Online Scheduling with Predictions

TIANMING ZHAO

Doctor of Philosophy

Supervisor: Professor Albert Y. Zomaya
Associate Supervisor: Doctor Wei Li

A thesis submitted in fulfilment of
the requirements for the degree of
Doctor of Philosophy

School of Computer Science
Faculty of Engineering
The University of Sydney
Australia

3 August 2023

Authorship Attribution Statement

Chapter 3 of this thesis is published as [1] and [2].
I initiated the study, designed the algorithms, conducted the analysis and experiments, and wrote the drafts of the manuscripts.

Chapter 4 of this thesis is published as [3].
I initiated the study, designed the algorithms, conducted the analysis, and wrote the drafts of the manuscript.

Chapter 5 of this thesis is published as [4].
I initiated the study, designed the algorithms, conducted the analysis and experiments, and wrote the drafts of the manuscript.

Chapter 6 of this thesis is published as [5].
I designed the algorithms, conducted the analysis and experiments, and wrote the drafts of the manuscript.

In addition to the statements above, in cases where I am not the corresponding author of a published item, permission to include the published material has been granted by the corresponding author.

Student Name: Tianming Zhao
Signature:
Date:

As supervisor for the candidature upon which this thesis is based, I can confirm that the authorship attribution statements above are correct.

Supervisor Name: Albert Y. Zomaya
Signature:
Date:

Publications

[1] T. Zhao, W. Li, and A. Y. Zomaya, “Uniform machine scheduling with predictions,” *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 32, pp. 413–422, Jun. 2022.

[2] T. Zhao, W. Li, and A. Y. Zomaya, “Learning-augmented scheduling,” *IEEE Transactions on Computers*, 2023. Submitted.

[3] T. Zhao, C. Li, W. Li, and A. Y. Zomaya, “Brief announcement: Towards a more robust algorithm for flow time scheduling with predictions,” in *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA ’22, (New York, NY, USA), p. 385–388, Association for Computing Machinery, 2022.

[4] T. Zhao, W. Li, and A. Y. Zomaya, “Real-time scheduling with predictions,” in *2022 IEEE Real-Time Systems Symposium (RTSS)*, pp. 331–343, 2022.

[5] T. Zhao, W. Li, B. Qin, L. Wang, and A. Y. Zomaya, “Pulsed power load coordination in mission and time critical cyber-physical systems,” *ACM Trans. Model. Perform. Eval. Comput. Syst.*, Dec. 2022.

Publications not included in the thesis:

[6] T. Zhao, W. Si, W. Li, and A. Y. Zomaya, “Optimizing the maximum vertex coverage attacks under knapsack constraint,” *IEEE/ACM Transactions on Networking*, vol. 29, no. 3, pp. 1088–1104, 2021.

[7] T. Zhao, W. Si, W. Li, and A. Y. Zomaya, “Towards minimizing the r metric for measuring network robustness,” *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 4, pp. 3290–3302, 2021.

Statement of Originality

This is to certify that to the best of my knowledge, the content of this thesis is my own work. This thesis has not been submitted for any degree or other purposes.

I certify that the intellectual content of this thesis is the product of my own work and that all the assistance received in preparing this thesis and sources have been acknowledged.

Name: Tianming Zhao

Signature: **Date**:

Student Plagiarism: Compliance Statement

I certify that:

I have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure;

I understand that failure to comply with the Student Plagiarism: Coursework Policy and Procedure can lead to the University commencing proceedings against me for potential student misconduct under Chapter 8 of the University of Sydney By-Law 1999 (as amended);

This Work is substantially my own, and to the extent that any part of this Work is not my own I have indicated that it is not my own by Acknowledging the Source of that part or those parts of the Work.

Name: Tianming Zhao

Signature: **Date**:

Abstract

Online scheduling is the process of allocating resources to tasks to achieve objectives with uncertain information about future conditions or task characteristics. This thesis presents a new online scheduling framework named online scheduling with predictions. The framework uses predictions about unknowns to manage uncertainty in decision-making. It considers that the predictions may be imperfect and include errors, surpassing the traditional assumptions of either complete information in online clairvoyant scheduling or zero information in online non-clairvoyant scheduling. The goal is to create algorithms with predictions that perform better with quality predictions while having bounded performance with poor predictions. The framework includes metrics such as consistency, robustness, and smoothness to evaluate algorithm performance. We prove the fundamental theorems that give tight lower bounds for these metrics. We apply the framework to central scheduling problems and cyber-physical system applications, including minimizing makespan in uniform machine scheduling with job size predictions, minimizing mean response time in single and parallel identical machine scheduling with job size predictions, and maximizing energy output in pulsed power load scheduling with normal load predictions. Analysis and simulations show that this framework outperforms state-of-the-art methods by leveraging predictions.

Acknowledgements

I would like to express my gratitude to my supervisor, Professor Albert Y. Zomaya, and Dr. Wei Li, for their supervision, support, and encouragement throughout my PhD journey. They have created an extraordinary academic environment for my research and provided me with all the necessary support along the way. Without them, the research outcomes would not have been possible. I would also like to extend my thanks to Dr. Weisheng Si for his supervision, support, and encouragement in my early academic career. He taught me valuable approaches to good academic writing that I will always remember and benefit from.

I am grateful to my family. My parents, Lei Zhao and Yin Zhao, provided me with a childhood filled with freedom and support for my interests. My grandfather, Zhengxue Zhao, taught me that the secret to successful learning is thinking. My grandmother, Shuying Sun, instilled in me a love for math. My uncle and aunt, Yuan Zhao and Yan Jiang, hosted me during my university studies.

Last but certainly not least, I would like to express my deepest appreciation to my wife, Min Li, for being my unwavering source of support and love throughout this journey. You have been my rock. You have been my cornerstone. You mean the world to me.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Scheduling is the process of allocating resources to tasks to achieve goals. Examples include matching CPUs to run programs, energy to perform operations, and human resources to projects. This thesis focuses on scheduling in computer systems where resources are machines and tasks are jobs, as well as scheduling in cyber-physical systems. Scheduling is critical to computer systems as it effectively matches the processing units and the computational tasks, ensuring low latency and high quality of service. The goal is that the right resources are allocated to the right tasks at the right time to optimize objectives. Quality scheduling is inevitable due to the constant conflict between limited resources and demanding tasks.

The challenges of scheduling involve decision-making under uncertainty, also known as online scheduling. In online scheduling, we often do not have complete information about certain parameters or the future when making decisions. Specifically, we do not know the existence of a future job until it arrives in the system, and sometimes we do not know the size of a job (i.e., the processing time) until it is completed. Under these uncertainties, we aim to optimize specific objectives, such as makespan and response time, and ensure that the schedule is effective even if the unknowns work against us. Two approaches to this goal have been taken in the literature. One, called *online clairvoyant scheduling*, assumes that we have all the information about a job upon its arrival. The other, called *online non-clairvoyant scheduling*, assumes that we do not know the job size until its completion. The former approach is optimistic, and the latter is pessimistic.

Both approaches have issues. Online clairvoyant scheduling can guarantee excellent performance, but it is too optimistic to assume known job sizes. Online non-clairvoyant scheduling may have low competitiveness, but it is too pessimistic to assume no information. With the increasing power of learning models and forecast algorithms, more accurate job size predictions are possible. Although they may not be entirely accurate, predictions can help us achieve effective scheduling. Therefore, the assumptions in online scheduling should be re-examined, and a new framework is required. This motivates the thesis, which develops a new framework called *online scheduling with predictions*.

The proposed framework assumes that instead of having complete or no information, we can access some predictions of unknowns, which are assumed to be (possibly) imperfect. This is a compromise and an extension of online clairvoyant scheduling and online non-clairvoyant scheduling. Under this framework, the goal is to develop algorithms that use predictions and have performance depending on prediction quality. We want to achieve near-optimal performance under good predictions and bounded performance under bad predictions. This thesis first formulates the framework of online scheduling with predictions and presents the performance metrics used in this framework and the fundamental results for these metrics. It then applies the framework to solve several central scheduling problems and concludes the technical chapters with an application to cyber-physical systems scheduling. The technical chapters make the following contributions.

In Chapter 2, we formulate the framework of online scheduling with predictions. We present the performance evaluation metrics, *consistency*, *robustness*, and *smoothness*, and prove the fundamental results on them. These results provide universal lower bounds for these metrics, prove the existence of algorithms that achieve these bounds, and link online scheduling with predictions to online clairvoyant scheduling and online non-clairvoyant scheduling.

In Chapter 3, we solve the problem of uniform machine scheduling to minimize makespan with job size predictions. Under the assumptions of knowing and not knowing the prediction error, we construct $O(\min\{\log m, \log \eta\})$-competitive algorithms using online doubling techniques, where m denotes the machine number and η the prediction error. The algorithms achieve optimal $O(1)$ consistency, optimal $O(\log m)$ robustness, and $O(\min\{\log m, \log \eta\})$ smoothness. We also extend the algorithms to static and dynamic scheduling and show that they retain the competitive ratio results. We perform extensive simulations to verify the theoretical results and show that the algorithms outperform the state-of-the-art.

In Chapter 4, we solve the problem of single machine scheduling to minimize the mean response time with job size predictions. We prove a sufficient condition for any algorithm to achieve the optimal $O(P)$ robustness and construct an algorithm with optimal $O(1)$ consistency and $O(P)$ robustness, where P denotes the maximum job size ratio. We also present an algorithm and conjecture that it achieves an $O(\eta^2)$ competitive ratio.

In Chapter 5, we solve the problem of single and parallel machine scheduling to minimize the mean response time with job size predictions. We propose *PEDRMLF*, *Predictions Enhanced Dynamic Randomized MultiLevel Feedback*, with optimal $O(\min\{\log \frac{n}{m}, \log P\})$ consistency and the best-known $O(\min\{\log n \log \frac{n}{m}, \log n \log P\})$ robustness. *PEDRMLF* incorporates job size predictions into a multi-level feedback queue scheduling policy. We show that the algorithm performance is deterministic under perfect predictions and matches the lower bounds. We also show that the (randomized) algorithm has the worst-case expected performance matching the best-known bounds under arbitrarily bad predictions. We perform extensive simulations to verify the theoretical results and compare the algorithm with the state-of-the-art. The simulations show that *PEDRMLF* outperforms the best-known non-clairvoyant algorithm by consistently achieving near-optimal objectives and staying close to the optimal clairvoyant algorithm.

In Chapter 6, we solve the problem of pulsed power load scheduling in cyber-physical systems to maximize energy output with normal load predictions. We formulate the pulsed power load scheduling model with normal load predictions and propose exact dynamic programming-based algorithms for constant and general normal loads. We prove the optimality of the algorithms and perform extensive simulations to compare the performance with existing algorithms. The simulations show that our algorithms outperform the state-of-the-art by having strong performance guarantees under the worst-case normal loads.

Many chapters in this thesis share a large portion of similar symbols and notations. However, due to the unavoidable difference in the problem settings, each chapter is assumed to have an independent set of notations to avoid the reader's confusion. Chapters 2 — 5 are closely related to each other as all are dedicated to the theory of online scheduling with predictions. The final technical chapter is a case study of cyber-physical system scheduling with predictions of some system parameters. These results not only solve the problems in this thesis but also extend to any application if the model fits the problem settings.

Online scheduling with predictions belongs to the general research agenda of algorithms with predictions. This field assumes that an algorithm can access the predictions of unknowns in decision-making, and the algorithm can use this (possibly imperfect) information to improve performance. Beginning in 2018, this young and active field has been a fast-growing area with many results not only in scheduling but also caching, auctioning, paging, clustering, and graphs [8]. The increasing power of learning models has enabled us with increased predictive capacity. These models are becoming accurate and lightweight, fitting all parts of a system nicely to support decision-making. The author believes that the next generation of computing paradigms will build upon algorithms and predictions.

CHAPTER 2

Fundamentals of Online Scheduling with Predictions

The major challenge of online scheduling is managing uncertainty. The literature has considered two typical information settings: clairvoyant and non-clairvoyant scheduling. The former assumes knowing complete information; the latter assumes knowing nothing. This chapter formulates the framework of online scheduling with predictions, an extension of clairvoyant and non-clairvoyant scheduling. The framework assumes the algorithms have access to predictions for some unknowns. We expect these predictions can improve our decision-making, although they may be imperfect. Our objective is algorithms with performance tied to prediction quality. We first introduce the concept of prediction error and several metrics that quantify prediction quality. We then present consistency, robustness, and smoothness that measure algorithm performance under our framework. We show the fundamental results of these metrics, connecting the proposed framework with clairvoyant and non-clairvoyant scheduling.

2.1 Online Scheduling with Predictions

Consider any online non-clairvoyant scheduling problem P^n: we have jobs to complete using the given (one or more) machines with the goal of optimizing an objective. The superscript n indicates non-clairvoyance. With Graham notations, P^n can be uniquely determined by $\alpha|\beta|\gamma$, where α denotes machine characteristics, β job characteristics, and γ the objective [9]. Two typical online settings are online over time and online over list. The former assumes that jobs arrive in the system over time; we do not know the existence of a job until it has arrived. The latter assumes that jobs arrive as a list; we do not know the existence of the following job until we have (irrevocably) assigned a machine to the current job. Independently of the online settings, two typical information settings are clairvoyance and non-clairvoyance. The former assumes that we have every arrived job's complete information, e.g., job size or processing time of a job; the latter assumes that we only have this information after the job's completion. We use P^c to denote the clairvoyant counterpart of P^n, i.e., P^c is the same problem as P^n except that the unknowns are known. The superscript c indicates clairvoyance. As an example for P^n, $Qm \mid$ *online-time-nclv*, *pmtn-restart* $\mid C_{\max}$, the problem we study in Chapter 3, denotes the problem of the uniform machine (machines have different processing speeds) scheduling for jobs arriving online over time to minimize makespan. The *pmtn-restart* means that jobs are preemptive, but once preempted, the job must restart running from scratch. The *online-time-nclv* indicates online over time and non-clairvoyance. The clairvoyant counterpart P^c, $Qm \mid$ *online-time-clv*, *pmtn-restart* $\mid C_{\max}$, is the same problem except that job size is known upon the job's arrival.

Augmenting P^n with predictions $\hat{p}$ for some unknowns (e.g., job sizes), we end up with a problem of online scheduling with predictions, which we denote by P. Our goal is to design algorithms that can use predictions in scheduling. Such algorithms are also called *learning-augmented* algorithms, as learning models are common sources of predictions. This is, however, not mandatory; the predictions can come from any source.

2.1.1 Prediction Error

The predictions are possibly imperfect. We use *prediction error* η to quantify prediction quality. No universal definition for prediction error exists. For convention, we assume prediction quality decreases as η increases, and there exists a lower bound $\eta_{\min}$ for η, where the predictions are perfect if $\eta = \eta_{\min}$, i.e., where the predictions equal the exact values. The value $\eta_{\min}$ varies as the prediction error metric varies. One example prediction error is $\eta = \max_{1 \leq j \leq n} \max\{\frac{p_j}{p_j^*}, \frac{p_j^*}{p_j}\}$, the maximum multiplicative gap between the exact value p_j^* and the prediction p_j, where predictions are for job sizes. The predictions are perfect if $\eta = 1$ ($\eta_{\min} = 1$ in this case); the worst predictions have unbounded η. This error metric has been used in [1, 3, 4, 10]. Another example is $\eta = \sum_{1 \leq j \leq n} |p_j^* - p_j|$, the sum of the absolute error between the exact value p_j^* and the prediction p_j. The predictions are perfect if $\eta = 0$ ($\eta_{\min} = 0$ in this case); the worst predictions, again, have unbounded η. This error metric has been used in [11–13]. It is possible to have predictions for different kinds of unknowns where each has an independent prediction error metric.

