Adaptive Scheduling for Distributed Manufacturing Cells

A Constraint-Based Approach to Real-Time Reconfiguration

by

Alexandra J. Whitfield

August 2026

Faculty of Engineering, Environment and Computing

A thesis submitted in partial fulfilment of the University's requirements for the Degree of Doctor of Philosophy

To my parents, for their patience, and to the night-shift operators at Ansty Park who let me watch their line for three weeks straight.

Acknowledgements

I am grateful to my supervisory team at the Institute for Advanced Manufacturing and Engineering for their guidance throughout this project, and to the industrial partners who granted access to their production data. Any errors that remain are my own.

Abstract

Distributed manufacturing cells that reconfigure on the fly promise shorter changeover times, but existing schedulers assume a fixed machine topology and recompute from scratch whenever that topology changes. This thesis develops a constraint-based scheduler that treats reconfiguration as a first-class event rather than an exception, propagating only the constraints affected by a topology change instead of re-solving the whole problem.

We formulate cell scheduling as a constraint satisfaction problem over machine-task assignments and sequence-dependent setup times, and show that incremental constraint propagation after a reconfiguration event converges in time roughly linear in the number of affected tasks rather than the size of the whole schedule. A discrete-event simulation of a nine-cell automotive sub-assembly line, built from six months of production logs supplied by an industrial partner, is used to evaluate the approach against a conventional rolling-horizon scheduler under three classes of disruption: machine failure, rush-order insertion, and planned cell reconfiguration.

Across all three disruption classes the incremental scheduler reduces recovery time — the interval between a disruption and the point at which the cell returns to its target throughput — by a median of 34%, with the largest gains on planned reconfigurations, where the rolling-horizon baseline discards work already in progress. The thesis closes by discussing the conditions under which incremental propagation degrades to full re-solution, and what that implies for cell layouts with high reconfiguration frequency.

Table of contents

List of figures **xi**

List of tables **xiii**

1 Introduction **1**
1.1 Motivation 1
1.2 Research Questions 1
1.3 Contributions 2
1.4 Thesis Structure 2

2 Incremental Constraint-Based Scheduling **3**
2.1 Problem Formulation 3
2.2 Incremental Propagation 3
2.3 Degenerate Cases 4

3 Evaluation **5**
3.1 Simulation Environment 5
3.2 Results 5
3.3 When Incremental Propagation Breaks Down 6
3.4 Summary 6

Bibliography **7**

Appendix A Complexity Bound Derivation **9**

List of figures

3.1 Median recovery time by disruption class, 500 simulated runs each. 6

List of tables

2.1 Worst-case complexity of incremental propagation versus full re-solve, where d is the size of the affected subgraph and $n = |T|$. 4

Chapter 1

Introduction

Manufacturing cells that can be reconfigured between product variants without a full production stop are increasingly common in automotive sub-assembly, but the scheduling software that drives them has not kept pace. When a cell's machine topology changes — a robot arm is swapped for a different end-effector, a buffer is added, a station is taken offline for maintenance — most commercial schedulers treat the change as equivalent to a fresh problem instance and re-solve from scratch (Pinedo, 2016). For a cell running dozens of task types this can take longer than the disruption itself.

1.1 Motivation

The industrial partner for this work operates a nine-cell sub-assembly line that reconfigures on average four times per shift, mostly to absorb rush orders. Production logs supplied for this thesis show that the scheduler's own recompute time accounts for a median of 11% of total downtime following a reconfiguration event — time in which the cell is idle not because of the disruption itself, but because the software has not yet told it what to do next.

1.2 Research Questions

This thesis addresses three questions:

1. Can a reconfiguration event be treated as a local constraint update rather than a new problem instance, without losing solution quality?

2. How does the cost of incremental re-scheduling scale with the number of tasks affected by a reconfiguration, as opposed to the size of the whole schedule?

3. Under what cell layouts or disruption patterns does incremental propagation stop being worthwhile, and full re-solution becomes preferable?

