Algorithms for dynamic scheduling in manufacturing, towards digital factories

Flexible Job Shop Scheduling Problems (FJSPs) with generalized time-lags and no-wait constraints

Bogdan Luca Paramon

Supervisor(s): Mathijs de Weerdt, Leon Planken, Kim van den Houten

[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: Bogdan Luca Paramon
Final project course: CSE3000 Research Project
Thesis committee: Mathijs de Weerdt, Leon Planken, Kim van den Houten, Jasmijn Baaijens

Abstract

This study investigates scheduling strategies for the stochastic duration flexible job-shop problem with no-wait and general time lags constraints (FJSP/NW-GTL). Progress in Constraint Programming (CP) and temporal-networks has renewed interest in assessing the strengths and limitations of different proactive and reactive scheduling approaches. This paper covers the application of a CP-based fully proactive method, a reactive method and STNU-based method on the FJSP/NW-GTL problem comparing results in terms of predetermined objectives and feasibility. In addition, the paper aims to answer how different distributions for task duration affect feasibility and performance. Our results show that strictly proactive methods are infeasible for no-wait constraints and very tight schedules, which lead to adding an online step in the proactive implementation to pursue the comparison between the approaches. With this change, plotting the average makespan across methods by distribution shows that there is not much fluctuation between distribution types in terms of makespan. Moreover, it appears that the proactive method performs the best, followed closely by the reactive method, while the STNU approach results in a notably higher makespan for the same instances. Notably, in terms of feasibility, the proactive and reactive approach have 100% rate of success compared to the STNU approach which is infeasible on 35% of the instances in the dataset.

1 Introduction

The rise of smart manufacturing systems is transforming industries through the integration of Industry 4.0 technologies [14], where digitalization plays a pivotal role. At the core of this transformation are advanced scheduling algorithms that manage the complexities of real-time production environments. Traditional job shop scheduling techniques, which rely on deterministic task durations and fixed sequences, are increasingly inadequate in the face of real-world challenges, where processing times are uncertain and delays are common.

Disturbances such as variable processing times and machine breakdowns render purely deterministic schedules brittle. This makes it necessary to use more advanced methods for reasoning about time. Temporal Constraint Networks (TCNs), first formalised by Dechter, Meiri and Pearl, offer a practical framework for managing and propagating constraints between time points [4]. Building on this, Simple Temporal Networks with Uncertainty (STNUs) extend TCNs with contingent (uncontrollable) durations and offer dynamic-controllability guarantees [11]. STNUs represent temporal constraints between tasks, where some task durations are uncertain, and allow for flexible scheduling that adapts as uncertainty is revealed.

Our research builds upon recent work by Van den Houten et al. (2024) [15], who developed proactive and reactive scheduling techniques using constraint programming applied to stochastic project scheduling problems with maximal time-lags (RCPSP/max). Their framework, based on temporal networks, is versatile and can in principle be applied to a variety of scheduling problems beyond RCPSP/max.

We focus on the Flexible Job Shop Problem (FJSP) [2], a variant of the Job Shop Problem, where each task can be assigned to multiple machines rather than a single one. The Flexible Job-Shop Scheduling Problem (FJSP) is provably NP-hard [7; 3], and recent surveys still underline its computational intractability and practical relevance [1]. FJSPs are critical in manufacturing environments due to the inherent flexibility they offer. However, when combined with generalized time-lags (which impose minimum and maximum delays between tasks) and no-wait constraints (which require immediate continuation of certain tasks), FJSPs become even more complex and remain largely unexplored in the context of constraint programming and STNUs. Generalized time-lags are essential in modeling real-world dependencies between operations, such as cooling or curing periods that must elapse before subsequent processing can begin, or deadlines that impose strict upper bounds on task sequencing. No-wait constraints are equally critical in continuous manufacturing processes like metalworking or chemical production, where any delay between consecutive operations (e.g., cutting and welding) can compromise product quality or cause defects. Those constraints closely simulate real-life scenarios, which further emphasizes the importance of their investigation.

While some research has addressed FJSPs with constraints such as time-lags and no-wait in deterministic settings, these models often fail to account for uncertainty in task durations. For instance, Aissani et al. [1] extend the FJSP with generalised time-lags using mixed-integer programming, but do not consider uncertainty or dynamic adaptation. Likewise, Ebrahimnejad and Khalilzadeh [5] study a no-wait FJSP with periodic maintenance via meta-heuristics, yet their approach lacks proactive and reactive scheduling capabilities.

This research aims to bridge this gap by investigating the use of proactive, reactive and hybrid scheduling strategies based on STNUs for FJSPs with generalized time-lags and no-wait constraints under uncertain task durations. By extending the PyJobShop package [10], this work will explore how these strategies influence the feasibility, robustness, and performance of scheduling systems in uncertain environments, providing insights into more adaptable scheduling methods for smart manufacturing.

2 Methodology and Background

This section presents the methodology employed in this research, outlines necessary background concepts, and formally describes the problem studied.

2.1 Problem Description

My research addresses the challenges of scheduling flexible job-shop problems (FJSPs) under uncertainty, incorporating generalized time-lags and no-wait constraints in task durations. The FJSP extends the classical job-shop model by allowing each operation to be carried out on any machine within a specified subset of eligible machines. In its most general form, the FJSP [3] can therefore be expressed as follows:

1. There is a set of n jobs to be processed on m machines.
2. The set of m machines is denoted by $M = \{ M_1, M_2, \ldots, M_m \}$.
3. Each job i consists of a sequence of n_i operations
$$(O_{i,1},\ O_{i,2},\ \ldots,\ O_{i,n_i}).$$
4. Every operation must be executed to complete its job. Operation $O_{i,j}$ requires exactly one machine chosen from the candidate set $M_{i,j} \subseteq M$. When $O_{i,j}$ is processed on machine $M_k \in M_{i,j}$, its processing time is $p_{i,j,k}$.
   Typically, the following assumptions are made for a general FJSP:
   (a) All machines are available at time $t = 0$.
   (b) All jobs are available at time $t = 0$.
   (c) Each operation can be processed by only one machine at a time.
   (d) There are no precedence constraints among operations of different jobs; therefore jobs are independent of each other.
   (e) No pre-emption of operations is allowed, i.e. an operation, once started, cannot be interrupted.
   (f) Transportation time between machines and setup time for an operation are included in its processing time.

