Greedy Scheduling with Dynamic Priority Elevations and Deadlines

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

Table of Contents

1 Introduction **2**

2 Project Proposal **2**
2.1 Survey . . . 2
2.1.1 Motivations . . . 2
2.1.2 Real-world Applications . . . 2
2.1.3 Known Results . . . 2
2.1.4 Special Cases . . . 3
2.1.5 Preliminary Direction . . . 3
2.1.6 Our Contribution . . . 3
2.2 Research Plan . . . 3
2.2.1 Aims . . . 3
2.2.2 Plan . . . 4

3 Problem Statement **5**
3.1 Definitions . . . 5
3.2 DPE Definition . . . 6
3.3 Objective . . . 6

4 Progress Report **6**
4.1 Achievements . . . 6
4.1.1 Test Scenarios Overview . . . 6
4.1.2 Algorithm Performance Differentiation . . . 7
4.1.3 Baseline Performance: Algorithm Equivalence . . . 8
4.1.4 Partial Failure Cases . . . 8
4.1.5 Critical Failure: Priority Inversion . . . 8
4.1.6 Implementation Validation . . . 9
4.1.7 DPE Performance Analysis . . . 10
4.1.8 Key Findings . . . 10
4.1.9 Conclusion . . . 10
4.2 Revised Plan . . . 11

1 Introduction

We study parallel machine scheduling with priority classes and propose Dynamic Priority Elevation (DPE), a novel threshold-based mechanism that prevents low-priority task starvation while maintaining high-priority deadline guarantees. Unlike traditional static priority scheduling, DPE dynamically promotes tasks based on deadline pressure, addressing a key limitation in parallel machine environments where priority inversion can indefinitely delay lower-priority work.

2 Project Proposal

2.1 Survey

2.1.1 Motivations

Our interest in this scheduling problem stems from its direct applications in manufacturing and task allocation systems, where production lines must balance urgent orders with regular manufacturing schedules. We were particularly inspired by the Week 2 workshop problems on greedy scheduling algorithms, which introduced us to the theoretical framework of optimal ordering strategies. These exercises demonstrated how elegant greedy approaches could solve complex scheduling scenarios, motivating us to explore whether similar techniques could handle the additional complexity of priority classes and parallel machines.

2.1.2 Real-world Applications

The scheduling algorithm can be applied to cloud computing by using two deadlines: high-priority tasks (premium customers), and low-priority tasks (standard customers), ensuring both groups meet their deadlines [6]. This algorithm can also be applied to emergency department patient management with two priority classes: high-priority patients (urgent cases) and low-priority patients (non-urgent cases). This approach addresses emergency department overcrowding while ensuring timely medical services for critical patients and efficient management of routine cases[1].

2.1.3 Known Results

- Graham's study shows that any 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. Therefore, we expect our algorithm should achieve better than the optimal $2 \times \omega_0$ makespan[2].
- Liu and Layland's work proved that Earliest Deadline First (EDF) scheduling achieves optimal processor utilization on single-processor systems by dynamically assigning priorities based on task deadlines[5].

- Lee and Pinedo's work develops a multi-phase approach of using different greedy strategies followed by local search optimization, which directly parallels our proposed methodology for handling complex parallel machine scheduling with priority constraints[4].

2.1.4 Special Cases

- **Case 1: Equal processing times** ($p_i = p$ for all tasks i)
  - Reduces to bin packing with priority deadline constraints
  - Enables polynomial-time optimal solutions for small instances
  - Simplifies analysis of DPE threshold effects
- **Case 2: Single machines** ($m = 1$)
  - Serves as the most fundamental and achievable case for this problem
- **Case 3: Single priority class**
  - Equivalent to classical multiprocessor scheduling without priorities
  - Validates our base algorithms against well-known benchmarks
  - DPE becomes inactive (no priority elevation needed)

2.1.5 Preliminary Direction

We plan to develop a multi-phase greedy scheduling framework that systematically explores different priority-based strategies. Our approach will generate and evaluate multiple greedy solutions using various rules (such as **shortest processing time first(STF)**, **earliest deadline first(EDF)**, and load balancing strategies) applied within each priority class[3].

2.1.6 Our Contribution

Our main innovation is the Dynamic Priority Elevation (DPE) mechanism that dynamically adjusts task priorities during execution based on deadline pressure, unlike existing scheduling algorithms that use static priority assignments throughout. While traditional approaches like EDF and fixed-priority scheduling can lead to low-priority task starvation in parallel machine environments, our threshold-based elevation system prevents indefinite postponement while maintaining high-priority task guarantees.

