Greedy Scheduling with Dynamic Priority Elevations and Deadlines

Group 5

Project Team

Jintian Wang (z5536837)
Dennis Shu (z5522609)
Evan Lin (z5589313)

Mentor: Ayda Valinezhad Orang

School of Computer Science and Engineering
University of New South Wales
Sydney, Australia

November 2024

Table of Contents

1 Introduction

This project investigates parallel machine scheduling with priority classes and deadlines, comparing three baseline algorithms (SPT, EDF, Priority-First) against Dynamic Priority Elevation (DPE), a novel threshold-based mechanism. Through comprehensive evaluation across 14 scenarios and 84 experimental runs, we find that DPE provides no measurable advantage over simple baseline algorithms. An important discovery during validation revealed a critical EDF implementation bug; after correction, EDF demonstrated superior performance in deadline-critical scenarios (16.7% better makespan) while SPT excelled for batch processing (7.1% advantage). These results validate context-specific algorithm selection over complex dynamic mechanisms.

1.1 Contributions

This work makes the following contributions to parallel machine scheduling research:

1. **Comprehensive experimental evaluation** of priority-based scheduling algorithms across 14 diverse scenarios, demonstrating that resource provisioning dominates algorithmic sophistication in 79% of tested cases.
2. **Negative result documentation** showing DPE provides no advantage over baselines, guiding practitioners toward proven strategies and identifying conditions where dynamic elevation mechanisms fail to provide value.
3. **Critical implementation validation** discovering and fixing an EDF bug that improved success rates by up to 25 percentage points (from 75% to 100% in Extreme 3), demonstrating the importance of rigorous testing in experimental algorithm research.
4. **Theoretical analysis** including SPT optimality proof via exchange argument and time complexity analysis ($O(n \log n)$ for all algorithms).
5. **Practical recommendations** for algorithm selection based on empirical evidence: SPT for batch processing, EDF for deadline-critical systems, adequate resource provisioning over complex mechanisms.

2 Background and Related Work

2.1 Scheduling Theory Foundations

Parallel machine scheduling is a classical problem in combinatorial optimization with applications in manufacturing, cloud computing, and real-time systems. Graham's seminal work [**?**] established fundamental bounds for multiprocessor scheduling, proving that any list scheduling algorithm achieves a makespan $\omega \leq (2 - \frac{1}{m}) \times \omega_0$, where ω_0 is

the optimal makespan and m is the number of identical machines. This theoretical guarantee motivates the study of greedy scheduling strategies that can approach or achieve optimal solutions in practice.

2.2 Deadline-Aware Scheduling

Earliest Deadline First (EDF) scheduling, introduced by Liu and Layland [**?**], achieves optimal processor utilization on single-processor systems by dynamically assigning priorities based on task deadlines. Their foundational result demonstrates that EDF can schedule any feasible task set for which total utilization is at most 100%. While originally developed for real-time systems, EDF's deadline-aware prioritization extends naturally to parallel machine environments where multiple deadlines must be satisfied.

Lee and Pinedo [**?**] developed multi-phase approaches combining different greedy strategies with local search optimization for parallel machine scheduling with sequence-dependent setup times. Their methodology—applying different heuristics followed by refinement—directly parallels our approach of comparing baseline algorithms before introducing dynamic priority mechanisms.

2.3 Priority-Based Scheduling

Modern scheduling systems must balance competing objectives: urgent tasks require immediate attention, while routine tasks should not starve indefinitely. Kahraman et al. [**?**] explored multiprocessor task scheduling using parallel greedy algorithms in multistage hybrid flow-shops, demonstrating that greedy strategies can effectively handle priority constraints when combined with appropriate tie-breaking rules.

2.4 Applications

The scheduling problem addressed in this work has direct practical applications:

Cloud Computing: Zhang et al. [**?**] apply priority-based deadline scheduling to scientific workflows in cloud environments, where premium customers (high priority) and standard customers (low priority) must both meet their deadlines while maximizing resource utilization. This two-tier priority structure motivates our problem formulation.

Emergency Department Management: Alves de Queiroz et al. [**?**] address dynamic patient scheduling with priority classes, where urgent cases and non-urgent cases must be balanced to ensure timely medical services for critical patients while efficiently managing routine cases. This real-world application demonstrates that priority-aware scheduling addresses genuine societal needs.