On top of this, generalized time-lags and no-wait constraints are enforced using already available methods found in the API Documentation of PyJobShop. For any ordered pair of operations $(\text{task}_1, \text{task}_2)$ part of the same job we impose
$$L \le \text{st}_2 - \text{end}_1 \le U,$$
where

- end_1 is the finish time of task_1,
- st_2 is the start time of task_2,
- L is the *minimum* allowable delay (which may be positive, zero, or even negative to allow overlap),
- U is the *maximum* allowable delay (and can be $+\infty$ to model only a lower bound).
- Setting $L = U = 0$ yields the familiar no-wait constraint $\text{st}_2 = \text{end}_1$; larger intervals simply relax this equality while keeping both bounds explicit.

The $objective$ of the experimentation is to observe hybrid scheduling strategies that efficiently manage these constraints to $optimize\ makespan$ while maintaining dynamic controllability, enabling real-time adaptability to observed duration variations.

2.2 Scheduling Methods

Following the framework of Van den Houten *et al.* [15], three algorithmic paradigms are benchmarked for the FJSP with generalised time-lags and no-wait links under stochastic task durations:

Proactive Schedule
The proactive approach aims to build a schedule offline and strictly adhere to it during the online phase. The offline part uses a CP solver to return a fixed start time for every operation, and no changes are allowed during the online phase. The online phase consists of sampling the real durations based on different strategies and during execution we check if constraints are met. We test several ways of obtaining the deterministic durations (robust and percentile sampling with different quantiles).

1. **Robust** - every activity is fixed to its maximum support value, yielding a highly feasible but typically long schedule.
2. **Quantile** - we test different quantiles 0.25, 0.5, 0.75, 0.9, with the expectation of higher percentage quantiles balancing higher feasibility and shorter schedules in terms of makespan and lower quantiles affecting feasibility considerably aiming for even smaller makespans

Reactive Schedule
The reactive method also starts with a deterministic scenario, following the same logic as described in the proactive approach. The differentiating factor of the reactive approach lays in progressively altering the schedule as time passes and tasks complete execution. It *re-optimises* the schedule whenever a job finishes later or earlier than predicted. At each event the remaining sub-instance is rebuilt with updated release dates and the CP solver is invoked again. This policy incurs extra solver calls yet greatly improves robustness in practice [15].

STNU-Based Hybrid Schedule
The third approach encodes the provisional plan as a *Simple Temporal Network with Uncertainty* (STNU) and extracts a *partial-order schedule* (POS). Dynamic-controllability analysis of the STNU guarantees that the schedule can actually adapt tasks and their real durations. Actual start times are decided in real time as soon as durations are revealed, while all temporal constraints (including time-lags and no-wait links) remain satisfied [11]. The STNU approach uses RTE* [9], an algorithm that is used to dynamically schedule tasks in presesnce of temporal constraints. RTE* extends standard RTE by allowing more flexibility in execution while still ensuring dynamic controllability, meaning that the schedule can adapt to outcomes in real time without violating temporal constraints. Empirical evidence in the SRCPSP/max domain shows that this hybrid method often matches the robustness of reactive rescheduling with far fewer online optimisation steps [15].

2.3 Methodology

The experimentation pipeline consists of a uniform modeling layer, a machine-selection policy, and a script used to collect evaluation metrics under different duration distributions.

Modelling the FJSP instance
All benchmark instances are encoded with PyJobShop [10] framework allowing for an effortless implementation of the temporal constraints. Extending on previous work, the

PyJobShopSTNUs[1]extension is used to facilitate the hybrid approach (STNU). The RTE* algorithm, previously mentioned in the hybrid approach, is used in the online phase after an STNU is built from a FJSP instance. If the temporal network is dynamically controllable, RTE* is used to execute the schedule adaptively, handling uncertain durations while preserving feasibility and minimizing rescheduling. Besides the standard precedence links, two additional temporal constructs are imposed:

1. **No-wait links.** Consecutive operations that must start back-to-back are modelled through a generalised time-lag with bounds $L = U = 0$, thus enforcing $\mathrm{st}_{k+1} = \mathrm{end}_k$.
2. **Generalised time-lags.** Whenever a lag between two operations (i, j) is required, the interval constraint $L \leq \mathrm{st}_j - \mathrm{end}_i \leq U$ is added directly to the STNU, where L and U are scenario-specific parameters.

To test robustness across different shop environments, each stochastic processing time d is replaced by a random duration drawn from an uniform distribution, with different noise levels as parameter: $\alpha > 0$:

$$\text{Uniform:} \quad \mathrm{Unif}\big(\max(1,\, d - \alpha\sqrt{d}),\, d + \alpha\sqrt{d}\big),$$

The resulting temporal network is fed to the RTE* dispatcher of Van den Houten *et al.* [15], which checks dynamic controllability and, if successful, returns a partial-order schedule ready for simulation.

