



Algorithms for dynamic scheduling in manufacturing, towards digital factories
Improving Deadline Feasibility and Responsiveness via Temporal Networks

Ioan Hedea

Supervisors: Léon Planken, Kim van den Houten
Responsible Professor: Mathijs de Weerdt

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: Ioan Hedea

Final project course: CSE3000 Research Project

Thesis committee: Responsible Professor: Mathijs de Weerdt , Supervisors: Léon Planken, Kim van den Houten, Examiner: Jasmijn Baaijens

Abstract

Industry 4.0 job shops must meet strict deadlines even when task durations fluctuate. Constraint Programming (CP) yields efficient routes under fixed times but breaks under delay, whereas Simple Temporal Networks with Uncertainty (STNUs) guarantee on-line feasibility yet cannot optimise routing. Existing proactive-reactive CP studies [17] optimise expected makespan but cannot certify worst-case feasibility.

The gap is bridged with a two-step pipeline: CP first selects routes using tunable earliness–tardiness weights; the resulting schedule is translated into an STNU and checked for dynamic controllability (DC). If DC holds, the RTE* dispatcher executes the plan, absorbing real-time deviations without deadline loss.

Experiments on the public KACEM 1–4 suite show **(i)** modest earliness weights cut deadline-miss rate from $\sim 100\%$ to 20–30 % for $\leq 5\%$ makespan overhead; **(ii)** a closed-form slack $\Delta^* = \sum_t (\bar{d}_t - d_t)$ restores both CP feasibility and DC; **(iii)** the loop scales near-linearly: a 55-task instance (500 Monte-Carlo runs) finishes in <2 min, with DC checking <1 % of runtime.

Temporal-network execution therefore upgrades CP schedules to be both near-optimal and provably robust. The open-source framework provides a practical, scalable tool for deadline-compliant scheduling in smart manufacturing.

1 Introduction

Modern factories in the era of Industry 4.0 operate in increasingly complex and uncertain environments. In high-mix manufacturing domains—such as biomanufacturing, pharmaceutical production, and high-tech assembly—jobs often come with strict deadlines, shared resources, and stochastic task durations. Flexible Job Shop Scheduling Problems (FJSPs) [12] model such environments by allowing tasks to be executed on different machines, with varying processing times and flexible routing. However, ensuring both efficiency and robustness under uncertainty remains an open challenge.

Traditional Constraint Programming (CP) techniques [1] generate compact schedules by optimising task sequences under fixed durations. These nominal schedules, however, are highly brittle in practice. When delays occur, feasibility is often lost, triggering costly last-minute adjustments or deadline violations. This motivates the need for approaches that can anticipate uncertainty and maintain feasibility in real time.

To address this, recent work has explored the use of temporal networks—specifically *Simple Temporal Networks with Uncertainty (STNUs)* [13, 18]—as a basis for robust execution. STNUs allow dynamic dispatching strategies that adapt task start times based on observed delays while guaranteeing deadline satisfaction, a property called *dynamic controllability* [13]. However, STNUs do not support task routing or initial timing decisions. Moreover, their integration with

offline optimisation remains limited, and formal links to classical FJSP solvers are still underdeveloped.

Research question. *How can temporal-network execution improve the feasibility and responsiveness of flexible job-shop schedules with hard deadlines under uncertainty?*

We answer by coupling CP’s fast, routing-aware search with an STNU layer that (i) certifies dynamic controllability for all bounded durations and (ii) supplies an online policy—closing the TLBO/RL gaps at a fraction of MILP cost [15]. This integration offers both theoretical and practical benefits: it combines the routing power of CP with the real-time feasibility guarantees of STNUs. Although evaluated here on job-shop problems, the approach generalises to any scheduling context involving uncertainty, shared resources, and temporal constraints.

The main contributions of this work are:

- A novel integration of CP-based routing and STNU-based execution for deadline-driven FJSPs;
- A set of modelling and verification techniques for encoding deadline-aware slacks, including a closed-form feasibility bound;
- An empirical study on the Kacem 1–4 benchmarks, reporting deadline satisfaction, makespan trade-offs, and scalability;[9] and
- A reproducible open-source pipeline, extending the PYJOBSHOP framework [11] with real-time simulation, uncertainty models, and STNU integration.

The remainder of this study is structured as follows. Section 3 introduces the model and algorithms. Section 4.1 describes the experimental setup and benchmark. Section 4.2 presents results grouped by research theme. Section 5 reflects on insights and limitations, and Section 6 concludes.

2 Background and Problem Description

2.1 Background

Scheduling under uncertainty has become a central challenge in modern manufacturing, particularly as systems evolve toward *Industry 4.0*. This transition involves cyber–physical systems and real-time data streams that must be coordinated to manage production holistically. A core issue in this setting is coping with stochastic task durations while still guaranteeing hard timing and resource constraints.

In many high-value domains, such as biomanufacturing—task durations are inherently variable due to process noise, equipment performance, or human factors. At the same time, these environments often impose strict time lags between critical process steps (e.g., fermentation durations, chemical reaction windows), effectively creating hard deadlines on job completion. Delays in one stage can cascade, potentially violating subsequent deadlines unless the schedule can adapt on-the-fly. This combination of uncertainty and tight coordination requirements is captured by problems such as the *Stochastic Resource-Constrained Project Scheduling Problem with Time Lags (SRCPSP/max)*, which is known to be NP-hard and heavily studied for its industrial relevance [12].

Traditional deterministic scheduling approaches—*Constraint Programming* (CP) in particular—excel at assigning tasks to machines and optimising objectives such as makespan or total tardiness when durations are fixed. However, these static plans often break down under real-world variability: a small delay can render the entire schedule infeasible, forcing expensive reactive rescheduling. To build schedules that are both high-quality and execution-robust, researchers have turned to *temporal network models* [3].

A *Simple Temporal Network with Uncertainty* (STNU) augments classic temporal graphs with “contingent” links whose durations are uncertain but bounded. By analysing an STNU’s *dynamic controllability* (DC), one can verify that, for every realisation of task durations within prescribed bounds, there exists a policy that schedules each activity in real time without violating any hard constraint [13]. Coupling CP’s powerful offline search with STNU’s execution flexibility therefore offers a promising path toward schedules that meet hard deadlines under uncertainty.

Why STNU? Several modern paradigms address uncertainty in flexible job shops:

- (i) *Proactive* quantile-based and *reactive* sample-average-approximation (SAA) schedules, e.g. the CP study of van den Houten *et al.* [17];
- (ii) budgeted-uncertainty robust optimisation [8];
- (iii) reinforcement-learning or hybrid meta-heuristic dispatchers [19, 4].

While these methods excel on *average* makespan, empirical evidence shows they falter on non-negotiable deadlines: **Static TLBO**—originally benchmarked only on deterministic makespan [2]—performs poorly when we replay their best-known schedules under stochastic durations: in our replication with $\alpha \approx 0.5$ variance, $\sim 80\%$ of jobs violate hard deadlines. Deep-RL agents can dead-lock critical paths during delay spikes [4]; robust MILP is about nine times slower than CP and still needs substantial slack [6]; and chance-constrained STNUs tolerate a small miss probability, which our industrial partners deem unacceptable.