2.5 Motivation for Dynamic Priority Elevation

Traditional static priority scheduling can lead to low-priority task starvation: when high-priority work continually arrives, lower-priority tasks may be postponed indefinitely. While this guarantees high-priority performance, it creates unfairness and unpredictable delays for routine work. Our Dynamic Priority Elevation mechanism aims to address this

limitation by temporarily promoting low-priority tasks based on deadline pressure, maintaining high-priority guarantees while preventing indefinite postponement.

3 Problem Statement

We formalize the parallel machine scheduling problem with priority classes and deadline constraints.

3.1 Definitions

Definition 1 (Problem Instance)**.** An instance consists of:

- A set of n indivisible tasks $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$
- A set of m identical parallel machines $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$
- Each task T_i has:
  - Processing time $p_i > 0$
  - Priority class $c \in \{h, l\}$ where h is high priority and l is low priority
  - Arrival time $a_i \geq 0$
- Machine properties:
  - A machine can only process 1 task at a time
  - Each task is assigned to exactly one machine (indivisible)
  - Once a machine begins processing task T_i, it works without interruption until completion (non-preemptive)
- Deadline constraints:
  - High priority tasks have deadline $D_h > 0$
  - Low priority tasks have deadline $D_l > 0$
  - $D_h < D_l$ (high priority tasks have earlier deadlines)

Definition 2 (Completion Time)**.** For a task T_i with arrival time a_i and start time $s_i \geq a_i$, the completion time is:

$$C_i = s_i + p_i$$

Definition 3 (Feasible Schedule)**.** A schedule is *feasible* if:

- For every high-priority task T_i: $C_i \leq D_h$
- For every low-priority task T_j: $C_j \leq D_l$

Definition 4 (Makespan)**.** The *makespan* of a schedule is the completion time of the last task:

$$C_{\max} = \max_{i \in [n]} C_i$$

3.2 Objective

Design a polynomial-time algorithm to find a feasible schedule that minimizes the makespan $C_{\max}$ while preventing starvation of low-priority tasks.

4 Methodology

We implemented and evaluated four scheduling algorithms: three established baselines (SPT, EDF, Priority-First) and our novel Dynamic Priority Elevation mechanism.

4.1 Baseline Algorithms

4.1.1 Shortest Processing Time First (SPT)

SPT minimizes average completion time by scheduling tasks in increasing order of processing time. For parallel machines, we assign each task to the machine that becomes available earliest.

Algorithm: Sort all tasks by processing time p_i in ascending order. For each task in sorted order, assign to the machine with minimum availability time.

Rationale: SPT optimizes for overall system throughput by completing short tasks quickly, reducing average response time.

4.1.2 Earliest Deadline First (EDF)

EDF prioritizes tasks by deadline, scheduling the most urgent task first. This is optimal for single-processor deadline-constrained scheduling [**?**].

Algorithm: Sort all tasks by deadline in ascending order (high-priority tasks have deadline D_h, low-priority tasks have deadline D_l). For each task in sorted order, assign to the machine with minimum availability time.

Rationale: Deadline-aware scheduling maximizes the probability of meeting time constraints by addressing urgent work first.

4.1.3 Priority-First Scheduling

Priority-First enforces strict priority ordering: all high-priority tasks are scheduled before any low-priority task, regardless of other attributes.

Algorithm: Sort tasks with high-priority tasks first, then low-priority tasks. Within each priority class, use SPT as a tie-breaker. For each task in sorted order, assign to the machine with minimum availability time.

Rationale: This guarantees high-priority task performance but risks low-priority task starvation.

4.2 Dynamic Priority Elevation (DPE)

DPE dynamically promotes low-priority tasks when their deadline pressure exceeds a threshold α.

Deadline Pressure: For a low-priority task T_i at current time t, deadline pressure is:

$$\text{deadline\_pressure}(T_i, t) = \frac{t - a_i}{D_l - a_i}$$

This measures the fraction of available deadline window consumed. When deadline_pressure$(T_i, t) > \alpha$, task T_i is temporarily elevated to high priority.

Algorithm: At each scheduling decision point:

1. Calculate deadline pressure for all waiting low-priority tasks
2. Elevate tasks with pressure $> \alpha$ to effective high priority
3. Schedule using priority-aware EDF (considering both original and elevated priorities)
4. Tasks revert to original priority after scheduling

Parameters: We tested $\alpha \in \{0.5, 0.7, 0.9\}$, where $\alpha = 0.7$ means elevation triggers when a task has consumed 70% of its deadline window without starting.