2.1.2 Competitive Framework

We measure algorithm performance with the classic competitive framework. A learning-augmented algorithm A for problem P will compare against an optimal offline algorithm A^*, which knows all the (present and future) information in advance and achieves the optimal objective. Let cost(I, A) denote the objective achieved by A on problem instance I. We say A is c-competitive or has competitive ratio c for problem P, if it satisfies that, for any problem instance I in problem P, cost$(I, A) \leq c \cdot$ cost(I, A^*) for some function c of problem input and prediction error η. The above definition works for deterministic algorithms. If instead A is a randomized algorithm with execution depending on some random choice β, we say A is c-competitive, if it satisfies that, for any problem instance I in P, $E_\beta[\text{cost}(I, A)] \leq c \cdot \text{cost}(I, A^*)$, where the expectation is taken over all possible β. The function c represents the performance of an algorithm; it represents the worst-case performance bound.

For the following discussions, we use symbol I, I^c, and I^n to indicate a problem instance for problem P, P^c, and P^n. An observation is the equality cost$(I, A^*) =$ cost$(I^c, A^{c*}) =$ cost(I^n, A^{n*}) holds for any optimal algorithms A^*, A^{c*}, and A^{n*} for problems P, P^c, and P^n, as they refer to the same scheduling problem.

2.1.3 Consistency, Robustness, and Smoothness

The work [11] proposes two metrics to measure how learning-augmented algorithms perform against changing prediction quality. Consistency measures the performance under the perfect predictions; (worst-case) robustness measures it under the worst predictions. We present their formal definitions.

Definition II.1 (Consistency)**.** *Algorithm A is said υ-consistent or has υ consistency, if A is υ-competitive under $\eta = \eta_{\min}$ for υ independent to η.*

Definition II.2 (Robustness)**.** *Algorithm A is said τ-robust or has τ robustness, if A is τ-competitive under any $\eta \geq \eta_{\min}$ for τ independent to η.*

The generalization of consistency and robustness is smoothness. We define smoothness as the competitive ratio of A under any η. Here, the competitive ratio is a function of η, indicating how smoothly the algorithm performance degrades with degrading predictions.

Definition II.3 (Smoothness)**.** *Algorithm A is said $\psi(\eta)$-smooth or has $\psi(\eta)$ smoothness, if A is $\psi(\eta)$-competitive under any $\eta \geq \eta_{\min}$.*

Smoothness covers what is provided by consistency and robustness: a $\psi(\eta)$-smooth learning-augmented algorithm is $\psi(\eta_{\min})$-consistent and $\sup_{\eta \geq \eta_{\min}} \psi(\eta)$-robust. It is, however, difficult for some algorithms to have an explicit smoothness function. In such cases, consistency and robustness are often used to outline the performance by showing the extreme behaviors of an algorithm. In the following sections, we present and prove several fundamental results for these metrics.

2.2 Fundamental Theorems of Consistency

Any learning-augmented scheduling algorithm that solves P has consistency bounded below by the competitive ratio lower bound of the clairvoyant counterpart P^c. Suppose the competitive ratio lower bound for P^c is C^c_{min}, i.e., any c-competitive algorithm for P^c must have $c \geq C^c_{\text{min}}$. Then, any learning-augmented algorithm that solves problem P cannot have consistency lower than C^c_{min}. The following theorem shows this.

Theorem II.4 (Universal Bound on Consistency)**.** *Any υ-consistent algorithm for problem P must have $\upsilon \geq C^c_{\text{min}}$, where C^c_{min} is the competitive ratio lower bound for P^c, the clairvoyant counterpart of P.*

Proof. We prove this by contradiction. Suppose there exists an υ-consistent algorithm A with $\upsilon < C^c_{\text{min}}$. For any problem instance I^c in problem P^c, run A with the information provided as part of the problem input (A treats the information as predictions). Since the information is exact, i.e., $\eta = \eta_{\min}$, we have

$$\text{cost}(I^c, A) = \text{cost}(I, A) \leq \upsilon \cdot \text{cost}(I^c, A^*)$$

where A^* is the optimal offline algorithm solving P^c. Therefore, A is υ-competitive for problem P^c with $\upsilon < C^c_{\text{min}}$, which contradicts with the C^c_{min} competitive ratio lower bound. Thus, it must be that

$v \geq C^c_{\min}$. The same result holds, by similar arguments, for randomized algorithms with the randomized competitive ratio lower bound. □

Conversely, any c-competitive clairvoyant scheduling algorithm for problem P^c can construct, by treating the predictions as exact values, a c-consistent learning-augmented algorithm for problem P. This gives us a recipe for consistent learning-augmented algorithms; it connects clairvoyant scheduling and online scheduling with predictions.

Theorem II.5 (Universal Existence of Consistency). *Any c-competitive algorithm for problem P^c constructs a c-consistent learning-augmented algorithm for problem P.*

Proof. Suppose algorithm A is c-competitive for problem P^c. Construct a learning-augmented algorithm A' that follows exactly what A does by treating the predictions as exact values. Algorithms A and A' see the same information and construct the same schedule when $\eta = \eta_{\min}$. Therefore, for any problem instance I in problem P, we have

$$\text{cost}(I, A') = \text{cost}(I^c, A) \leq c \cdot \text{cost}(I, A^*)$$

under $\eta = \eta_{\min}$, where A^* is the optimal offline algorithm solving P. It must hold that A' is c-competitive under $\eta = \eta_{\min}$ and thus c-consistent (c is independent to η by definition). □

2.3 Fundamental Theorems of Robustness

Any learning-augmented scheduling algorithm that solves P has robustness bounded below by the competitive ratio lower bound of the non-clairvoyant counterpart P^n. Suppose the competitive ratio lower bound for P^n is $C^n_{\min}$, i.e., any c-competitive algorithm for P^n must have $c \geq C^n_{\min}$. Then, any learning-augmented algorithm that solves problem P cannot have robustness lower than $C^n_{\min}$. The following theorem shows this.

Theorem II.6 (Universal Bound on Robustness). *Any τ-robust algorithm for problem P must have $\tau \geq C^n_{\min}$, where $C^n_{\min}$ is the competitive ratio lower bound for P^n, the non-clairvoyant counterpart of P.*

Proof. We prove this by contradiction. Suppose there exists a τ-robust algorithm A with $\tau < C^n_{\min}$. Construct a non-clairvoyant scheduling algorithm A' that follows exactly what A does as if the predictions are all set to 0, i.e., A' always uses 0 as the prediction for every unknown. For any problem instance I^n in problem P^n, run A'. We have

$$\text{cost}(I^n, A') = \text{cost}(I_{(\hat{p}=0)}, A) \leq \psi(\eta') \cdot \text{cost}(I^n, A^*) \leq \sup_{\eta \geq \eta_{\min}} \psi(\eta) \cdot \text{cost}(I^n, A^*) \leq \tau \cdot \text{cost}(I^n, A^*)$$

where A^* is the optimal offline algorithm solving P^n and η' is the prediction error for 0 as predictions ($I_{(\hat{p}=0)}$ denotes the problem instance I with predictions set to 0). Therefore, A' is τ-competitive for problem P^n with $\tau < C^n_{\min}$, which contradicts with the $C^n_{\min}$ competitive ratio lower bound. Thus, it must be that $\tau \geq C^c_{\min}$. The same result holds, by similar arguments, for randomized algorithms with the randomized competitive ratio lower bound. $\square$

Similarly, as in consistency, any c-competitive non-clairvoyant scheduling algorithm for problem P^n can construct, by ignoring the predictions, a c-robust learning-augmented algorithm for problem P. This gives us a recipe for robust learning-augmented algorithms; it connects non-clairvoyant scheduling and online scheduling with predictions.

Theorem II.7 (Universal Existence of Robustness)**.** *Any c-competitive algorithm for problem P^n constructs a c-robust learning-augmented algorithm for problem P.*

Proof. Suppose algorithm A is c-competitive for problem P^n. Construct a learning-augmented algorithm A' that follows exactly what A does by ignoring the predictions. Algorithms A and A' see the same information and construct the same schedule. Therefore, for any problem instance I in problem P, we have

$$\text{cost}(I, A') = \text{cost}(I^n, A) \leq c \cdot \text{cost}(I, A^*)$$

under any $\eta \geq \eta_{\min}$, where A^* is the optimal offline algorithm solving P. It must hold that A' is c-competitive under any $\eta \geq \eta_{\min}$ and thus c-robust (c is independent to η by definition). $\square$

2.4 Fundamental Theorems of Smoothness

The previous theorems bound the smoothness.

Theorem II.8 (Universal Bound on Smoothness)**.** *Any $\psi(\eta)$-smooth algorithm for problem P must have*

$$C^c_{\min} \leq \psi(\eta_{\min}) \text{ and } C^n_{\min} \leq \sup_{\eta \geq \eta_{\min}} \psi(\eta)$$

where $C^c_{\min}$ and $C^n_{\min}$ are the competitive ratio lower bounds for P^c and P^n.

Proof. It immediately follows from Theorems II.4 and II.6. $\square$

Unlike single metric consistency and robustness, designing algorithms that achieve the lower bounds of consistency and robustness simultaneously is non-trivial. There is no simple recipe unless we have the additional information — the prediction error.

2.4.1 The Effect of Knowing the Prediction Error

Suppose the prediction error η is also part of the input along with the predictions. In this case, we still do not have the exact values but the maximum "distance" between a prediction and the exact value. We show that this information suffices to build a learning-augmented algorithm that achieves optimal consistency and robustness simultaneously. Fix any problem P. Let A^c be a $C^c_{\min}$-competitive clairvoyant algorithm for problem P^c. Let A^n be a $C^n_{\min}$-competitive non-clairvoyant algorithm for problem P^n. Both A^c

Algorithm 1: Trivial learning-augmented algorithm

```
Data   : scheduling problem inputs, predictions $\hat{p}$, and the prediction error $\eta$
Result : schedule with optimal consistency and robustness
1 if $\eta = \eta_{\min}$ then
2     run algorithm $A^c$ by feeding $\hat{p}$ as exact values.
3 else
4     run algorithm $A^n$ by ignoring $\hat{p}$.
```

and A^n exist by the definition of (tight) lower bound on the competitive ratio. Construct a learning-augmented algorithm A as follows. Algorithm A runs A^c by treating all the predictions as the exact values if $\eta = \eta_{\min}$; A runs A^n by ignoring the predictions if $\eta > \eta_{\min}$. The pseudo-code is in Algorithm 1. The following theorem shows the effect of knowing the prediction error.

Theorem II.9 (Universal Existence of Smoothness)**.** *There exists, assuming a known prediction error η, a (trivial) $\psi(\eta)$-smooth algorithm for problem P that simultaneously achieves the lower bounds of consistency and robustness:*

$$C^c_{\min} = \psi(\eta_{\min}) \text{ and } C^n_{\min} = \sup_{\eta \geq \eta_{\min}} \psi(\eta)$$

where $C^c_{\min}$ and $C^n_{\min}$ are the competitive ratio lower bounds for P^c and P^n.

Proof. We show that Algorithm 1 achieves the desired property. When $\eta = \eta_{\min}$, we run A^c which gives us $C^c_{\min} = \psi(\eta)$ by Theorems II.4 and II.5. When $\eta > \eta_{\min}$, we run A^n which gives us $C^n_{\min} = \sup_{\eta > \eta_{\min}} \psi(\eta)$ by Theorems II.6 and II.7. Observe that $C^c_{\min} \leq C^n_{\min}$: the competitive ratio lower bound for P^c is at most that for P^n. Therefore, we have $C^n_{\min} = \sup_{\eta \geq \eta_{\min}} \psi(\eta)$. □

Although Algorithm 1 achieves optimal consistency and robustness, it discards the predictions even if they are close (but not equal) to the exact values. This is not desirable for online scheduling with predictions, as our primary goal is to improve performance with (possibly imperfect) predictions. Thus, we are interested in the non-trivial algorithms that either (1) do not assume a known prediction error or (2) do not use the known prediction error to distinguish cases of perfect predictions and others explicitly.

Theorem II.9 does not guarantee that learning-augmented algorithms with smoothness optimal at two ends are always achievable. This is because, more often, the optimal clairvoyant or the optimal non-clairvoyant algorithms achieving the competitive ratio lower bounds are challenging to find, i.e., many problems have a gap between the competitive ratio lower bound and the best-known competitive ratio. It is also possible that matching-bound algorithms exist for one end (e.g., clairvoyant counterpart) but not for the other (e.g., non-clairvoyant counterpart). Theorem II.9 can be extended to these cases. Rather than constructing algorithms with smoothness optimal at two ends, we can follow the same recipe to construct an algorithm with consistency matching the competitive ratio of the best-known clairvoyant algorithm and robustness matching the competitive ratio of the best-known non-clairvoyant algorithm. Thus, any advance in clairvoyant or non-clairvoyant scheduling advances online scheduling with predictions. The converse also holds.

Nevertheless, the assumption of knowing the prediction error is realistic. In particular, when predictions come from learning algorithms, the models are often trained on large historical data sets mirroring the characteristics of future jobs; the prediction error found in training and testing at least approximates the actual prediction error. With a known prediction error, our objective should be non-trivial learning-augmented algorithms. By not explicitly using the prediction error to distinguish perfect predictions from the others, the non-trivial algorithms (smoothly) bridge clairvoyant and non-clairvoyant scheduling. This also indicates that the design of learning-augmented algorithms can start with the clairvoyant or the non-clairvoyant competitive algorithms.

2.5 Conclusions

We have introduced the framework of online scheduling with predictions and shown how it extends from and connects clairvoyant and non-clairvoyant scheduling. We have introduced the prediction error metric and performance metrics — consistency, robustness, and smoothness — that measure the performance of learning-augmented algorithms. Fundamental results for these metrics have been shown. These results lower bound the metrics and thus define the optimum. Finally, we have considered the effect of knowing the prediction error: this information suffices in constructing algorithms with the best-known consistency and robustness. These results are the fundamentals of online scheduling with predictions.

CHAPTER 3

Uniform Machine Scheduling with Predictions

This chapter contributes to the research agenda of online scheduling with predictions by studying the makespan minimization in uniformly related machine non-clairvoyant scheduling with job size predictions. Our task is to design online algorithms that use predictions and have performance guarantees tied to the quality of predictions. We first propose a simple algorithm-independent prediction error metric to quantify prediction quality. Then we design an offline improved 2-relaxed decision procedure approximating the optimal schedule to effectively use predictions. With the decision procedure, we propose an online $O(\min\{\log \eta, \log m\})$-competitive ($O(\min\{\log \eta, \log m\})$-smooth) static scheduling algorithm assuming a known prediction error. We use this algorithm to construct a robust $O(\min\{\log \eta, \log m\})$-competitive ($O(\min\{\log \eta, \log m\})$-smooth) static scheduling algorithm that does not assume a known error. Finally, we extend these static scheduling algorithms to address dynamic scheduling, where jobs arrive over time. The dynamic scheduling algorithms attain the same competitive ratios and smoothness as the static ones. All the algorithms achieve optimal consistency and robustness. They require only moderate predictions to break the well-known $\Omega(\log m)$ lower bound, showing the potential of predictions in managing uncertainty.

3.1 Introduction

Managing uncertainty is the focus of online optimization. Against the uncertainty from problem inputs, traditional methods strive to bound the worst-case performance under the assumption of incomplete information [14, 15]. However, dealing with uncertainty under such assumptions incurs high costs in the solution quality compared with when the needed information is known [16, 17]. To reduce the costs, recent works have introduced augmenting algorithms with advice [18, 19]. With additional relevant information, one can improve the performance and achieve cost-efficiency [20–22]. At the same time, recent works in learning theory have introduced models with accurate predictions to wide applications [23, 24]. Online optimization with predictions, a new framework that combines the advice model and learning theory techniques emerges.