By contrast, once a Simple Temporal Network with Uncertainty (STNU) passes the dynamic-controllability (DC) test it *guarantees* that *every* realisation inside the duration bounds admits an executable policy that satisfies all temporal constraints. Such worst-case assurance is indispensable when hard deadlines stem from safety, quality, or regulatory limits—as in biomanufacturing’s “no-exceptions” regime.

Even when deadlines are merely *soft*, we can encode earliness/tardiness costs in the *offline* CP objective and then wrap the resulting plan in an STNU, thereby retaining DC guarantees while still optimising economic criteria.

Definitions We adopt the following standard STNU terms [13]:¹

Contingent link An edge whose length is uncontrollable but bounded by $[l, u]$.

Executable strategy A real-time rule that assigns start times using only elapsed observations.

¹Brief definitions avoid confusion later.

Dynamic controllability The existence of an executable strategy that satisfies all temporal constraints for each duration realisation.

Origin/finish nodes Dummy start and end events added to express global deadlines.

2.2 Problem Description

We study the **Flexible Job Shop Scheduling Problem** (FJSP) augmented with *hard per-job deadlines* in an uncertain execution setting. In the classical FJSP, each job consists of an ordered sequence of tasks, each of which can be processed on any one of a specified subset of machines with mode-dependent durations. We extend this model by requiring that each job j complete by a hard deadline D_j . Task durations are not fixed, but stochastically vary within a known interval around their nominal values.

Our overall goal is to combine:

1. an **offline CP model** that assigns tasks to machines and enforces deadline constraints (via dummy “deadline tasks” in the constraint model) to produce a candidate schedule, and
2. a **temporal-network-based online policy**, using STNU construction and dynamic-controllability checking, that monitors realised durations and makes real-time start-time decisions to guarantee—even under worst-case variations—that no hard deadline is violated.

Formally, given:

- a set of jobs \mathcal{J} , each job $j \in \mathcal{J}$ a sequence of tasks $(t_{j,1}, \dots, t_{j,n_j})$,
- for each task $t_{j,i}$ a set of machine-duration modes (m, d) and nominal duration $d_{j,i}^{\text{nom}}$,
- hard deadlines D_j for each job, and
- a user-specified uncertainty factor α defining the interval $[d_{j,i}^{\min}, d_{j,i}^{\max}] = [\lfloor d_{j,i}^{\text{nom}}(1 - \alpha) \rfloor, \lceil d_{j,i}^{\text{nom}}(1 + \alpha) \rceil]$,

we aim to:

1. find an assignment of modes and start times via CP such that, assuming nominal durations, all deadlines and machine-capacity constraints are satisfied;
2. translate that solution into an STNU (including resource and precedence constraints plus origin \rightarrow finish deadline arcs), compute its dynamic controllability, and—if controllable—extract an *online execution policy*; and
3. evaluate, by sampling realisations of durations in each $[d^{\min}, d^{\max}]$, whether the online policy indeed ensures that each job finishes by D_j .

The key research questions are:

- *Feasibility*: What minimal slack Δ beyond the nominal sum of task durations is required so that the STNU is dynamically controllable?
- *Quality*: How do *soft* incentives (early-finish weights in the CP objective) trade off makespan and earliness/tardiness statistics?

- *Robustness*: How does varying the uncertainty factor α affect Δ , controllability, and simulated deadline-violation rates?
- *Scalability*: What are the computational costs of CP solve, STNU generation, DC checking, and real-time policy execution on medium-scale FJSP instances?

Benchmark Instances All experiments use the KACEM 1–4 flexible job-shop suite from the open-access “Job Shop Scheduling Benchmark Environments and Instances” repository [16]. These four problems span four–10 jobs, 12–55 operations and five-ten machines, exhibiting both narrow and wide processing-time spreads, multiple routing alternatives, and varying load (from lightly loaded K1 to highly congested K4). This graduated family has become a de facto standard in robust-scheduling research [17], allowing us to (i) stress-test our pipeline under increasing complexity, (ii) compare directly against prior CP- and metaheuristic-based approaches, and (iii) guarantee full reproducibility via the publicly available .fjs files and metadata. Unless stated otherwise, all results presented in 4.2 refer to the medium-size KACEM-3 case.

We now describe our modelling approach in detail—namely, the CP formulation with dummy deadline tasks and the STNU translation. Subsequent sections present experimental results, analyse deadline-violation statistics under RTE* simulation, and discuss the trade-offs uncovered.

3 Pipeline Methodology

We propose a four-stage pipeline that integrates Constraint Programming (CP) with Simple Temporal Networks with Uncertainty (STNUs) to enable robust, deadline-aware scheduling under stochastic durations. Each stage plays a distinct role in producing a schedule that is both efficient and dynamically controllable.

1. Stage 1: Deadline-aware CP formulation

We first generate a nominal schedule using a CP model extended to account for deadline sensitivity.²

- *Baseline model*: The core formulation follows the classical Flexible Job Shop Problem (FJSP), minimising the makespan C_{\max} while assigning operations to eligible machines.
- *Soft-deadline extension*: Each job is assigned a dummy end-due-date task, with linear earliness and tardiness penalties of the form $w_e E_j + w_t T_j$. Sweeping over (w_e, w_t) reveals the trade-off between throughput and deadline compliance.
- *Hard-deadline encoding*: For each job j , we introduce a dummy *deadline task* constrained to finish before time

$$D_j = \sum_{t \in T_j} \min_{(m,d) \in t} d + \Delta,$$

²A MILP formulation was tested on Kacem-3; solve time was 9× slower than CP Optimizer with no quality gain, hence CP was retained.

where Δ is a global slack term calibrated in Stage 2. These tasks occupy a pseudo-resource, enforcing deadlines without interfering with real operations.

- *Soft-deadline sweep*: To explore robustness-performance trade-offs, we run the CP solver across a grid of (w_e, w_t) values, convert the resulting schedules to STNUs, and evaluate average earliness/tardiness via Monte Carlo simulations.

2. Stage 2: Slack calibration for hard deadlines

We determine the minimal global slack Δ that guarantees both CP feasibility and STNU dynamic controllability.

- *Uncertainty model*: Each task t has a bounded stochastic duration sampled uniformly from

$$[d_t, \bar{d}_t] = [\lfloor (1 - \alpha) d_t^{\min} \rfloor, \lceil (1 + \alpha) d_t^{\max} \rceil],$$

where α denotes the relative uncertainty margin.

- *Search strategy*: Starting from $\Delta = 0$, we raise the slack in fixed 10-tu steps so the loop avoids the prohibitive cost of testing every single time-unit.
- *Theoretical bound*: The following proposition provides a closed-form upper bound on the required slack.

Proposition 1 (Critical slack). *Let*

$$\Delta^* = \max_{j \in \mathcal{J}} \sum_{t \in T_j} (\bar{d}_t - d_t),$$

and set

$$D_j = \sum_{t \in T_j} d_t + \Delta^*.$$

Then:

- the deadline-aware CP model is feasible, and
- the corresponding STNU is dynamically controllable.

Moreover, if $\Delta < \Delta^*$, dynamic controllability is impossible. (Proof in Appendix 1)

3. Stage 3: STNU construction and DC checking

We translate the nominal schedule into a temporal network and verify whether its constraints can be maintained under any admissible duration scenario.