Machine selection

Choosing a mode for every operation can be performed either *offline*, before execution starts, or *online*, together with the start-time decisions. Including modes in the online search, however, increases the branching factor of both the reactive and the STNU dispatcher. Hence we adopt a fully *offline* policy:

1. Each deterministic projection of the stochastic instance (robust or different quantiles) is solved once with CP.
2. The mode chosen by the solver for that projection is *frozen* and reused during all subsequent online decisions, removing the mode variable from the real-time search space.

This strategy keeps the comparison fair across proactive, reactive and hybrid algorithms while maintaining tractable online running times.

Performance metrics

Four performance indicators have been used in the evaluation and analysis of the different approaches:

Feasibility ratio percentage of simulated scenarios that satisfy all temporal (no-wait and general time lags) and resource constraints.

Offline time CPU time required to generate the initial schedule (including mode selection).

Online time cumulative CPU time spent in real-time decisions (rescheduling or STNU dispatching).

Makespan completion time of the last operation; objective is to minimize makespan.

3 Experimental Setup

All tests are carried out on the **Fattahi** benchmark family for the Flexible Job-Shop Scheduling Problem (FJSP). The 20 instances were originally introduced by Fattahi, Mehrabad and Jolai [6] and are now publicly available in the SchedulingLab repository (https://github.com/SchedulingLab/fjsp-instances)[13].[2] These instances already contain the routing alternatives required for FJSP. Instead of enriching the data files themselves, we inject the additional *no-wait* links and *generalised time-lags programmatically* during modeling. The general time lags follow the following strategy to become deterministic, thus reproducible:

1. First, we set the hyperparameters that define how wide the generalized time windows between tasks can be. (e.g. lower fraction = 0.25, width fraction = 0.50)
2. Second, we convert the filename into a deterministic random seed using zlib.crc32. This ensures that the same input file always results in the same time-lags LB and UB making experiments reproducible.
3. Third, we compute the average (mean) duration for each two consecutive operations. The two means are averaged and buffered in a variable that will later scale LB and UB meaningfuly.
4. Last step is to compute the LB and UB. The lower bound is sampled using the seed from step two and the lower fraction. An intermediate value follows the same formula as LB, but using the width fraction instead. The upper bound is calculated by adding the intermediate value to the LB. With the hyperparameters set to the values metnioned in the first step, this would result in:

$$0 \leq LB \leq 0.25 \cdot \text{scale}, \quad LB \leq UB \leq 0.75 \cdot \text{scale}$$

Keeping the raw benchmark intact preserves comparability with earlier deterministic FJSP studies while satisfying best-practice recommendations to separate immutable input data from derived artefacts and to document every processing step transparently [12; 8].

Uncertainty settings. For every nominal processing time d we sample uncertain durations with two noise levels, $\alpha \in \{1.0,\, 2.0\}$ from an unifor distribution as described in the previous section. The filename of the instance is used as the RNG seed so that every method sees exactly the same realisation stream.

[1]Repository available at https://github.com/kimvandenhouten/PyJobShopSTNUs/tree/main.

[2]The repository keeps the exact plain-text files from the authors and is version-controlled, which simplifies future replication.

Deterministic reductions. All deterministic projections (robust, and different percentiles) are modeled in PyJobShop and solved with IBM CP Optimizer (via the library's `cpoptimizer` backend) to obtain both a baseline schedule and a concrete mode assignment.

Evaluated algorithms. The three algorithms of Section 2.2- proactive, reactive, and the STNU-based hybrid- are run on each instance / noise combination. Different amounts of independent duration samples have been used for the experiments. For general comparison of average makespan for different distribution modes 10 samples were used per instance, giving a total of Instances(20) $\times$ Noise(2) $\times$ Samples(10) simulation runs for every method. For feasibility experiments, 50 samples were used per instance, which leads to a total of Instances(20) $\times$ Noise(2) $\times$ Samples(50)

Hardware platform. All experiments were conducted on a MacBook Pro (Model Identifier: MacBookPro17,1) equipped with an Apple M1 chip featuring 8 cores (4 performance and 4 efficiency) and 16GB of unified memory, running macOS Sequoia version 15.3.1. The project was developed and executed using Python 3.9 within a PyCharm IDE environment. All solvers and scheduling pipelines ran natively on the Apple Silicon architecture without virtualization

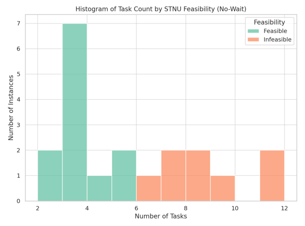
Reproducibility. All tests are carried out on the **Fattahi** Flexible Job-Shop benchmark suite [6], which we leave *unchanged on disk*. Experiments were executed within a controlled Python virtual environment managed in PyCharm. Solver versions (e.g., CP Optimizer, PyJobShop) and dependencies were specified in a requirements.txt file. The results were logged with timestamped directories and labeled by method type and noise configuration, with per-instance feasibility, start times, makespan, and online/offline computation times recorded. Additionally, the parameters used are transparently mentioned in the paper such that they can be reused to reproduce the experiments in future work.

4 Results and Discussion

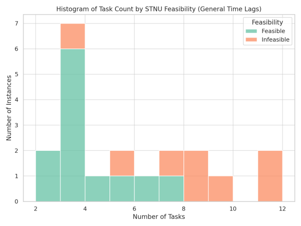
In this section we present and interpret the most significant outcomes of our experiments, focusing on key performance metrics across all scheduling approaches. We evaluate each metric separately to gain a clearer understanding of the performance trade-offs between approaches.

4.1 Feasibility Overview

In contrast to prior work where feasibility ratios vary, due to our implementation particularities, our study shows that feasibility is always achieved for the Proactive and Reactive approaches. However, a few infeasible instances were observed under the STNU-based method, particularly for instance with higher number of tasks. The correlation can be observed both in the plots obtained from the no-wait and general time lags constraints from (Figures 1a and 1b). A table showing the data used in the plots can be found in Appendix A.