One of the most significant fields boosted by the framework is online scheduling [1, 3]. Multiple works have shown that predictions can improve the theoretical performance bounds. Online scheduling with predictions is the new trend for scheduling studies. [11] develops a preferential round-robin algorithm using job size predictions for single-machine scheduling to minimize total completion time. [10] develops an online rounding algorithm using machine weight predictions for parallel machine scheduling under restricted assignments to minimize makespan. [25] develops greedy and binning algorithms using job size predictions for single-machine scheduling to minimize weighted flow time. [26] revisits the problem of single-machine scheduling to minimize total completion time, and develops a new prediction error measurement and an improved round-robin algorithm using job size predictions. All these works reduce the attainable competitive ratios via predictions. This work extends scheduling with predictions to another online scheduling problem: makespan minimization in uniformly related machine non-clairvoyant scheduling. We study how to use imperfect predictions to improve online optimization and present the first such algorithms for uniformly related machine non-clairvoyant scheduling.

We develop algorithms with job size predictions, yielding significantly improved performance bounds. The development begins with defining a simple algorithm-independent prediction error measurement η. With this error measurement and the job size predictions, our algorithm simultaneously schedules jobs and computes (under-)estimations on the actual job sizes and the error η on the fly. We propose an (improved) offline approximation compared with [27] and four online algorithms. The first two online algorithms deal with *static scheduling* where all jobs are available for processing at time 0. The first algorithm assumes a known prediction error; the second does not. Then, we extend these two algorithms to deal with *dynamic scheduling* where jobs have arbitrary release times. Again, we develop one algorithm that assumes a known prediction error, and the other one does not. We use online doubling techniques to deal with uncertain job sizes, optimal makespan, and prediction errors. We use a simple but effective rerunning strategy to deal with uncertain release times. We prove that our proposed online algorithms achieve $O(\min\{\log\eta, \log m\})$-competitive ratio with m machines, breaking the previous $\Omega(\log m)$ lower bound. Our contributions are summarized below.

(1) An improved offline 2-relaxed decision procedure for approximating the optimal schedule. (Theorem III.1)
(2) An online $O(\min\{\log\eta, \log m\})$-competitive static scheduling algorithm with known prediction error (Theorem III.2) and a robust online $O(\min\{\log\eta, \log m\})$-competitive static scheduling algorithm with unknown prediction error (Theorem III.3).
(3) An online $O(\min\{\log\eta, \log m\})$-competitive dynamic scheduling algorithm with known prediction error (Theorem III.4) and a robust online $O(\min\{\log\eta, \log m\})$-competitive dynamic scheduling algorithm with unknown prediction error (Theorem III.5).
(4) Extensive experiments to evaluate the theoretical results and the practicality of the proposed algorithms.

3.2 Preliminaries

3.2.1 Problem Definition

We study the makespan minimization problem in uniformly related machine non-clairvoyant scheduling. There are m uniformly related parallel machines and n independent jobs. A machine is denoted by M_i, $1 \leq i \leq m$, and a job is denoted by J_j, $1 \leq j \leq n$. Job J_j has size p_j^*, but this value remains unknown until J_j is completed. This setting is known as *non-clairvoyant* in literature. In this work, we assume $p_j^* \geq 1$ for all $1 \leq j \leq n$, which is known to the scheduler. Machines are heterogeneous in processing power, and machine M_i has a processing speed s_i. For convenience, we sort the machines by non-increasing speeds, i.e., $s_1 \geq s_2 \geq ... \geq s_m$. If job J_j is assigned to machine M_i, the processing time for this job will be $\frac{p_j^*}{s_i}$. The jobs are priority-free and preemptive-restart. Preemptive-restart means that running jobs are non-preemptive but can be canceled and restarted later on any machine. Job J_j has a release time r_j. The system does not know the existence of J_j until it arrives at r_j. We will first assume $r_j = 0$ for every job, i.e., static scheduling. Later, we remove this assumption and study dynamic scheduling. Our objective is to minimize the *makespan*, $C_{\max}$, the time when the last job completes. Our scheduling problem can be defined as $Qm \mid \textit{online-time-nclv}, \textit{pmtn-restart} \mid C_{\max}$ for static scheduling, or $Qm \mid \textit{online-time-nclv}, \textit{pmtn-restart}, r_j \mid C_{\max}$ for dynamic scheduling [9]. In addition, the scheduler is allowed to make decisions with the predictions of some parameters to manage uncertainty. However, the predictions are possibly imperfect. The performance of the scheduler could be adversely affected by prediction quality. We discuss the metrics of algorithm performance and prediction quality below.

3.2.2 Predictions and Prediction Error

When integrating predictions into a scheduler, one must consider what to predict and how the prediction quality is measured. In principle, we expect the predictions to improve the competitive ratio. The scheduler should guarantee a bounded performance under poor predictions and a near-optimal solution with accurate predictions. Last, the learning problem itself must be feasible. A counterexample is predicting the optimal schedule, which simplifies the scheduling algorithm but makes learning extremely hard.

This work considers job size predictions. Recent works [28–30] have shown that job sizes are highly predictable in many scenarios, e.g., cloud, clusters, and factories. Our online algorithms can access the job size prediction p_j for every job J_j of size p_j^*. Similar to the works [1, 3, 10] that measure the error by ratios, we define the error for job J_j as $\eta_j = \max\{\frac{p_j}{p_j^*}, \frac{p_j^*}{p_j}\}$, the multiplicative gap between the prediction and the exact value, and the overall prediction error η as

$$\eta = \max_{1 \leq j \leq n} \eta_j = \max_{1 \leq j \leq n} \max\{\frac{p_j}{p_j^*}, \frac{p_j^*}{p_j}\}$$

Observe that $\eta_j \geq 1$ for any j, so $\eta \geq 1$. The prediction is perfect if and only if $\eta = 1$. The prediction error measurement is simple, natural, and algorithm-independent.

3.2.3 Performance Evaluation

We evaluate the performance of the algorithms by the competitive framework [14] and the metrics of consistency, robustness, and smoothness introduced in Chapter 2.

3.2.4 Related Results

Let us review some important related results: (1) the problem is NP-complete in the strong sense for m arbitrary, (2) the offline version of the problem has a polynomial-time 2-relaxed decision procedure, (3) the online version has a lower bound $\Omega(\log m)$ on the competitive ratio, and (4) there exists an $O(\log m)$-competitive algorithm matching this lower bound.

For computational complexity, we consider the simpler offline problem with identical machines where the job sizes are known. This problem is NP-complete in the strong sense for m arbitrary [31]. A simple reduction to our problem shows that the offline version of the uniformly related machine scheduling is NP-complete in the strong sense for m arbitrary.

There exists a 2-relaxed decision procedure for the offline version [27]. An algorithm is a 2-relaxed decision procedure if, given a makespan d, the algorithm either determines that no schedule of length d exists or produces a schedule of at most $2d$ makespan. With this procedure, a bisection search leads to a 2-approximation. Our online algorithms will use a similar 2-relaxed decision procedure, with slight improvement, to approximate a near-optimal schedule.

The best result that has resolved this problem for decades is [27]. It has been shown that any online deterministic algorithm has a competitive ratio of $\Omega(\log m)$, and the authors give an $O(\log m)$-competitive algorithm matching this lower bound. This algorithm is, therefore, asymptotically optimal in non-clairvoyant scheduling. The recent advances in learning theory lead to higher and higher prediction accuracy, thus enabling new ways to minimize makespan.

3.3 Robust Online Scheduling with Job Size Predictions

3.3.1 Algorithm Overview

The development of our solutions involves four stages and produces five algorithms. We consider, in the first three stages, static scheduling, where all jobs are available for processing at time 0. We consider, in the final stage, dynamic scheduling, where jobs have arbitrary release times. Figure III.1 shows the connection between the proposed algorithms.

Our first stage assumes that job size p_j^* and the optimal makespan d are known. There is a 2-relaxed decision procedure for producing a schedule of length at most $2d$ [27]. We will use this procedure, with slight improvement, as our base algorithm. The second stage solves the problem online by no longer using the actual job sizes and the optimal makespan d. Instead, we use the job size prediction p_j and the known prediction error η. Here, we note that accessing the prediction error η is a reasonable assumption as the predictive model is often trained offline on large labeled datasets — the error in the training stage will likely reflect the error in run time. We propose an $O(\min\{\log \eta, \log m\})$-competitive algorithm for this case. The algorithm estimates the actual job sizes in at most $O(\min\{\log \eta, \log m\})$ rounds. The 2-relaxed decision procedure is executed each round while the optimal makespan is searched online

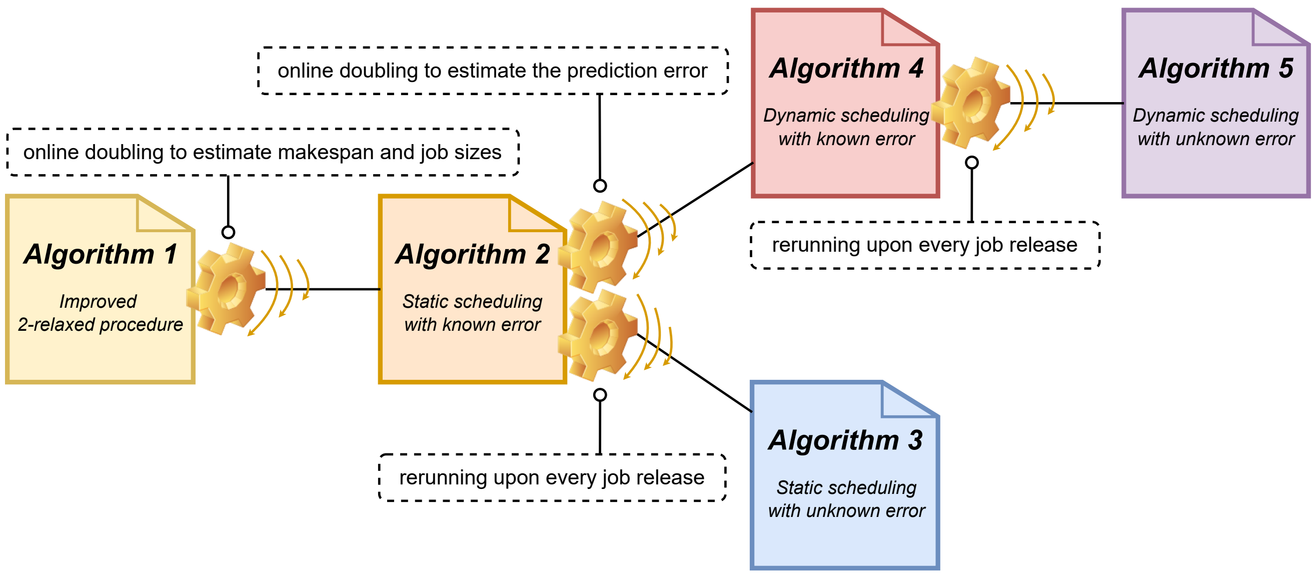

online doubling to estimate the prediction error
online doubling to estimate makespan and job sizes
Algorithm 1
Improved 2-relaxed procedure
Algorithm 2
Static scheduling with known error
Algorithm 4
Dynamic scheduling with known error
Algorithm 5
Dynamic scheduling with unknown error
rerunning upon every job release
rerunning upon every job release
Algorithm 3
Static scheduling with unknown error

Figure III.1: The connection between the algorithms. Algorithm 2 is a decision procedure that decides if a given set of jobs (sizes) can be completed within a given makespan. Algorithm 3 uses Algorithm 2 as a subroutine to estimate the optimal makespan and the actual job sizes online via a doubling technique. Algorithm 5 reruns Algorithm 3 as a subroutine upon every job release to handle release times. Algorithm 4 uses Algorithm 3 as a subroutine to estimate the prediction error online via a doubling technique. Finally, Algorithm 6 reruns Algorithm 4 as a subroutine upon every job release to handle release times.

using a standard doubling method. Estimating the job sizes incurs a cost of $O(\min\{\log \eta, \log m\})$ factor in makespan, while the doubling method introduces a multiplicative factor of 2 to the competitive ratio. In the third stage, we further remove the assumption of knowing the prediction error η. The prediction error will be searched online using a doubling method. The search will impose an extra $\log \eta$ term on the competitive ratio. To provide an $O(\log m)$ strong worst-case guarantee, we switch to the guaranteed $O(\log m)$-competitive algorithm once we believe the prediction error is too large. This will give us back an $O(\min\{\log \eta, \log m\})$-competitive algorithm. In the final stage, we extend the online static scheduling algorithms to deal with the release times by rerunning from the scratch the whole static scheduling algorithm upon any job arrival. We prove that this rerunning strategy retains the $O(\min\{\log \eta, \log m\})$ competitive ratio.

3.3.2 An Improved 2-Relaxed Procedure

Consider offline scheduling with all job sizes p_j^* and the makespan d given. Our proposed 2-relaxed procedure ensures to produce a schedule of length at most $2d$, or otherwise indicate that no d-length schedule exists. The procedure works as follows. When a machine M_i is idle, it starts processing either the largest uncompleted job that can be completed in period d or the smallest job not yet started. Specifically, machine M_i will first consider the largest uncompleted job J_j with $p_j^* \leq s_i \cdot d$ that is either not started or currently being processed on another machine M_k with $p_j^* > s_k \cdot d$. If there is no such job, the machine will process the smallest non-running job. Machine M_i stays idle if all the above

Algorithm 2: Improved 2-relaxed procedure

```
Data   : job sizes $p_j^*$ and a makespan $d$
Result : schedule of length at most $2d$, or output **no** indicating that it is impossible to construct a schedule
         of length $d$
1  Function GETJOB($i$) // function that returns the next job for machine $M_i$
2      set $J_{\text{next1}} \leftarrow J_j$ with $p_j^* \leq s_i \cdot d$ and either $J_j$ not begin or is being processed on $M_k$ with $p_j^* > s_k \cdot d$,
         such that $p_j^*$ is maximal.
3      set $J_{\text{next2}} \leftarrow J_j$ with $p_j^* > s_i \cdot d$ and $J_j$ not begin, such that $p_j^*$ is minimal.
4      return the first nonempty $J_{\text{next}p}$ (smallest $p$) or null.

5  Event Function MACHINEIDLE() // machine $M_i$ is idle
6      set $J_j \leftarrow$ GETJOB ($i$).
7      $M_i$ starts processing $J_j$ or stays idle if $J_j$ is null.

8  Event Function SCHEDULECOMPLETE() // all jobs have been completed
9      return the whole schedule completes.

10 Event Function TIMEOUT() // run time exceeds $2d$
11     return no.
```

conditions fail. As a rule, if, at any time, there are multiple idle machines, event function *MachineIdle* is executed sequentially for the machines in the non-increasing order of machine speed. The entire procedure lasts until all jobs are processed or until time $2d$. If there are uncompleted jobs at time $2d$, the procedure outputs **no** indicating that it is impossible to construct a schedule of length d. Otherwise, it has constructed a schedule of length at most $2d$.

We outline the improvement of our 2-relaxed procedure over the original one [27]. First, our procedure uses the largest job first strategy to assign the large jobs to fast machines for processing. It provides slow machines opportunities to complete relatively small jobs that could be otherwise allocated to fast machines. This strategy contributes to a smaller makespan as letting fast machines process small jobs often increases the makespan. Second, our procedure can invoke idle machines to complete large jobs that can be completed in $2d$ but not in d time. In contrast, the procedure [27] leaves such machines idle. If there exists a schedule with makespan at most $2d$, our procedure can generate one with a shorter length than that by [27]. The pseudo-code is shown in Algorithm 2. Our first result is the correctness of the decision procedure stated as follows.

Theorem III.1 (Theorem III.6 Restated)**.** *The improved 2-relaxed procedure can produce a schedule of length at most $2d$, or otherwise confirms that it is impossible to have a schedule of length d.*

3.3.3 Online Static Scheduling with Job Size Predictions and a Known η

We consider the online static scheduling where job sizes and the optimal makespan are unknown. Instead, the scheduler knows the job size prediction p_j and the prediction error η.