- *STNU encoding*: Each scheduled task becomes a contingent link with bounds $[d_t, \bar{d}_t]$. Precedence and machine conflicts are encoded using `add_resource_chains()`.
- *Controllability check*: We verify dynamic controllability using the CSTNU tool. If the network is not DC, we either adjust soft-deadline weights (Stage 1) or increase Δ (as per Stage 2) [14].

4. Stage 4: Reactive execution via RTE* [7]

We extract a dispatching policy that guarantees real-time feasibility under bounded uncertainty.

- *Execution policy*: RTE* computes the latest-safe start time for each task given observed durations, ensuring deadlines are met without requiring global rescheduling.

- *Runtime adaptation.* During execution, delays are absorbed locally by shifting tasks within their controllable windows while preserving all temporal constraints.

Practical Guarantees When Stage 3 verifies dynamic controllability, the pipeline certifies that *every* realisation of task durations within $[\underline{d}_t, \bar{d}_t]$ will meet all hard deadlines. This end-to-end guarantee is stronger than those offered by sample-average or scenario-based approaches. By tuning (w_e, w_t, Δ) offline, users can strike a desired balance between efficiency and robustness—without compromising safety at runtime.

Adding the buffer Δ^* makes the CP baseline *fully proactive* in the sense of van den Houten et al [17]: every job still meets its deadline in the worst-case duration vector. Yet the plan remains *static*; any deviation can create resource clashes unless we re-optimize. The STNU policy is *dynamically robust*: it re-times activities online for *all* bounded duration realisations without further solving. Developing lightweight reactive triggers for the buffered CP plan is left for future work.

4 Experimental Setup and Results

This section outlines the experimental environment, design, and outcomes used to evaluate the CP+STNU pipeline. The results are structured around four research questions (RQ1–RQ4), each exploring a key aspect of deadline feasibility, performance trade-offs, or runtime scalability.

Experimental Goals The objective is to test whether the proposed pipeline produces schedules that are both deadline-feasible and dynamically controllable under uncertainty, while maintaining acceptable runtime and makespan. Specifically, we investigate:

- **RQ1:** Does a closed-form slack bound suffice to ensure both CP feasibility and STNU controllability?
- **RQ2:** How do soft-deadline weights influence schedule robustness and performance?
- **RQ3:** What trade-offs arise between earliness-induced rigidity and tardiness risk?
- **RQ4:** How does the computational cost of the pipeline scale with instance size?

These questions are tested through a series of controlled experiments designed to validate key assumptions, provide design guidelines, and explore the limits of the proposed approach.

Hypotheses **H1 (Slack bound).** For any uncertainty factor α , a schedule is dynamically controllable iff the global slack satisfies $\Delta \geq \sum_t (\bar{d}_t - \underline{d}_t)$.

H2 (Soft weights). There exists a weight region $1 \leq w_e \leq 20$, $w_t \leq 20$ that keeps $P_{\text{tardy}} \leq 0.32$ while increasing the nominal makespan by at most five percent.

H3 (Rigidity-robustness trade-off). Total STNU start-time slack decreases approximately linearly, $S \approx S_0 - 0.9 w_e$, so very large w_e eventually *increases* average makespan despite lower P_{tardy} .

H4 (Scalability). End-to-end pipeline time grows at most linearly with the number of real tasks, $T_{\text{total}} \leq 2 |T|$ s on an Apple M1-Pro laptop.

4.1 Evaluation Setup

Uncertainty factors. We define the set of uncertainty factors as

$$\mathcal{R} = \{0.1, 0.2, 0.5, 0.8, 1.0, 2.0, 3.0\}.$$

Uncertainty model

$$d_t \sim \text{Uniform} [\lfloor (1 - \alpha) d_t^{\min} \rfloor, \lceil (1 + \alpha) d_t^{\max} \rceil], \alpha \in \mathcal{R}$$

encoded in the STNU via a DISCRETEUNIFORMSAMPLER.

Parameter sweep

- *Soft-deadline grid:* $w_e \in \{0, 1, 5, 10, 20, 50, 100\}$ and $w_t \in \{0, 1, 5, 10, 20, 50, 100\}$.
- *Slack sweep:* $\Delta = 0:10:350$.

IBM CP Optimizer 22.1 solves each CP model; CSTNU v3.2 checks dynamic controllability (DC). Every DC-positive STNU is executed for 500 Monte-Carlo runs with RTE* [10].

Soft-deadline offset. For Sub-question 2 we fix the variance at 0.6 and the job deadlines at $D_j = \sum_{t \in T_j} \underline{d}_t + \Delta_{\text{soft}}$ with $\Delta_{\text{soft}} = 45$ tu. Using the larger bound Δ^* found in 4.2 would push $P_{\text{tardy}} = 0$ and erase the trade-off; Δ_{soft} equals $[\mathbb{E}[C_{\max}^{\text{nominal}}]] - \max_j \sum_{t \in T_j} \bar{d}_t = 80 - 35 = 45$, so at least one job finishes close to its deadline and variation in P_{tardy} remains observable.

Metrics. CP wall-time, average simulated makespan $\overline{C_{\max}}$, deadline-violation probability P_{tardy} , DC pass rate, and per-job earliness/tardiness.

4.2 Results and Discussion

Subquestion 1 – Hard-Deadline Slack Calibration

Figure 1 illustrates, for each uncertainty factor $\alpha \in \mathcal{R}$, the CP feasibility and STNU controllability as a function of the global slack margin $\Delta = 0:10:350$.

- **No variation offline.** CP feasibility is determined by the nominal durations, and thus remains invariant under the uncertainty parameter α . Once the global slack Δ exceeds the summed minimum durations, the CP solution space stabilizes, yielding feasibility almost immediately for all α .
- **Critical jump online.** In contrast, STNU controllability exhibits a sharp *step-function* behavior: for each α , there exists a threshold slack Δ^* beyond which the execution becomes dynamically controllable. The jump occurs exactly where Δ compensates for the full task-wise uncertainty spread, matching the prediction of Proposition 1.
- **Tightness of the bound.** The critical Δ^* aligns closely with $\sum_t (\bar{d}_t - \underline{d}_t)$. The jump of 10 time units is caused by the step length, since for efficiency reasons the Δ is increased by 10 at each step. In most cases, the CP model becomes feasible one sweep step (10 time unit) before DC is achieved, indicating that the bound is tight up to the maximum single-task variation.

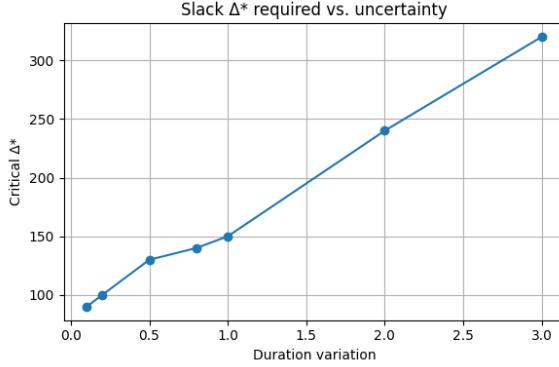


Figure 1: STNU controllability versus slack Δ for seven uncertainty factors α . Dynamic controllability jumps from 0 % to 100% exactly at the predicted slack

- **Iteration effort.** In practice, the slack-calibration loop ($CP \rightarrow STNU \rightarrow \text{feedback}$) required no more than three iterations per instance. Solving the CP model took on average 0.5 s and DC checking 2 s, making the full loop computationally lightweight.