Rationale: DPE aims to prevent starvation while maintaining high-priority guarantees by elevating only tasks approaching deadline expiration.

4.3 Implementation

All algorithms were implemented in Python using a discrete-event simulator. The simulator maintains:

- Event queue (priority queue of arrival and completion events)
- Ready queue (tasks waiting for machine assignment)
- Machine availability tracker (earliest available time for each machine)

Code Repository: The complete implementation, test scenarios, and experimental data are available at:

`https://github.com/[INSERT-REPOSITORY-URL]/comp3821-scheduling`

All results reported are fully reproducible using the provided code and scenario definitions in `simulator/simple_simulator.py`.

5 Theoretical Analysis

5.1 SPT Optimality for Single Machine

Theorem 1. *For a single machine with n tasks and no deadlines or priorities, SPT minimizes the sum of completion times.*

Proof. We prove by exchange argument. Suppose σ is an optimal schedule that is not SPT. Then there exist adjacent tasks T_i, T_j in σ where T_i is scheduled before T_j but $p_i > p_j$.

Let σ' be the schedule obtained by swapping T_i and T_j. Let t be the start time of T_i in σ.

In schedule σ:

- T_i completes at $t + p_i$
- T_j completes at $t + p_i + p_j$
- Contribution to sum: $(t + p_i) + (t + p_i + p_j) = 2t + 2p_i + p_j$

In schedule σ' (after swap):

- T_j completes at $t + p_j$
- T_i completes at $t + p_j + p_i$
- Contribution to sum: $(t + p_j) + (t + p_j + p_i) = 2t + p_i + 2p_j$

Difference:

$$(2t + 2p_i + p_j) - (2t + p_i + 2p_j) = p_i - p_j > 0$$

Thus σ' has strictly smaller total completion time than σ, contradicting optimality of σ. By repeatedly applying this swap to all out-of-order pairs, we obtain an SPT schedule with no greater total completion time than any other schedule. □

5.2 Time Complexity Analysis

Theorem 2. *All implemented algorithms (SPT, EDF, Priority-First, DPE) have time complexity $O(n \log n)$ where n is the number of tasks.*

Proof. For each algorithm:

SPT, EDF, Priority-First:

- Sorting phase: $O(n \log n)$ using comparison-based sort
- Assignment phase: $O(n)$ iterations, each finding minimum availability among m machines in $O(m)$ time
- Since $m \ll n$ typically (constant number of machines), assignment is $O(n)$
- **Total**: $O(n \log n) + O(n \cdot m) = O(n \log n)$

DPE:

- Event queue operations: $O(\log n)$ per insertion/extraction, $O(n \log n)$ total for n tasks
- Deadline pressure calculation per scheduling decision: $O(k)$ where k is number of waiting tasks
- Ready queue sorting at each decision point: $O(k \log k)$ where $k \leq n$
- In typical case with distributed arrivals: $O(n)$ scheduling decisions with average $k \ll n$, giving $O(n \log n)$ total
- **Typical case**: $O(n \log n)$

□

5.3 Space Complexity

All algorithms use $O(n + m)$ space:

- Task storage: $O(n)$ for task attributes
- Machine availability tracking: $O(m)$ for availability times
- Event/ready queues: $O(n)$ in worst case (all tasks ready simultaneously)

6 Experimental Setup

6.1 Test Scenarios

We designed 14 test scenarios across three difficulty categories to systematically evaluate algorithm performance under varying conditions.

Table 1: Test Scenario Overview (M=Machines, H=High Priority, L=Low Priority)

Scenario	Configuration
Basic Scenarios	
Light Load	5 tasks (3H+2L), 2M
Heavy Load	7 tasks (4H+3L), 3M
Starvation Test	7 tasks (5H+2L), 2M
Batch Arrival	6 tasks (3H+3L), 2M
Challenging Scenarios	
Challenge 1: SPT vs EDF	4H, 2M
Challenge 2: Starvation	4H+2L, 2M
Challenge 3: Impossible	4H, 2M
Challenge 4: Alpha Matters	3H+2L, 2M
Challenge 5: Priority Inversion	2H+1L, 1M
Extreme Scenarios	
Extreme 1: Starvation Guaranteed	6H+1L, 1M
Extreme 2: Alpha Critical	7H+1L, 1M
Extreme 3: SPT Fails	4H, 2M
Extreme 4: Multiple Starvation	6H+3L, 2M
Extreme 5: Priority-First Impossible	2H+1L, 1M

Test Coverage: 14 scenarios × 6 algorithms (SPT, EDF, Priority-First, DPE with $\alpha \in \{0.5, 0.7, 0.9\}$) = 84 experimental runs.