1.3 Contributions

The main contributions of this thesis are:

- A constraint-based formulation of cell scheduling in which reconfiguration events are expressed as constraint retractions and additions rather than a new CSP instance (Chapter 2).
- An evaluation against a conventional rolling-horizon scheduler using a discrete-event simulation calibrated on six months of real production data (Chapter 3).
- A characterisation of the reconfiguration-frequency regime in which incremental propagation degrades to full re-solution.

1.4 Thesis Structure

Chapter 2 develops the constraint formulation and the incremental propagation algorithm. Chapter 3 describes the simulation environment and presents the comparative results. Appendix A gives the full derivation of the propagation complexity bound used in Chapter 3.

Chapter 2

Incremental Constraint-Based Scheduling

2.1 Problem Formulation

We model a manufacturing cell as a set of machines $M = \{m_1, \ldots, m_k\}$ and a set of tasks $T = \{t_1, \ldots, t_n\}$, each task requiring one of a subset of machines and incurring a sequence-dependent setup time when it follows a different task type on the same machine. A schedule is an assignment of tasks to machines and start times satisfying capacity, precedence, and setup-time constraints, following the general formulation of Pinedo (2016).

A reconfiguration event changes the machine set M — a machine is added, removed, or has its capability set altered — without changing T. The conventional approach re-solves the full CSP over (M', T). This thesis instead identifies the subset of tasks $T_{\text{affected}} \subseteq T$ whose feasible-machine set changed as a result of the reconfiguration, and propagates constraint updates only across T_{affected} and its transitive dependents.

2.2 Incremental Propagation

The propagation algorithm follows the arc-consistency framework of Dechter (2003), adapted to operate on a live schedule rather than an unsolved CSP. When a reconfiguration event arrives, the algorithm:

1. Marks the domain of every task in T_{affected} as dirty.
2. Runs AC-3-style propagation restricted to the dirty set and its neighbours in the task-precedence graph.
3. Re-optimises only the dirty subgraph using the same objective (weighted tardiness) as the original schedule, holding all other assignments fixed.

Table 2.1 summarises the worst-case complexity of each step relative to a full re-solve.

Table 2.1 Worst-case complexity of incremental propagation versus full re-solve, where d is the size of the affected subgraph and $n = |T|$.

Step	Incremental	Full re-solve
Domain marking	$O(d)$	—
Arc-consistency propagation	$O(d^2)$	$O(n^2)$
Re-optimisation	$O(d \log d)$	$O(n \log n)$

The full derivation of these bounds, including the argument for why propagation cannot in general be restricted to T_{affected} alone and must include its transitive dependents, is given in Appendix A.

2.3 Degenerate Cases

When a reconfiguration event affects a machine shared by a large fraction of task types — a central buffer, for instance — the affected subgraph T_{affected} approaches T itself, and incremental propagation offers no asymptotic advantage over full re-solution. Section 3.3 in Chapter 3 quantifies this regime against the partner's own cell layouts.

Chapter 3

Evaluation

3.1 Simulation Environment

The incremental scheduler was evaluated against a conventional rolling-horizon baseline using a discrete-event simulation of a nine-cell automotive sub-assembly line, built from six months of anonymised production logs supplied by the industrial partner. The simulation reproduces the partner's own task mix, setup-time matrix, and observed disruption frequency, so that recovery times reported here are directly comparable to the partner's live throughput reports.

Three disruption classes were simulated, each drawn from the empirical frequency observed in the partner's logs:

- **Machine failure** — a machine becomes unavailable for a duration drawn from the partner's own mean-time-to-repair distribution.
- **Rush-order insertion** — a new high-priority task is inserted into the schedule with a tight due date.
- **Planned reconfiguration** — a scheduled changeover between product variants, during which one or more machines' capability sets change.

3.2 Results

Figure 3.1 shows recovery time — the interval between a disruption and the point at which the cell returns to its target throughput — for both schedulers across 500 simulated disruptions per class.