The pseudo-code is shown in Algorithm 3. Initially, the job size estimate of job J_j is set by $p_j^e = \frac{p_j}{\eta}$. This estimate will last k rounds of doubling where $p_j^* \leq 2^k \cdot \frac{p_j}{\eta} < 2 \cdot p_j^*$. The number of rounds is bounded

Algorithm 3: Online static scheduling with known error

```
Data   : job size predictions $p_j$ and the prediction error $\eta$
Result : schedule with makespan $O(\min\{\log \eta, \log m\}) \cdot C^*_{\max}$
1  set sup ← $\min\{k \mid \sum_{i=1}^{k} s_i \geq \frac{1}{2}\sum_{i=1}^{m} s_i, 1 \leq k \leq m\}$.
2  Function GETJOB($i$) // function that returns the next job for machine $M_i$
3      if $i \leq$ sup then
4          set $J_{\text{next1}} \leftarrow J_j$ with $p^e_j \leq s_i \cdot d$ and either $J_j$ not begin or is being processed on $M_k$ with
             $p^e_j > s_k \cdot d$ or $k >$ sup, such that $p^e_j$ is maximal.
5      else
6          set $J_{\text{next1}} \leftarrow J_j$ with $p^e_j \leq s_i \cdot d$ and either $J_j$ not begin or is being processed on $M_k$ with
             $p^e_j > s_k \cdot d$, such that $p^e_j$ is maximal.
7      set $J_{\text{next2}} \leftarrow J_j$ with $p^e_j > s_i \cdot d$ and $J_j$ not begin, such that $p^e_j$ is minimal.
8      return the first nonempty $J_{\text{next}p}$ (smallest $p$) or null.
9  set $p^e_j \leftarrow \frac{p_j}{\eta}$, $\forall\, 1 \leq j \leq n$. // job size estimates
10 set $d \leftarrow \frac{\max_{1\leq j\leq n} p_j}{\eta \cdot s_1}$. // makespan estimate
11 set Time ← $0$. // Time variable that increases as the real-time
12 Event Function MACHINEIDLE() // machine $M_i$ is idle
13     set $J_j \leftarrow$ GETJOB ($i$).
14     $M_i$ starts processing $J_j$ or stays idle if $J_j$ is null.
15 Event Function SCHEDULECOMPLETE() // all jobs have been completed
16     return the whole schedule completes.
17 Event Function TIMEOUT() // Time $> 2d$
18     if there is an uncompleted job not started by any machine $M_i$ with $i \leq$ sup and $p^e_j \leq s_i \cdot d$ before
         time $d$ or machine $M_1$ processes any job for more than $d$ time then
19         set $d \leftarrow 2d$. // double the makespan estimate
20     else
21         for job $J_j$ that has been processing on $M_i$ for time $t$ do
22             set $p^e_j \leftarrow \max\{2p^e_j, 2s_i \cdot t\}$.
23     set Time ← $0$.
24     idle all machines $M_i$ $\forall\, 1 \leq i \leq n$.
```

and we show that $k \leq [2\log\eta + 2]$. Meanwhile, the algorithm also estimates the optimal makespan $C^*_{\max}$ in rounds. Initially, we set the makespan estimate $d = \frac{\max_{1\leq j\leq n} p_j}{\eta \cdot s_1}$, where $d \leq C^*_{\max}$. Then we run a similar procedure as Algorithm 2 to test if the makespan estimate is achievable. This estimate will undergo k' rounds of doubling where $C^*_{\max} \leq 2^{k'} \cdot \frac{\max_{1\leq j\leq n} p_j}{\eta \cdot s_1} < 2 \cdot C^*_{\max}$. Finally, to bound, under the worst-case predictions, the number of doubling rounds for job size estimation, we refine *getJob* function in Algorithm 2. This refinement induces some fast machines to carry out more jobs. Define a critical machine index threshold *sup* $= \min\{k \mid \sum_{i=1}^{k} s_i \geq \frac{1}{2}\sum_{i=1}^{m} s_i, 1 \leq k \leq m\}$ (according to [27]). For machine M_i with $i >$ *sup*, the definition of *getJob* remains the same. Otherwise, the definition of J_{next1} changes to $J_{\text{next1}} \leftarrow J_j$ with the maximal p^*_j such that $p^*_j \leq s_i \cdot d$, where J_j is not started or currently processed on another $M_{k''}$ with $p^*_j > s_{k''} \cdot d$ or $k'' >$ *sup*. It means that a fast machine M_i ($i \leq$ *sup*) can

Algorithm 4: Online static scheduling with unknown error

```
   Data   : job size predictions $p_j$
   Result : schedule with makespan $O(\min\{\log \eta, \log m\}) \cdot C^*_{\max}$
1  set $\eta^e \leftarrow 1$.
2  while $(\eta^e)^2 < m$ do
3      run Algorithm 3 with job size predictions $p_j$ for uncompleted jobs and prediction error $\eta^e$; stop when
         detecting any job $J_j$ with $\frac{p_j}{\eta^e} > p^*_j$ upon its completion.
4      if all jobs have been completed then
5          return the whole schedule completes.
6      set $\eta^e \leftarrow 2\eta^e$.
7      idle all machines $M_i$ $\forall\ 1 \le i \le n$.
8  if there are uncompleted jobs then
9      run Algorithm 3 with $p_j = 1$ for all uncompleted job $J_j$ and $\eta = \infty$.
10 return the whole schedule completes.
```

cancel a running job on a slow machine $M_{k''}$ ($k'' > sup$) regardless its completion time. The benefit of this refinement is to bound the number of doubling rounds for job size estimation by $[\log m + 2]$.

In Algorithm 3, *Time* variable is initially set to 0 and increases as the real-time. When *Time* reaches $2d$, event function *TimeOut* is triggered; at the end of *TimeOut*, *Time* is set back to 0. When *Time* is increasing in $[0, 2d]$, machines process jobs outputted by *getJob* function, as described in event function *MachineIdle*. For the following discussion, we refer *inner procedure* as one circle of execution from *Time* variable starting with 0 until either it reaches $2d$ or the schedule completes. The inner procedure will run multiple rounds until it completes all jobs. After each round, the job size estimates are updated, or the makespan estimate is updated. These updates could change the output of *getJob* function in the following rounds. As a rule, if, at any time, there are multiple idle machines, event function *MachineIdle* is executed sequentially for the machines in the non-increasing order of machine speed. We will show that the doubling strategy for estimating the job sizes incurs a cost of $O(\min\{\log \eta, \log m\})$ multiplicative factor in the competitive ratio, and the doubling strategy for estimating the optimal makespan incurs only a cost of constant 2 multiplicative factor. Combining these, we obtain the following significant result.

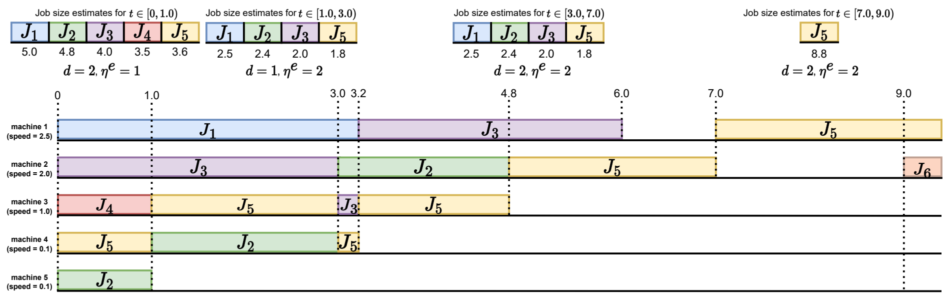
Theorem III.2 (Theorem III.12 Restated). *Given the job size predictions p_j and the prediction error $\eta = \max_{1 \le j \le n} \max\{\frac{p_j}{p^*_j}, \frac{p^*_j}{p_j}\}$, Algorithm 3 has an $O(\min\{\log \eta, \log m\})$ competitive ratio in online makespan minimization for uniformly related machine static scheduling.*

3.3.4 Online Static Scheduling with Job Size Predictions and Unknown η

We now consider a more general situation where the scheduler has no knowledge of the prediction error but just the predictions. The pseudo-code is given in Algorithm 4. We will first assume the predictions are perfect, i.e., $\eta^e = 1$. Run Algorithm 3 with η^e until we detect that η^e underestimates the prediction error. To detect underestimation, we check if any job J_j can complete on machine M_i within time t and $s_i \cdot t < \frac{p_j}{\eta^e}$. Once Algorithm 3 detects an underestimation, it stops; it doubles η^e; it runs again with the updated η^e. These operations will repeat at most $O(\log \eta)$ times before reaching the actual prediction

	J_1	J_2	J_3	J_4	J_5	J_6
Release time	0.0	0.0	0.0	0.0	0.0	9.0
Job size	8.0	3.6	7.0	1.0	9.0	1.0
Job size prediction	5.0	4.8	4.0	3.5	3.6	2.0

(a)



(b)

Figure III.2: Algorithm execution. Subfigure (a) is a dynamic scheduling workload. Subfigure (b) is the execution (up to time 9.0) of Algorithm 6 with 5 machines ($s_1 = 2.5, s_2 = 2.0, s_3 = 1.0, s_4 = 0.1, s_5 = 0.1$ and, thus, *sup* $= 3$) against the workload. Jobs J_1, J_2, J_3, J_4, J_5 are available at time 0.0 when the algorithm starts with an inner procedure round with $d = 2$ and $\eta^e = 1$. At time 1.0 when job J_4 is completed, the algorithm detects $\frac{p_4}{\eta^e} = p_4 = 3.5 > 1.0 = p_4^*$ (an underestimation of the prediction error); thus, it cancels the current inner procedure; then, it doubles the prediction error estimate η^e; correspondingly, it updates the job size estimates by dividing every estimate by 2 and the makespan estimate by setting $d = 1$. Time $[1.0, 3.0)$ covers the first complete inner procedure round; at the round termination, the algorithm doubles the makespan estimate. Time $[3.0, 4.0)$ covers the second complete inner procedure round during which the algorithm completes jobs J_1, J_2, J_3; at the round termination, the algorithm updates the job size estimate of J_5 by setting $p_5^e = 8.8$. The algorithm is supposed to run the following inner procedure round in $[7.0, 11.0)$, but the arrival of J_6 at time 9.0 interrupts. At time 9.0, the algorithm must rerun Algorithm 4 with jobs J_5 and J_6. In the following run, job J_5 will be allocated to machine M_1 and J_6 to M_2. Algorithm 6 will complete the workload with makespan 10.6; for just the first five jobs (a static scheduling workload), the algorithm will still complete with the same makespan (10.6). For comparison, the optimal makespan for the first five jobs is 6.0 (J_1, J_3 on M_1, J_4, J_5 on M_2, and J_2 on M_3) and that for all six jobs is 9.4.

error. To also bound the performance for arbitrarily bad predictions, Algorithm 4 assumes $\eta = \infty$ when it realizes $(\eta^e)^2 \geq m$. With $\eta = \infty$, Algorithm 3 is $O(\log m)$-competitive. We obtain a general online static scheduling algorithm with job size predictions.

Theorem III.3 (Theorem III.17 Restated)**.** *Given only the job size predictions p_j, Algorithm 4 has an $O(\min\{\log \eta, \log m\})$ competitive ratio in online makespan minimization for uniformly related machine static scheduling, where η is the prediction error and $\eta = \max_{1 \leq j \leq n} \max\{\frac{p_j}{p_j^*}, \frac{p_j^*}{p_j}\}$.*

3.3.5 Online Dynamic Scheduling with Job Size Predictions

We extend the static scheduling algorithms (Algorithms 3 and 4) to deal with release times, i.e., dynamic scheduling. We show that the simple strategy of rerunning a static scheduling algorithm upon every job release is effective enough to yield (asymptotically) the same competitive ratio as the static scheduling algorithm. The performance bound for this strategy follows the observation that the optimal makespan is at least the latest release time. Thus, rerunning incurs at most a cost of an additive one in the competitive

```
Algorithm 5: Online dynamic scheduling with known error
  Data   : job size predictions $p_j$ and the prediction error $\eta$
  Result : schedule with makespan $O(\min\{\log\eta, \log m\}) \cdot C^*_{\max}$
1 Event Function JobRelease() // Job $J_j$ is released at time $r_j$
2     idle all machines $M_i\ \forall\ 1 \le i \le n$.
3     run Algorithm 3 with job size predictions $p_j$ for uncompleted jobs and prediction error $\eta$.
```

```
Algorithm 6: Online dynamic scheduling with unknown error
  Data   : job size predictions $p_j$
  Result : schedule with makespan $O(\min\{\log\eta, \log m\}) \cdot C^*_{\max}$
1 Event Function JobRelease() // Job $J_j$ is released at time $r_j$
2     idle all machines $M_i\ \forall\ 1 \le i \le n$.
3     run Algorithm 4 with job size predictions $p_j$ for uncompleted jobs.
```

ratio. Formally, we present Algorithms 5 and 6 for dynamic scheduling with known and unknown errors. Figure III.2 presents a sample run of Algorithm 6; the execution shows the scenarios of updating the prediction error estimate, doubling the makespan estimate, doubling the job size estimates, and rerunning the static scheduling algorithm.

Theorem III.4 (Theorem III.24 Restated)**.** *Given the job size predictions p_j and the prediction error $\eta = \max_{1\le j\le n}\max\{\frac{p_j}{p_j^*}, \frac{p_j^*}{p_j}\}$, Algorithm 5 has an $O(\min\{\log\eta, \log m\})$ competitive ratio in online makespan minimization for uniformly related machine dynamic scheduling.*

Theorem III.5 (Theorem III.25 Restated)**.** *Given only the job size predictions p_j, Algorithm 6 has an $O(\min\{\log\eta, \log m\})$ competitive ratio in online makespan minimization for uniformly related machine dynamic scheduling, where η is the prediction error and $\eta = \max_{1\le j\le n}\max\{\frac{p_j}{p_j^*}, \frac{p_j^*}{p_j}\}$.*

3.4 Analysis

In this section, we prove the above theorems. See Table III.1 for the table of notations.

S	Meaning	S	Meaning
n	number of jobs	m	number of machines
J_j	job with index j	M_i	machine with index i
p_j^*	job size of J_j	p_j	job size prediction of J_j
η_j	prediction error of J_j	η	total prediction error
r_j	release time of J_j	s_i	processing speed of M_i
sup	machine threshold	p_j^e	job size estimate of J_j
d	makespan estimate	η^e	prediction error estimate

Table III.1: A table of notations.

3.4.1 Correctness for Algorithm 2

Theorem III.6 (Correctness of Improved 2-Relaxed Procedure)**.** *The improved* 2*-relaxed procedure either produces a schedule of length at most* $2d$ *or otherwise ensures no* d*-length schedule exists.*

Proof. The whole schedule completes only when the run time does not exceed $2d$. Thus, it suffices to show that no valid d-length schedule exists if the procedure outputs **no**. Consider an uncompleted job J_j. No valid d-length schedule exists if $p_j^* > s_1 \cdot d$; we thus assume $p_j^* \leq s_1 \cdot d$. Let k be the largest machine index that satisfies $p_j^* \leq s_k \cdot d$. For this statement to hold, machines $M_1, ..., M_k$ must be fully occupied by jobs with size at least p_j^* before time d. Indeed, otherwise, the function *getJob* will let one of these k machines process J_j before time d and complete J_j by time $2d$. To construct, if there exists, a d-length schedule, any job with size at least p_j^* must be processed on one of the first k machines, $M_1, ..., M_k$. Observe that, by the existence of J_j and the definition of *getJob*, no job is canceled on the first k machines before time d. Then, the sum of the size of the jobs that must be processed on machines M_i ($i \leq k$) is already more than $\sum_{i=1}^{k} s_i \cdot d$. Thus, no d-length schedule exists. □

3.4.2 Performance Bounds for Algorithm 3

We show the competitive ratio of Algorithm 3. Recall that we refer *inner procedure* one circle of execution from *Time* variable starting with 0 until either it increases to $2d$ or the schedule completes. We prove the following bounds. (1) For a given makespan d, the number of rounds of the inner procedure is bounded by either $O(\log \eta)$ or $O(\log m)$, whichever is smaller. (2) Every inner procedure incurs a cost of $O(1) \cdot C_{\max}^*$ in makespan, where $C_{\max}^*$ denotes the optimal makespan. (3) The rounds that end up doubling makespan estimate d incur, in total, a constant multiplicative factor cost in makespan. Combining these results establishes the proof for the $O(\min\{\log \eta, \log m\})$ competitive ratio.