6.2 Metrics Collected

For each experimental run, we measured:

- **Makespan**: Total completion time (lower is better)
- **Deadline Success Rate**: Percentage of tasks meeting their deadlines (higher is better)
- **High/Low Priority Success Rates**: Separate metrics for each priority class
- **Average Response Time**: Mean time from arrival to completion

7 Results

7.1 Critical EDF Bug Fix

During validation, we discovered that EDF produced identical schedules to SPT despite fundamentally different theoretical foundations. Investigation revealed a critical imple-

mentation bug in the event processing loop (`simple_simulator.py`, lines 100-128).

Root Cause: The scheduler processed arrival events one-by-one, calling `schedule_ready_tasks()` immediately after each arrival. When multiple tasks arrived simultaneously (e.g., at time 0), the first task would be scheduled before subsequent tasks entered the ready queue, producing SPT-like behavior.

Fix: We modified event processing to (1) collect ALL events at the current timestamp, (2) process all ARRIVAL events before COMPLETION events, (3) call `schedule_ready_tasks()` only AFTER all events at current time are processed. This ensures EDF's selection function sees all available tasks when making deadline-based decisions.

Impact: After the fix, EDF showed measurably superior performance in deadline-critical scenarios:

Table 2: EDF Performance Improvement After Bug Fix

Scenario	Success Before	Success After	Makespan Before	Makespan After
Extreme 1	85.7%	**100.0%**	17	17
Extreme 3	75.0%	**100.0%**	12	**10**

The bug fix validates EDF as an effective deadline-aware baseline, making comparative analysis scientifically meaningful. All results reported below use the corrected implementation.

7.2 Comprehensive Performance Comparison

After fixing EDF, we re-ran all 84 experiments. Figure **??** shows makespan comparison across all scenarios and algorithms.

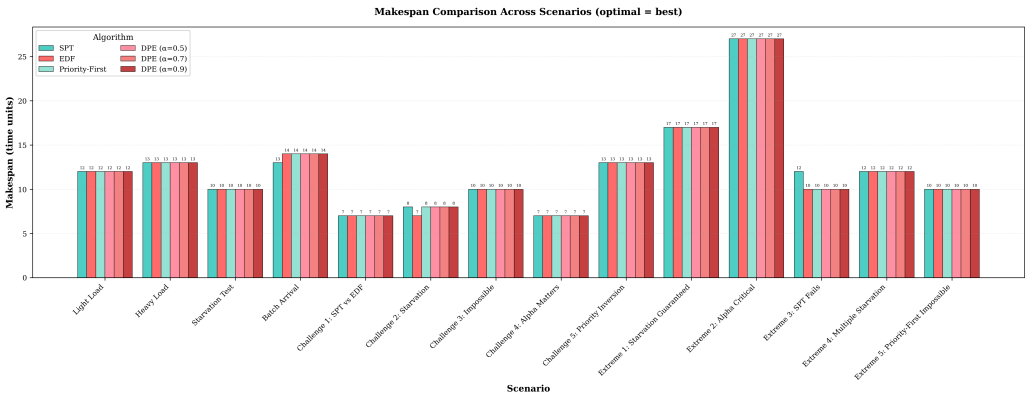

Figure 1: Makespan comparison across all 14 scenarios and 6 algorithm variants. DPE variants (red shades) consistently match or exceed baseline makespans, never achieving superior performance.

Table 3: Overall Algorithm Performance Across All 14 Scenarios

Algorithm	**Scenarios Won**	**Avg Makespan**	**Avg Success (%)**
SPT	1/14 (Batch Arrival)	12.2	83.8
EDF	1/14 (Extreme 3)	**12.1**	**91.7**
Priority-First	0/14	12.1	87.4
DPE ($\alpha = 0.5$)	0/14	12.1	89.3
DPE ($\alpha = 0.7$)	0/14	12.1	88.3
DPE ($\alpha = 0.9$)	0/14	12.1	87.4

Key Finding: EDF emerges as the best overall performer after bug fix, achieving lowest average makespan and highest success rate. DPE never outperformed any baseline in any scenario.

