required to reconfigure the machine from task t to task t'. Sequence-dependent setup times are generally considered one of the more challenging aspects of scheduling problems [12]. SDST can be visualized as gaps between tasks on the same machine as illustrated by the differences in Figure 1 a and Figure 1 b

Stochastic Task Durations

In real-world applications such as semiconductor fabrication or bio-based manufacturing, task durations are typically uncertain. We assume that task durations follow a stochastic distribution. This extension of the FJSP is referred to as the Stochastic FJSP (SFJSP), and it is considered even more difficult to solve due to the uncertainty involved in task durations [6].

Task durations are modeled as independent random variables denoted by d_j, representing the duration of task j. Each duration becomes known upon task completion. We assume that each d_j follows a discrete uniform distribution, i.e., $d_j \sim \text{DiscreteUniform}(lb_j, ub_j)$, where lb_j and ub_j are the minimum and maximum possible durations for task j [20]. Figure 1 c visualizes this stochasticity by illustrating how 4 samples of d_j might yield 4 distinct schedules.

2.1 CP Model Formulation

CP is a paradigm used to solve constraint satisfaction problems. These problems consist of a finite set of variables, each having a discrete domain, and a set of constraints that must be satisfied. Applying this to FJSP, we introduce the following interval variables:

$$\begin{aligned}
&\forall t \in T,\ \forall m \in M : \quad && \tau_{t,m} : [\text{start}, \text{end}], \\
&\forall t \in T : && \tau_t : [\text{start}, \text{end}],
\end{aligned}$$

The CP model given those interval variables and the constraints presented above:

$$\begin{aligned}
\min \quad & \max_{t \in T} \texttt{EndOf}(\tau_t) && \text{s.t.} \\
& \forall t \in T : & \texttt{Alternative}\big(\tau_t, \{\tau_{t,m}\}_{m \in M}\big) \quad & (1) \\
& \forall j \in J,\ \forall (t_j, t_{j+1}) : & \texttt{EndBeforeStart}(\tau_{t_j}, \tau_{t_{j+1}}) \quad & (2) \\
& \forall m \in M : & \texttt{NoOverlap}\big(\{\tau_{t,m}\}_{t \in T}; \text{transition} = s_{t,t',m}\big) \quad & (3)
\end{aligned}$$

The objective function minimizes the makespan, defined as the maximum completion time of all tasks. Constraint (1) ensures that each task is assigned to exactly one machine from its set of available alternatives. Constraint (2) enforces the precedence relations within each job. Constraint (3) maintains that machines process at most one task at a time, incorporating sequence-dependent setup times.

This formulation is adapted from Echeverria et al. (2024) [5] with noOverlap (3) now being subject to sequence dependent set-up times. The syntax used is consistent with the IBM CP Solver syntax introduced by Laborie et al. (2018) [11].

3 Scheduling Methods

In the stochastic scheduling literature, two primary paradigms are distinguished: **proactive** and **reactive** approaches. Proactive scheduling seeks to construct a robust schedule a priori, anticipating uncertainties before execution, whereas reactive scheduling continually adjusts the scheduling in real time at each decision moment. These paradigms represent opposing ends of a methodological continuum; in both practice and literature, hybrid approaches are common [20]. In Flexible Job Shop Scheduling literature in particular, approaches commonly rely on sampling and optimization, leaning firmly on the more proactive side [3].

In this section, we briefly summarize the scheduling methods put forward by van den Houten et al.'s, adapted for FJSP with SDST and implemented in PyJobShop, an open-source Python library for solving scheduling problems with constraint programming [13]. Implementation details will only be mentioned where relevant. Pseudocode to aid the readers understanding can be found in Appendix A

3.1 Proactive Method

The proactive paradigm produces a robust schedule entirely offline; the feasibility of this schedule is then verified.

- **Offline:** Replace each uncertain processing window $[\ell_j, u_j]$ by a fixed *quantile* duration $\hat{d}_j$ (i.e. the γ–quantile). Solve the resulting deterministic problem to obtain start times S_j and machine assignments that remain feasible given that the realized durations do not exceed $\hat{d}_j$ (Proposition 1 in Van den Houten *et al.* [20]).
- **Online:** As soon as a task finishes, reveal its true duration and verify that all yet-to-start operations can still begin at their pre-computed S_j without violating any constraints.

The hyper-parameter γ governs the trade-off between robustness and makespan: $\gamma = 1$ yields the most conservative (upper-bound) schedule, whereas smaller γ values shorten the planned makespan at the risk of constraint violations if durations overrun $\hat{d}_j$.

3.2 Reactive Method

Reactive scheduling repeatedly reoptimizes once an operation's duration deviates from its offline estimate.

- **Offline:** Same as offline step of proactive method.
- **Online:** At every decision moment (when a task finishes), we resolve the deterministic problem but fix all the tasks we have already completed which we know now the actual durations of. We use the estimated durations for the tasks not yet completed.

Each realization of a duration immediately triggers a deterministic CP solve, dramatically increasing the online computation time, especially for larger instances. Therefore, the time limit T_{on} for resolving must be chosen carefully: too small, and rescheduling may fail to find any feasible partial schedule; too large, and online computation time may become prohibitive.

3.3 STNU-based Method

The Simple Temporal Network with Uncertainty (STNU) approach integrates proactive robustness with reactive adaptability through partial order schedules (POS). We use the concept of dynamic controllability (DC), as defined by Morris (2014) [15], guarantees that scheduling decisions remain feasible across all possible outcomes of these uncertain durations.

The STNU method comprises three principal phases:

- **Offline**: Same as offline step of proactive and reactive methods, but uses $\gamma = 1$.
- **Construct STNU**: The Partial Order Schedule (POS) derived from the CP solution is used to construct the STNU.
- **DC Check**: We verify the STNU's DC property. If DC, the STNU provides input to the Real-Time Execution (RTE*) algorithm [10], which adaptively schedules tasks in real-time, adjusting to actual operation durations as they occur.

In our STNU framework, each sequence-dependent setup time is encoded as a single ordinary edge from the finish node of task i to the start node of task j on machine m with weight $s_{t,t',m}$, thereby enforcing the deterministic delay directly in the temporal graph. An alternative would have been to model each setup as a "dummy" task (introducing a start and finish node plus a zero-variance contingent link), but that approach would inflate the graph by two nodes per setup, slow down dynamic-controllability checks, and require special-case handling in the reactive dispatcher. By using ordinary edges, we keep the STNU compact, semantically clear, and efficient in both DC checking and online execution.

4 Experimentation

We evaluate the three scheduling paradigms on the FJSP–SDST benchmark of Fattahi *et al.* (2007) [19], taken from the Job-Shop Benchmark Suite of Reijnen *et al.* (2023) [18]. Stochastic variants are generated by sampling durations from discrete uniform distributions for noise levels $\epsilon = \{1, 2\}$.

The two independent variables studied are the noise level ϵ and the scheduling methods, We record the (i) solution quality, split into makespan and feasibility, and the (ii) CPU time, split into offline initialization and online execution. For each of the 20 benchmark instances we generate 10 independent samples, resulting in 2×10 runs per method.

All CP models are solved in PyJobShop with the IBM CP solver, which has native support for sequence-dependent setup times [4].

We selected the robust $\gamma = 1$ schedule to guarantee feasibility under all realized processing-

Table 1: Performance and Feasibility Summary for Proactive across γ values, subset of instances (1-18)

Mode	Feasibility Ratio	Avg Makespan	Avg Offline Time
quantile_0.25	0.09	160.76471	0.05916
quantile_0.5	0.13	192.47826	0.08113
quantile_0.75	0.30	296.53704	0.18309
quantile_0.9	0.55	373.80808	0.50558
robust	**1.00**	**460.66667**	**0.83114**

time variations, thereby eliminating any risk of constraint violations during online verification. Although this choice produces the longest planned makespan, lowering γ to 0.9 cuts feasibility to just over 55% for only a modest reduction in average makespan as seen in Table 1.

We select $\gamma = 0.9$ for reactive the reactive method because it delivers the best balance between makespan and online computation effort, as seen in Table 2. Compared to the fully robust (upper-bound) and more optimistic (mean or lower quantiles) settings, the 0.9 quantile minimizes total online time without incurring excessive solution quality deterioration. We chose to evaluate the hyperparameter for the larger instances (10 - 20) because this is where the computational online time becomes prohibitive.