2.2 Research Plan

2.2.1 Aims

Primary Focus: Implement established algorithms first, and then add DPE innovation as the experimental twist.

2.2.2 Plan

1. Foundation (Week 3 - 4)
    - Finalize the problem definitions to make the problem statement concise and accurate
    - Create comprehensive test suite with 20+ examples covering: single machine, multiple machine, all high-priority, all low-priority edge cases, and general cases (from simple to complex)
    - Research and design specifications for 3 baseline algorithms: SPT (Shortest Processing Time), EDF (Earliest Deadline First), and Priority-First scheduling
2. Algorithm Design I (Week 5 - 6)
    - **Implement 3 established greedy algorithms** for single machine case:
        - SPT (Shortest Processing Time First)
        - EDF (Earliest Deadline First)
        - Static Priority-First scheduling
    - Develop infeasibility detection method for all algorithms
    - Test and validate all baseline algorithms on example suite
    - **Week 6 Goal:** Working baseline implementations with performance benchmarks
3. Proof and Analysis (Week 7)
    - Prove correctness for the best-performing baseline algorithm
    - Analyze time complexity for implemented algorithms
    - Establish baseline performance metrics (makespan, deadline satisfaction, fairness)
4. Algorithm Design II (Week 8 - 9)
    - **Implement DPE (Dynamic Priority Elevation) as innovative twist** on best baseline algorithm·
    - Experimental comparison: DPE-enhanced vs. all 3 baseline approaches
    - Measure and analyze when/why DPE outperforms or underperforms baselines
    - Document scenarios where innovation provides measurable improvement
5. Final documentation (Week 10) By week 10, we will have the technical report and poster presenting our comparative study of baseline algorithms vs. DPE innovation. The deliverables will include experimental results, performance analysis, and lessons learned about the effectiveness of our innovative twist.

3 Problem Statement

We consider the problem of scheduling tasks with different priority classes and deadline constraints on identical parallel machines to minimize total completion time while ensuring all deadline requirements are satisfied.

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
- Machine properties:
  - A machine can only process 1 task at a time, but can process arbitrarily many tasks sequentially
  - Each task is assigned to exactly one machine (indivisible - no task splitting)
  - Once the machine begins to process a task T_i, it works without interruption until completing the task.
- Deadline constraints
  - All tasks T_i have deadlines $D_h > 0$ or $D_l > 0$
  - $D_h < D_l$, where 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 time when all tasks have finished, which is equivalent to the the completion time of the last task:

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

3.2 DPE Definition

We introduce **Dynamic Priority Elevation (DPE)** for parallel machine scheduling:

A low-priority task T_i gets temporarily elevated to high-priority when its deadline pressure exceeds threshold α:

$$\text{deadline\_pressure}(T_i) = \frac{\text{current\_time} - \text{arrival\_time}(T_i)}{D_i - \text{arrival\_time}(T_i)} > \alpha$$

where $\alpha = 0.7$ (task has consumed 70% of its deadline window).

This means: elevate low-priority tasks when they've used up 70% of their allowed time but haven't started processing yet.

3.3 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 Progress Report

Summary: We conducted comprehensive experimental evaluation comparing DPE against three baseline algorithms (SPT, EDF, Priority-First) across 14 diverse scenarios. Our results demonstrate that DPE provided no measurable advantage over baseline approaches in any tested configuration. While this negative result may indicate limitations in the current DPE implementation or insufficient stress in our test scenarios, the evidence suggests that simple, context-appropriate baseline algorithms outperform complex dynamic priority mechanisms. Future work should either refine the DPE design or identify specific conditions where dynamic elevation provides value; otherwise, practitioners should favor proven baseline strategies.

4.1 Achievements

4.1.1 Test Scenarios Overview

We designed 14 test scenarios across three categories to systematically evaluate algorithm performance. Table 1 summarizes each scenario; detailed specifications are provided in Appendix **??**.

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 $\times$ 6 algorithms (SPT, EDF, Priority-First, DPE with $\alpha \in \{0.5, 0.7, 0.9\}$) = 84 experimental runs.