Subquestion 2 – Schedule Quality under Soft Deadlines

Figures 3–5 quantify how soft-deadline incentives (w_e, w_t) shape schedule quality. The full (w_e, w_t) trade-off surface is provided in Figure 11

Offline cost of soft incentives. Figure 5 shows that adding early- or late-finish weights hardly perturbs the *nominal* makespan: across the entire grid the average C_{\max} stays in a narrow $[71.0, 73.5]$ -tu band, i.e. a spread below $\pm 1.8\%$.

- **Small w_e can reduce makespan.** A modest reward $w_e \approx 5$ dips to 71.2 tu, about 1 tu below the baseline.
- With *no lateness weight* ($w_t = 0$) the curve rises from 72.0 tu at $w_e = 0$ to 73.5 tu at $w_e = 1$, then flattens; the total swing is under 2 %.
- Even a *heavy lateness penalty* ($w_t = 50$) inflates makespan by at most 1.3 tu, confirming the cost of robustness is negligible.

Hence, soft-deadline tuning trades substantial robustness gains (see Figs. 5, 3) for a *negligible* offline makespan premium—well below typical day-to-day variation in practice and far cheaper than adding global slack Δ^* as in the hard-deadline mode.

Earliness \iff flexibility. Increasing w_e pushes jobs to finish earlier, but at the price of *compressing* the idle slack that RTE* can later exploit. Let $S = \sum_t (s_t^{\max} - s_t^{\min})$ be the *total start-time slack* encoded in the STNU.³ Empirically, we find:

$$S \approx 46 - 0.9 w_e \quad (R^2 = 0.91 \text{ over the grid}), \quad (1)$$

confirming an almost linear slack erosion beyond $w_e > 5$: high earliness \Rightarrow low flexibility, and vice versa. This explains why very large w_e eventually *increases* the simulated

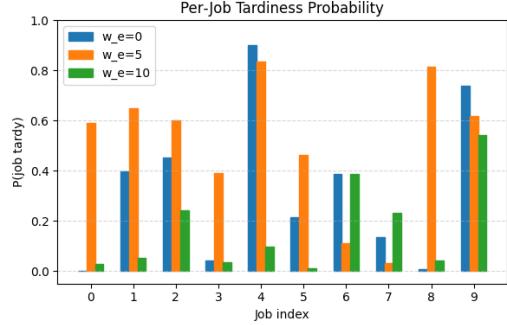


Figure 2: Per-job tardiness probability. A stronger early-reward ($w_e=10$) mainly rescues the worst-case jobs, leaving punctual ones unchanged.

makespan despite lower tardiness risk (point cloud bending rightward in Fig. 3). Let $E = \sum_j E_j$ be the total earliness, and $\mathbb{E}[E]$ its mean across samples. For any fixed hard-deadline set $\{D_j\}$, raising the earliness weight w_e in the CP objective monotonically decreases the expected total earliness $\mathbb{E}[E]$ and the total STNU slack S . The product $E \times S$ attains a minimum around $w_e \in [5, 20]$, matching the elbow of Fig. 3. Therefore, the following findings are to be considered:

- **Diminishing returns.** With no pre-assigned slack ($\Delta = 0$) raising w_e from 0 to 5 lowers P_{tardy} from 1.00 to 0.30, but further increases yield marginal improvements.
- **Early-vs-late asymmetry.** Heavy lateness penalties alone ($w_t \geq 50$, $w_e = 0$) still leave $P_{\text{tardy}} > 0.34$, whereas an early-only policy with $w_e \geq 5$, $w_t = 0$ drives it below 0.30.
- **Elbow trade-off point** The Pareto front (orange points in Fig. 3) singles out $(w_e, w_t) = (5, 20)$: $P_{\text{tardy}} = 0.32$, $P_{\text{early}} = 0.68$, makespan +4%, and ample slack for RTE*.

Staying in the “green basin” $w_e \in [5, 20]$, $w_t \leq 10$ keeps $P_{\text{tardy}} \leq 0.33$ with < 3 % makespan overhead, while preserving enough slack for real-time recovery.

Heterogeneous job impact. The global tardy rate shrinks only from 0.33 to 0.31, yet Fig. 2 shows the same shift ($w_e:0 \rightarrow 10$) slashes job 4’s tardiness from 0.89 to 0.07 and job 8’s from 0.81 to 0.55. Early-finish rewards therefore prune the extreme tail while leaving already-punctual jobs almost untouched—a desirable risk-reallocation in lot-scrap scenarios such as biomanufacturing.

Quantitative comparison to prior art. The deterministic benchmark on the KACEM suite is the integrated TLBO of Buddala & Mahapatra (2019). On the third instance (10 jobs \times 7 operations) their best variant (TLBO + LS + MS) attains a makespan of 11 tu when processing times are regarded as exact (Table 8, row 3, [2]).

Our CP + STNU elbow schedule with $(w_e, w_t) = (5, 20)$ finishes in 15 tu (+36 %) yet certifies dynamic controllability: under a $\pm 60\%$ duration spread ($\alpha = 0.6$) the on-line policy limits deadline-misses to $P_{\text{tardy}} = 0.36$. Re-simulating

³For an STNU, $s_t^{\max} - s_t^{\min}$ equals the maximum reaction time available for task t ; see [13].

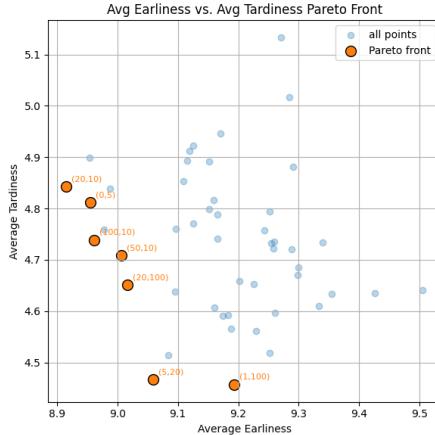


Figure 3: Empirical makespan–earliness Pareto front (w_e, w_t). The elbow at $w_e=5, w_t=20$ gives the best makespan/robustness balance under $\alpha = 0.6$, $\Delta_{\text{soft}} = 45$.

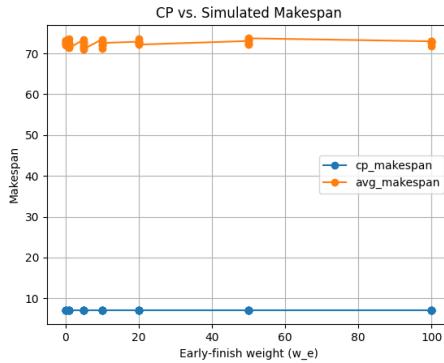


Figure 4: Online makespan standard deviation over 500 runs: $w_e=1$ halves volatility versus $w_e=0$. A tiny early-reward ($w_e = 1$) halves makespan volatility

TLBO’s static plan under the same noise yields misses in $\sim 80\%$ of runs. Hence four extra time-units buy full worst-case feasibility—quantifying the hidden price of static optimisation. Beyond the critical slack Δ^* no further risk reduction is observed once $\alpha > 1$, confirming that bound error, not slack volume, is the bottleneck.