Note that the reactive method is always feasible because despite 'freezing' completed tasks, it resolves the remaining problem, accounting for any sequencing constraints applied to the CP solver. Since we do not have hard deadlines or resource constraints, the CP can just spread out tasks to compensate for discrepancies.

Table 2: Performance Summary for Reactive across γ values, subset of instances (10-20)

Mode	Avg. Makespan	Avg. Online Time	Avg. Offline Time
mean	731.91	67.80	221.47
robust	738.70	64.90	331.67
quantile$_{0.9}$	**735.51**	**65.39**	**264.46**
quantile$_{0.75}$	731.35	66.22	222.84
quantile$_{0.5}$	728.86	67.20	212.35
quantile$_{0.25}$	723.71	67.62	104.51

Benchmark Instances

In this section we will analyse the composition, distribution and patterns within the 20 benchmark instances which is necessary to sufficiently understand our results. All figures and values referred to in this section can be found in Appendix B. The following properties hold across all instances:

- Every job has the same amount of tasks.
- Every task can be scheduled on every available machine.

- A non-zero setup time exists between every task pair on every machine.

In our dataset, infeasible transitions between tasks on machine m are encoded by a setup time of $S_\infty = 1{,}000{,}000$. This value vastly the analytical worst-case horizon $H_{\max}$ ($\lesssim 33\,000$), enforcing that any transition with $s_{t,t',m} \geq S_\infty$ is infeasible. These infeasible assignments reduces the domain of the CP model. In large instances, up to 90% of setup times are S_∞, effectively pruning those infeasible assignments and drastically shrinking the search space so the problem becomes tractable.

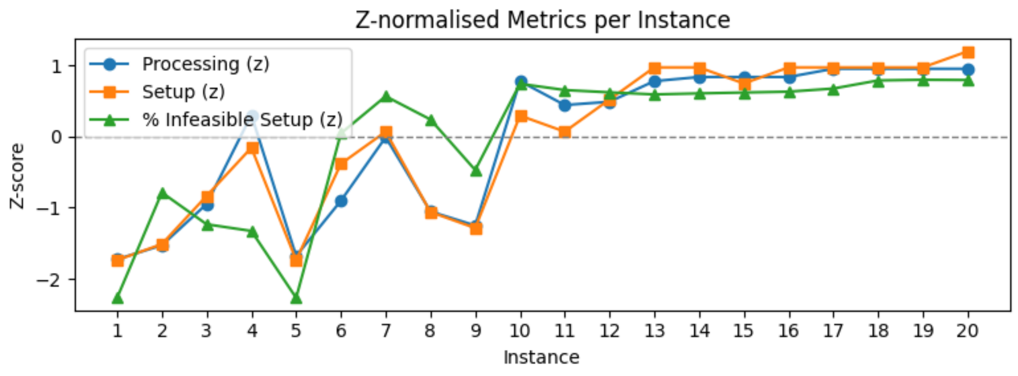


Figure 2: Z-normalized median processing times, setup times, and proportion of feasible setup times

To compare instance characteristics on a common scale, we report z-normalized medians of processing times, setup times, and the proportion of feasible setups in Figure **??**. Notably, instances 10 through 20 form a plateau across all three metrics, indicating consistent task durations, setup-time distributions, and infeasibility ratios. This homogeneity implies that, for these latter instances, observed changes in solution quality and computational effort can be attributed primarily to increases in problem size rather than to shifts in underlying instance structure. This assumption is necessary for answering research sub-question 3 concerning scalability. We nevertheless acknowledge that, without a detailed analysis of the underlying structural graphs, this conclusion must remain tentative.

5 Results

We begin by examining the general trends between methods. Then we break down findings to answer our sub-research questions, pertaining to uncertainty, sequence-depednent setup times, and scalability respectively. Further results can be found in Appendix C

Solution Quality

Across all instances of FJSP with SDST, the reactive policy consistently attains the lowest makespan compared to both proactive and STNU methods.

From Table 3 we can observe the following relationship of the partial ordering between methods.

Test	Legend	react–stnu	stnu–proactive	proactive–reactive
Wilc. Quality	[n] z (p)	[400] 13.650 (*)	[400] -9.850 (*)	[400] -18.450 (*)
Prop. Quality	[n] prop (p)	[400] 0.843 (*)	[400] 0.252 (*)	[400] 0.037 (*)
Magn. Quality	[n] t (p)	[400] -21.496 (*)	[400] 11.796 (*)	[400] 28.616 (*)
	norm. avg.	react: 0.896	stnu: 1.058	$\text{pro}_{0.9}$: 1.047
	norm. avg.	stnu: 1.104	$\text{pro}_{0.9}$: 0.942	react: 0.953
Wilc. Offline	[n] z (p)	[400] 6.450 (*)	[400] -4.850 (*)	[400] -13.950 (*)
Prop. Offline	[n] prop (p)	[400] 0.338 (*)	[400] 0.377 (*)	[400] 0.850 (*)
Magn. Offline	[n] t (p)	[400] 6.517 (*)	[400] 6.695 (*)	[400] -16.675 (*)
	norm. avg.	react: 1.071	stnu: 1.059	$\text{pro}_{0.9}$: 0.869
	norm. avg.	stnu: 0.929	$\text{pro}_{0.9}$: 0.941	react: 1.131
Wilc. Online	[n] z (p)	[400] 19.950 (*)	[400] -19.850 (*)	[400] -19.950 (*)
Prop. Online	[n] prop (p)	[400] 0.000 (*)	[400] 0.002 (*)	[400] 1.000 (*)
Magn. Online	[n] t (p)	[400] 463.854 (*)	[400] 277.109 (*)	[400] -23411.810 (*)
	norm. avg.	react: 1.955	stnu: 1.942	$\text{pro}_{0.9}$: 0.001
	norm. avg.	stnu: 0.045	$\text{pro}_{0.9}$: 0.058	react: 1.999

Table 3: Pairwise test results for $\epsilon = all$, grouping=1-20.

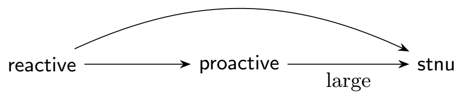


Figure 3: Summarizing illustration of the partial ordering of the different methods for solution quality

The reactive method has an edge due to its ability to dynamically adjust the schedule in real-time, effectively managing unforeseen variations in task durations. Its continuous adaptation allows for consistently tighter scheduling and optimal machine routing. However, it is important to acknowledge that this advantage might stem partially from the significantly higher online computation time required by the reactive method compared to the other two methods. Future research involving detailed hyperparameter tuning of the time-out settings for the reactive method could ensure a more equitable comparison and potentially provide additional insights.

In contrast to our initial hypothesis, the proactive method outperforms the STNU-based approach in terms of makespan, particularly for larger and more complex instances. The need to satisfy the DC property, inherently embedding significant conservatism to guarantee feasibility under worst-case scenarios might explain this difference. In our implementation, we explicitly model each SDST as contingent links might further compounds this conservatism, resulting in substantial cumulative slack throughout the schedule. As instances grow larger and more complex, the increased number of contingent SDST links significantly raises the density and complexity of the STNU graph. Consequently, the stringent DC constraints might restrict schedule flexibility, forcing tasks to start later than necessary and inflating the makespan. Alternative implementations of the SDSTs in the STNU might avoid this over-conservatism.

Offline Time

$$\begin{tikzcd}\text{reactive} & \text{proactive} \arrow[l] \arrow[r] & \text{stnu} \arrow[ll, bend left]\end{tikzcd}$$

Figure 4: Partial ordering of the different methods for offline computation time

The following partial orderings in terms of offline computation time make sense given the underlying implementation. The STNU and proactive methods start with the exact same deterministic single point solution calculation, and then the STNU method goes on to contsruct an STNU, increasing its offline time. The reactive method uses $\gamma = 0.9$ and therefore has to recompute the bounds before performing this same deterministic calculation as above, where additional offline computation time might be incurred.