4.1.2 Algorithm Performance Differentiation

Two scenarios revealed meaningful performance differences among baseline algorithms, demonstrating context-specific advantages. SPT excels when tasks arrive simultaneously by minimizing average completion time through shortest processing time first scheduling, achieving 7.7% better makespan. EDF outperforms when deadline constraints are primary concerns, achieving 14.3% superior makespan through deadline-aware prioritization. DPE provided no advantage in either scenario.

Table 2: Scenarios Where Algorithms Differ

Scenario	Algorithm	Success (%)	Makespan	Improvement
Batch Arrival	SPT	100.0	**13**	**7.7% better**
Batch Arrival	EDF	100.0	14	baseline
Batch Arrival	Priority-First	100.0	14	baseline
Batch Arrival	DPE (all α)	100.0	14	baseline
Challenge 4	SPT	100.0	7	baseline
Challenge 4	EDF	100.0	**6**	**14.3% better**
Challenge 4	Priority-First	100.0	7	baseline
Challenge 4	DPE (all α)	100.0	7	baseline

4.1.3 Baseline Performance: Algorithm Equivalence

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

Table 3: Scenarios with 100% Success and Identical Performance

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: Starvation Guaranteed	7	1	17
Extreme 2: Alpha Critical	8	1	21
Extreme 4: Multiple Starvation	9	2	12
Extreme 5: Priority-First Impossible	3	1	10

This indicates that when machine capacity sufficiently exceeds task demands, scheduling strategy becomes inconsequential—all algorithms achieve optimal or identical results. This demonstrates that problem structure and resource availability dominate algorithm choice in determining feasibility.

4.1.4 Partial Failure Cases

Three scenarios revealed partial deadline satisfaction where all algorithms performed identically, indicating inherent infeasibility.

Finding: All algorithms failed identically in these scenarios, demonstrating that when deadlines are too tight relative to processing requirements and machine capacity, no scheduling strategy can achieve complete success regardless of sophistication.

4.1.5 Critical Failure: Priority Inversion

Challenge 5 demonstrated the most severe failure case where all algorithms achieved 0% high-priority success.

Table 4: Partial Failure Scenarios - Identical Algorithm Performance

Scenario	High Tasks	High Success (%)	Makespan	Interpretation
Challenge 1: SPT vs EDF	4	75.0	7	Partial feasibility
Challenge 3: Impossible	4	50.0	10	Severe constraints
Extreme 3: SPT Fails	4	75.0	12	Resource-time limits

Table 5: Challenge 5: Priority Inversion - Complete High-Priority Failure

Algorithm	High Success (%)	Low Success (%)	Makespan	Avg Response
SPT	0.0	100.0	13	8.67
EDF	0.0	100.0	13	9.0
Priority-First	0.0	100.0	13	9.0
DPE (all α)	0.0	100.0	13	9.0

Analysis: Zero high-priority success despite 100% low-priority success exposes fundamental resource-time constraints in non-preemptive scheduling. When a low-priority task (processing=8) occupies the single machine at time 0, urgent high-priority tasks arriving at times 2 and 3 with deadlines 6 and 8 cannot be scheduled in time. All algorithms including DPE performed identically, confirming that non-preemptive constraints create mathematical limits no scheduling strategy can overcome.

4.1.6 Implementation Validation

Visual demonstrations revealed unexpected behaviors requiring investigation.

Table 6: Demo 1: SPT vs EDF - Expected vs Actual Behavior

Algorithm	Task	Completion	Deadline	Result
SPT (Actual)	A	2	50	Met
	B	12	11	**Missed**
EDF (Actual)	A	2	50	Met
	B	12	11	**Missed**
EDF (Expected)	B	10	11	Should meet
	A	12	50	Should meet

Finding: EDF produced an identical schedule to SPT (A→B) instead of the expected deadline-prioritized schedule (B→A). This discrepancy indicates a potential implementation issue where EDF's deadline-based prioritization logic may not be functioning correctly. The expected behavior would prioritize task B (deadline=11) before task A (deadline=50), allowing B to complete at time 10 and meet its deadline, followed by A completing at time 12 (well within deadline 50).