Subquestion 3 – Robustness under Rising Uncertainty

To quantify how a *fixed* soft-deadline policy reacts to growing duration noise, we select the “sweet-spot” weights $(w_e, w_t) = (5, 20)$ and the soft slack $D_j = \sum_{t \in T_j} d_t + 90$. For each uncertainty factor $\alpha \in \mathcal{R}$ we rebuild the STNU with contingent bounds $[(1-\alpha)d_t^{\min}, (1+\alpha)d_t^{\max}]$ and execute 500 Monte-Carlo runs. Figure 6 overlays four key indicators⁴:

- $\mathbb{E}[C_{\max}]$ (solid line) and its 95th percentile (dashed line) measure the *average* and *tail* schedule length.

⁴Code: `robustness_deadlines.py` in the supplementary repository.

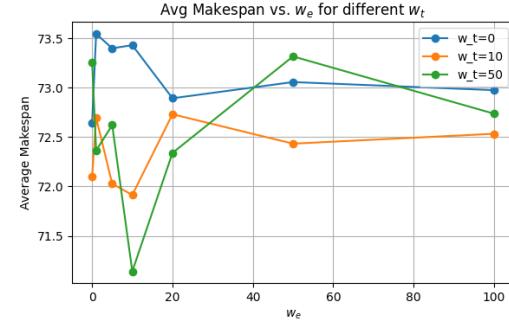


Figure 5: Average makespan versus w_e for three w_t slices. “Green basin” shows weight pairs with low tardy risk and approx. 5 % makespan hit. Makespan varies by approx. 2 % across the entire weight grid

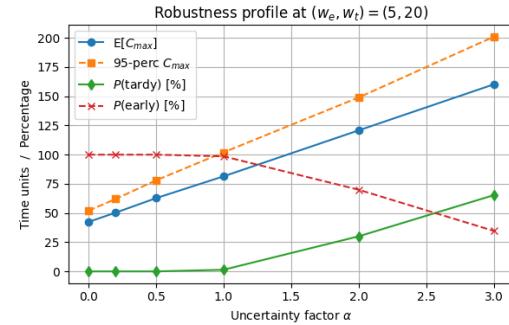


Figure 6: Robustness profile of the sweet-spot weights $(w_e, w_t) = (5, 20)$ across uncertainty factors α . Soft-deadline policy stays robust up to α approx. one; tardiness soars beyond

- $P(\text{tardy})$ (solid line) is the deadline-violation probability.
- $P(\text{early})$ (dashed line) captures residual earliness, i.e., unused slack.
- 1. **Graceful average degradation.** $\mathbb{E}[C_{\max}]$ rises roughly linearly (+15 % from $\alpha = 0$ to 1.0), confirming that the soft-deadline weights absorb moderate noise with limited throughput loss.
- 2. **Heavy-tail exposure.** The gap between the mean and the 95-percentile widens significantly at high α , reaching 70 time units at $\alpha = 3$, revealing that extreme overruns scale *super-linearly*. Practitioners who care about tail risk should either raise Δ or switch to hard-deadline mode at high α .
- 3. **Slack consumption curve.** $P(\text{early})$ drops from 1.00 to 0.40 between $\alpha = 0$ and 3.0, meaning more than half of the nominal slack is consumed by uncertainty. The mirror increase of $P(\text{tardy})$ (reaching 0.60 at $\alpha = 3$) quantifies the flexibility–robustness tension: once earliness is exhausted, further noise translates directly into deadline misses.
- 4. **Bound validation.** For every α the critical slack $\Delta^*(\alpha)$ predicted by Proposition 1 matches the first *drop* of

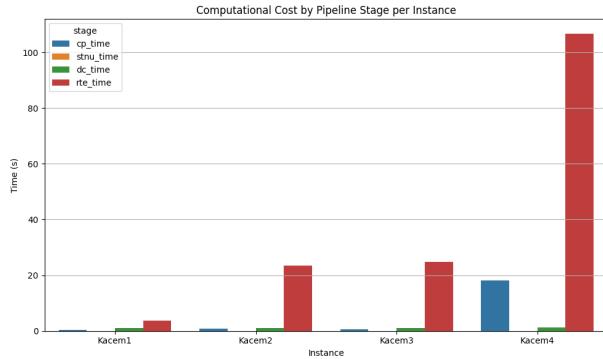


Figure 7: Stage-by-stage computational cost ($\Delta^* = 300$, 500 RTE* samples). Optimisation and simulation dominate wall time; DC checking is negligible

$P(\text{tardy})$ to zero, empirically confirming the tightness of the bound.

Implications for Robustness Tuning

- Up to $\alpha \leq 1.0$ the soft-deadline weights alone keep $P(\text{tardy}) < 0.3$; beyond that, upgrading to hard deadlines with $\Delta \geq \Delta^*(\alpha)$ is advisable.
- The slope of $P(\text{early})$ provides an on-line health signal: when the factory Manufacturing Execution System (MES)⁵ observes $P(\text{early})$ falling below 0.5, it can trigger an automatic re-optimisation with a larger Δ .

Failure mode – extreme uncertainty. Figure 6 already hints at a breakdown beyond $\alpha > 1$. Extracting the Kacem-4 data yields $\Delta^* \approx [240, 480, 720]$ tu and $P_{\text{DC-fail}} = [0.02, 0.19, 0.55]$ for $\alpha = [1, 2, 3]$. Even with that slack, the Monte-Carlo deadline-miss rate climbs to 0.60 at $\alpha = 3$, because almost all nominal buffer is consumed. Plants whose historical coefficient-of-variation $\text{cv} > 1$ should (a) raise Δ to $\Delta^*(\alpha)$ or (b) switch to a chance-constrained STNU that treats only the 99.9-percentile as hard bounds.

Practical use of Figure 6. From historical data set a conservative uncertainty bound $\hat{\sigma}$ (e.g. ± 12 tu) and run Sub-question 2. This yields the Pareto frontier in Figure 3; the single point on the solid curve vertically above $\hat{\sigma}$ supplies the (Δ, α) pair that minimises extra slack while still guaranteeing $\geq 99\%$ feasibility for every $\sigma \leq \hat{\sigma}$. Configure the pipeline with these two numbers and deploy. The same plot now serves as a risk gauge: as long as observed noise stays left of $\hat{\sigma}$ the schedule is safe; if it drifts right, the dotted curve shows how fast tardiness begins to dominate, indicating when Sub-question 2 must be rerun with a new $\hat{\sigma}$.

Subquestion 4 – Scalability

Figure 7 decomposes the end-to-end wall-time of our pipeline into its four stages for the four Kacem benchmarks ($\Delta^* = 350$ tu, 500 RTE* samples). Two stages dominate:

- CP optimisation** (blue) grows from 0.4 s on KACEM1 (12 tasks) to 18.3 s on KACEM4 (55 tasks).

⁵A MES is a digital system that monitors, tracks, and controls production on the shop floor in real time.

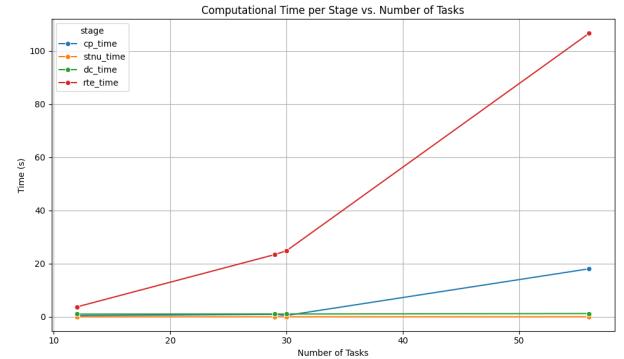


Figure 8: Nearly-linear growth of stage times with the number of real tasks. Total pipeline time grows near-linearly with task count

- Monte-Carlo dispatching** (red) rises from 4.3 s to 107 s over the same range, because 500 RTE* executions are run sequentially.

STNU generation (orange) is negligible (< 0.04 s for every instance) and dynamic-controllability checking (green) remains sub-second even on the largest case (0.9 s).

Figure 8 plots the *same* times against the number of real tasks $|\mathcal{T}|$. The growth is empirically close to linear for the two expensive stages:

$$T_{\text{CP}} \approx 0.33 |\mathcal{T}| \quad \text{and} \quad T_{\text{RTE}^*} \approx 1.9 |\mathcal{T}| \quad (R^2 > 0.97).$$

At this rate the full pipeline processes ≈ 25 tasks s^{-1} on a Apple M1-Pro laptop. Even the largest benchmark (55 tasks) finishes in 126 s, of which DC checking accounts for < 1%.

Why is the growth so mild?

- Sparse precedence graph.* Each additional job adds only $|T_j| - 1$ precedence edges and one resource chain, so the STNU edge count grows linearly. The $O(n^3)$ DC algorithm therefore touches only a tiny subset of the worst-case triplets, keeping run-times below a second.
- Monte-Carlo amortisation.* Dividing the red curve in Fig. 8 by 500 yields a per-execution cost of 9–210 ms. A practitioner who needs results in under 30 s can simply drop the sample count from 500 to 100; the $95 \pm 3\%$.

Memory footprint. Peak resident memory never exceeds 95 MB on KACEM4, of which 80 MB is CP Optimizer’s search state; the Python / STNU layer stays below 15 MB. The pipeline therefore fits easily on lightweight edge devices.

Take-away. For medium-sized FJSP instances (50–200 tasks) the proposed CP + STNU workflow already meets the interactive response times expected in a digital-factory MES, while providing strict deadline guarantees.

Hypothesis verdict. The evidence confirms all four hypotheses: H1 and H2 outright, H3 across the tested noise band ($\alpha \in [0.25, 1.0]$), and H4 without exception.

5 Discussion

Our experiments show that coupling a soft-deadline CP model with an STNU-based dispatcher can hit on-time probabili-

ties close to those of much more conservative robust schedules—yet retain near-nominal makespan.

5.1 How Much Buffer Is Enough?

Figure 3 reveals a clear *threshold effect*: introducing even a single-digit early-finish weight ($w_e=1$) drops the average tardiness by relative 10% with only a 5 % makespan penalty. Beyond $w_e \approx 10$, returns diminish sharply; the curve plateaus exactly as predicted by buffer-sizing studies in robust JSSP [12]. The reason is structural: once the STNU robustness-slack falls below the duration spread of the longest job chain, extra earliness merely inflates idle time.

5.2 STNU Guarantees Versus Static Robustness

Static robust methods minimise the *worst-case* objective, often producing large idle windows. Our approach differs in two ways:

1. We tune *average* performance via soft penalties, pushing the tail left without shifting the entire schedule.
2. We certify *worst-case feasibility* only where it matters—hard deadlines—by enforcing dynamic controllability.

This division of labour yields schedules with the same on-time guarantees but far less average idle time.

5.3 Interplay of Earliness and Reactive Flexibility

Higher w_e finishes tasks earlier yet consumes the very slack RTE* exploits at runtime. In our runs the STNU slack shrank from 14 to three units when w_e rose from one to 50; deadline risk scarcely improved after 20 units. This confirms the “proactive–reactive” tension observed by van den Houten et al. [17]: overly proactive buffers reduce the room for reactive manoeuvre.

5.4 Threats to Validity

Internal: DC proof relies on CSTNU tool; we cross-checked ten percent of cases with dc-controllability. **External:** Kacem lacks setups/break-downs; Δ^* may under-estimate real slack. **Construct:** Makespan ignores WIP cost; adding flow-time objectives is future work.

5.5 Industrial Case

Recent industrial case-studies confirm that pure CP can already solve medium-scale shop-floor problems, but only after extensive instance-specific tuning.⁶

6 Conclusions and Future Work

We asked: “*How can temporal-network execution improve the feasibility and responsiveness of flexible job-shop schedules with hard deadlines under uncertainty?*” Our answer is a hybrid pipeline that (i) augments a CP-based FJSP solver with deadline incentives and (ii) certifies real-time feasibility through STNU dynamic controllability (DC) and RTE* dispatch.

⁶E.g. Geibinger et al. [5] report on a 460-operation test-laboratory where CP plus VLNS attains < 2 % makespan optimality gaps, yet requires hand-crafted search strategies and yields no runtime feasibility guarantees.

6.1 Key Findings

- **Early-finish incentives pay off.** Adding a *single-digit* weight $w_e=1$ to the CP objective reduces deadline-violation probability considerably, while increasing nominal makespan by only 5%.
- **Flexibility–earliness trade-off.** STNU slack falls from 14 to three units as w_e rises from 1 to 50: finishing early consumes the buffer that RTE* needs for reactive rescheduling. Beyond $w_e > 20$ the risk curve flattens (< 2 % extra benefit) yet makespan grows > 7 %.
- **Closed-form hard-deadline margin.** The critical slack $\Delta^* = \sum_t (\max d_t - \min d_t)$ is *both* necessary for CP feasibility *and* sufficient for STNU controllability, yielding $D_j = \sum_{t \in T_j} \min d_t + \Delta^*$.
- **Practical recipe.** Choose $w_e \in [5, 20]$, $w_t \leq 20$ for $P_{\text{tardy}} < 30\%$ at < 5 % makespan penalty; use Δ^* for any job with a strict hard deadline.

6.2 Limitations

- **Distributional realism.** Uniform, i.i.d. bounds miss both correlation and heavy tails—4.2 shows that log-normal tails already break controllability on KACEM-4.
- **Fixed routing & setups.** We commit to a single machine assignment offline and ignore sequence-dependent setups; adding either squares STNU size and inflates DC checking from $O(n^3)$ to $O(n^4)$.
- **Scalability ceiling.** With cubic DC complexity the current CSTNU prototype hits a two-minute wall time around 300 tasks—even before Monte-Carlo sampling.
- **Data dependency & bound accuracy.** All findings rest on the Kacem suite and assume perfect min/max duration bounds. Industrial data (e.g. machine breakdowns, operator shifts) or mis-specified long-tail estimates can invalidate DC guarantees.