(a) Feasibility vs Task Count (NW)



(b) Feasibility vs Task Count (GTL with LB 0.25% and UB 0.75%)

Figure 1: Histogram of task counts by STNU feasibility under different temporal constraints.

Feasibility in the Flexible Job Shop Scheduling Problem (FJSP) with *no-wait (NW)* and *generalized time-lag (GTL)* constraints can fail under two main scenarios:

1. When time-lag bounds between operations are violated due to real-time delays or sampling noise.
2. When no-wait constraints, which enforce strict chaining between tasks, cannot be satisfied on shared resources due to timing conflicts.

Each approach handles these constraints differently:

Proactive approach. Schedules are precomputed using a *constraint programming (CP) solver* with a deterministic duration vector, where both GTL and NW constraints are modeled as hard constraints for a feasible baseline. The solver is only allowed to return feasible solutions, thus the offline schedules guarantees feasibility. In the online phase, theoretically, there should be no rescheduling. The new finish times should be calculated using the new durations derived for each sample and then tested for feasibility. However, one of the main conclusions of the paper shows that a fully proactive method is infeasible for no-wait constraints. No-wait constraints are broken not only when tasks have longer durations, but also when they have shorter durations and they do not follow immediately one after the other. Thus, the proactive method has been adapted in the study in the following manner: real durations are sampled from a noise-perturbed distribution, the model is rebuilt using those new durations and the instance is re-solved using the offline solution as a warm start for the solver. The online feasibility guarantee is handled by the CP solver, which finds a feasible solution if it exists, given enough time. In essence, the proactive method is not fully proactive anymore as tasks are shifted during execution to respect the no-wait constraints. Moreover, calling the solver only once during the online execution with all the new durations as input simulates a 'look ahead' behavior.

Reactive approach. This method shares similarities in the offline phase with the proactive approach as it begins from a precomputed feasible schedule and triggers re-optimization only when real-time durations diverge from the expected ones. At each re-optimization step, a full CP model with embedded NW and GTL constraints is resolved. Throughout the experiments, all reactive reschedulings succeeded, ensuring full feasibility.

STNU-based approach. In this approach, constraints are embedded into a *Simple Temporal Network with Uncertainty (STNU)*. Feasibility depends on whether the network is *dynamically controllable*, meaning it can guarantee all constraints regardless of how uncertain durations resolve. This stricter requirement, coupled with tight no-wait constraints and resource limitations, often leads to infeasibility, especially when task chains are long and overlapping as we see in instances with a higher number of tasks. As previously mentioned this is showed in (Figures 1a and 1b).

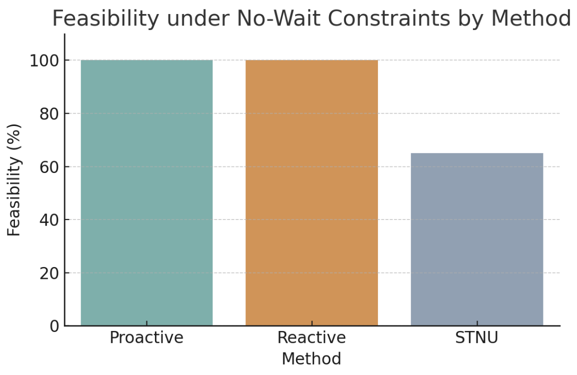


Figure 2: Feasibility under no-wait constraints across scheduling methods. Proactive and Reactive always yield feasible solutions, while STNU fails on 35-40% of instances on average due to dynamic controllability limits.

For completeness, the same plot for general time lag constraints can be found in Appendix A.

4.2 Offline Execution Time

The *offline execution time* refers to the pre-processing cost required to produce a ready-to-run schedule. As shown in Figure 3, the STNU-based approach incurs the highest offline overhead, a direct consequence of the temporal reasoning required to generate a partial order schedule that allows for task flexibility.

Both the *proactive* and *reactive* methods incur lower offline costs due to their reliance on simpler constraint programming (CP) formulations. This result is consistent with expectations, as the STNU must compute not only valid task sequences but also temporal flexibility margins, which increase computational complexity.

4.3 Online Execution Time

The *online execution time* measures the runtime overhead incurred during schedule execution. In Figure 4, we can observe how our implementation choice affects the proactive method, which still exhibits a low online time, but due to the warm start re-solve during the online phase, it causes it to take slightly more time than the STNU approach. The reactive method, by contrast, triggers a full re-optimization at runtime upon any deviation, leading to significantly higher online costs.

The STNU-based method strikes as the lowest computationally intensive online approach. It makes online decisions dynamically, but leverages the precomputed partial order and the RTE* algorithm to determine feasible task activations. This allows it to remain more responsive than the reactive method while still adapting to execution-time variability. This comes at a cost however, as it can be observed later in the paper that in terms of makespan the STNU performs considerably less optimal due to choosing robustness over optimality in terms of makespan.

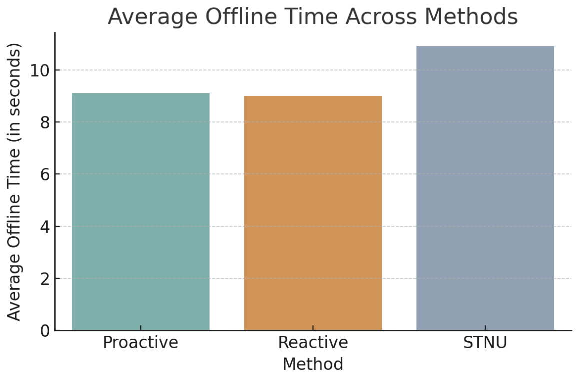


Figure 3: Average offline execution time across Proactive, Reactive, and STNU-based scheduling approaches.