4.1.7 DPE Performance Analysis

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

Table 7: 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	7.7% worse makespan
Inferior to EDF (Challenge 4)	1/14	14.3% worse makespan
Matched best algorithm (Challenge 2)	1/14	Equal performance
Total scenarios where DPE won	**0/14**	**Never superior**

Analysis: DPE's threshold-based elevation mechanism provided no measurable advantage across any test scenario. In scenarios with adequate resources (11/14 cases), algorithm choice was irrelevant and all approaches succeeded identically. In the two scenarios where performance differentiated (Batch Arrival and Challenge 4), DPE matched the inferior algorithms rather than matching or exceeding the best performer. The α parameter (tested at 0.5, 0.7, 0.9) made no observable difference across any scenario, suggesting either: (a) test scenarios lacked sufficient deadline pressure to trigger differential elevation behavior, or (b) the elevation mechanism does not function as theoretically designed.

4.1.8 Key Findings

1. **DPE Provides No Advantage:** Across 14 scenarios, DPE never outperformed baseline algorithms, challenging the hypothesis that dynamic priority elevation improves upon static strategies.
2. **Specific Selection** SPT excels in batch processing (7.7% better), EDF in deadline-critical cases (14.3% better). Simple, targeted algorithms outperform universal mechanisms.
3. **Resources Matters** In 79% of scenarios (11/14), all algorithms performed identically. When well-provisioned or infeasible, algorithmic choice is irrelevant.
4. **Implementation Issues Discovered:** EDF behaved identically to SPT despite different theoretical foundations, indicating implementation bugs that may invalidate comparative results.
5. **Alpha Parameter Has No Effect:** Testing $\alpha \in \{0.5, 0.7, 0.9\}$ produced no measurable differences, suggesting the elevation mechanism is non-functional.

4.1.9 Conclusion

Our comprehensive experimental evaluation across 14 scenarios and 84 runs demonstrates that simple, context-appropriate baseline algorithms (SPT for batch processing, EDF for deadline-critical systems) outperform the complex DPE mechanism. When systems are well-provisioned, algorithm choice is irrelevant; when resource-constrained,

simple algorithms excel in their target contexts. The evidence suggests that resource provisioning and correct implementation quality matter more than algorithmic sophistication. Dynamic priority elevation, as currently designed and tested, provides no measurable advantage over static scheduling strategies in parallel machine environments.

4.2 Revised Plan

Week 8 (November 2-8):

- Fix EDF implementation bug (currently behaves identically to SPT)
- Investigate why DPE shows no improvement:
  - Analyze DPE elevation trigger conditions in existing scenarios
  - Add logging to track when/if priority elevations occur
  - Verify DPE implementation against algorithm specification
- Design 5+ new scenarios specifically targeting DPE advantages:
  - High deadline pressure scenarios where low-priority tasks approach starvation
  - Scenarios with continuous task arrivals (not batch)
  - Cases where static priority causes demonstrable low-priority delays

Week 9 (November 9-15):

- Re-run all experiments with corrected EDF
- Test new DPE-focused scenarios with multiple α values ($\alpha \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$)
- Refine DPE mechanism if initial design shows fundamental flaws:
  - Consider alternative elevation criteria (e.g., absolute waiting time, relative slack time)
  - Test hybrid approaches combining DPE with baseline strategies
- Prove correctness for SPT (single-machine case)
- Analyze time complexity: all algorithms are $O(n \log n)$ due to sorting
- Begin technical report writing

Week 10 (November 16-22):

- Complete technical report with all experimental results
- Create poster highlighting comparative performance
- Finalize all documentation and code

References

[1] Thiago Alves de Queiroz, Manuel Iori, Arthur Kramer, and Yong-Hong Kuo. Dynamic scheduling of patients in emergency departments. *European Journal of Operational Research*, 310(1):100–116, 2023.

[2] R. L. Graham. Bounds for certain multiprocessing anomalies. *The Bell System Technical Journal*, 45(9):1563–1581, 1966.

[3] Cengiz Kahraman, Orhan Engin, İhsan Kaya, and R. Elif Öztürk. Multiprocessor task scheduling in multistage hybrid flow-shops: A parallel greedy algorithm approach. *Applied Soft Computing*, 10(4):1293–1300, 2010. Optimisation Methods Applications in Decision-Making Processes.

[4] Young Hoon Lee and Michael Pinedo. Scheduling jobs on parallel machines with sequence-dependent setup times. *European Journal of Operational Research*, 100(3):464–474, 1997.

[5] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, January 1973.

[6] Longxin Zhang, Liqian Zhou, and Ahmad Salah. Efficient scientific workflow scheduling for deadline-constrained parallel tasks in cloud computing environments. *Information Sciences*, 531:31–46, 2020.