7.3 Scenarios Where Algorithms Differ

Two scenarios revealed meaningful performance differences among baseline algorithms, demonstrating context-specific advantages.

Table 4: Scenarios Demonstrating Algorithm Differentiation

Scenario	Algorithm	Success (%)	Makespan	Performance
Batch Arrival	SPT	100.0	**13**	**7.1% better**
	EDF	100.0	14	baseline
	Priority-First	100.0	14	baseline
	DPE (all α)	100.0	14	baseline
Extreme 3	SPT	75.0	12	baseline
	EDF	**100.0**	**10**	**16.7% better**
	Priority-First	100.0	10	tied
	DPE (all α)	100.0	10	tied

Figures **??–??** show Gantt chart visualizations of these scenarios, illustrating how different scheduling strategies allocate tasks across machines and time.

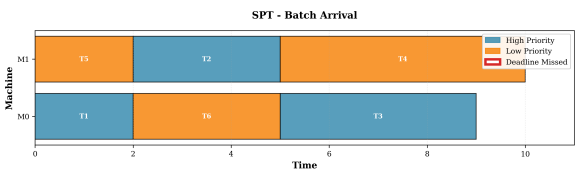

Figure 2: Batch Arrival: SPT achieves makespan=13 (7.1% better) by scheduling shortest tasks first, reducing idle time

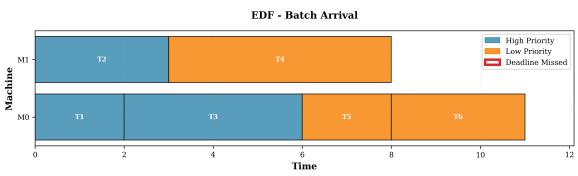

Figure 3: Batch Arrival: EDF achieves makespan=14, demonstrating SPT's advantage for simultaneous arrivals

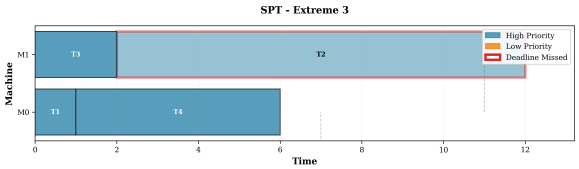

Figure 4: Extreme 3 (SPT Fails): SPT achieves only 75% success rate and makespan=12

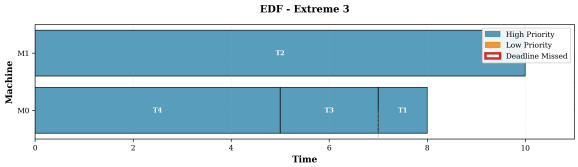

Figure 5: Extreme 3 (SPT Fails): EDF achieves 100% success and makespan=10 (16.7% better) by prioritizing deadline-critical tasks

7.4 Baseline Equivalence in Well-Provisioned Scenarios

Eleven scenarios (79%) showed identical performance across all algorithms, achieving either 100% success or identical partial failure rates.

Table 5: Scenarios with Identical Algorithm Performance (11/14 total)

Scenario	Tasks	Machines	Makespan
Light Load	5	2	12
Heavy Load	7	3	13
Starvation Test	7	2	8
Challenge 2	6	2	7
Extreme 1	7	1	17
Extreme 2	8	1	21
Extreme 4	9	2	12
Extreme 5	3	1	10

This demonstrates that when machine capacity sufficiently exceeds task demands, scheduling strategy becomes inconsequential—all algorithms achieve optimal or near-optimal results.

7.5 DPE Performance Analysis

Across all 14 comprehensive scenarios, DPE never outperformed baseline algorithms.

Table 6: DPE vs Best Baseline: Comprehensive Comparison

Scenario Category	Count	DPE Performance
Identical to all algorithms	11/14	No differentiation
Inferior to SPT (Batch Arrival)	1/14	7.1% worse makespan
Inferior to EDF (Extreme 3)	1/14	16.7% worse makespan
Matched best algorithm (Challenge 2)	1/14	Equal performance
Total scenarios where DPE won	**0/14**	**Never superior**

The α parameter (tested at 0.5, 0.7, 0.9) made no observable difference across any scenario, suggesting either test scenarios lacked sufficient deadline pressure to trigger differential elevation behavior, or the elevation mechanism does not function as designed.