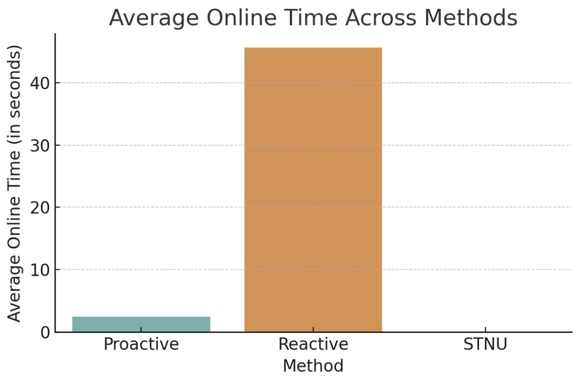


Figure 4: Average online execution time across all methods. Results averaged over all noise factors and modes for feasible schedules only.

4.4 Objective Performance

The objective performance of each scheduling method was evaluated in terms of average makespan across different duration distribution modes, as illustrated in Figure 5. Overall, the STNU-based approach consistently produces the highest makespan values across all modes. This outcome is expected given its strict requirement for dynamic controllability, which enforces conservative timing decisions to remain robust under all possible realizations of uncertainty. The STNU policy avoids any risk of temporal constraint violations, thereby prolonging the overall schedule.

In contrast, the proactive and reactive methods demonstrate significantly lower makespan values, with proactive scheduling yielding the lowest average makespan across all tested modes. This can be attributed to the constraint programming (CP) solver's ability to fully optimize task ordering and machine assignments upfront, with the ability to 'look ahead' and know all the deterministic durations before they take place due to the implementation strategy. Interestingly, the reactive approach performs only slightly worse than proactive in terms of makespan, primarily due to the overhead of re-optimization during execution, not having the ability to 'look ahead'. However, it still outperforms the STNU approach, indicating that adaptivity with optimization offers a better trade-off than robustness alone when makespan minimization is a priority with such demanding constraints in terms of lags.

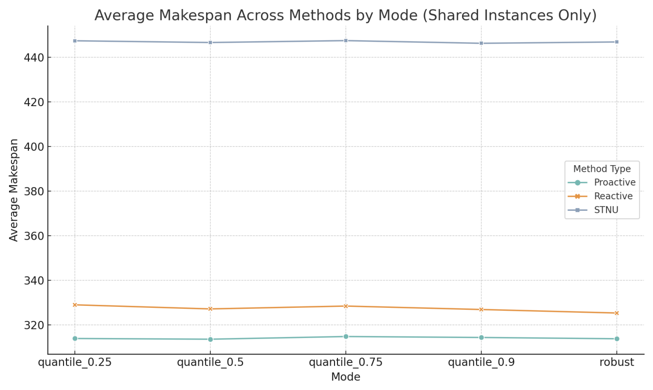


Figure 5: Average makespan across method types by duration distribution mode, based on shared feasible instances only from the Fattahi dataset.

Notably, we can observe the flatness of the curves representing the average makespan of the methods influenced by different modes for the duration distributions in Figure 5. This was an unexpected result, but it shows that no-wait and general time lag constraints strongly restrict the relative timing between tasks. For the no-wait constraints the difference in makespan between modes appears to come mostly from the difference in duration of the tasks since there is no lag in between tasks and the samples follow the offline schedule.

Moreover, the two noise levels tested have been shown to have identical impact on makespan, online time or offline time. The data supporting this observation can be found in Appendix A.

4.5 Makespan vs. Online Time (Trade-Off Curve)

Figure 6 presents a scatter plot of *average makespan* versus *average online time*, highlighting how the better performing methods balance schedule quality against runtime responsiveness. Each type of point represents a specific mode within a method type.

Several patterns emerge:

- Proactive method occupies the left region of the plot, offering low online cost with very similar performance in terms of makespan. It appears however, that for instances with a higher makespan due to complexity/task count, the reactive approach performs slightly better in terms of makespan
- Reactive methods dominate the upper-right area, incurring high online execution time in exchange for modest improvements in makespan.
- Modes converge to about the same result, thus modes can be ignored in the analysis of this plot

This analysis reveals that the two methods produce very similar schedules in terms of makespan, but the cost in online responsiveness of the reactive approach may outweigh these gains in real-time systems. It is important to note however, that the proactive approach 'looks ahead' and has access to all of the durations at one time which is not applicable in real, dynamic environments, but was kept as a viable method for comparison purposes.

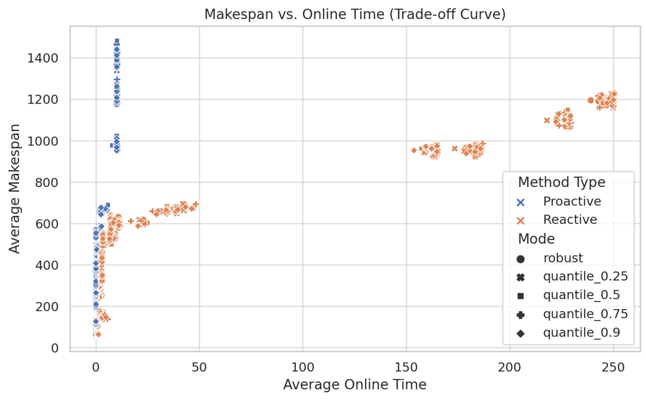


Figure 6: Trade-off between makespan and online execution time for all feasible method-mode pairs. Lower-left points represent better overall performance.