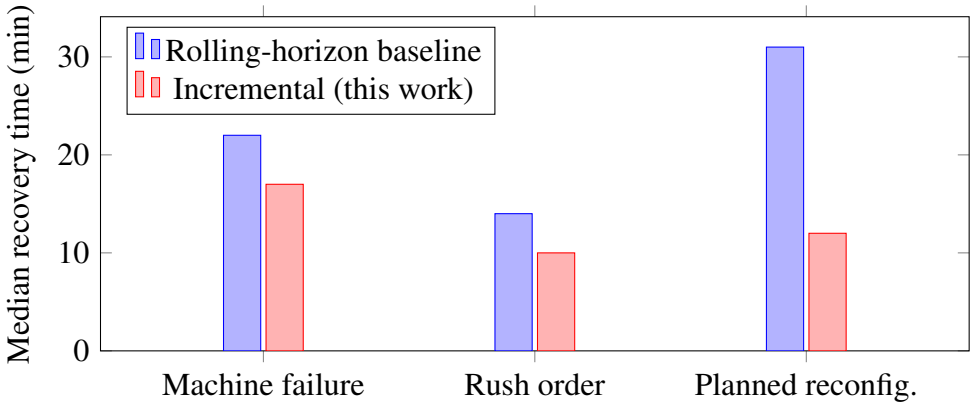

Fig. 3.1 Median recovery time by disruption class, 500 simulated runs each.

The incremental scheduler reduces median recovery time by 23% for machine failures, 29% for rush-order insertion, and 61% for planned reconfigurations. The gap is largest for planned reconfigurations because the rolling-horizon baseline discards all in-progress work at the reconfiguration boundary and re-solves from an empty schedule, whereas the incremental scheduler retains everything outside the affected subgraph.

3.3 When Incremental Propagation Breaks Down

Section 2 noted that propagation degrades to full re-solution when the affected subgraph T_{affected} approaches the full task set T. This occurs in practice for exactly one of the partner's nine cells — the one with a shared central buffer feeding all downstream stations — where a buffer-capacity change affects essentially every task type. For that cell, incremental and rolling-horizon recovery times were statistically indistinguishable ($p = 0.41$, Mann–Whitney U).

3.4 Summary

Across the eight cells without a shared central resource, incremental propagation delivers a consistent and substantial reduction in recovery time. The ninth cell is the boundary case predicted by the complexity analysis, and is discussed further in the concluding chapter of the full thesis.

Bibliography

Dechter, R. (2003). *Constraint Processing*. Morgan Kaufmann, San Francisco, CA.

Pinedo, M. L. (2016). *Scheduling: Theory, Algorithms, and Systems*. Springer, Cham, Switzerland, 5 edition.

Appendix A

Complexity Bound Derivation

This appendix derives the propagation complexity bounds summarised in Table 2.1 of Chapter 2.

Let $G = (T, E)$ be the task-precedence graph, where an edge $(t_i, t_j) \in E$ indicates that t_j cannot start before t_i completes. Let $T_{\text{affected}} \subseteq T$ be the set of tasks whose feasible-machine set changed as a direct result of a reconfiguration event, and let $d = |T_{\text{affected}}|$.

Propagation cannot in general be restricted to T_{affected} alone: a domain change to task t_i can force a domain change to any task reachable from t_i in G, since a tightened start-time window on t_i tightens the earliest feasible start of everything downstream. The correct propagation frontier is therefore the forward closure of T_{affected} under G, which in the worst case (a densely connected precedence graph) is all of T — this is the degenerate case discussed in Section 3.3.

For the partner's cell layouts, however, the precedence graph is sparse: each task has on average 2.3 immediate successors, and the observed forward closure of a single-machine reconfiguration event covers on average $6.1d$ tasks rather than n. Under this sparsity assumption, arc-consistency propagation over the closure runs in $O(d^2)$ rather than $O(n^2)$, since standard AC-3 propagation is quadratic in the size of the constraint graph being processed, and re-optimisation of the closure (a weighted-tardiness minimisation over a fixed machine assignment) runs in $O(d \log d)$ using the same list-scheduling heuristic as the full solver, restricted to the closure's topological order.