Lemma III.7 (Bound on Inner Procedure Rounds by $\log \eta$)**.** *For a fixed makespan estimate* d*, the number of inner procedure rounds is at most* $[2 \log \eta + 2]$.

Proof. Execute the inner procedure once. We double the makespan estimate d when there is an uncompleted job J_j not started on any machine M_i with $i \leq$ *sup* and $p_j^e \leq s_i \cdot d$ before time d or machine M_1 processes any job for more than d time. Otherwise, the inner procedure continues the following round with the same makespan estimate, which happens only if there are at most *sup* $- 1$ uncompleted jobs. Suppose, then, the job size estimates undergo k doubling rounds until either the whole schedule completes or the makespan estimate is doubled. Consider the inner procedure after the $(k-1)$-th round of doubling. We claim, for any uncompleted job J_j (uncompleted after the k-th round), that $p_j^e < p_j^*$ at the termination of the $(k-1)$-th round. Indeed, otherwise, $p_j^e \geq p_j^*$; as the algorithm runs the k-th round without ending up doubling the makespan estimate, J_j must start running on some machine M_i with $p_j^e \leq s_i \cdot d$ before time d. Then, by time $2d$ in the k-th round, the job will have been completed since $p_j^* \leq p_j^e \leq s_i \cdot d$. This contradicts J_j being uncompleted. Finally, observe that every round (at least) double the job size estimate — we obtain

$$2^{k-1} \cdot \frac{p_j}{\eta} \leq p_j^e < p_j^*$$

which implies

$$2^{k-1} < \eta \cdot \frac{p_j^*}{p_j} = \eta \cdot \eta_j \leq \eta^2 \implies k \leq [2 \log \eta + 1]$$

and the number of inner procedure rounds is at most $k + 1 \leq [2 \log \eta + 2]$ $\square$

Lemma III.8 (Bound on Inner Procedure Rounds by $\log m$)**.** *For a fixed makespan estimate d, the number of inner procedure rounds is at most* $[\log m + 2]$.

Proof. Execute the inner procedure once. Similar to Lemma III.7, the inner procedure continues, with the same makespan estimate, the following round only if there are at most *sup* $- 1$ uncompleted jobs each running on one of $M_2, ..., M_{sup}$ for more than d time. Without loss of generality, rename these uncompleted jobs be $J_1, ..., J_r$ ($r <$ *sup*). After updating the job size estimates for the first time, it follows that $p_j^e \geq 2 \cdot s_{sup} \cdot d$, $j \leq r$. Suppose, then, the job size estimates undergo k doubling rounds before either the whole schedule completes or the makespan estimate doubles. Consider the inner procedure after the $(k-1)$-th round of doubling. Any uncompleted job J_j must start running on some machine M_i with $p_j^e \leq s_i \cdot d$. We let p_{j1}^e and $p_{j(k-1)}^e$ denote the job size estimate of J_j after the first and after the $(k-1)$-th doubling. We obtain

$$2^{k-2} \cdot 2 \cdot s_{sup} \cdot d \leq 2^{k-2} \cdot p_{j1}^e \leq p_{j(k-1)}^e \leq s_i \cdot d \leq s_1 \cdot d$$

which implies

$$2^{k-1} \leq \frac{s_1}{s_{sup}} \implies k \leq [\log \frac{s_1}{s_{sup}} + 1]$$

We show, by definition of *sup*, that $\frac{s_1}{s_{sup}} \leq m$. Recall *sup* $\leftarrow \min\{k \mid \sum_{i=1}^{k} s_i \geq \frac{1}{2} \sum_{i=1}^{m} s_i, 1 \leq k \leq m\}$. If $s_1 \geq \frac{1}{2} \sum_{i=1}^{m} s_i$, then *sup* $= 1$ and apparently $\frac{s_1}{s_{sup}} = 1 \leq m$. From now on, we assume $s_1 < \frac{1}{2} \sum_{i=1}^{m} s_i$. Suppose, for contradiction, that $\frac{s_1}{s_{sup}} > m$. Then, $s_{sup+1}, ..., s_m \leq s_{sup} < \frac{s_1}{m}$; $\sum_{i=1}^{sup-1} s_i = \sum_{i=1}^{m} s_i - \sum_{i=sup}^{m} s_i > \sum_{i=1}^{m} s_i - m \cdot \frac{s_1}{m} > \frac{1}{2} \sum_{i=1}^{m} s_i$. This contradicts the definition of *sup*. It follows that $\frac{s_1}{s_{sup}} \leq m$. The number of inner procedure rounds, therefore, is at most $k + 1 \leq [\log \frac{s_1}{s_{sup}} + 2] \leq [\log m + 2]$. $\square$

Lemma III.9 (Bound on Inner Procedure Rounds)**.** *For a fixed makespan estimate d, the number of inner procedure rounds is bounded by* $\min\{[2 \log \eta + 2], [\log m + 2]\}$.

Proof. It immediately follows from Lemma III.7 and III.8. $\square$

Next we prove that every inner procedure incurs a cost of $O(1) \cdot C^*_{\max}$ in makespan. We consider the worst case when every job running on machines M_i ($i >$ *sup*) are canceled by some machines M_k ($k \leq$ *sup*), as if only the fastest *sup* machines are processing jobs. We first show that, for any job set, the optimal makespan achieved with just the fastest *sup* machines is at most $3 \cdot C^*_{\max}$. It then follows that Algorithm 3 completes the whole schedule if the makespan estimate $d \geq 6 \cdot C^*_{\max}$ and every inner procedure costs $O(1) \cdot C^*_{\max}$ time.

Lemma III.10 (Bound on Optimal Makespan with Just Fast Machines)**.** *Let the optimal makespan be $C^*_{\max}$. The optimal makespan achieved with just the fast machines $M_1, ..., M_{sup}$ is at most $3 \cdot C^*_{\max}$.*

Proof. It suffices to show that with just the fastest *sup* machines, there exists a schedule with makespan at most $3 \cdot C^*_{\max}$. The schedule is constructed as follows. For the first $2 \cdot C^*_{\max}$ time, run all the jobs arbitrarily on machines $M_1, ..., M_{sup}$: whenever a machine becomes idle, it selects a non-running job to process; the machine stays idle if every job has been completed. No jobs are canceled before time $2 \cdot C^*_{\max}$. At time $2 \cdot C^*_{\max}$, there must be at least one idle machine. Indeed, otherwise, if every fast machine is busy at time $2 \cdot C^*_{\max}$, the sum of job size exceed $\sum_{i=1}^{sup} 2 \cdot s_i \cdot C^*_{\max} > \sum_{i=1}^{m} s_i \cdot C^*_{\max}$, which contradicts the optimal makespan $C^*_{\max}$. If there is at least one idle machine at time $2 \cdot C^*_{\max}$, the number of uncompleted jobs is at most $sup - 1$. Without loss of generality, we rename these uncompleted jobs as $J_1, ..., J_r$ ($r < sup$). Cancel these jobs at time $2 \cdot C^*_{\max}$. Consider the optimal schedule with all the machines; in this schedule, jobs $J_1, ..., J_r$ must be assigned on some machines $M_{i1}, M_{i2}, ..., M_{il}$ with $i1 < i2 < ... < il$ and $il \leq r < sup$. With just the fast machines, we process these remaining jobs that are assigned to machine M_{i1} (in the optimal schedule) on machine M_1, jobs to machine M_{i2} on machine M_2, and so on. Formally, process the remaining jobs that are assigned to machine M_{ip} on machine M_p. Since $s_{ip} \geq s_p$, the time for processing these remaining jobs on the fast machines with this assignment is no more than $C^*_{\max}$. This constructs a (at most) $3 \cdot C^*_{\max}$-length schedule with just the fast machines $M_1, ..., M_{sup}$. □

Lemma III.11 (Bound on Makespan Estimate)**.** *The whole schedule completes after several rounds of the inner procedure if the makespan estimate $d \geq 6 \cdot C^*_{\max}$.*

Proof. The proof follows three observations. (1) For every problem instance I with optimal makespan C, if we double every job size, the newly created (doubled) problem instance has optimal makespan at most $2 \cdot C$. (2) When an inner procedure terminates, it follows $p^e_j < 2 \cdot p^*_j$ for every uncompleted job J_j. (3) The makespan estimate is doubled only when the algorithm confirms that no d-length schedule exists with just the fast machines $M_1, ..., M_{sup}$.

Observation (1) is trivial. If every job size is doubled, the optimal schedule is also doubled. Thus, the optimal makespan for the newly created problem is at most $2 \cdot C$. Observation (2) follows from the proof of Lemma III.7. Suppose, with a fixed makespan estimate d, the job size estimates undergo k doubling rounds. We have shown that after the $(k-1)$-th round, any uncompleted job J_j has $p^e_j < p^*_j$. Then, after the final doubling round, it follows that $p^e_j < 2 \cdot p^*_j$ for any uncompleted job J_j. Observation (3) is supported by similar arguments in Theorem III.6. If machine M_1 processes any job for more than d time, no valid d-length schedule exists with just the fast machines $M_1, ..., M_{sup}$. If there is an uncompleted job J_j not started by any machine M_i with $i \leq sup$ and $p^e_j \leq s_i \cdot d$ before time d, every machine $M_1, ..., M_q$ (q is the largest index such that $p^e_j \leq s_q \cdot d$ and $q \leq sup$) must be busy processing the jobs that cannot complete on machine M_{q+1} or slower before time d, which implies that no valid d-length schedule exists with just the fast machines $M_1, ..., M_{sup}$. Observation (3), therefore, holds.

Now we combine these observations. First, by Lemma III.10, there exists a schedule of length at most $3 \cdot C^*_{\max}$ with just the fast machines $M_1, ..., M_{sup}$. At any time, we have $p^e_j < 2 \cdot p^*_j$ for every uncompleted job (observation (2)); there exists, if every job J_j has size p_j^e, a schedule of length at most $6 \cdot C^*_{\max}$ with just the fast machines $M_1, ..., M_{sup}$ (observation (1)). Then, if the makespan estimate $d \geq 6 \cdot C^*_{\max}$, the algorithm will not double d in the following inner procedure rounds (observation (3)); subsequently, after a bounded number of inner procedure rounds (Lemma III.9), the algorithm will complete the whole schedule. □

It immediately follows that, since an inner procedure lasts at most $2d$ time, an inner procedure costs at most $12 \cdot C^*_{\max}$ time. By putting all the above lemmas together, we obtain the performance bound for Algorithm 3.

Theorem III.12 (Performance Bound for Algorithm 3)**.** *Given the job size predictions p_j and the prediction error $\eta = \max_{1 \leq j \leq n} \max\{\frac{p_j}{p_j^*}, \frac{p_j^*}{p_j}\}$, Algorithm 3 computes a schedule with makespan $C_{\max}$ at most $48 \cdot \min\{[2\log\eta + 2], [\log m + 2]\} \cdot C^*_{\max}$, where $C^*_{\max}$ denotes the optimal makespan.*

Proof. Initially, the algorithm sets the makespan estimate $d \leftarrow d_0 := \frac{\max_{1 \leq j \leq n} p_j}{\eta \cdot s_1}$ and executes the inner procedure. Clearly $d_0 \leq C^*_{\max}$. Suppose the algorithm will undergo x rounds of doubling makespan estimate with $d = 2d_0, 2^2 d_0, ..., 2^x d_0$. Fixing a makespan estimate d, the number of inner procedure rounds is at most $\min\{[2\log\eta + 2], [\log m + 2]\}$ by Lemma III.9, with each spending at most $2d$ time. Observe that $2^{x-1} d_0 < 6 \cdot C^*_{\max}$, since otherwise, it would have completed the whole schedule before the x-th round by Lemma III.11. Thus, the makespan of the entire schedule is bounded by

$$\begin{aligned} C_{\max} &\leq 2 \cdot (d_0 + ... + 2^x d_0) \cdot \min\{[2\log\eta + 2], [\log m + 2]\} \\ &< 2 \cdot (2^{x+1} d_0) \cdot \min\{[2\log\eta + 2], [\log m + 2]\} \\ &< 48 \cdot \min\{[2\log\eta + 2], [\log m + 2]\} \cdot C^*_{\max} \end{aligned}$$ □

Remark III.13. *It follows from Theorem III.12 that Algorithm 3 has an $O(\min\{\log\eta, \log m\})$ competitive ratio. Let $C_{\max}$ be the makespan of the produced schedule. With $\eta = 1$, we have $C_{\max} < 96 \cdot C^*_{\max}$; the algorithm is 96-consistent or $O(1)$-consistent. With $\eta = \infty$, we have $C_{\max} < 48 \cdot [\log m + 2] \cdot C^*_{\max}$; the algorithm is $48 \cdot [\log m + 2]$-robust or $O(\log m)$-robust. Finally, under any $\eta \geq 1$, we have $C_{\max} < 48 \cdot \min\{[2\log\eta + 2], [\log m + 2]\} \cdot C^*_{\max}$; the algorithm is $48 \cdot \min\{[2\log\eta + 2], [\log m + 2]\}$-smooth or $O(\min\{\log\eta, \log m\})$-smooth.*

Theorem III.14 (Optimality of Consistency and Robustness for Algorithm 3)**.** *Algorithm 3 has asymptotically optimal consistency and asymptotically optimal robustness.*

Proof. The lower bound for the competitive ratio of any clairvoyant algorithm is $\Omega(1)$. The lower bound for the competitive ratio of any non-clairvoyant algorithm is $\Omega(\log m)$. By the Fundamental Theorems of Consistency (Theorem II.4) and the Fundamental Theorems of Robustness (Theorem II.6), Algorithm 3 has consistency and robustness both matching the corresponding lower bound. □

Note that, even in the worst case of $\eta = \infty$, our algorithm returns a schedule of makespan at most $48 \cdot [\log m + 2] \cdot C^*_{\max}$, which outperforms the algorithm proposed in [27] with a $48 \cdot [\log m + 6] \cdot C^*_{\max}$ bound.

3.4.3 Performance Bounds for Algorithm 4

We prove the performance bound for Algorithm 4. Unknowing the prediction error, the algorithm estimates η in rounds. Our proof builds on the following key observation. With a fixed prediction error estimate η^e, the time spent in running an inner procedure round is bounded by $O(1) \cdot C^*_{\max}$ if $\frac{p_j}{\eta^e} > p^*_j$ for some job J_j and is bounded by $O(\min\{\log \eta, \log m\}) \cdot C^*_{\max}$ otherwise. To bound overall performance, we maintain $(\eta^e)^2 < m$ but switch to the $O(\log m)$-competitive algorithm if $(\eta^e)^2 \geq m$. With such a strategy, the makespan can be bounded by $O(\log m) \cdot C^*_{\max}$ even if η is arbitrarily large.

Lemma III.15 (Bound on Time with Fixed η^e for Case of $\frac{p_j}{\eta^e} > p^*_j$ for some j)**.** *Fix η^e with $\frac{p_j}{\eta^e} > p^*_j$ for some uncompleted job J_j. The time spent in running Algorithm 3 is at most $24 \cdot C^*_{\max}$.*

Proof. Let $J_{j'}$ denote the largest overestimated job, i.e., $\frac{p_{j'}}{\eta^e} > p^*_{j'}$ with $\frac{p_{j'}}{\eta^e}$ maximal. We show that, with a fixed prediction error estimate η^e and a makespan estimate d, an inner procedure either detects an underestimation of η^e or doubles d until $d \geq 3 \cdot C^*_{\max}$. With every inner procedure round spending at most $2d$ time, the total time spent is then bounded by $O(1) \cdot C^*_{\max}$.