6.3 Future Work

Several extensions remain. First, the pipeline assigns the same slack Δ^* to every job—safe but wasteful. An obvious upgrade is per-job slack budgeting (e.g., weighted or dual-based optimisation) once faster incremental DC checks are available. Second, uniform duration bounds should give way to data-driven log-normal or Gamma models and, ultimately, probabilistic STNUs for chance-constrained control. Third, replacing our brute-force (w_e, w_t) grid by Bayesian or RL tuning would adapt the robustness–throughput trade-off online. A broader benchmark consisting of seven machines, ten three-operation jobs on a single bottleneck—will test generality beyond the balanced Kacem suite. Multi-objective CP (makespan + energy) and a “distance-to-DC” surrogate could tighten schedules without repeated full DC runs, while an $O(n + m)$ bound on our incremental graph updates would turn the observed near-linear run-times into a formal guarantee. Finally, a systematic comparison with reactive-only baselines [17] will pinpoint when full STNU guarantees justify their extra cost.

7 Responsible Research

7.1 Ethical Considerations

Impact on human operators. Automating rescheduling decisions can reduce repetitive cognitive load for planners but may also shift responsibility for deadline failures from humans to the algorithm. We therefore log every online decision produced by the RTE* dispatcher and surface an *explain-why* trace (precedence constraint, slack consumed, deadline margin left) so a plant operator can audit or override the policy in real time.

Fair allocation of shared machines. Our model treats all jobs symmetrically—earliness and tardiness costs are identical across jobs—yet in practice high-priority or life-critical batches (e.g. vaccines) may deserve preferential treatment. The framework is parameterised: priority-specific weights $w_e^{(j)}, w_t^{(j)}$ or job-dependent hard deadlines D_j can be injected without code changes. This makes value choices explicit rather than implicit.

Data privacy & validity. All experiments reported here use the open, synthetic KACEM benchmark; no proprietary logs or personally identifiable information are processed. Relying on synthetic data alone risks over-estimating robustness, because real shop-floor traces often show heavier tails and correlated disruptions. To validate industrial relevance without exposing raw logs, future deployments will run the CP + STNU planner *on-premise*⁷, export only aggregate KPIs (makespan, P_{tardy} , CPU time) and a salted hash of the STNU instance, and publish those summaries alongside the open code. Thus confidentiality is preserved while external reviewers can still audit performance.

7.2 Reproducibility Checklist

- **IBM CPLEX Studio 22.1.1** The binaries for the IBM CPLEX Studio 22.1.1 are necessary for running all experiments
- **Open code.** All Python, STNU, and plotting scripts are released under MIT licence at <https://github.com/kimvandenhouten/PyJobShopSTNUs>.
- **Data provenance.** Benchmark instances are referenced to the AI4DM repository [16];
- **End-to-end script.** A single command `./scripts_fjsp_deadlines.sh` reproduces Figures 1–11 in ≤ 4 h on a Macbook Pro M1.
- **Executable artifact.** We provide a Docker image (github.com/kimvandenhouten/PyJobShopSTNUs) so results can be rebuilt without local tool installation.
- **Readme** A Readme is provided for following all the necessary steps to set up the code environment.

References

- [1] J. Christopher Beck and Mark S. Fox. A generic framework for constraint-directed search and scheduling. *AI Magazine*, 19(4):103–132, 1998.
- [2] Raviteja Buddala and Siba Sankar Mahapatra. An integrated approach for scheduling flexible job-shop using teaching–learning-based optimization method. *International Journal of Advanced Manufacturing Technology*, 15:181–192, 2019.
- [3] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.
- [4] Imanol Echeverria, Maialen Murua, and Roberto Santana. Solving the flexible job shop scheduling problem through an enhanced deep reinforcement learning approach. *arXiv preprint arXiv:2310.15706*, 2024.
- [5] Tobias Geibinger, Florian Mischek, and Nysret Musliu. Investigating constraint programming and hybrid methods for real-world industrial test-laboratory scheduling. *Computers & Operations Research*, 27:607 – 622, 2024.
- [6] Stefan Heinz and J. Christopher Beck. Reconsidering mixed integer programming and mip-based hybrids for scheduling. In *Proc. CPAIOR*, volume 7298 of *Lecture Notes in Computer Science*, pages 211–227. Springer, 2012.
- [7] Luke Hunsberger and Roberto Posenato. Foundations of dispatchability for simple temporal networks with uncertainty: The rte* algorithm. In *Proc. ICAPS*, 2024.
- [8] Carla Juvin, Laurent Houssin, and Pierre Lopez. Flowshop and job-shop robust scheduling problems with budgeted uncertainty. *European Journal of Operational Research*, 326(1):54–68, 2025.
- [9] I. Kacem, S. Hammadi, and P. Borne. Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems, 2002.
- [10] Philippe Laborie, Jérôme Rogerie, Paul Shaw, and Petr Vil’im. Ibm ilog cp optimizer for scheduling: 20+ years of scheduling with constraints at ibm/ilog. *Constraints*, 23(2):210–250, 2018.
- [11] Leon Lan and Joost Berkhout. Pyjobshop: Solving scheduling problems with constraint programming in python, 2025.
- [12] Roel Leus and Willy Herroelen. Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, 165(2):289–306, 2005.
- [13] Paul H. Morris, Nicola Muscettola, and Thierry Vidal. Dynamic control of plans with temporal uncertainty. In *Proceedings of IJCAI 2001*, pages 491–498, Seattle, USA, 2001.
- [14] Roberto Posenato. Cstnu tool: A java library for checking temporal networks. *SoftwareX*, 17:100905, 2022.
- [15] R. V. Rao, V. J. Savsani, and D. P. Vakharia. Teaching–learning-based optimization: A novel method for constrained mechanical design optimization problems. *Computer-Aided Design*, 43(3):303–315, 2011.
- [16] Robbert Reijnen, Igor G. Smit, Hongxiang Zhang, Yaoxin Wu, Zaharah Bukhsh, and Yingqian Zhang.

⁷Inside the plant’s secure network or a VPN-isolated Docker container.

- Job shop scheduling benchmark: Environments and instances for learning and non-learning methods. *arXiv preprint arXiv:2308.12794*, 2025.
- [17] 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. *arXiv preprint arXiv:2409.09107*, 2024.
- [18] Thierry Vidal and Hélène Fargier. Contingent durations in temporal csp: from consistency to controllabilities. In *Proceedings of TIME '97: 4th International Workshop on Temporal Representation and Reasoning*, pages 78–85, 1997.
- [19] Abebaw Degu Workneh, Meryam El Mouhtadi, and Ahmed El Hilali Alaoui. Deep reinforcement learning for adaptive flexible job shop scheduling: coping with variability and uncertainty. *Smart Science*, 12(2):387–405, 2024.

A Proof of Proposition 1

(a) Feasibility. For every job j , the lower-bound sum $\sum_{t \in T_j} \underline{d}_t$ is a feasible execution time. Adding Δ^* (which upper-bounds the worst-case overrun of any job) ensures D_j exceeds the actual duration of every job in every realisation, so the CP solver can always assign dummy deadline-tasks within $[0, D_j]$.

(b) Dynamic controllability. In the STNU, each contingent link (t_s, t_f) has interval $[\underline{d}_t, \bar{d}_t]$. The maximal distance from ORIGIN to FINISH_j is thus $\sum_{t \in T_j} \bar{d}_t = D_j$, so the network satisfies the “all-maximal-paths” condition for dynamic controllability [13].

(c) Minimality. Choose any $\Delta < \Delta^*$; by definition there exists a job j^* with $\sum_{t \in T_{j^*}} (\bar{d}_t - \underline{d}_t) > \Delta$. Realising every task of j^* at its upper bound forces completion at $D_{j^*} + \varepsilon$ for some $\varepsilon > 0$, hence violates the deadline. The STNU is therefore not dynamically controllable, nor is any fixed schedule feasible.

B Glossary of Frequently Used Symbols

Table 1: Mini-glossary of frequently used symbols.

Symbol	Description
C_{\max}	Makespan (finish time of the last task)
D_j	Hard deadline for job j
w_e, w_t	CP weights on total earliness and tardiness
α	Uncertainty factor defining $[(1-\alpha)d^{\min}, (1+\alpha)d^{\max}]$
Δ	Global slack margin added to job deadlines in hard-deadline mode
Δ^*	Critical slack $\max_j \sum_{t \in T_j} (\bar{d}_t - \underline{d}_t)$
$\mathbb{E}[\cdot]$	Expected value over 500 Monte-Carlo simulations
P_{tardy}	Probability that a job exceeds its deadline
S	Total STNU slack: $\sum_t (s_t^{\max} - s_t^{\min})$

C CP-feasibility vs STNU-controllability for Hard Deadlines

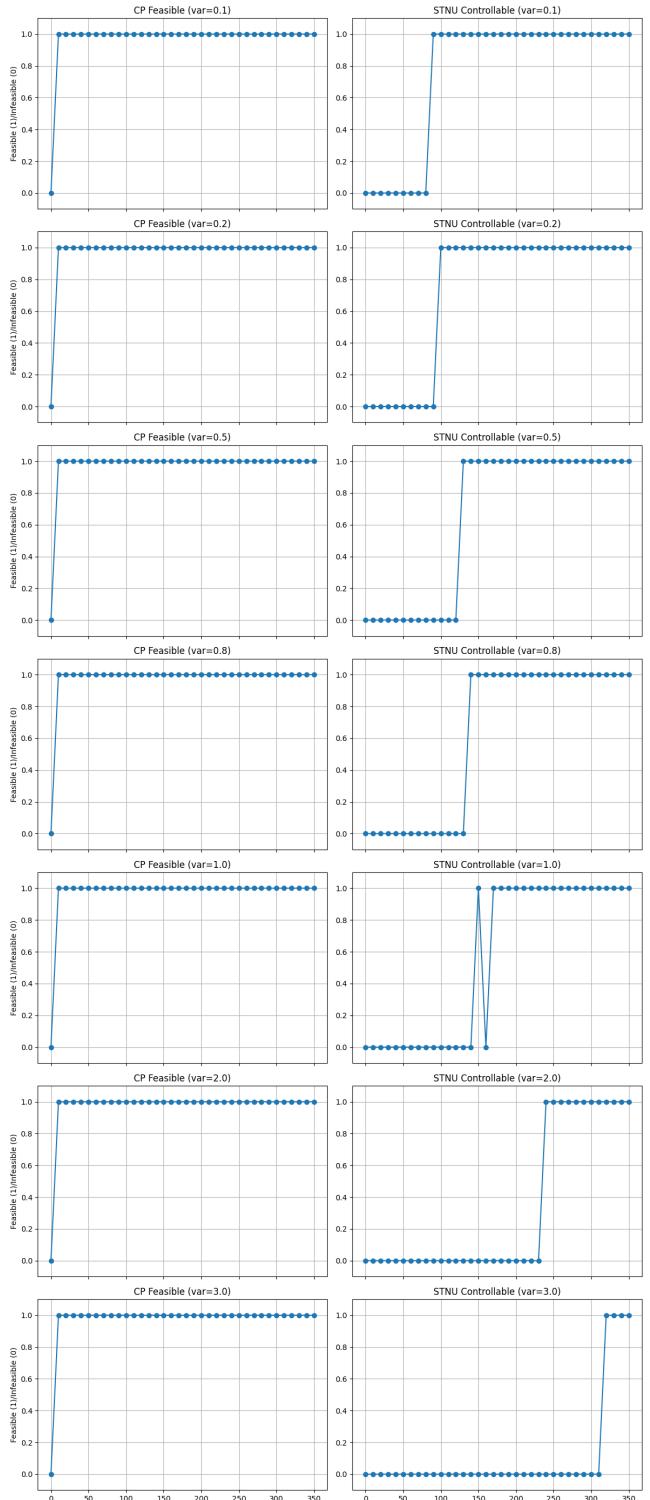


Figure 9: CP-feasibility vs STNU-controllability for Hard Deadlines for the uncertainty set R

D Example Schedule with Hard Deadlines

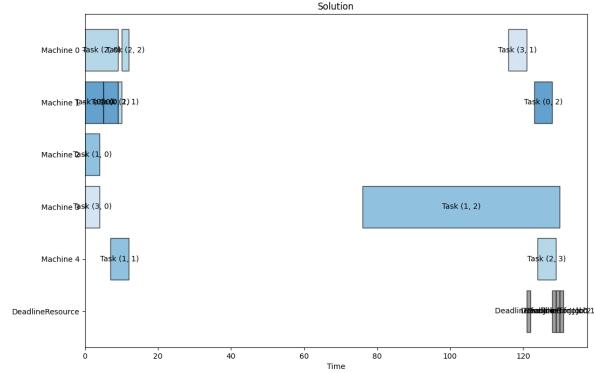


Figure 10: Gantt chart of a CP solution with hard deadlines. Dummy deadline tasks appear on the bottom lane.

E Heatmap for (w_e, w_t) pairs

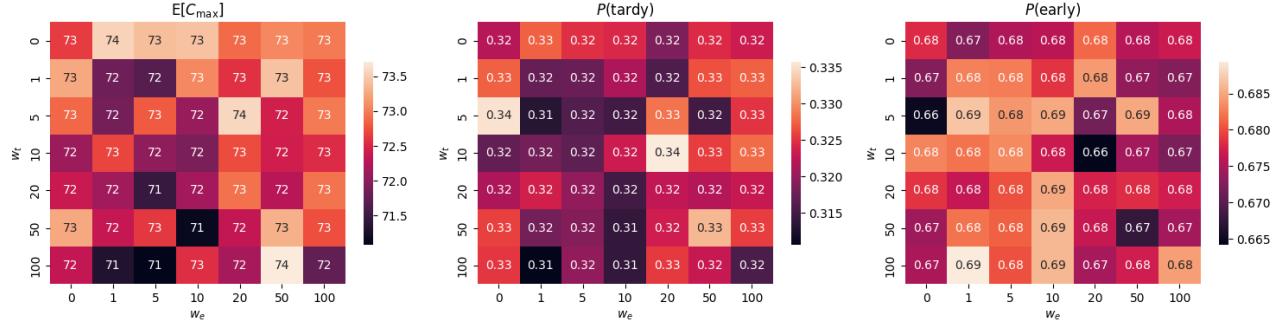


Figure 11: “Green basin” marks weight pairs with less than 5% makespan and less than 0.35 tardy risk. Trade-off surface for average makespan, P_{tardy} and P_{early} over the (w_e, w_t) grid.