5 Responsible Research

Open artefacts. Every script or parameter file required to reproduce the experiments is archived at https://github.com/your-lab/fjsp-stnu-experiments . The repository also stores the raw CSV outputs for all simulation runs plus a Dockerfile that pins solver and library versions. In addition, it contains visual representations of the schedules, clearly labeled for transparency and verifiability. Because the enrichment of time-lags is driven by deterministic hash seeds (Section 3) and the addition of constraints takes place in the modeling phase, any user can regenerate identical augmented instances from the original Fattahi data set.

Assumptions and scope. To keep the comparison tractable we adopt (1) fixed routing alternatives selected offline, (2) perfect knowledge of the processing-time distributions, and (3) instantaneous rescheduling with no switch-over losses. Resources are assumed to be reliable and identical copies of each machine type. These simplifications follow common practice in FJSP studies but may under- state the challenges of highly dynamic shops or of scenarios with learning, breakdowns or heterogeneous workers. We therefore caution against direct industrial deployment without validating the models under plant-specific conditions.

Ethical and societal aspects. The work investigates optimisation algorithms; it does not process personal data, nor does it automate safety-critical decisions. Potential negative impacts are limited to energy consumption, already quantified above, and to possible misuse for excessive production targets. We encourage practitioners to weigh efficiency gains against worker well-being and sustainability objectives, in line with the "ten simple rules" for reproducible and responsible computational research [12; 8].

6 Conclusions and Future Work

This study aims to evaluate how different scheduling strategies perform under uncertainty in the context of the Flexible Job Shop Scheduling Problem (FJSP) with no-wait and generalized time-lag (GTL) constraints. We implemented and compared three approaches: proactive, reactive, and STNU-based and analyzed them across key metrics including feasibility, offline and online execution time, and makespan.

Our results show both expected and unexpected insight due to the novelty of the approaches in combination with the tight temporal constraints. We found during the research that a strictly proactive approach in combination with no-wait constraints is infeasible, due to the need of shifting tasks in the online phase. We thus opted for a proactive method with minimal computation during the execution of tasks, which lead to an interesting problem that strays away from the current research. The problem considers comparing a proactive method with 'look ahead' capabilities to a reactive method that reschedules the tasks 'step by step'. This was touched upon in the analysis of the results in the research paper, but future work can consider investigating this further. Our research demonstrates that the proactive and reactive methods achieve full feasibility even under strict temporal constraints, whereas STNU-based approaches fail approximately 35-40% of times on the dataset used. We learn that this is primarily due to the strong requirements of dynamic controllability, which limit the flexibility of the STNU in combination with tight schedules. In terms of makespan, we observe the same pattern, the hybrid approach aims for robustness rather than optimizing the makespan, thus the reactive and proactive outperform the STNU-based approach considerably in terms of minimizing schedule length. Notably, we observed that the choice of distribution mode (robust or different quantiles) had surprisingly little impact on the overall makespan, especially under no-wait constraints.

For future work, several options emerge. First, a better approach for the proactive method can be studied and experimented with in relation with no-wait and general time lag constraints. Second, the comparison between the proactive 'look ahead' approach and the 'step by step' reactive strategy might an interesting topic of research as we observed that with more complex instances the reactive approach performed slightly better in terms of makespan, but it took considerably more time to create the schedule. As a last suggestion, following research on the matter could consider analyzing the transition from no-wait constraints to tight general time lags that transition into more relaxed lags and compare feasibility during the transition.

Overall, the aim of this research was to expose how different methods of solving dynamic scheduling problems perform on a problem that was not studied before in combination with the no-wait and general time lag constraints. The insights gained during the research show how proactive, reactive and hybrid approaches based on Simple Temporal Networks with Uncertainty perform when used on Flexible Job Shop Problems with the previously mentioned constraints.

References

[1] Nabil Aissani et al. The flexible job shop scheduling problem: A review. *European Journal of Operational Research*, 2023. DOI: https://doi.org/10.1016/j.ejor.2023.05.045.

[2] Peter Brucker and Rainer Schlie. Job-shop scheduling with multi-purpose machines. *Computing*, 45(4):369–375, 1990. Published December 1990.

[3] Imran A. Chaudhry and Abdul A. Khan. A research survey: Review of flexible job shop scheduling techniques. *International Transactions in Operational Research*, 23(3):551–591, 2016. Available at https://www.academia.edu/14895542/A_research_survey_review_of_flexible_job_shop_scheduling_techniques.

[4] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1–3):61–95, 1991.

[5] M. Ebrahimnejad and M. Khalilzadeh. Efficient scheduling of a no-wait flexible job shop with periodic maintenance activities. *Journal of Quality Engineering and Production Optimization*, 7(2):73–84, 2021. Available at https://jqepo.shahed.ac.ir/article_4047.html.

[6] P. Fattahi, M. S. Mehrabad, and F. Jolai. Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. *Journal of Intelligent Manufacturing*, 18(3):331–342, 2007.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[8] Verena Heise, Constance Holman, Hung Lo, et al. Ten simple rules for implementing open and reproducible research practices after attending a training course. *PLoS Computational Biology*, 18(12):e1010750, 2022.

[9] 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 (ICAART 2024)*, volume 2, pages 253–263, 2024.

[10] L. Lan and J. Berkhout. Pyjobshop: Solving scheduling problems with constraint programming in python, 2025. Available at https://arxiv.org/abs/2502.13483.

[11] Nicola Muscettola, Paul Morris, and Ioannis Tsamardinos. Reformulating temporal plans for efficient execution. In *Proc. 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR '98)*, 1998.