First consider a special case when $\frac{p_{j'}}{\eta^e} = \frac{\max_{1\leq j\leq n} p_j}{\eta^e} \geq s_1 \cdot C^*_{\max}$. In such case, the first makespan estimate $d := d_0 = \frac{p_{j'}}{s_1 \eta^e} \geq C^*_{\max}$. By Algorithm 3, machine M_1 first selects an overestimated job to process; then, the algorithm confirms an underestimation of η^e within at most $\frac{\max_{1\leq j\leq n} p^*_j}{s_1} \leq C^*_{\max}$ time. The prediction error estimate η^e is then doubled, and the time spent is, apparently, no more than $24 \cdot C^*_{\max}$. From now on, we assume that $\frac{p_{j'}}{\eta^e} \leq s_1 \cdot C^*_{\max}$ so the first makespan estimate $d := d_0 \leq C^*_{\max}$.

Under a fixed η^e, Algorithm 3 experiences several rounds of doubling makespan estimate d. We show that if $d < 3 \cdot C^*_{\max}$, an inner procedure either detects an underestimation of η^e or doubles d at the end — the job size estimates never get updated. We prove this by contradiction. Suppose the algorithm updates the job size estimates at the end of an inner procedure round. Then, job $J_{j'}$ must run on a machine M_i with $i \leq \mathit{sup}$ and $p^e_{j'} \leq s_i \cdot d$ before time d. However, $p^*_{j'} < \frac{p_{j'}}{\eta^e} \leq p^e_{j'} \leq s_i \cdot d$, indicating that job $J_{j'}$ should have been completed and that an underestimation of η^e is confirmed, which is a contradiction.

Next we show that Algorithm 3 must detect an underestimation of η^e if $d \geq 3 \cdot C^*_{\max}$. We prove, by contradiction, that job $J_{j'}$ must start running on a machine M_i with $i \leq \mathit{sup}$ and $p^e_{j'} \leq s_i \cdot d$ before time d. Suppose job $J_{j'}$ fails to do so. Then, every machine $M_1, ..., M_q$, where q is the largest machine index such that $p^e_{j'} \leq s_q \cdot d$ and $q \leq \mathit{sup}$, must be busy processing, before time d, the (non-overestimated) jobs with job size estimate at least $p^e_{j'}$; these jobs have size at least $p^e_{j'}$ and cannot be processed on a slower machine if we aim to achieve makespan d with just the fast machines $M_1, ..., M_{\mathit{sup}}$. The existence of $J_{j'}$ and function *getJob* ensure that no job is canceled on the first q machines before time d. Then, the sum of the size of the jobs that, if we aim to achieve makespan d with just the fast machines $M_1, ..., M_{\mathit{sup}}$, must be processed on machines M_i $(i \leq q)$ is already more than $\sum_{i=1}^{q} s_i \cdot d$. This, however, contradicts Lemma III.10, which shows that there exists a schedule to complete all jobs in $3 \cdot C^*_{\max}$ time with just the fast machines $M_1, ..., M_{\mathit{sup}}$. Thus, job $J_{j'}$ must start running on a machine M_i with $i \leq \mathit{sup}$ and $p^e_{j'} \leq s_i \cdot d$ before time d; since $p^*_{j'} < p^e_{j'} \leq s_i \cdot d$, job $J_{j'}$ is completed before time $2d$. Upon completing $J_{j'}$, Algorithm 3 detects an underestimation of η^e.

Finally, suppose Algorithm 3 terminates after x rounds of doubling d with $d = 2d_0, 2^2d_0, ..., 2^xd_0$, where $d_0 = \frac{\max_{1\leq j\leq n} p_j}{s_1}$. With a fixed makespan estimate d, the number of inner procedure rounds is at most 1. Each round spends at most $2d$ time. Observe that $2^{x-1}d_0 < 3 \cdot C^*_{\max}$. Thus, before confirming an underestimation of η^e, the total time spent in running Algorithm 3 is at most

$$2 \cdot (d_0 + ... + 2^x d_0) < 24 \cdot C^*_{\max}$$

□

Lemma III.16 (Bound on Time with Fixed η^e for Case of $\frac{p_j}{\eta^e} \leq p^*_j$ for all j)**.** *Fix η^e with $\frac{p_j}{\eta^e} \leq p^*_j$ for every uncompleted job J_j. Algorithm 3 completes the whole schedule with makespan at most* $48 \cdot \min\{[2\log\eta^e + 2], [\log m + 2]\} \cdot C^*_{\max}$.

Proof. The lemma holds by the same arguments for Theorem III.12. All bounds apply. □

Theorem III.17 (Performance Bound for Algorithm 4)**.** *Given only the job size predictions p_j, Algorithm 4 computes a schedule of makespan $C_{\max}$ at most* $\min\{[120\log\eta + 216], [60\log m + 216]\} \cdot C^*_{\max}$*, where $C^*_{\max}$ denotes the optimal makespan and η denotes the prediction error and* $\eta = \max_{1\leq j\leq n}\max\{\frac{p_j}{p^*_j}, \frac{p^*_j}{p_j}\}$.

Proof. First, assume that the algorithm returns the whole schedule when $(\eta^e)^2 < m$. Suppose the algorithm undergoes x rounds of doubling η^e with $\eta^e = 1, 2, ..., 2^x$. There must be a job J_j with $\frac{p_j}{\eta^e} > p^*_j$ when $\eta^e = 1, 2, ..., 2^{x-1}$, since otherwise the algorithm would experience fewer rounds of doubling η^e. Every round spends at most $24 \cdot C^*_{\max}$ time by Lemma III.15. Thus, the first x rounds spend at most $24 \cdot x \cdot C^*_{\max}$ time. After setting $\eta^e = 2^x$, the algorithm, by Lemma III.16, completes the schedule within $48 \cdot \min\{[2\log 2^x + 2], [\log m + 2]\} \cdot C^*_{\max} = 48 \cdot (2x + 2) \cdot C^*_{\max}$ time (since $(\eta^e)^2 < m$). Observe that $2^{x-1} < \eta$. Thus, the makespan of the whole schedule, if returned when $(\eta^e)^2 < m$, is at most

$$24 \cdot x \cdot C^*_{\max} + 48 \cdot (2x + 2) \cdot C^*_{\max} \leq [120\log\eta + 216] \cdot C^*_{\max}$$

Next, assume that the algorithm returns the whole schedule by running Algorithm 3 with $p_j = 1$ for every uncompleted job J_j and $\eta = \infty$. Suppose the algorithm undergoes $x - 1$ rounds of doubling η^e with $\eta^e = 1, 2, ..., 2^{x-1}$ before breaking the while loop. Every round spends at most $24 \cdot C^*_{\max}$ time. Thus, the while loop spends at most $24 \cdot x \cdot C^*_{\max}$ time. With $(2^{x-1})^2 < m$, it follows that $24 \cdot x \cdot C^*_{\max} < (12\log m + 24) \cdot C^*_{\max}$. After breaking the while loop, running Algorithm 3 costs at most $48 \cdot [\log m + 2] \cdot C^*_{\max}$. Thus, the makespan of the whole schedule, if returned when $(\eta^e)^2 \geq m$, is at most

$$24 \cdot x \cdot C^*_{\max} + 48 \cdot [\log m + 2] \cdot C^*_{\max} \leq [60\log m + 120] \cdot C^*_{\max}$$

The whole schedule, therefore, has makespan $C_{\max}$ at most $\min\{[120\log\eta + 216], [60\log m + 216]\} \cdot C^*_{\max}$. □

Remark III.18. *It follows from Theorem III.17 that Algorithm 4 has an $O(\min\{\log\eta, \log m\})$ competitive ratio. The algorithm is 216-consistent or $O(1)$-consistent, $[60\log m + 120]$-robust or $O(\log m)$-robust, and $\min\{[120\log\eta + 216], [60\log m + 216]\}$-smooth or $O(\min\{\log\eta, \log m\})$-smooth.*

Theorem III.19 (Optimality of Consistency and Robustness for Algorithm 4)**.** *Algorithm 4 has asymptotically optimal consistency and asymptotically optimal robustness.*

Proof. The theorem holds by the same arguments for Theorem III.14, the Fundamental Theorems of Consistency (Theorem II.4), and the Fundamental Theorems of Robustness (Theorem II.6). □

3.4.4 Performance Bounds for Algorithms 5 and 6

We extend the static scheduling results to derive performance bounds for Algorithms 5 and 6. The heart of the proof is that any competitive static scheduling algorithm can construct, with a negligible cost in makespan, a competitive dynamic scheduling algorithm via rerunning itself upon every job release. We prove this claim via the following series of lemmas, followed by the competitiveness results.

Lemma III.20 (Bound on Makespan for a Subset of Jobs)**.** *Let $C^*_{\max}$ denote the optimal makespan for a set of jobs with no release times and $C'^*_{\max}$ denote that for any subset of these jobs. We have $C'^*_{\max} \le C^*_{\max}$.*

Proof. The optimal schedule can construct a new schedule by keeping (from the schedule) just the jobs in the subset. The constructed schedule has a makespan at most $C^*_{\max}$. By optimality of $C'^*_{\max}$, we have $C'^*_{\max} \le C^*_{\max}$. □

Lemma III.21 (Bound on Optimal Static Makespan)**.** *Let $C^*_{\max}$ denote the optimal makespan for a set of jobs with arbitrary release times and $C^{\{0\}*}_{\max}$ the optimal makespan if we ignore the release times. We have $C^{\{0\}*}_{\max} \le C^*_{\max}$.*

Proof. The optimal schedule for the jobs with arbitrary release times is also a valid schedule for these jobs if we ignore release times. By optimality of $C^{\{0\}*}_{\max}$, we have $C^{\{0\}*}_{\max} \le C^*_{\max}$. □

Lemma III.22 (Bound on Release Time)**.**

$$\max_{1\le j\le n} r_j \le C^*_{\max}$$

where $C^*_{\max}$ denotes the optimal makespan for a set of jobs with arbitrary release times.

Proof. A job can complete no earlier than its release time. Thus, the optimal makespan is no less than the latest release time. □

We now prove the key result for the rerunning strategy by combining the above lemmas.

Theorem III.23 (Performance Bound for Rerunning Strategy)**.** *Given a static scheduling algorithm A with competitive ratio α, construct a dynamic scheduling algorithm A^+ by rerunning A against the uncompleted jobs upon every job release. It follows that A^+ has a competitive ratio of $\alpha + 1$ for dynamic scheduling.*

Proof. Fix any dynamic scheduling problem instance with jobs $J_1, ..., J_n$. Run A^+ against this instance. Let $C^*_{\max}$ denote the optimal makespan. Let $r_{\max}$ denote the latest release time, i.e., $r_{\max} = \max_{1\le j\le n} r_j$. By Algorithm A^+, A will run from the stretch against the present uncompleted jobs when the last job arrives. Consider the set of uncompleted jobs at $r_{\max}$ immediately after the last job arrives.

Let $C'^*_{\max}$ denote the optimal makespan for these jobs if we ignore release times and $C^{\{0\}*}_{\max}$ the optimal makespan for the whole set of jobs $J_1, ..., J_n$ if we ignore release times. Combining Lemma III.20 and III.21, we have

$$C'^*_{\max} \leq C^{\{0\}*}_{\max} \leq C^*_{\max}$$

If $C_{\max}$ is the makespan of the schedule produced by A^+, we have, by the competitiveness of A, that

$$C_{\max} \leq r_{\max} + \alpha \cdot C'^*_{\max} \leq C^*_{\max} + \alpha \cdot C'^*_{\max} \leq (\alpha + 1) \cdot C^*_{\max}$$

where the second inequality holds by Lemma III.22. Therefore, A^+ has competitive ratio $\alpha + 1$ for dynamic scheduling. □

Theorem III.24 (Performance Bound for Algorithm 5)**.** *Given the job size predictions p_j and the prediction error $\eta = \max_{1 \leq j \leq n} \max\{\frac{p_j}{p^*_j}, \frac{p^*_j}{p_j}\}$, Algorithm 5 computes a schedule with makespan $C_{\max}$ at most $(48 \cdot \min\{[2\log\eta + 2], [\log m + 2]\} + 1) \cdot C^*_{\max}$, where $C^*_{\max}$ denotes the optimal makespan.*

Proof. It immediately follows from Theorem III.12 and III.23. □

Theorem III.25 (Performance Bound for Algorithm 6)**.** *Given only the job size predictions p_j, Algorithm 6 computes a schedule of makespan $C_{\max}$ at most $\min\{[120\log\eta + 217], [60\log m + 217]\} \cdot C^*_{\max}$, where $C^*_{\max}$ denotes the optimal makespan and η denotes the prediction error and $\eta = \max_{1 \leq j \leq n} \max\{\frac{p_j}{p^*_j}, \frac{p^*_j}{p_j}\}$.*

Proof. It immediately follows from Theorem III.17 and III.23. □

Remark III.26. *It follows from Theorem III.24 that Algorithm 5 has an $O(\min\{\log\eta, \log m\})$ competitive ratio. The algorithm is 97-consistent or $O(1)$-consistent, $(48[\log m] + 97)$-robust or $O(\log m)$-robust, and $(48 \cdot \min\{[2\log\eta + 2], [\log m + 2]\} + 1)$-smooth or $O(\min\{\log\eta, \log m\})$-smooth. It follows from Theorem III.25 that Algorithm 6 has an $O(\min\{\log\eta, \log m\})$ competitive ratio. The algorithm is 217-consistent or $O(1)$-consistent, $[60\log m + 121]$-robust or $O(\log m)$-robust, and $\min\{[120\log\eta + 217], [60\log m + 217]\}$-smooth or $O(\min\{\log\eta, \log m\})$-smooth.*

Theorem III.27 (Optimality of Consistency and Robustness for Algorithms 5 and 6)**.** *Algorithms 5 and 6 have asymptotically optimal consistency and asymptotically optimal robustness.*

Proof. The theorem holds by the same arguments for Theorem III.14, the Fundamental Theorems of Consistency (Theorem II.4), and the Fundamental Theorems of Robustness (Theorem II.6). □

3.4.5 Correctness

We give the correctness proof for the proposed Algorithms 3 — 6.

Theorem III.28 (Correctness of Algorithms 3 — 6)**.** *Both of Algorithms 3 and 4 construct a valid schedule for $Qm \mid$ online-time-nclv, pmtn-restart $\mid C_{\max}$ (static scheduling). Both of Algorithms 5 and 6 construct a valid schedule for $Qm \mid$ online-time-nclv, pmtn-restart, $r_j \mid C_{\max}$ (dynamic scheduling).*

Proof. The correctness follows the established bounds. Pick Algorithm 3 or 4 or 5 or 6. The algorithm terminates within a bounded number of inner procedure rounds. Every round terminates within a bounded time by Theorem III.6. Therefore, the algorithm terminates in a bounded time; by the termination condition that no job is left uncompleted, a valid schedule is constructed upon the termination. □

3.4.6 Complexity Analysis

Finally, we give the complexity results for the proposed algorithms. Maintaining uncompleted jobs in ordered sets sorted by job size estimate, both of the function *getJob* and event function *MachineIdle* take $O(\log n)$ time per call. At the end of an inner procedure when *TimeOut* is triggered, doubling makespan estimate takes $O(1)$ time, doubling job size estimates $O(\mathit{sup}) \subseteq O(m)$ time, and idling all machines $O(m \log n)$ time. Both Algorithms 5 and 6 rerun the static counterpart upon every job release, which needs to idle all machines and thus costs $O(m \log n)$ time. Moreover, we can achieve these time complexities with $O(n)$ space.

Theorem III.29 (Time and Space Complexity)**.** *The proposed algorithms admit an implementation with complexities:*

(1) Function getJob takes $O(\log n)$ time per call.
(2) Event function MachineIdle takes $O(\log n)$ time per call.
(3) Event function TimeOut takes $O(m \log n)$ time per call.
(4) Event function JobRelease takes $O(m \log n)$ time per call.
(5) $O(n)$ space is used.

Proof. Consider three partitions of uncompleted jobs: non-running jobs, jobs running on machines $M_{\mathit{sup}+1}, ..., M_m$, and jobs running on machines $M_1, ..., M_{\mathit{sup}}$ but the machine is not fast enough to complete it in makespan estimate time. Maintain each partition as a balanced binary search tree (BST) sorted by job size estimate. Apparently, the data structures use $O(n)$ space.

The function *getJob* accesses at most 3 BSTs for searching a job, resulting in $O(\log n)$ time per call. Event function *MachineIdle* may delete a job from a BST (when completing the job), call *getJob*, and move a job from one BST to another (due to an update in the job status), resulting in $O(\log n)$ time per call. Event function *TimeOut* either updates the makespan estimate ($O(1)$ time) or the job size estimates ($O(\mathit{sup}) \subseteq O(m)$ time as this happens only if there are at most *sup* jobs). Finally, *TimeOut* idles all machines, which moves $O(m)$ jobs from one BST to the BST of non-running jobs, resulting in $O(m \log n)$ time per call. Event function *JobRelease* reruns the static counterpart, which must idle all machines first, resulting in $O(m \log n)$ time per call. □

Remark III.30. *Theorem III.29 shows the complexity per call for every (event) function. At any time instant, there are at most n jobs moving between the BSTs. Thus, the real-time complexity for any of the proposed algorithms is bounded by $O(n \log n)$ at any time.*

3.5 Experimental Evaluation

This section performs an experimental evaluation to show the competitiveness of the proposed algorithms. For static scheduling, we implement four algorithms: (1) an offline near-optimal algorithm based on Algorithm 2, (2) the $O(\log m)$-competitive algorithm [27], (3) Algorithm 3 with a known prediction error, and (4) Algorithm 4 without a known error. We denote these algorithms by A_{opt}, A_{classic}, $A_{\text{knowError}}$, and $A_{\text{unknowError}}$. Finding the optimal solution is computationally infeasible due to NP-hardness. Instead, we implement A_{opt} with a bisection method on Algorithm 2 as a 2-approximation; this gives us a baseline for the other algorithms. The reported performance ratios are, therefore, approximations of the analytical ones — every reported performance ratio is no less than half of the analytical competitive ratio.

For dynamic scheduling, we extend the static scheduling algorithms via the rerunning strategy to deal with release times. We end up with four dynamic scheduling algorithms $A^{+}{}_{\text{opt}}$, $A^{+}{}_{\text{classic}}$, $A^{+}{}_{\text{knowError}}$, and $A^{+}{}_{\text{unknowError}}$. Each algorithm reruns the static counterpart upon every job release. As the rerunning strategy incurs an additive cost of one optimal makespan (Theorem III.23), the baseline $A^{+}{}_{\text{opt}}$ turns into a 3-approximation. Thus, every reported performance ratio, for dynamic scheduling, is no less than a third of the analytical one.

3.5.1 Workload Generation

Parameter	Range	Default value
number of jobs (n)	256	256
number of machines (m)	$[2, 128]$	32
job size of J_j (p_j^*)	$[1, 1024]$	N/A
processing speed of M_i (s_i)	$[1, 64]$	N/A
release time of J_j (r_j)	$[0, 64]$	0
total prediction error (η)	$[1, \infty)$	N/A
job size prediction of J_j (p_j)	$[\frac{p_j^*}{\eta}, \eta \cdot p_j^*]$	N/A

Table III.2: The workload parameters, with time-related ones in milliseconds.

We generate 140000 problem instances to evaluate, under different settings, the performance ratios of the proposed algorithms. Table III.2 lists the range and the default value of every parameter in workload generation. The number of jobs (n) is fixed at 256 to observe performance variations caused by other variables: prediction error (η), the number of machines (m), and release times (r_j). Our experiment turns every problem instance with arbitrary release times, i.e., every dynamic scheduling instance, into a static scheduling instance by ignoring the release times — by doing so, we can observe the effect of introducing release times. With a fixed prediction error η, we randomly generate a job size prediction for every job J_j (with actual size p_j^*) from $[\max\{1, \frac{p_j^*}{\eta}\}, \eta \cdot p_j^*]$. To minimize measurement deviation caused by randomly generated predictions, every problem instance is executed 50 times. We evaluate both the average-case execution performance and the worst-case execution performance.

3.5.2 Experimental Results: Static Scheduling

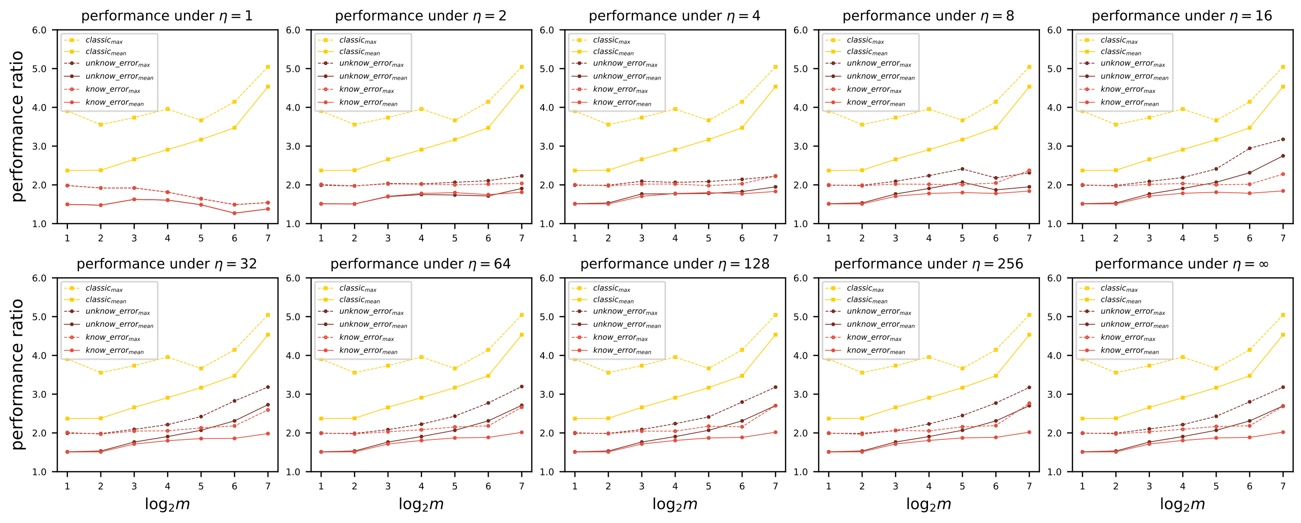


Figure III.3: Performance ratios for static scheduling algorithms with varying m and η. Every algorithm has two figures: the dashed line representing the maximum performance ratio for all problem instances and the solid line representing the mean. The figures for the max performance ratio have observable fluctuations as they represent the extremes of the data and thus are sensitive to workload generation.

Figure III.3 shows the performance comparisons between A_{classic}, $A_{\text{knowError}}$, and $A_{\text{unknowError}}$ under increasing machines m and prediction error η. We observe that performance ratios generally increase as we increase machines. The intuition is that, for any job set, additional machines allow the optimal scheduler to show a greater advantage in job allocation and complete the jobs as soon as possible; thus, additional machines lower the relative performance of the other algorithms. The experimental results show that $A_{\text{knowError}}$ and $A_{\text{unknowError}}$ consistently outperform A_{classic} in both mean and max performance ratios, even under arbitrarily bad predictions, which confirms that our improved 2-relaxed procedure ensures the worst-case performance of our algorithms is better than the performance of A_{classic}. Unlike A_{classic}, both $A_{\text{knowError}}$ and $A_{\text{unknowError}}$ have a small gap between the max (worst-case) performance and the mean (average-case) performance. Our algorithms, therefore, have a slight performance deviation: the max performance ratios are just a constant distance away from the mean. In addition, the performance ratios of our algorithms increase sub-linearly as $\log m$ increases, revealing the theoretical result of $O(\min\{\log \eta, \log m\})$-competitive ratio. At almost all times, our algorithms bound the performance ratio by small constants: their performance is within three times the optimum.

Next we discuss the performance difference between $A_{\text{knowError}}$ and $A_{\text{unknowError}}$. They have the same performance under $\eta = 1$. If the predictions are perfect, the performance no longer depends on the machine number, as the competitive ratio $O(\min\{\log \eta, \log m\}) = O(\log \eta) = O(1)$ turns to a constant regardless of m. Indeed, the experimental results (under $\eta = 1$) show that, under $\eta = 1$, the performance ratios for both $A_{\text{knowError}}$ and $A_{\text{unknowError}}$ are bounded by 2.0 at all times. On the other extreme of $\eta = \infty$, $A_{\text{knowError}}$ outperforms $A_{\text{unknowError}}$ since the latter spends extra time in estimating η before realizing that η is too large and that it is better to switch to the $O(\log m)$-competitive algorithm by setting $\eta^e = \infty$.

Regardless of η, the performance of $A_{\text{knowError}}$ and $A_{\text{unknowError}}$ stay close to each other when m is small ($m \leq 8$). This is because, even under large η, when one may expect that $A_{\text{unknowError}}$ should have wasted significant time in realizing the large prediction error, the performance is bounded by the $\log m$ term. We also observe that knowing the error is favorable only if both η and m are large. If η is small ($\eta \leq 4$), the performance of $A_{\text{knowError}}$ and $A_{\text{unknowError}}$ stay close to each other as $A_{\text{unknowError}}$ requires only a few more rounds of doubling η to estimate the actual prediction error precisely.

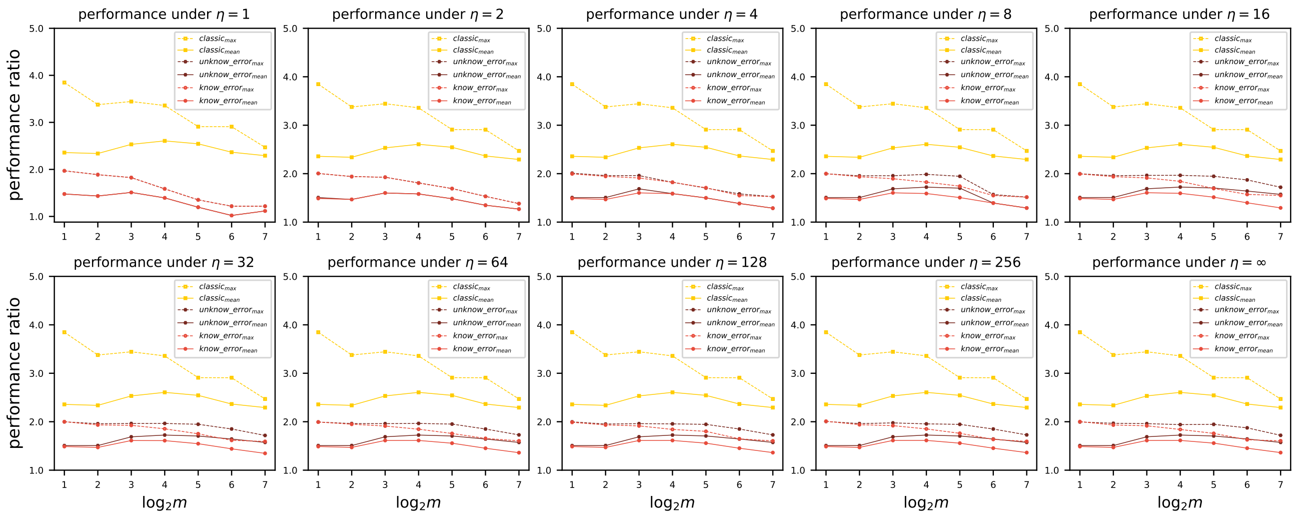


Figure III.4: Performance ratios for dynamic scheduling algorithms with varying m and η.

3.5.3 Experimental Results: Dynamic Scheduling

This section concerns dynamic scheduling. We observe the effect of release times and the performance of Algorithms 5 and 6 ($A^{+}_{\text{knowError}}$ and $A^{+}_{\text{unknowError}}$). Figure III.4 shows the performance comparisons in dynamic scheduling.

A static scheduling instance, if we enforce a release time to every job, is "relaxed" in that a near-optimal performance becomes easier. Direct evidence is that every dynamic scheduler, shown in Figure III.4, achieves a lower performance ratio than the static counterpart. The intuition is that release times force the scheduler to wait until $r_{\max}$, the latest release time, before completing all jobs. Suppose, for scheduling jobs except for the last, A^{+}_{opt} uses t time, but another algorithm A^{+} uses $t + \Delta t$ time ($\Delta t > 0$); if $r_{\max} \geq t + \Delta t$, both algorithms need to wait, after completing all the other jobs, for the last job to arrive and A^{+}_{opt}, in turn, loses the advantage of a Δt earlier completion; even if $t < r_{\max} < t + \Delta t$, release times weaken the advantage of A^{+}_{opt} from a Δt earlier completion to a $t + \Delta t - r_{\max}$ ($< \Delta t$) earlier completion. The previous Figure III.2 shows the case where the advantage of an optimal scheduler is weakened by the release time of job J_6. If we have more machines, the release times weaken the advantage of A^{+} further: additional machines reduce not only t but also Δt, increasing the chance for $r_{\max} \geq t + \Delta t$ or $t < r_{\max} < t + \Delta t$ to happen. As a result, we see from Figure III.4 that performance ratios decrease as we increase machines, revealing the effect of release times.

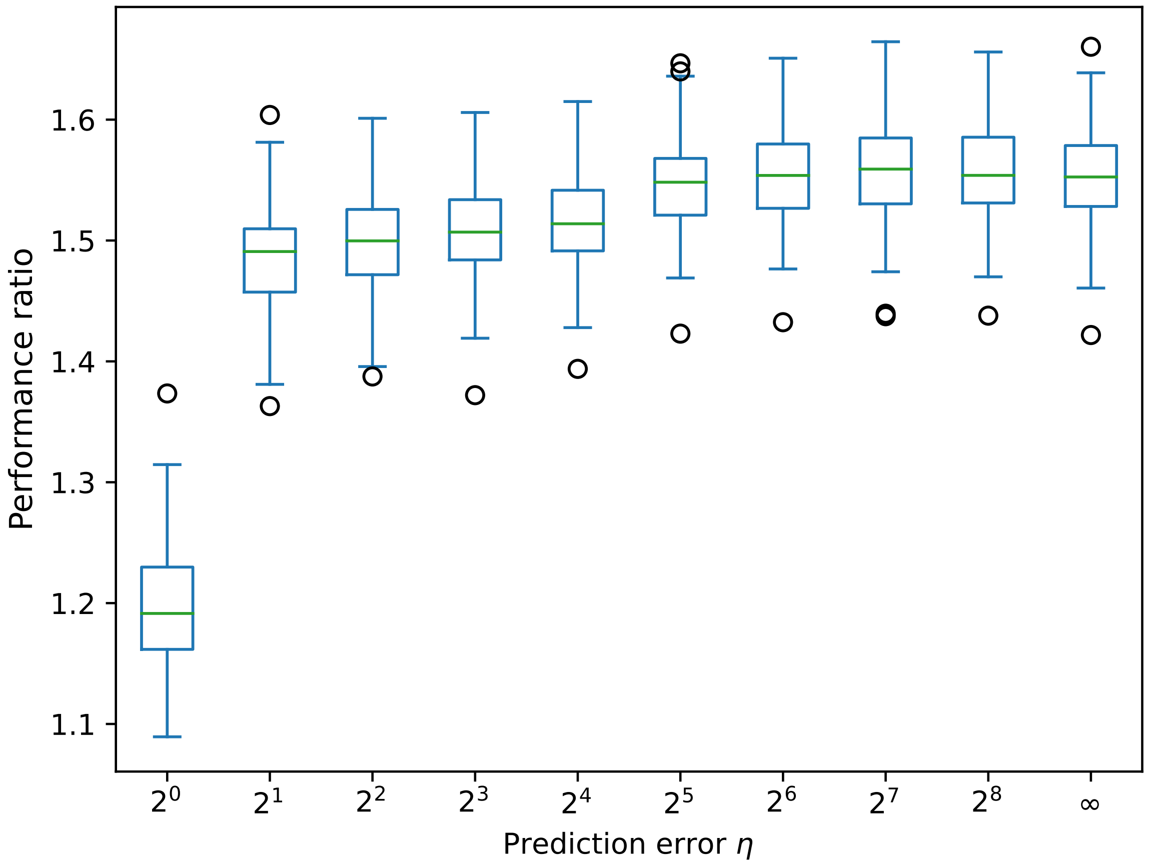


Figure III.5: Performance ratio of Algorithm 5 with varying η. The horizontal bars across the middle of the boxes represent the median of each boxplot. The box covers the middle 50% of the data points, while the lower and upper boundary of the box represents the 25th (75th) percentile.

In addition to the observation of the release times effect, the data display the same comparison results as those observed in static scheduling; they are: (1) $A^+{}_{\text{knowError}}$ and $A^+{}_{\text{unknowError}}$ consistently outperforms $A^+{}_{\text{classic}}$, even under arbitrarily bad predictions, (2) our algorithms have a slight performance deviation, (3) knowing the error η is favorable only if both η and m are large, to list three.

To take a closer look at how η affects the performance, we fix the machine number m at 32 and randomly sample 100 problem instances to plot the performance ratios of $A^+{}_{\text{knowError}}$ and $A^+{}_{\text{unknowError}}$ with increasing η (Figures III.5 and III.6). Algorithm 5 experiences a performance drop from $\eta = 1$ to $\eta = 2$; after $\eta \geq 2$, the performance ratio increases very slightly as η increases — the algorithm is crying for high-quality predictions. Nevertheless, the performance ratio is bounded approximately between 1.4 and 1.6 under $\eta = \infty$, showing the robustness of the algorithm under bad predictions. In comparison, Algorithm 6 experiences a two-stage performance drop. The first drop happens when predictions change from perfect ($\eta = 1$) to reasonable ($2 \leq \eta \leq 4$); the second drop happens when predictions change from reasonable to poor ($\eta \geq 8$). If the predictions have reasonable accuracy ($\eta \in [2, 4]$), $A^+{}_{\text{unknowError}}$ bounds the performance ratio approximately between 1.4 and 1.6, which is similar to the performance of $A^+{}_{\text{knowError}}$; with reasonable predictions, unknowing the prediction error

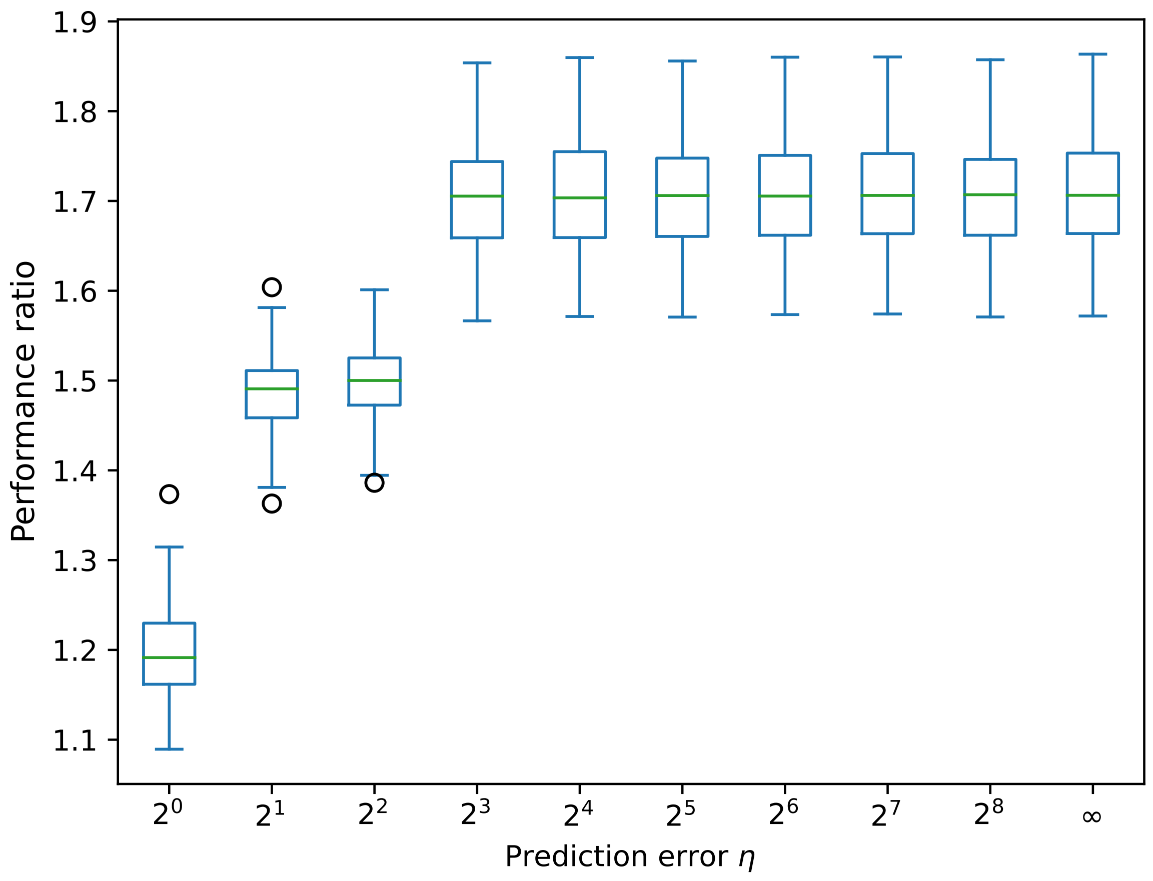


Figure III.6: Performance ratio of Algorithm 6 with varying η.

does not degrade, compared with knowing the error, the performance much. If the predictions have poor accuracy ($\eta \geq 8$), unknowing the error comes with a high cost. Algorithm 6 still attains a bounded performance ratio (approximately between 1.6 and 1.9) under bad predictions, but knowing the error can, in this case, improve the performance by about 16.67%.

3.6 Conclusions and Future Work

We study online scheduling with predictions and show how job size predictions improve the competitive ratio for makespan minimization in uniformly related machine scheduling. We first design an offline improved 2-relaxed decision procedure approximating the optimal schedule with given job sizes. The online algorithms use this procedure as a base. With a simple algorithm-independent prediction metric η and the decision procedure, we design an $O(\min\{\log \eta, \log m\})$-competitive ($O(\min\{\log \eta, \log m\})$-smooth) algorithm assuming a known prediction error for online static scheduling. Built upon this algorithm, we design a robust $O(\min\{\log \eta, \log m\})$-competitive ($O(\min\{\log \eta, \log m\})$-smooth) static scheduling algorithm that does not assume a known error. Both algorithms improve the known $\Omega(\log m)$ lower bound via the predictions. We extend these algorithms to address dynamic scheduling under the known and unknown prediction error. Both dynamic scheduling algorithms achieve an $O(\min\{\log \eta, \log m\})$

competitive ratio and $O(\min\{\log \eta, \log m\})$ smoothness. All the proposed algorithms with predictions achieve optimal consistency and optimal robustness. We prove the performance bounds and conduct extensive experiments to evaluate the algorithms.

Many interesting problems remain open in online optimization with predictions. An immediate direction is to extend this work to other online scheduling or optimization problems. It will be interesting to see how the proposed error metric and the algorithm design technique lead to improved competitiveness. Finally, with predictions exposing rich information to algorithms, finding the theoretical lower bounds for smoothness is another direction. These lower bounds will help us better understand the potential of predictions in managing uncertainty.

CHAPTER 4

Single Machine Response Time Scheduling with Predictions

This chapter considers the problem of online scheduling on a single machine to minimize the mean response time with job size predictions. The existing algorithm achieves 2-consistency to predictions, but no algorithm can simultaneously attain bounded robustness. We find a sufficient condition for any algorithm to achieve the optimal $O(P)$-robustness, where P denotes the maximum ratio of any two job sizes. We present the first algorithm that achieves, using the condition, optimal robustness up to a constant multiplicative factor and optimal consistency. Finally, for small prediction errors, we present an algorithm that we conjecture to achieve the optimal $O(\eta^2)$ competitive ratio ($O(\eta^2)$ smoothness), where η denotes the prediction error.

4.1 Introduction

Recent advance in learning theory has boosted the development of online scheduling with predictions [11, 25, 32]. In this work, we study minimizing the mean response time or, equivalently, the total response time, i.e., the sum of job service time, in single machine online non-clairvoyant scheduling with job size predictions. A recent work [25] considers this problem and develops a $2\eta^4$-competitive algorithm, where η is the prediction error. This algorithm is 2-competitive when the predictions are perfect ($\eta = 1$). However, it performs poorly in the worst-case scenario where the predictions are arbitrarily bad since $2\eta^4$ approaches ∞ while η approaches ∞. The algorithm has unbounded robustness. To the best of our knowledge, no algorithm can simultaneously achieve constant consistency and bounded robustness.

It is challenging to obtain near-optimal consistency and bounded robustness simultaneously. The intuition is that if an algorithm relies on the predictions heavily, it can perform well if the predictions are perfect but would fail quickly when the predictions are bad. On the other hand, if the algorithm has no trust in the predictions, it loses the benefits of quality predictions. We overcome this challenge by observing a sufficient condition for bounded robustness. The idea is that the performance is bounded if the time spent in processing the partial jobs, the ones that have started processing, is small. We achieve this condition by using the predictions conservatively: we intentionally underestimate the job sizes via job size predictions. Analysis shows that our approach uses predictions just enough to achieve optimal consistency and bounded robustness simultaneously.

We propose the first algorithm that is consistent and robust to the predictions. The proposed algorithm is $O(1)$-consistent and $O(P)$-robust, where P denotes the maximum ratio of any two job sizes. First, we prove a sufficient condition for any algorithm to achieve an $O(P)$ competitive ratio (Theorem IV.1), which is significant for proving the performance bound for our algorithm and designing robust algorithms in the future. Then, we design an algorithm that meets this condition by using the predictions to underestimate job sizes intentionally (Algorithm 7). This robustness result is optimal since any non-clairvoyant algorithm has an $\Omega(P)$ competitive ratio [33]. Finally, we consider the case when η is guaranteed to be reasonably small so that we allow the competitive ratio depends purely on η. We present an algorithm that we conjecture to achieve the optimal $O(\eta^2)$ competitive ratio.

4.2 Problem Definition

We study the response time minimization problem in single machine online non-clairvoyant scheduling. Assume n independent jobs arrive at the machine over time. Job J_j has arrival time r_j, but the machine does not know J_j until it arrives. Each job J_j also has a size p_j^*, but this value remains unknown until the job is completed. This setting is known as *non-clairvoyant* in literature. The jobs are priority-free and preemptive. For a schedule, denote C_j as the completion time of J_j so that the *response time* of J_j is $F_j = C_j - r_j$. Our objective is to minimize the *mean response time*, $\frac{1}{n}\sum_{j=1}^{n} F_j$, equivalently to minimize the total response time or $\sum_{j=1}^{n} F_j$ for any problem instance. Our scheduling problem can be defined as $1 \mid$ *online-time-nclv*, *pmtn* $\mid \sum F_j$ in Graham notation [9]. In addition, an oracle that provides a job size prediction for every job is accessible to the system. The scheduler is allowed to make decisions using the predictions. However, the prediction is possibly imperfect. Let the job size prediction for J_j be p_j. We define the prediction error for job J_j as $\eta_j = \max\{\frac{p_j}{p_j^*}, \frac{p_j^*}{p_j}\}$, and the total prediction error η as

$$\eta = \max_{1\le j\le n} \eta_j = \max_{1\le j\le n} \max\{\frac{p_j}{p_j^*}, \frac{p_j^*}{p_j}\}$$

Observe that $\eta \ge 1$ and $\frac{p_j}{\eta} \le p_j^* \le \eta \cdot p_j$ for any J_j. The prediction is perfect if and only if $\eta = 1$. Our proposed prediction error metric η for response time minimization is very similar to the one used in the previous work [25]. The quantity η is crucial in our work and will be used directly in the algorithm.

4.2.1 Consistency and Robustness

We evaluate the performance of the algorithms by the competitive framework [14] and the metrics of consistency, robustness, and smoothness introduced in Chapter 2.

4.3 Robust Single Machine Response Time Scheduling with Job Size Predictions

Let $p^*_{\max}$, $p^*_{\min}$ denote the maximum and the minimum job sizes, respectively. Define $P = p^*_{\max}/p^*_{\min}$, the ratio of the two extremes. This section gives a 1-consistent and $3P$-robust algorithm with predictions. Before presenting the algorithm, we first introduce a sufficient condition for any algorithm to achieve an $O(P)$ competitive ratio. This condition helps prove our algorithm's performance bound and design general robust algorithms for response time minimization. For an algorithm A, we define S_A as the schedule produced by A, and $\delta_A(t)$ as the number of unfinished jobs at time t in S_A. It is known that the total response time of S_A, denoted by F_A, can be re-written as $F_A = \sum_{j=1}^n F_j = \int_{t\geq 0} \delta_A(t)\, dt$. By integrating δ over t, it follows that A is c-competitive if $\delta_A(t) \leq c \cdot \delta_{A^*}(t)$ for all t. Our goal is to bound $\delta_A(t)$. To limit $\delta_A(t)$, we desire the machine spends the most time processing the jobs that can be finished by t rather than those left unfinished at t. This is because every unfinished job, regardless of how long it has been processed, increases $\delta_A(t)$ by 1. Define $P_A(t)$ as the set of unfinished jobs at time t, and $x_j^A(t)$ the time spent in processing job J_j before t in S_A. We expect $\sum_{J_j\in P_A(t)} x_j^A(t)$ to be bounded. In addition, we prefer algorithm A to be non-lazy, i.e., it does not unnecessarily idle the machine. It turns out that bounding $\sum_{J_j\in P_A(t)} x_j^A(t)$ is crucial in bounding $\delta_A(t)$ by P for any non-lazy algorithm, which is stated in the following theorem.

Theorem IV.1 (Sufficient Condition for $O(P)$ Competitive Ratio)**.** *Fix any constant $c > 0$ and consider any non-lazy algorithm A. At any time t, it follows that $\delta_A(t) \leq (c+1)\cdot P \cdot \delta_{A^*}(t)$ if $\sum_{J_j\in P_A(t)} x_j^A(t) \leq c \cdot p^*_{\max}$.*

Proof. Let Q_A and Q_{A^*} denote the set of unfinished jobs at time t in S_A and S_{A^*}, respectively. Define $Q_U = Q_A \cap Q_{A^*}$, $m = |Q_{A^*} - Q_U|$, $l = |Q_A - Q_U|$, and $o = |Q_U|$. Then, we have $\delta_A(t) = l + o$ and $\delta_{A^*}(t) = m + o$. It immediately follows that $\delta_A(t) \leq \delta_{A^*}(t) \leq (c+1)\cdot P \cdot \delta_{A^*}(t)$, if $l \leq m$. Therefore, we assume $l > m$ for the rest of the proof, which contains the following two cases.

Case 1 ($m = 0$). First, suppose that $o = 0$ with $\delta_{A^*}(t) = 0$. By non-laziness of A, $\delta_A(t) = 0$ and the result follows. Suppose $o \geq 1$. We have

$$\frac{\delta_A(t)}{\delta_{A^*}(t)} = \frac{l+o}{m+o} = \frac{l}{o} + 1 \leq l + 1$$

Since $m = 0$ indicates $Q_{A^*} = