Algorithms for Scheduling under Uncertainty

Author: Mayte Steeghs | m.c.steeghs@student.tudelft.nl
Supervisors: Kim van den Houten & Léon Planken
EEMCS, Delft University of Technology, The Netherlands

TU Delft

Introduction

- **Industry 4.0** the rise of "digital factories" requires job shop scheduling algorithms with stochastic durations.
- **Job shop scheduling problems (JSP)** are NP-Hard combinatorial problems.
- Previously, Mixed Integer Programming (MIP) methods like BACCUS and SORU-H were considered state-of-the-art. A recent paper shows **Constraint Programming (CP)** outperforms MIP across a wide range of benchmarking scheduling problem instances [1].
- This opened up research gap for finding robust **proactive** schedules or **reactive** approaches with rescheduling during execution which was previously considered too computationally heavy.
- van den Houten et al. filled this research gap by presenting **proactive**, **reactive**, and **hybrid** approach using latest CP advancements for the **SRCPSP/max problem** [2].
- Applications to different scheduling problem variants under different temporal constraints unexplored for these approaches.

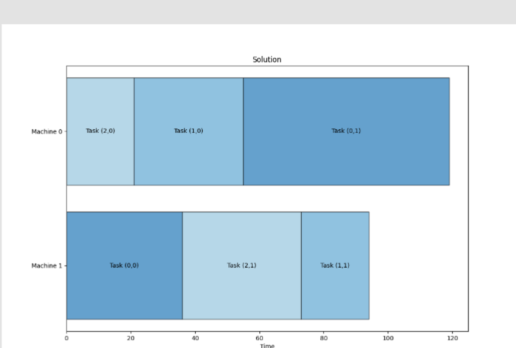
Research Question

Does an **STNU-based** method yield superior solution quality and runtime for the stochastic Flexible Job Shop Scheduling Problem with sequence-dependent set-up times compared to **proactive** and **reactive** CP methods?

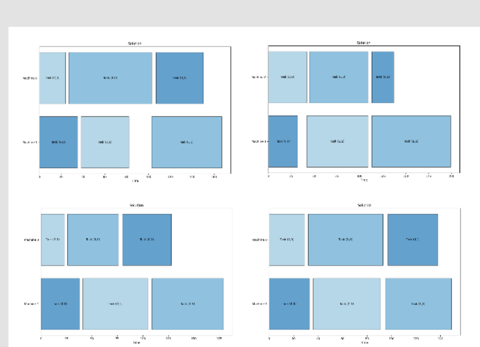
1. How does **uncertainty** affect performance and feasibility of solutions?
2. To what extent do **sequence-dependent set-up** times affect makespan, the feasibility ratio and computational time offline and online?
3. How robust are the different methodologies when **scaling** input problem size?

Problem Definition

- The **Flexible Job-Shop Problem Scheduling with Sequence-Dependent Set-Up Times** (FJSP-SDST) extends the classical job-shop by allowing multiple alternative machines per task and by imposing set-up times that depend on the order of consecutive tasks on the same machine.
- In the stochastic variant, durations follow a stochastic distribution. The duration d_j is an independent random variable.



Gannt chart of SFJSP without SDST



Gannt chart of Stochastic FJSP

Software

- **PyJobShop**: a scheduling problem CP solver in Python. [3]
- **IBMs CP Solver**: PyJobShop makes use of this solver. This solver was selected over Googles OR Tools for its direct implementation of SDST and associated performance benefits.
- Reactive, Proactive and Hybrid Approaches were taken from the code base of **van den Houten et al**. [2] and adapted for **PyJobShop**.

CP Methods

Proactive Method

- **Model**: draw duration samples / quantiles and solve a scenario-based CP-optimization to get one buffered start-time vector
- **Offline step:** heavy search to minimize expected makespan while remaining feasible for all sampled scenarios
- **Online execution**: zero computation during runtime

STNU Method

- **Model**: Construct POS from resource chain of single-point solution. Encode each task as contingent links inside an STNU and prove dynamic controllability (DC) → a policy always exists for every possible duration realization
- **Offline step:** Solve deterministic single-point solution using PyJobShop. Construct STNU from POS, check DC
- **Online execution:** RTE* dispatcher.

Reactive Method

- **Model**: Begin with single-point solution; treat every actual finish time as a trigger to re-solve.
- **Offline Step**: Solve deterministic single-point solution using PyJobShop.
- **Online loop:** observe → re-optimise → dispatch new schedule

Experimentation

Datset: Job Shop Scheduling Benchmark: Environments and Instances for Learning and Non-learning Methods [4].