Online Time

$$\begin{tikzcd}\text{reactive} & \text{proactive} \arrow[l] \arrow[r] & \text{stnu} \arrow[ll, bend left]\end{tikzcd}$$

Figure 5: Partial ordering of the different methods for online computation time

At first glance, from Table 3 it is obvious that the reactive method has dramatically higher online computation time, explained by the repeated CP resolves. Furthermore, the fact that proactive has the smallest online time can be explained by the minimal constant time check performed to verify if the scheduled constructed under estimated durations is still feasible under the realization durations.

5.1 Uncertainty

Comparing the three methods across noise levels $\gamma = 1, 2$ as done in Table 14, we notice the following:

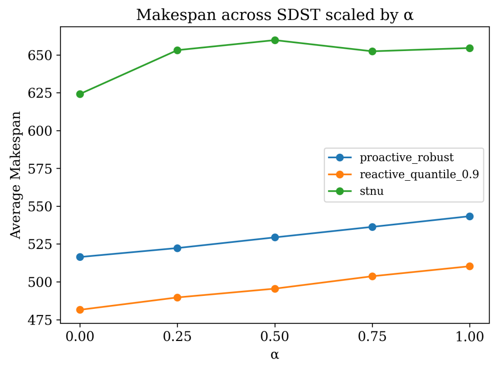
- The proactive method has considerably lower makespan for $\epsilon = 1$
- The online and offline computation times of the proactive and reactive methods are much smaller for $\gamma = 1$ compared to $\gamma = 2$. There is no statistically significant difference in computation times for the STNU method.

Test	Metric	reactive ($\epsilon = 1$ vs $\epsilon = 2$)	proactive ($\epsilon = 1$ vs $\epsilon = 2$)	stnu ($\epsilon = 1$ vs n2)
Wilc. Quality	[n] z (p)	[200] 0.636 (0.525)	[200] 12.374 (*)	[200] 1.909 (0.056)
Prop. Quality	[n] prop (p)	[200] 0.475 (0.525)	[200] 0.940 (*)	[200] 0.570 (0.056)
Magn. Quality	[n] t (p) norm. avg. norm. avg.	[200] -0.327 (0.744) $\epsilon = 1$: 0.999 $\epsilon = 2$: 1.001	[200] -22.108 (*) $\epsilon = 1$: 0.969 $\epsilon = 2$: 1.031	[200] -2.017 (*) $\epsilon = 1$: 0.995 $\epsilon = 2$: 1.005
Wilc. Offline	[n] z (p)	[200] 11.243 (*)	[200] -0.071 (0.944)	[200] 0.212 (0.832)
Prop. Offline	[n] prop (p)	[200] 0.900 (*)	[200] 0.500 (0.944)	[200] 0.490 (0.832)
Magn. Offline	[n] t (p) norm. avg. norm. avg.	[200] -17.123 (*) $\epsilon = 1$: 0.805 $\epsilon = 2$: 1.195	[200] -2.938 (*) $\epsilon = 1$: 0.968 $\epsilon = 2$: 1.032	[200] 0.468 (0.640) $\epsilon = 1$: 1.002 $\epsilon = 2$: 0.998
Wilc. Online	[n] z (p)	[200] 10.536 (*)	[200] 4.738 (*)	[200] 0.495 (0.621)
Prop. Online	[n] prop (p)	[200] 0.875 (*)	[200] 0.330 (*)	[200] 0.480 (0.621)
Magn. Online	[n] t (p) norm. avg. norm. avg.	[200] -18.441 (*) $\epsilon = 1$: 0.888 $\epsilon = 2$: 1.112	[200] 4.998 (*) $\epsilon = 1$: 1.045 $\epsilon = 2$: 0.955	[200] 0.161 (0.872) $\epsilon = 1$: 1.001 $\epsilon = 2$: 0.999

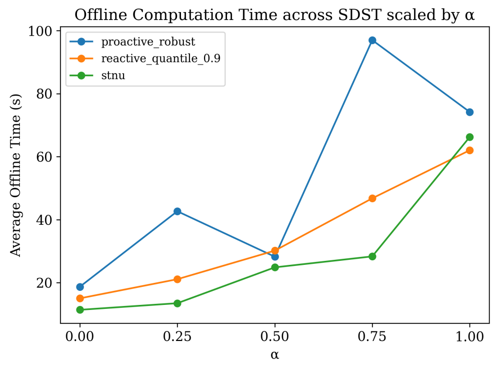
Table 4: Noise 1 vs Noise 2: pairwise tests per method (legend shows metric).

5.2 Sequence Dependent Setup Times

We performed an experiment in which we draw 10 samples for each method with different magnitudes of sequence-dependent setup times. This is achieved by scaling the datasets original SDST by a scale factor, denoted $\alpha = (0, 0.25, 0.5, 0.75, 1$.



(a) Average makespan vs. α



(b) Average offline time vs. α

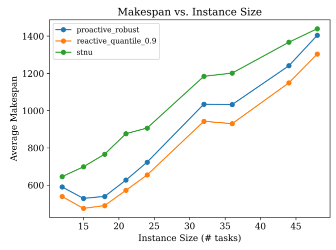
Figure 6: Impact of setup-time scaling (α) on makespan and offline computation time.

In Figure 6 we observe a fairly linear increase in makespan as the sequence-dependent setup times increase. Furthermore, we notice a more dramatic increase in computation time offline as we scale the SDST, seeing almost a 4 fold increase.

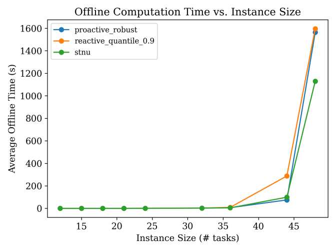
For the online computation time we observed no statistically significant difference across any methods.

5.3 Scalability

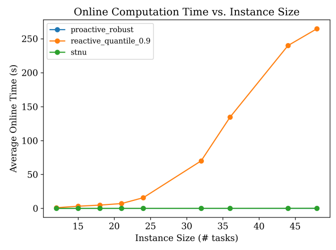
As explored in the Benchmark Section above, we can assume some features of the instances are homogeneous, such as distribution of setup times, task durations and feasibility ratios of setup times for instances 10 - 20. Based on this assumption, we will investigate how scaling the input size (total # of tasks) impacts the makespan, offline and online computation times.



(a) Average makespan vs. instance size



(b) Average offline time vs. instance size



(c) Average online time vs. instance size

Figure 7: Performance metrics (makespan, offline time, online time) as functions of instance input size for instances 10–20.

We notice fairly linear increases in makespan as input size increases, consistent across scheduling methods. We can assume that the inconsistencies are due to some underlying structural factors in the instances we did not control for.

Interestingly, the average computation time offline exponentially increases dramatically as the input size increases, observing a strong spike between 30 and 45 tasks. This pattern is consistent across the three methods.

Finally, we observe that the online time increases quite steeply for the reactive method, unlike the two other methods. This is inline with the fact that the reactive method recomputes at every decision point, therefore, it is expected that as the input size increases this resolving time will also increase.

6 Responsible Research

6.1 Reproducibility

All code, data, and environment specifications required to reproduce our experiments are publicly available at `https://github.com/kimvandenhouten/PyJobShopSTNUs/tree/fjsp-sdst/PyJobShopIntegration`. The implementation corresponding to my contribution is located in the `fjsp-sdst` branch. We conducted all experiments on the 20 FJSP–SDST benchmark instances from Fattahi et al. [19] ("Job-Shop Benchmark Suite") to ensure comparability with prior work in the literature [18].

Our repository is structured as follows:

- **Core modeling:**
  - `FJSP.py` (basic CP model)

    - `PyJobShopSTNU.py` (STNU extension)

- **Experimental pipelines:**
    - `FJSP_pipeline.py` (includes STNU method)
    - `FJSP_pipeline_scaled.py`

- **Reactive vs. Proactive strategies:**
    - `reactive.py`
    - `proactive.py`

- **Evaluation:**
    - `evaluate_gantt_plots.py` (Gantt-chart analyses)
    - `evaluator.py` (metrics computation)
    - `instance_exploration.py` (instance-feature analysis, Appendix B)

The `README` file provides instructions to reproduce this experiment. Hyperparameter settings used to perform our experiments can be found in Appendix C.

Experiments were run on a TU Delft server equipped with an AMD EPYC 7662 "Rome" CPU (12 virtual cores, single-threaded), using IBM CP Optimizer v22.1.1. All runtimes report the wall-clock execution time (in seconds), as measured by Python's `time.time()` function.

The number of worker threads was not explicitly fixed; PyJobShop defaults to using all available CPU cores. This decision acknowledges that hardware configurations—such as core counts, clock speeds, and cache sizes—vary widely across different systems, and by relying on the default "use all cores" behavior we ensure that our results naturally adapt to each execution environment.

6.2 Ethics

In conducting research on stochastic flexible job–shop scheduling, it is crucial to recognize that responsible research extends beyond algorithmic performance and reproducibility. Even though no human subjects are directly involved in our computational experiments, the schedules we generate may be deployed in manufacturing settings where aggressive makespan minimization can translate into intensified workloads, unrealistic deadlines, or de facto surveillance of shop-floor workers. On the other hand you could argue, that a better understanding of the uncertainty in processing times might reveal opportunities for slack and flexibility that deterministic schedules conceal, reducing strain on workers.

Furthermore, this project was conducted collaboratively by a team of five students, each addressing a distinct subproblem related to STNU-based scheduling under uncertainty in parallel, upholding a transparent, respectful, and constructive workflow by each owning a distinct subproblem and temporal constraints while jointly sharing code, providing feedback, and exchanging ideas. This collaborative approach aligns with TU Delft's Integrity Statement, specifically promoting: (h) transparency in day-to-day operations; (i) openness to constructive feedback; (k) a healthy work environment; and (l) a 'DIRECT' culture characterized by Diversity, Integrity, Respect, Engagement, Courage, and Trust [2].

7 Discussion

This section aims to reflect on the methodology choices and reliability of the experiment.

First, we adopted a sampling regime of 10 independent realizations per instance to estimate makespan and feasibility ratios, mirroring prior work and facilitating direct comparison with van den Houten et al. [20]. However, 10 samples may limit statistical confidence in our results. Though we found sufficiently low p-values for our results, increasing the number of samples would reduce variance in the estimated metrics and could reveal subtler differences between methods at instance granularity. On the other hand, larger sample sets impose greater computational cost, especially for the reactive method, where each deviation triggers a full CP solve.

Second, we modeled task durations as discrete uniform random variables, however, this might not capture real-world duration variability, which often exhibits skewness or heavy tails [14]. Future studies should investigate alternative distributions to assess the robustness of each scheduling paradigm under more realistic and diverse uncertainty profiles.

Third, our assumption that instances 10–20 "scale" homogeneously relies on the plateau observed in z-normalized medians of processing times, setup times, and infeasibility ratios (as seen in Benchmark Instances). This homogeneity suggests that increases in problem size predominantly drive solution quality changes. However, without a deeper graph-structural analysis this conclusion remains tentative. A more granular characterization of instance topology could disentangle the effects of scale from structural complexity.

Fourth, all runtime measurements were recorded as wall-clock times on a shared TU Delft server, which may have experienced fluctuating loads from concurrent users. Capturing CPU time and performing more extensive hyperparameter tuning on the time outs of the CP solvers in concurrence with the hardware might produce more consistent results.

Fifth, in our scaled SDST experiment we increased the magnitude of sequence-dependent setup times while preserving the proportion of infeasible arcs. This design choice maintained the original graph's connectivity pattern, simplifying isolation of magnitude effects. Nonetheless, it may obscure interactions between infeasibility density and scheduling behavior.

Finally, our study does not include comparisons against MIP-based benchmarks such as BACCUS [8] or SORU-H [7]. Incorporating a mixed-integer programming baseline would provide a more comprehensive perspective on the state-of-the-art, closing the literature gap presented in the introduction.

8 Conclusions and Future Work

This research investigated the performance of Simple Temporal Networks with Uncertainty (STNU) applied to the Stochastic Flexible Job-Shop Scheduling Problem with Sequence-Dependent Setup Times (SFJSP-SDST), comparing it to proactive and reactive Constraint Programming (CP) methods. Our results demonstrated that each scheduling paradigm possesses distinct strengths and limitations, but general concludes that the reactive method has a distinct advantage due to its flexibility.

The reactive CP method consistently achieved the lowest makespan, benefitting from its ability to dynamically reoptimize schedules based on real-time task durations. However, this advantage was counterbalanced by significantly higher online computational demands, particularly noticeable as problem sizes increased. The proactive CP method offered computational efficiency with minimal online demands, yet its schedules proved fragile under conditions where actual task durations exceeded pre-planned buffers. The STNU-based approach struck a balance between robustness and responsiveness, ensuring feasibility through dynamic controllability while maintaining minimal online time. However, the addition of sequence dependent times to the STNU implementation, especially for complex instances, made the network very dense, which might explain the increased conservatism leading to higher makespans.

To answer our sub-questions proposed in the introduction we conclude the following:

1. **Uncertainty:** Raising the noise level from $\epsilon = 1$ to $\epsilon = 2$ inflated the proactive makespan and increased both its offline and online times, while the reactive and STNU makespans stayed unchanged; however, higher noise dramatically increased the reactive method's online computation.

2. **SDST**: increasing SDST linearly affected makespan and exponentially impacted offline computational times

3. **Scalability**: Moreover, scalability tests revealed that computational requirements grew significantly with problem size, particularly affecting the reactive method's online computational time.

The foremost caveat of this conclusion is that the fairly unbound online timeout of the reactive method might inequitably bias the results. Performing hyperparamater tuning on our dataset and setting the timeouts in an equitable way across methods, might yield new insights. This would also better reflect real-life requirements for scheduling approaches.

Future work should explore alternative distributions for task duration uncertainty to better reflect real-world manufacturing variability. Additionally, refining STNU implementations to reduce inherent conservatism could further improve performance. Finally, including mixed-integer programming methods as comparative benchmarks would enhance understanding of the full spectrum of available scheduling solutions, bridging existing literature gaps.

A Scheduling Approaches Pseudocode

A.1 Proactive Method

Algorithm 1 Proactive Scheduling Method (with N samples)

```
Require: A Flexible Job Shop instance $\mathcal{I}$ with SDST, quantile $\gamma$, offline time limit $T_{\text{off}}$
Ensure: Robust start-time assignment $\{s_j\}$ or "infeasible"
  For each operation $j$, let $\ell_j, u_j$ be bounds of its duration; compute
      $$\widehat{d_j} = \begin{cases} \ell_j, & \ell_j = u_j, \\ \lfloor \ell_j + \gamma\,(u_j - \ell_j + 1) - 1 \rfloor, & \text{otherwise.} \end{cases}$$
  Build CP model $\mathcal{M}$ for FJSP-SDST using durations $\{\widehat{d_j}\}$
  Solve $\mathcal{M}$ with CP Optimizer within $T_{\text{off}}$, yielding schedule $\mathcal{S}^*$
  if $\mathcal{S}^*$ infeasible then return infeasible
  end if
  Extract offline start times start_times[1..$n$] from $\mathcal{S}^*$
  Initialize array makespan[1..$N$]
  for $t = 1$ to $N$ do
      Draw duration_sample[$j$] $\sim$ Uniform($\ell_j, u_j$) for each $j$
      if CheckFeasible(start_times, duration_sample) = false then
          return infeasible
      else
          makespan[$t$] $\leftarrow \max_j$(start_times[$j$] + duration_sample[$j$]) + set-up delay[$j$]
      end if
  end for
  return $\{s_j\}$, makespan[1..$N$]
```

A.2 Reactive Method

Algorithm 2 Reactive Method

```
Require: A Flexible Job Shop instance $\mathcal{I}$, duration estimates $\{\tilde{d}_j\}$, offline time limit $T_{\text{off}}$,
    online time limit $T_{\text{on}}$
Ensure: Robust start-time assignment $\{s_j\}$ and makespan

  Offline Initialization:
  Build CP model $\mathcal{M}$ for FJSP-SDST using $\{\tilde{d}_j\}$
  Solve $\mathcal{M}$ with CP Optimizer within $T_{\text{off}}$, yielding schedule $\mathcal{S}^*$
  if $\mathcal{S}^*$ infeasible then return infeasible
  end if
  Extract est_starts[$j$] from $\mathcal{S}^*$; set current_sol $\leftarrow \mathcal{S}^*$
  Initialize completed $\leftarrow \emptyset$ and $\{$real_fin[$j$]$\}$ undefined
  durations $\leftarrow \{\tilde{d}_j\}$

  Online Execution:
  while $|$completed$| < n$ do
      Observe real finish times for all $j \notin$ completed:

              real_fin[$j$] $\leftarrow$ est_starts[$j$] $+$ realized_duration[$j$]

      Let $j' = \arg\min_{j \notin \text{completed}}$ real_fin[$j$]; set $t_{\text{now}} \leftarrow$ real_fin[$j'$]
      Add $j'$ to completed
      if est_starts[$j'$] $+ \tilde{d}_{j'} \neq$ real_fin[$j'$] then          ▷ Deviation detected
          For each $j \notin$ completed, set $\tilde{d}_j \leftarrow$ realized_duration[$j$]
          Build CP model $\mathcal{M}'$ with updated $\{\tilde{d}_j\}$
          Warm-start $\mathcal{M}'$ with current_sol
          Solve $\mathcal{M}'$ within $T_{\text{on}}$, yielding $\mathcal{S}'$
          if $\mathcal{S}'$ infeasible then return infeasible
          end if
          Extract new est_starts[$j$] from $\mathcal{S}'$
          current_sol $\leftarrow \mathcal{S}'$
      end if
  end while
  makespan $\leftarrow \max_j ($real_fin[$j$]$)$
  return makespan
```

A.3 STNU Method

Algorithm 3 STNU-Based Method

```
Require: FJSP instance $\mathcal{I}$, offline CP solver time limit $T_{offline}$
  Construct deterministic CP model $M$.
  Solve $M$ offline within $T_{offline}$, obtaining partial order schedule $S^*$.
  if $S^*$ is infeasible then return infeasible
  end if
  Initialize STNU network $N = (V, E)$.
  for all operations $j$ do
      Add nodes $s_j, e_j$ to $V$.
      Add contingent link $(s_j, e_j$ with interval $[\ell_j, u_j]$ to $E$.
  end for
  for all constraints $j \prec k$ do
      Add edge $s_j + d_j + setup_{j,k} \leq s_k$ based on $S*$.
  end for
  Check DC of $N$.
  if $N$ is not dynamically controllable then return infeasible
  end if
  Online Execution (RTE*):
  Sample actual durations $\hat{d}_j \in [\ell_j, u_j]$ for each $j$.
  Execute RTE* using sampled durations to determine actual execution times.
  Compute realized makespan: makespan $= \max_j(e_j)$
  return makespan
```

B Instance Exploration

Instance	Jobs	T/Job	$\lvert J\rvert \times \lvert T\rvert$	Machines	μ_p	σ_p^2	μ_s	$\mu_{s/p}$	$H_{\max}$	% Infeasible
1	2	2	4	2	39.25	273.19	4.19	0.16	270	0.00
2	2	2	4	2	46.17	285.47	5.67	0.16	273	43.8
3	3	2	6	2	77.80	944.16	8.90	0.15	777	30.6
4	3	2	6	2	109.80	1009.76	12.06	0.12	1144	27.8
5	3	2	6	2	41.17	210.97	3.97	0.13	437	0.00
6	3	3	9	3	90.47	2768.38	10.30	0.23	1419	68.3
7	3	3	9	5	113.28	2096.65	13.68	0.16	1826	83.7
8	3	3	9	4	68.00	1430.67	7.49	0.17	1136	74.1
9	3	3	9	3	55.11	463.54	6.20	0.16	1053	53.1
10	4	3	12	5	132.35	1947.03	15.65	0.14	2810	88.9
11	5	3	15	6	122.94	1691.27	13.57	0.15	3879	86.3
12	5	3	15	7	128.82	2008.66	14.69	0.16	4428	85.3
13	6	3	18	7	138.50	2725.38	16.29	0.17	6727	84.5
14	7	3	21	7	145.52	2781.07	16.90	0.16	8759	84.9
15	7	3	21	7	143.38	2786.16	16.25	0.16	8635	85.3
16	8	3	24	7	151.10	3169.06	17.03	0.16	11018	85.6
17	8	4	32	7	152.99	2947.88	16.61	0.15	16838	86.9
18	9	4	36	8	150.50	2957.09	16.74	0.16	18974	90.3
19	11	4	44	8	153.74	3127.18	16.46	0.15	25823	90.6
20	12	4	48	8	162.38	4060.45	17.44	0.15	32318	90.5

Table 5: Instance-level statistics. Columns report: number of jobs, tasks per job (T/Job), input size ($|J| \times |T|$), number of machines; processing-time mean and variance (μ_p, σ_p^2); setup-time mean (μ_s); mean of the task-level setup/processing ratio ($\mu_{s/p}$); $H_{\max}$ gives the analytical worst-case horizon $\sum_{j \in T} \max_m d_{j,m} + \sum_{(j,j') \in E} \max_m s_{j,j',m}$; and the percentage of infeasible setup arcs (% Infeasible).