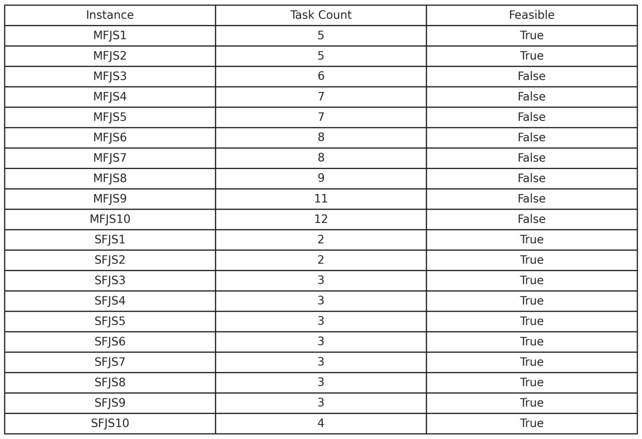
[12] Geir Kjetil Sandve, Anton Nekrutenko, James Taylor, and Eivind Hovig. Ten simple rules for reproducible computational research. *PLoS Computational Biology*, 9(10):e1003285, 2013.

[13] SchedulingLab. Flexible job-shop instances repository. https://github.com/SchedulingLab/fjsp-instances, 2025. commit <`hash`> accessed 10 Jun 2025.

[14] TNO. Digitalising smarter production, 2024. Accessed: 2025-04-27.

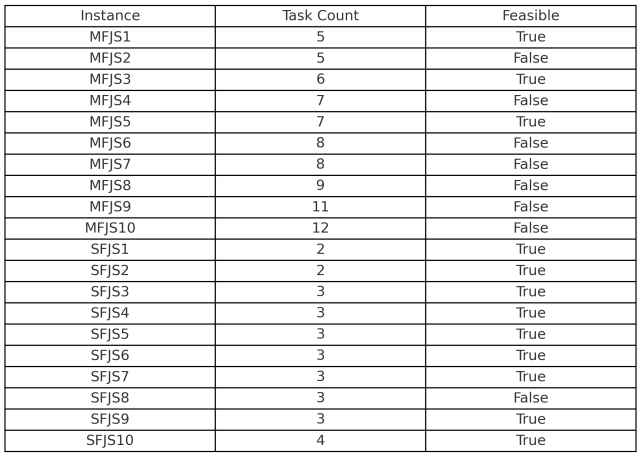
[15] Kim van den Houten, Leon Planken, Esteban Freydell, David M.J. Tax, and Mathijs de Weerdt. Proactive and reactive constraint programming for stochastic project scheduling with maximal time-lags, 2024. Available at https://arxiv.org/abs/2409.00000.

A Results



Instance	Task Count	Feasible
MFJS1	5	True
MFJS2	5	True
MFJS3	6	False
MFJS4	7	False
MFJS5	7	False
MFJS6	8	False
MFJS7	8	False
MFJS8	9	False
MFJS9	11	False
MFJS10	12	False
SFJS1	2	True
SFJS2	2	True
SFJS3	3	True
SFJS4	3	True
SFJS5	3	True
SFJS6	3	True
SFJS7	3	True
SFJS8	3	True
SFJS9	3	True
SFJS10	4	True

Figure 7: STNU feasibility results for each benchmark instance, showing task count and feasibility outcome under no-wait constraints.



Instance	Task Count	Feasible
MFJS1	5	True
MFJS2	5	False
MFJS3	6	True
MFJS4	7	False
MFJS5	7	True
MFJS6	8	False
MFJS7	8	False
MFJS8	9	False
MFJS9	11	False
MFJS10	12	False
SFJS1	2	True
SFJS2	2	True
SFJS3	3	True
SFJS4	3	True
SFJS5	3	True
SFJS6	3	True
SFJS7	3	True
SFJS8	3	False
SFJS9	3	True
SFJS10	4	True

Figure 8: STNU feasibility results for each benchmark instance, showing task count and feasibility outcome under general time lag constraints.