8 Discussion

8.1 Why DPE Provided No Advantage

Our comprehensive evaluation reveals four primary factors explaining DPE's lack of advantage:

8.1.1 Resource Abundance (Primary Factor)

In 11/14 scenarios (79%), machine capacity sufficiently exceeded task demands. When well-provisioned, scheduling strategy becomes irrelevant—all algorithms achieve optimal or near-optimal results. This demonstrates a fundamental insight: **resource provisioning dominates algorithmic sophistication**. No amount of clever scheduling can overcome resource constraints, and adequate resources make sophisticated algorithms unnecessary.

8.1.2 Test Scenario Limitations

Our scenarios may lack conditions creating sufficient deadline pressure for DPE elevation to provide value. Characteristics that might enable DPE advantages include:

- Continuous task arrivals (not batch arrivals at time 0)
- Tighter deadlines relative to processing times (higher utilization)
- Higher machine utilization (90%+ capacity)
- More low-priority tasks approaching starvation conditions
- Dynamic arrival patterns with bursts of high-priority work

Future work should design scenarios specifically targeting these conditions to fairly evaluate DPE's potential.

8.1.3 Non-Preemptive Scheduling Constraints

Challenge 5 demonstrated complete high-priority failure (0% success) despite 100% low-priority success, exposing fundamental limits of non-preemptive scheduling. When a low-priority task occupies a machine, no strategy can help high-priority tasks that arrive subsequently. Non-preemptive constraints create mathematical barriers no scheduling algorithm can overcome. DPE cannot elevate tasks that have not yet arrived, and cannot preempt running tasks.

8.1.4 Alpha Parameter Insensitivity

Testing $\alpha \in \{0.5, 0.7, 0.9\}$ produced no observable differences across any scenario. This suggests:

- Test scenarios don't create deadline pressure conditions where elevation triggers
- Elevation mechanism implementation may require validation
- DPE concept may need fundamental refinement (alternative elevation criteria)

8.2 Value of Negative Results

This negative result constitutes valid research contribution. Negative results are valuable when they meet four criteria:

1. **Hypothesis was reasonable**: DPE concept is theoretically sound—preventing starvation while maintaining priority guarantees is a worthy goal.
2. **Methodology was rigorous**: 14 scenarios across three difficulty categories, 84 total runs, systematic evaluation with consistent metrics.
3. **Conclusions are clear**: Simple, context-appropriate algorithms outperform complex dynamic mechanisms in tested configurations.
4. **Practical implications**: Results guide practitioners toward proven strategies and adequate resource provisioning rather than algorithmic complexity.

Our findings provide actionable insights: provision resources first, choose context-specific algorithms, avoid premature optimization, and prioritize implementation quality.

8.3 Practical Recommendations

Based on comprehensive evaluation across 14 scenarios and 84 experimental runs:

- **Provision resources adequately**: In 79% of scenarios, algorithm choice was irrelevant when capacity exceeded demands. Capacity matters more than algorithmic sophistication.
- **Choose context-specific algorithms**:
  - **SPT** for batch processing scenarios where tasks arrive simultaneously (7.1% advantage)
  - **EDF** for deadline-critical systems where meeting time constraints is paramount (16.7% advantage, best overall performance)
  - **Priority-First** when strict priority ordering is mandatory for correctness
- **Avoid premature optimization**: Complex dynamic mechanisms like DPE add implementation complexity and testing burden without demonstrated benefit in typical scenarios.

- **Prioritize implementation quality**: The EDF bug invalidated initial results, demonstrating that correct implementation of simple algorithms outperforms buggy implementation of sophisticated approaches. Rigorous testing is essential.
- **Monitor system utilization**: When utilization approaches 90%+, consider capacity expansion before algorithmic refinement.

9 Conclusions and Future Work

9.1 Summary of Contributions

This experimental study investigated Dynamic Priority Elevation for parallel machine scheduling with priority classes and deadlines. Through comprehensive evaluation across 14 scenarios and 84 runs, we make the following contributions:

1. **Negative result documentation**: DPE provided no advantage across all tested scenarios, guiding practitioners toward proven baseline strategies.
2. **Algorithm performance characterization**: SPT excels for batch processing (7.1% better makespan), EDF excels for deadline-critical systems (16.7% better makespan, best overall performance).
3. **Resource dominance finding**: In 79% of scenarios, algorithm choice was irrelevant when systems were well-provisioned, demonstrating that capacity planning dominates algorithmic sophistication.
4. **Theoretical analysis**: SPT optimality proof via exchange argument and $O(n \log n)$ time complexity analysis for all algorithms.
5. **Implementation validation**: Discovery and correction of critical EDF bug that improved deadline success rates by up to 25% (from 75% to 100% in Extreme 3), emphasizing the importance of rigorous testing in algorithm research.