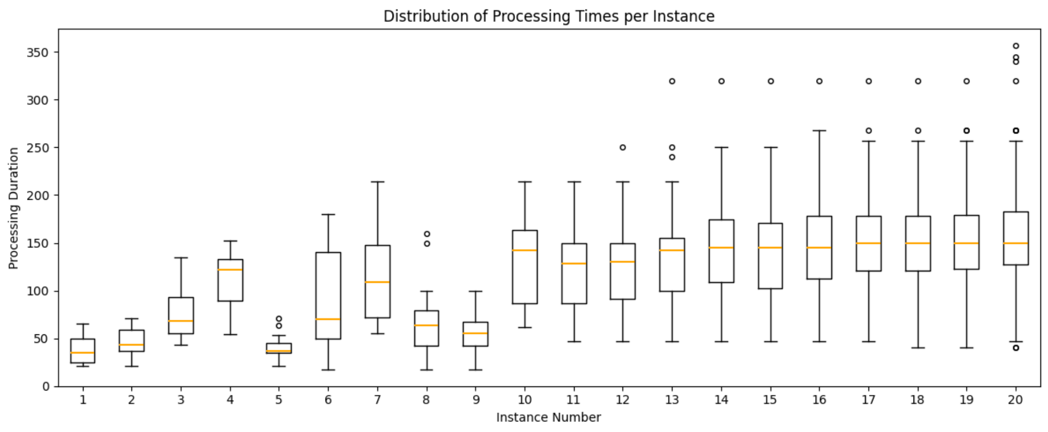


Figure 8: Distribution of Processing Times per Instance

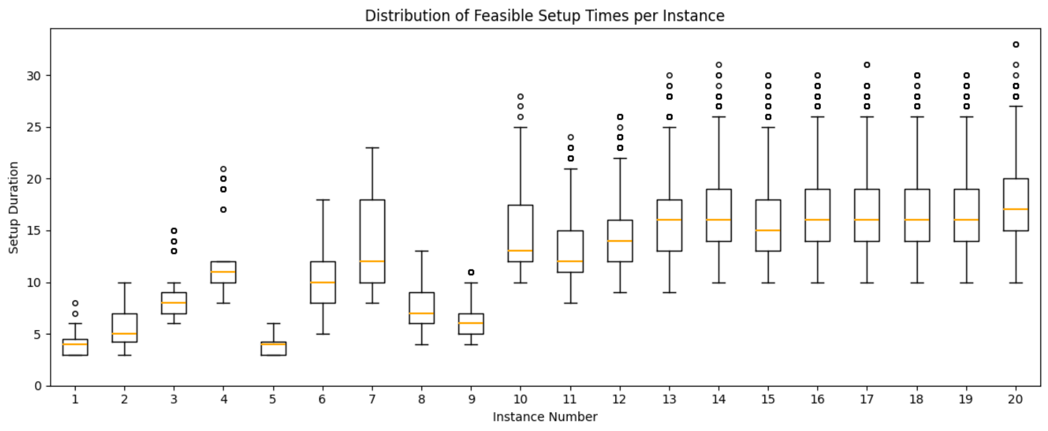


Figure 9: Distribution of Feasible Setup Times per Instance

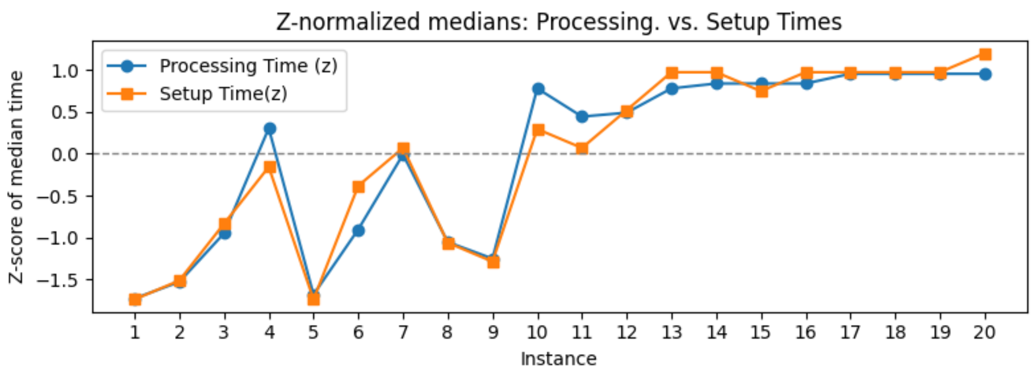


Figure 10: Z-normalized medians: Processing vs. Setup Times

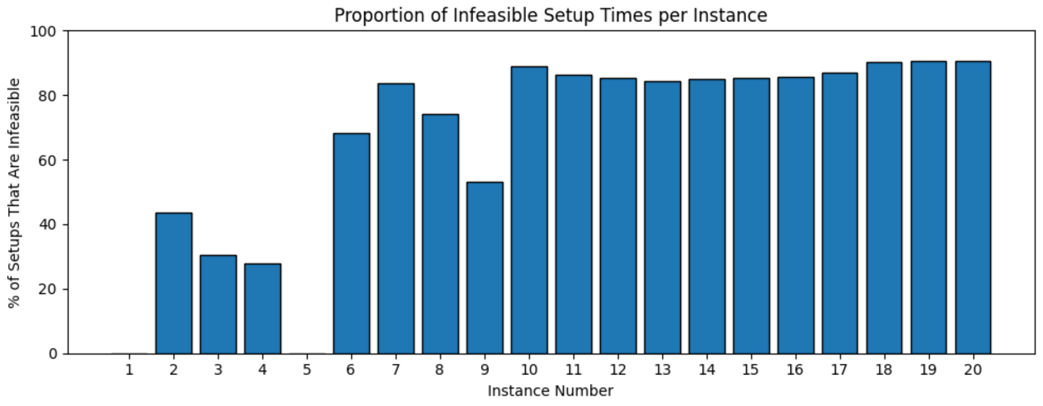


Figure 11: Proportion of Infeasible Setup Times per Instance

C Results

Test	Legend	react–stnu	stnu–proactive	proactive–reactive
Wilc. Quality	[n] z (p)	[180] 3.950 (*)	[180] -0.671 (0.502)	[180] -11.106 (*)
Prop. Quality	[n] prop (p)	[180] 0.650 (*)	[180] 0.528 (0.502)	[180] 0.083 (*)
Magn. Quality	[n] t (p)	[180] -6.580 (*)	[180] 0.296 (0.767)	[180] 15.320 (*)
	norm. avg.	react: 0.951	stnu: 1.002	$\text{pro}_{0.9}$: 1.047
	norm. avg.	stnu: 1.049	$\text{pro}_{0.9}$: 0.998	react: 0.953
Wilc. Offline	[n] z (p)	[180] 1.267 (0.205)	[180] -7.677 (*)	[180] -7.975 (*)
Prop. Offline	[n] prop (p)	[180] 0.550 (0.205)	[180] 0.211 (*)	[180] 0.800 (*)
Magn. Offline	[n] t (p)	[180] -2.061 (*)	[180] 9.482 (*)	[180] -11.127 (*)
	norm. avg.	react: 0.969	stnu: 1.129	$\text{pro}_{0.9}$: 0.900
	norm. avg.	stnu: 1.031	$\text{pro}_{0.9}$: 0.871	react: 1.100
Wilc. Online	[n] z (p)	[180] 13.342 (*)	[180] -13.342 (*)	[180] -13.342 (*)
Prop. Online	[n] prop (p)	[180] 0.000 (*)	[180] 0.000 (*)	[180] 1.000 (*)
Magn. Online	[n] t (p)	[180] 391.481 (*)	[180] 1728.361 (*)	[180] -32230.109 (*)
	norm. avg.	react: 1.918	stnu: 1.968	$\text{pro}_{0.9}$: 0.001
	norm. avg.	stnu: 0.082	$\text{pro}_{0.9}$: 0.032	react: 1.999

Table 6: Pairwise test results for $\epsilon = all$, grouping=1-9.

Test	Legend	react–stnu	stnu–proactive	proactive–reactive
Wilc. Quality	[n] z (p)	[220] 14.765 (*)	[220] -13.956 (*)	[220] -14.765 (*)
Prop. Quality	[n] prop (p)	[220] 1.000 (*)	[220] 0.027 (*)	[220] 0.000 (*)
Magn. Quality	[n] t (p)	[220] -33.606 (*)	[220] 24.138 (*)	[220] 29.342 (*)
	norm. avg.	react: 0.851	stnu: 1.103	$\text{pro}_{0.9}$: 1.047
	norm. avg.	stnu: 1.149	$\text{pro}_{0.9}$: 0.897	react: 0.953
Wilc. Offline	[n] z (p)	[220] 9.911 (*)	[220] -0.337 (0.736)	[220] -11.529 (*)
Prop. Offline	[n] prop (p)	[220] 0.164 (*)	[220] 0.514 (0.736)	[220] 0.891 (*)
Magn. Offline	[n] t (p)	[220] 11.771 (*)	[220] 0.178 (0.859)	[220] -13.012 (*)
	norm. avg.	react: 1.155	stnu: 1.002	$\text{pro}_{0.9}$: 0.845
	norm. avg.	stnu: 0.845	$\text{pro}_{0.9}$: 0.998	react: 1.155
Wilc. Online	[n] z (p)	[220] 14.765 (*)	[220] -14.630 (*)	[220] -14.765 (*)
Prop. Online	[n] prop (p)	[220] 0.000 (*)	[220] 0.005 (*)	[220] 1.000 (*)
Magn. Online	[n] t (p)	[220] 924.066 (*)	[220] 159.748 (*)	[220] -15392.850 (*)
	norm. avg.	react: 1.985	stnu: 1.920	$\text{pro}_{0.9}$: 0.001
	norm. avg.	stnu: 0.015	$\text{pro}_{0.9}$: 0.080	react: 1.999

Table 7: Pairwise test results for $\epsilon = all$, grouping=10-20.

C.0.1 $\gamma = 1$

Test	Legend	react–stnu	stnu–proactive	proactive–reactive
Wilc. Quality	[n] z (p)	[200] 9.546 (*)	[200] -8.273 (*)	[200] -13.081 (*)
Prop. Quality	[n] prop (p)	[200] 0.840 (*)	[200] 0.205 (*)	[200] 0.035 (*)
Magn. Quality	[n] t (p)	[200] -15.194 (*)	[200] 10.555 (*)	[200] 24.653 (*)
	norm. avg.	react: 0.898	stnu: 1.071	$\text{pro}_{0.9}$: 1.032
	norm. avg.	stnu: 1.102	$\text{pro}_{0.9}$: 0.929	react: 0.968
Wilc. Offline	[n] z (p)	[200] 1.061 (0.289)	[200] -2.899 (*)	[200] -7.283 (*)
Prop. Offline	[n] prop (p)	[200] 0.460 (0.289)	[200] 0.395 (*)	[200] 0.760 (*)
Magn. Offline	[n] t (p)	[200] -2.068 (*)	[200] 5.908 (*)	[200] -8.358 (*)
	norm. avg.	react: 0.975	stnu: 1.076	$\text{pro}_{0.9}$: 0.948
	norm. avg.	stnu: 1.025	$\text{pro}_{0.9}$: 0.924	react: 1.052
Wilc. Online	[n] z (p)	[200] 14.071 (*)	[200] -13.930 (*)	[200] -14.071 (*)
Prop. Online	[n] prop (p)	[200] 0.000 (*)	[200] 0.005 (*)	[200] 1.000 (*)
Magn. Online	[n] t (p)	[200] 310.254 (*)	[200] 142.944 (*)	[200] -13096.748 (*)
	norm. avg.	react: 1.950	stnu: 1.936	$\text{pro}_{0.9}$: 0.001
	norm. avg.	stnu: 0.050	$\text{pro}_{0.9}$: 0.064	react: 1.999

Table 8: Pairwise test results for $\epsilon = 1$, grouping=1-20.

Test	Legend	react–stnu	stnu–proactive	proactive–reactive
Wilc. Quality	[n] z (p)	[90] 2.635 (*)	[90] 0.738 (0.461)	[90] 7.906 (*)
Prop. Quality	[n] prop (p)	[90] 0.644 (*)	[90] 0.544 (0.461)	[90] 0.922 (*)
Magn. Quality	[n] t (p)	[90] -4.718 (*)	[90] -1.462 (0.147)	[90] -13.544 (*)
	norm. avg.	react: 0.951	stnu: 0.985	$\text{pro}_{0.9}$: 0.966
	norm. avg.	stnu: 1.049	$\text{pro}_{0.9}$: 1.015	react: 1.034
Wilc. Offline	[n] z (p)	[90] 3.479 (*)	[90] 4.111 (*)	[90] 3.689 (*)
Prop. Offline	[n] prop (p)	[90] 0.689 (*)	[90] 0.722 (*)	[90] 0.300 (*)
Magn. Offline	[n] t (p)	[90] -4.953 (*)	[90] -5.669 (*)	[90] 5.068 (*)
	norm. avg.	react: 0.901	stnu: 0.881	$\text{pro}_{0.9}$: 1.022
	norm. avg.	stnu: 1.099	$\text{pro}_{0.9}$: 1.119	react: 0.978
Wilc. Online	[n] z (p)	[90] 9.381 (*)	[90] 9.381 (*)	[90] 9.381 (*)
Prop. Online	[n] prop (p)	[90] 0.000 (*)	[90] 1.000 (*)	[90] 0.000 (*)
Magn. Online	[n] t (p)	[90] 293.272 (*)	[90] -1290.840 (*)	[90] 25285.300 (*)
	norm. avg.	react: 1.910	stnu: 0.033	$\text{pro}_{0.9}$: 1.998
	norm. avg.	stnu: 0.090	$\text{pro}_{0.9}$: 1.967	react: 0.002

Table 9: Pairwise test results for $\epsilon = 1$, grouping=1-9.

Test	Legend	react–stnu	stnu–proactive	proactive–reactive
Wilc. Quality	[n] z (p)	[110] 10.393 (*)	[110] -10.393 (*)	[110] -10.393 (*)
Prop. Quality	[n] prop (p)	[110] 1.000 (*)	[110] 0.000 (*)	[110] 0.000 (*)
Magn. Quality	[n] t (p)	[110] -23.120 (*)	[110] 19.524 (*)	[110] 26.692 (*)
	norm. avg.	react: 0.854	stnu: 1.116	$\text{pro}_{0.9}$: 1.030
	norm. avg.	stnu: 1.146	$\text{pro}_{0.9}$: 0.884	react: 0.970
Wilc. Offline	[n] z (p)	[110] 4.672 (*)	[110] -0.095 (0.924)	[110] -6.388 (*)
Prop. Offline	[n] prop (p)	[110] 0.273 (*)	[110] 0.491 (0.924)	[110] 0.809 (*)
Magn. Offline	[n] t (p)	[110] 2.857 (*)	[110] 2.695 (*)	[110] -7.520 (*)
	norm. avg.	react: 1.035	stnu: 1.041	$\text{pro}_{0.9}$: 0.923
	norm. avg.	stnu: 0.965	$\text{pro}_{0.9}$: 0.959	react: 1.077
Wilc. Online	[n] z (p)	[110] 10.393 (*)	[110] -10.202 (*)	[110] -10.393 (*)
Prop. Online	[n] prop (p)	[110] 0.000 (*)	[110] 0.009 (*)	[110] 1.000 (*)
Magn. Online	[n] t (p)	[110] 611.668 (*)	[110] 80.303 (*)	[110] -7999.137 (*)
	norm. avg.	react: 1.983	stnu: 1.910	$\text{pro}_{0.9}$: 0.001
	norm. avg.	stnu: 0.017	$\text{pro}_{0.9}$: 0.090	react: 1.999

Table 10: Pairwise test results for $\epsilon = 1$, grouping=10-20.

C.0.2 $\gamma = 2$

Test	Legend	react–stnu	stnu–proactive	proactive–reactive
Wilc. Quality	[n] z (p)	[200] 9.687 (*)	[200] -5.586 (*)	[200] -12.940 (*)
Prop. Quality	[n] prop (p)	[200] 0.845 (*)	[200] 0.300 (*)	[200] 0.040 (*)
Magn. Quality	[n] t (p)	[200] -15.179 (*)	[200] 6.346 (*)	[200] 23.851 (*)
	norm. avg.	react: 0.894	stnu: 1.044	$\text{pro}_{0.9}$: 1.062
	norm. avg.	stnu: 1.106	$\text{pro}_{0.9}$: 0.956	react: 0.938
Wilc. Offline	[n] z (p)	[200] 7.990 (*)	[200] -3.889 (*)	[200] -12.374 (*)
Prop. Offline	[n] prop (p)	[200] 0.215 (*)	[200] 0.360 (*)	[200] 0.940 (*)
Magn. Offline	[n] t (p)	[200] 10.857 (*)	[200] 3.512 (*)	[200] -17.337 (*)
	norm. avg.	react: 1.168	stnu: 1.042	$\text{pro}_{0.9}$: 0.791
	norm. avg.	stnu: 0.832	$\text{pro}_{0.9}$: 0.958	react: 1.209
Wilc. Online	[n] z (p)	[200] 14.071 (*)	[200] -14.071 (*)	[200] -14.071 (*)
Prop. Online	[n] prop (p)	[200] 0.000 (*)	[200] 0.000 (*)	[200] 1.000 (*)
Magn. Online	[n] t (p)	[200] 352.870 (*)	[200] 536.853 (*)	[200] -28553.743 (*)
	norm. avg.	react: 1.959	stnu: 1.947	$\text{pro}_{0.9}$: 0.001
	norm. avg.	stnu: 0.041	$\text{pro}_{0.9}$: 0.053	react: 1.999

Table 11: Pairwise test results for $\epsilon = 2$, grouping=1-20.

Test	Legend	react–stnu	stnu–proactive	proactive–reactive
Wilc. Quality	[n] z (p)	[90] 2.846 (*)	[90] -1.792 (0.073)	[90] -7.695 (*)
Prop. Quality	[n] prop (p)	[90] 0.656 (*)	[90] 0.600 (0.073)	[90] 0.089 (*)
Magn. Quality	[n] t (p)	[90] -4.570 (*)	[90] -0.928 (0.356)	[90] 11.493 (*)
	norm. avg.	react: 0.950	stnu: 0.989	$\text{pro}_{0.9}$: 1.061
	norm. avg.	stnu: 1.050	$\text{pro}_{0.9}$: 1.011	react: 0.939
Wilc. Offline	[n] z (p)	[90] 1.581 (0.114)	[90] -6.641 (*)	[90] -7.484 (*)
Prop. Offline	[n] prop (p)	[90] 0.411 (0.114)	[90] 0.144 (*)	[90] 0.900 (*)
Magn. Offline	[n] t (p)	[90] 1.804 (0.075)	[90] 8.003 (*)	[90] -13.672 (*)
	norm. avg.	react: 1.037	stnu: 1.139	$\text{pro}_{0.9}$: 0.822
	norm. avg.	stnu: 0.963	$\text{pro}_{0.9}$: 0.861	react: 1.178
Wilc. Online	[n] z (p)	[90] 9.381 (*)	[90] -9.381 (*)	[90] -9.381 (*)
Prop. Online	[n] prop (p)	[90] 0.000 (*)	[90] 0.000 (*)	[90] 1.000 (*)
Magn. Online	[n] t (p)	[90] 278.902 (*)	[90] 1172.436 (*)	[90] -25636.864 (*)
	norm. avg.	react: 1.926	stnu: 1.969	$\text{pro}_{0.9}$: 0.001
	norm. avg.	stnu: 0.074	$\text{pro}_{0.9}$: 0.031	react: 1.999

Table 12: Pairwise test results for $\epsilon = 2$, grouping=1-9.

Test	Legend	react–stnu	stnu–proactive	proactive–reactive
Wilc. Quality	[n] z (p)	[110] 10.393 (*)	[110] -9.249 (*)	[110] -10.393 (*)
Prop. Quality	[n] prop (p)	[110] 1.000 (*)	[110] 0.055 (*)	[110] 0.000 (*)
Magn. Quality	[n] t (p)	[110] -24.350 (*)	[110] 15.319 (*)	[110] 32.497 (*)
	norm. avg.	react: 0.849	stnu: 1.089	$\text{pro}_{0.9}$: 1.063
	norm. avg.	stnu: 1.151	$\text{pro}_{0.9}$: 0.911	react: 0.937
Wilc. Offline	[n] z (p)	[110] 9.249 (*)	[110] -0.667 (0.505)	[110] -9.821 (*)
Prop. Offline	[n] prop (p)	[110] 0.055 (*)	[110] 0.536 (0.505)	[110] 0.973 (*)
Magn. Offline	[n] t (p)	[110] 16.370 (*)	[110] -3.101 (*)	[110] -12.401 (*)
	norm. avg.	react: 1.276	stnu: 0.963	$\text{pro}_{0.9}$: 0.766
	norm. avg.	stnu: 0.724	$\text{pro}_{0.9}$: 1.037	react: 1.234
Wilc. Online	[n] z (p)	[110] 10.393 (*)	[110] -10.393 (*)	[110] -10.393 (*)
Prop. Online	[n] prop (p)	[110] 0.000 (*)	[110] 0.000 (*)	[110] 1.000 (*)
Magn. Online	[n] t (p)	[110] 712.660 (*)	[110] 529.075 (*)	[110] -32012.444 (*)
	norm. avg.	react: 1.987	stnu: 1.929	$\text{pro}_{0.9}$: 0.000
	norm. avg.	stnu: 0.013	$\text{pro}_{0.9}$: 0.071	react: 2.000

Table 13: Pairwise test results for $\epsilon = 2$, grouping=10-20.

C.0.3 $\gamma = 1$ vs $\gamma = 2$

Test	Metric	reactive (n1 vs n2)	proactive (n1 vs n2)	stnu (n1 vs n2)
Wilc. Quality	[n] z (p)	[200] 0.636 (0.525)	[200] 12.374 (*)	[200] 1.909 (0.056)
Prop. Quality	[n] prop (p)	[200] 0.475 (0.525)	[200] 0.940 (*)	[200] 0.570 (0.056)
Magn. Quality	[n] t (p)	[200] -0.327 (0.744)	[200] -22.108 (*)	[200] -2.017 (*)
	norm. avg.	n1: 0.999	n1: 0.969	n1: 0.995
	norm. avg.	n2: 1.001	n2: 1.031	n2: 1.005
Wilc. Offline	[n] z (p)	[200] 11.243 (*)	[200] -0.071 (0.944)	[200] 0.212 (0.832)
Prop. Offline	[n] prop (p)	[200] 0.900 (*)	[200] 0.500 (0.944)	[200] 0.490 (0.832)
Magn. Offline	[n] t (p)	[200] -17.123 (*)	[200] -2.938 (*)	[200] 0.468 (0.640)
	norm. avg.	n1: 0.805	n1: 0.968	n1: 1.002
	norm. avg.	n2: 1.195	n2: 1.032	n2: 0.998
Wilc. Online	[n] z (p)	[200] 10.536 (*)	[200] 4.738 (*)	[200] 0.495 (0.621)
Prop. Online	[n] prop (p)	[200] 0.875 (*)	[200] 0.330 (*)	[200] 0.480 (0.621)
Magn. Online	[n] t (p)	[200] -18.441 (*)	[200] 4.998 (*)	[200] 0.161 (0.872)
	norm. avg.	n1: 0.888	n1: 1.045	n1: 1.001
	norm. avg.	n2: 1.112	n2: 0.955	n2: 0.999

Table 14: Noise $\gamma = 1$ vs $\gamma = 2$: pairwise tests per method

Table 15: Hyperparameter settings

Hyperparameter	**Proactive**	**STNU**	**Reactive**
γ	robust	1	0.9
Time limit CP	5000s	5000s	5000s and 5s
Solver	IBM CP	IBM CP	IBM CP

References

[1] Imran Ali Chaudhry and Abdul Asad Khan. A research survey: review of flexible job shop scheduling techniques. *International Transactions in Operational Research*, 23(3):551–591, 2016.

[2] Committee Reassessment Integrity Policy. TU Delft Vision on Integrity 2018–2024. Technical report, Delft University of Technology, Delft, The Netherlands, September 2018. Includes the TU Delft Code of Conduct and Integrity Statement.

[3] Stéphane Dauzère-Pérès, Junwen Ding, Liji Shen, and Karim Tamssaouet. The flexible job shop scheduling problem: A review. *European Journal of Operational Research*, 314(2):409–432, 2024.

[4] PyJobShop Developers. Pyjobshop: Job shop scheduling with python – quick examples, 2024. Accessed: 2025-06-02.

[5] Imanol Echeverria, Maialen Murua, and Roberto Santana. Leveraging constraint programming in a deep learning approach for dynamically solving the flexible job-shop scheduling problem, 2024.

[6] Mario Flores-Gómez, Valeria Borodin, and Stéphane Dauzère-Pérès. Maximizing the service level on the makespan in the stochastic flexible job-shop scheduling problem. *Computers Operations Research*, 157:106237, 2023.

[7] Na Fu, Hoong Lau, Pradeep Varakantham, and Fei Xiao. Robust local search for solving rcpsp/max with durational uncertainty. *Journal of Artificial Intelligence Research (JAIR)*, 43:43–86, 05 2012.

[8] Na Fu, Pradeep Varakantham, and Hoong Lau. Robust partial order schedules for rcpsp/max with durational uncertainty. *Proceedings of the International Conference on Automated Planning and Scheduling*, 26:124–130, 03 2016.

[9] M.R. Garey, D.S. Johnson, and Ravi Sethi. Complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976. Cited by: 2392.

[10] Luke Hunsberger and Roberto Posenato. Foundations of dispatchability for simple temporal networks with uncertainty. pages 253–263, 01 2024.

[11] Philippe Laborie, Jérôme Rogerie, Paul Shaw, and Petr Vilím. Ibm ilog cp optimizer for scheduling. *Constraints*, 23(2):210–250, 2018.

[12] Manuel Laguna. A heuristic for production scheduling and inventory control in the presence of sequence-dependent setup times. *IIE Transactions*, 31(2):125–134, 1999.

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

[14] Ernst Limpert, Werner A. Stahel, and Markus Abbt. Log-normal distributions across the sciences: keys and clues. *BioScience*, 51(5):341–352, 2001.

[15] Paul Morris. 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*, volume 11, pages 464–479. Springer, 2014.

[16] Bahman Naderi, Rubén Ruiz, and Vahid Roshanaei. Mixed-integer programming vs. constraint programming for shop scheduling problems: New results and outlook. *INFORMS Journal on Computing*, 35(4):817–843, 2023.

[17] Nicola Policella, Stephen F Smith, Amedeo Cesta, and Angelo Oddi. Generating robust schedules through temporal flexibility. In *ICAPS*, volume 4, pages 209–218, 2004.

[18] Robbert Reijnen, Kjell van Straaten, Zaharah Bukhsh, and Yingqian Zhang. Job shop scheduling benchmark: Environments and instances for learning and non-learning methods. *arXiv preprint arXiv:2308.12794*, 2023.

[19] Mohammad Saidi-Mehrabad and Parviz Fattahi. Flexible job shop scheduling with tabu search algorithms. *The International Journal of Advanced Manufacturing Technology*, 32:563–570, 03 2007.

[20] 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. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2025. Preprint.