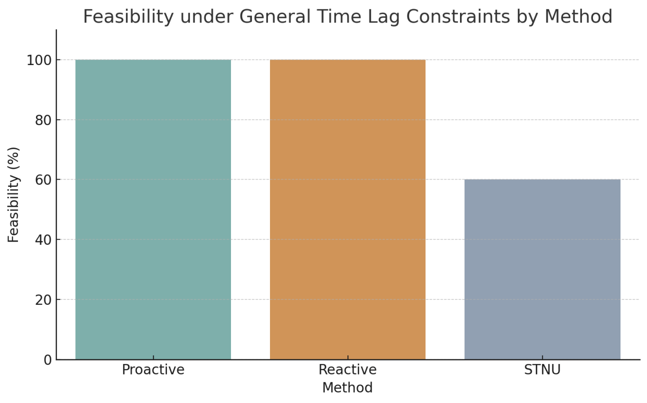
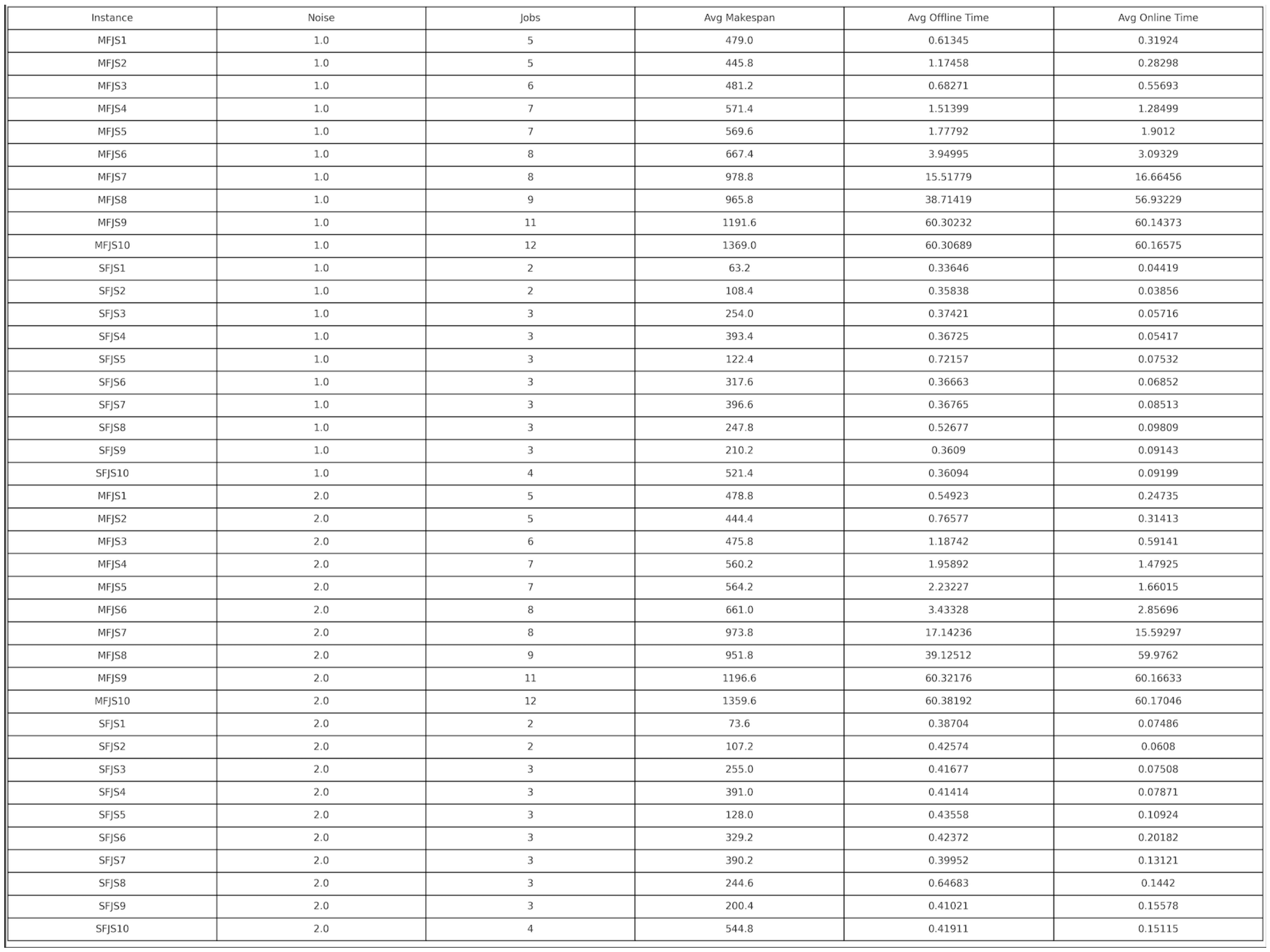


Figure 9: The feasibility ratios across methods for general time lag constraints.



Instance	Noise	Jobs	Avg Makespan	Avg Offline Time	Avg Online Time
MFJS1	1.0	5	479.0	0.61345	0.31924
MFJS2	1.0	5	445.8	1.17458	0.28298
MFJS3	1.0	6	481.2	0.68271	0.55693
MFJS4	1.0	7	571.4	1.51399	1.28499
MFJS5	1.0	7	569.6	1.77792	1.9012
MFJS6	1.0	8	667.4	3.94995	3.09329
MFJS7	1.0	8	978.8	15.51779	16.66456
MFJS8	1.0	9	965.8	38.71419	56.93229
MFJS9	1.0	11	1191.6	60.30232	60.14373
MFJS10	1.0	12	1369.0	60.30689	60.16575
SFJS1	1.0	2	63.2	0.33646	0.04419
SFJS2	1.0	2	108.4	0.35838	0.03856
SFJS3	1.0	3	254.0	0.37421	0.05716
SFJS4	1.0	3	393.4	0.36725	0.05417
SFJS5	1.0	3	122.4	0.72157	0.07532
SFJS6	1.0	3	317.6	0.36663	0.06852
SFJS7	1.0	3	396.6	0.36765	0.08513
SFJS8	1.0	3	247.8	0.52677	0.09809
SFJS9	1.0	3	210.2	0.3609	0.09143
SFJS10	1.0	4	521.4	0.36094	0.09199
MFJS1	2.0	5	478.8	0.54923	0.24735
MFJS2	2.0	5	444.4	0.76577	0.31413
MFJS3	2.0	6	475.8	1.18742	0.59141
MFJS4	2.0	7	560.2	1.95892	1.47925
MFJS5	2.0	7	564.2	2.23227	1.66015
MFJS6	2.0	8	661.0	3.43328	2.85696
MFJS7	2.0	8	973.8	17.14236	15.59297
MFJS8	2.0	9	951.8	39.12512	59.9762
MFJS9	2.0	11	1196.6	60.32176	60.16633
MFJS10	2.0	12	1359.6	60.38192	60.17046
SFJS1	2.0	2	73.6	0.38704	0.07486
SFJS2	2.0	2	107.2	0.42574	0.0608
SFJS3	2.0	3	255.0	0.41677	0.07508
SFJS4	2.0	3	391.0	0.41414	0.07871
SFJS5	2.0	3	128.0	0.43558	0.10924
SFJS6	2.0	3	329.2	0.42372	0.20182
SFJS7	2.0	3	390.2	0.39952	0.13121
SFJS8	2.0	3	244.6	0.64683	0.1442
SFJS9	2.0	3	200.4	0.41021	0.15578
SFJS10	2.0	4	544.8	0.41911	0.15115

Figure 10: Experiment summary showing the similarity in results between the two different noise factors.