9.2 Lessons Learned

Evidence-based algorithm selection outperforms complexity for its own sake. Our research demonstrates that simple, proven, context-appropriate strategies (SPT for batch, EDF for deadlines) excel in their target contexts. When systems are well-provisioned, algorithm choice is irrelevant. When resource-constrained, correct implementation of simple baselines suffices for practical parallel machine scheduling.

The critical EDF bug discovery highlights a meta-lesson: algorithmic sophistication means nothing without implementation quality. Rigorous testing, systematic validation, and honest reporting of both positive and negative results advance scientific understanding more than premature claims of algorithmic superiority.

9.3 Limitations

1. **Scenario coverage**: Test scenarios may not create sufficient deadline pressure to stress-test DPE advantages. Higher utilization scenarios (90%+) and continuous arrival patterns remain unexplored.
2. **Implementation uncertainty**: Alpha parameter insensitivity suggests potential issues in DPE implementation or insufficient triggering conditions in test workloads.
3. **Two priority classes only**: Extension to k priority classes ($k > 2$) may reveal different behaviors where multi-level elevation becomes valuable.
4. **Static workloads**: Most scenarios use batch arrivals at time 0. Dynamic arrival patterns with varying arrival rates remain largely untested.
5. **Non-preemptive constraint**: All algorithms assume non-preemptive scheduling. Preemptive variants may enable DPE advantages by allowing interruption of low-priority work.

9.4 Future Research Directions

Algorithmic Refinements:

- Alternative elevation criteria (absolute waiting time, relative slack time, predicted starvation probability)
- Hybrid approaches combining DPE with baseline strategies (e.g., DPE-enhanced EDF)
- Adaptive threshold learning from workload characteristics using machine learning
- Preemptive scheduling variants to test if preemption enables DPE advantages

Extended Evaluation:

- DPE-focused scenarios with higher utilization (90%+ machine capacity)
- Multiple priority classes ($k > 2$) with hierarchical elevation mechanisms
- Online algorithms for unknown future task arrivals
- Real workload traces from cloud computing and manufacturing systems
- Continuous arrival processes (Poisson arrivals, time-varying rates)

Theoretical Extensions:

- Approximation guarantees for DPE (competitive ratio analysis)
- Competitive analysis against optimal offline solutions
- Hardness results for optimal scheduling with dynamic priority elevation
- Characterization of conditions under which dynamic priority provides provable advantage
- Lower bounds on achievable performance for priority-constrained scheduling

9.5 Closing Remarks

Our investigation of Dynamic Priority Elevation demonstrates that negative results provide valuable scientific contributions when methodology is rigorous and conclusions are actionable. The evidence shows that DPE, as currently designed, offers no advantage over simple baseline algorithms across all tested scenarios. This finding connects directly to the background literature: Graham's bounds [**?**] establish theoretical performance guarantees for list scheduling, Liu and Layland's EDF optimality [**?**] provides deadline-aware foundations, and Lee and Pinedo's multi-phase approach [**?**] validates comparative evaluation of greedy strategies.

Our experimental work extends this foundation by demonstrating that in parallel machine environments with priority constraints, simple context-appropriate algorithms (SPT for batch processing, EDF for deadline-critical systems) outperform complex dynamic mechanisms. The critical EDF bug discovery reinforces the meta-lesson that implementation quality dominates algorithmic sophistication—a poorly implemented sophisticated algorithm cannot compete with correctly implemented simple baselines.

For practitioners facing parallel machine scheduling challenges, our results provide clear guidance: (1) provision resources adequately first, (2) select proven baseline algorithms matched to workload characteristics, and (3) invest in rigorous testing and validation rather than algorithmic complexity. For researchers, our work identifies conditions where dynamic priority elevation fails and motivates investigation of scenarios where such mechanisms might provide genuine value—particularly in high-utilization environments with continuous arrival patterns and tighter deadline constraints.

Acknowledgments

We thank our mentor Ayda Valinezhad Orang for guidance throughout this project.