- Contains 20 FJSP-SDST instances which will be converted to stochastic instances using sampler.

Experiments

- Each method is sampled 10 times and executed on every instance for noise levels ε = 1 and 2.
- SDST were scaled using α = (0, 0.25, 0.5, 0.75, 1) to answer sub-question 2.
- Proactive method used γ=1, reactive used γ=0.9

Evaluation Metrics:

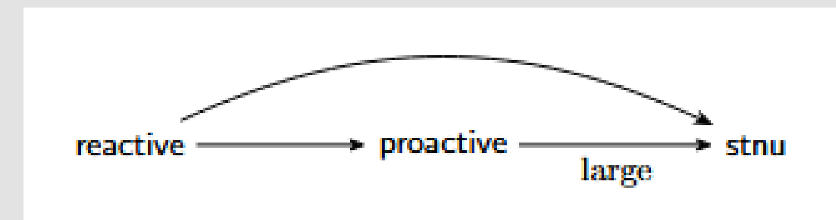
1. Makespan of the solution
2. Offline CPU time
3. Online CPU time
4. Feasibility Ratio

Statistical Test:

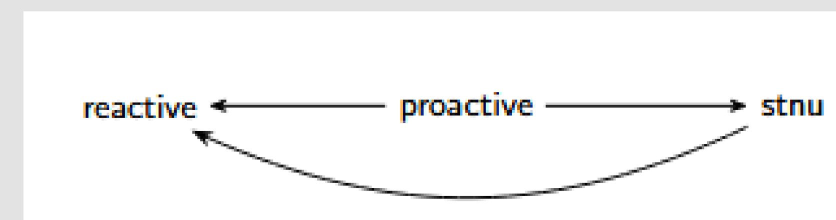
- Wilcoxon matched-pairs signed-rank test (consistency),
- Binomial proportion test (win counts)
- Paired t-test on "double hits" (magnitude).

Results

Solution Quality: Makespan



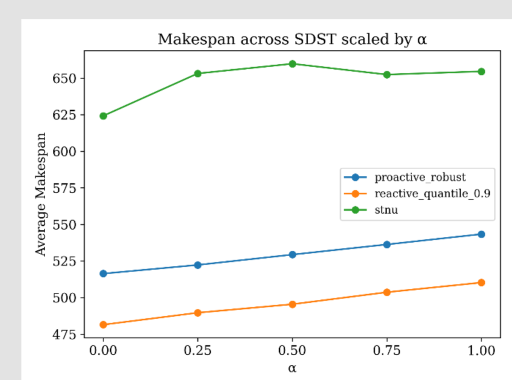
Computation Time (Online and Offline)

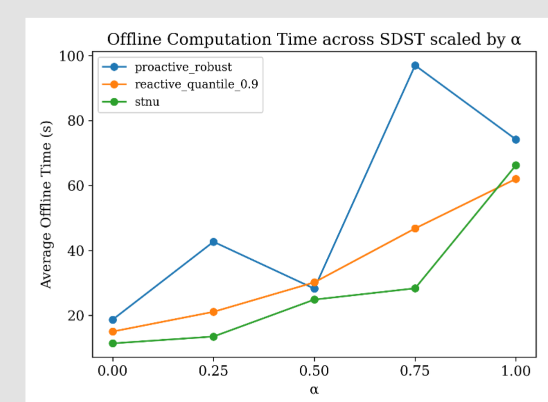


1. Uncertainty

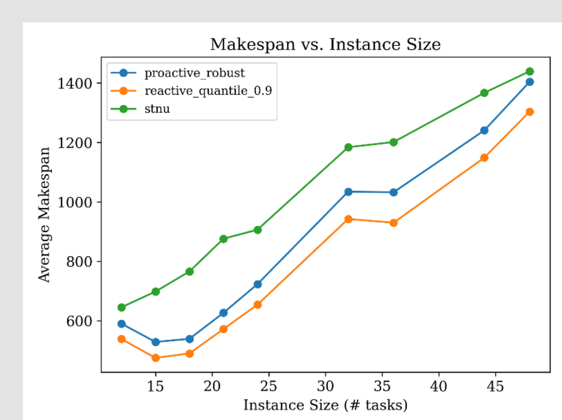
- Raising the noise level from ϵ = 1 to ϵ = 2 inflated the proactive makespan and increased both its offline and online times, while thea reactive and STNU makespans stayed unchanged;
- However, higher noise dramatically increased the reactive method's online computation.

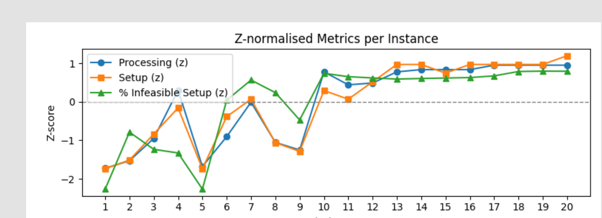
2. Sequence Dependent Setup Times



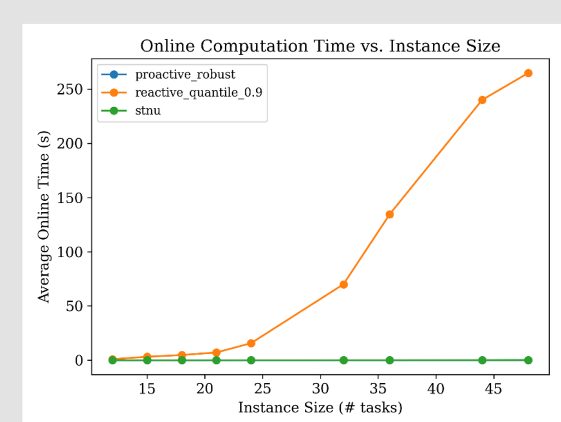


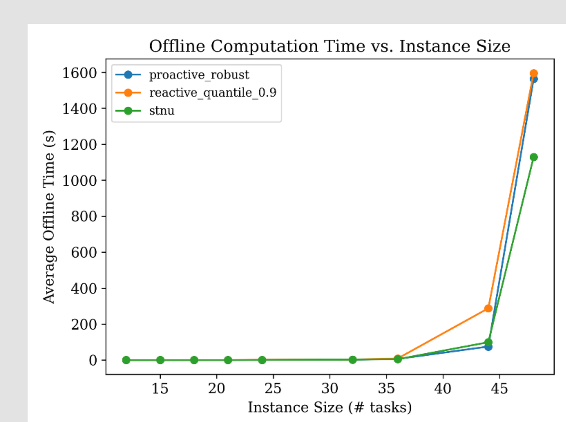
3. Scalability





- Instances 10 – 20 have similar setup time, task time and SDST feasibility ratios
- Plots comparing makespan, online and offline computation time





Conclusion

Reactive CP method

- Dynamically **reschedules** at every task completion and is very flexible, handling stochasticity in combination with sequencing constraints well.
- Delivers the consistently **lowest makespan** of all methods and better machine assignments of tasks.
- **Much higher online computation time** (frequent re-solving)

STNU method

- Contrary to hypothesis, **proactive CP beats STNU** on makespan for large/complex instances
- STNU must satisfy **dynamic controllability (DC)** which adds conservative slack to guarantee feasibility under worst-case uncertainty
- Modelling every sequence-dependent setup time (SDST) as a **contingent link** densifies the STNU graph, compounding conservatism.
- Underlying implementation factors might be the cause of this difference.

Recommendations

- High bound on **online timeout** of the reactive method might inequitably bias the results.
- Performing **hyperparamater** tuning on our dataset and setting the timeouts in an equitable way across methods
- Comparing **MIP** solutions to **CP** might further close the literature gap.
- **Alternative distributions** of task duration estimations might be closer to real life scenarios.
- Increase # **samples** per method to increase statistical robustness of results.
- Fix amount of threads for CP solvers and standardize CPU time measurements to increase reproducibility.
- Perform deeper investigation in the underlying graph structures of instances including how infeasible SDST arcs prune domain.

References

[1] Naderi, B., Ruiz, R., & Roshanaei, V. (2023). Mixed-integer programming vs. constraint programming for shop scheduling problems: new results and outlook. INFORMS Journal on Computing, 35(4), 817-843.
[2] Houten, K. V. D., Planken, L., Freydell, E., Tax, D. M., & de Weerdt, M. (2024). Proactive and Reactive Constraint Programming for Stochastic Project Scheduling with Maximal Time-Lags. arXiv preprint arXiv:2409.09107.
[3] Lan, L., & Berkhout, J. (2025). PyJobShop: Solving scheduling problems with constraint programming in Python. arXiv preprint arXiv:2502.13483.
[4] Reijnen, R., van Straaten, K., Bukhsh, Z., & Zhang, Y. (2023). Job shop scheduling benchmark: Environments and instances for learning and non-learning methods. arXiv preprint arXiv:2308.12794.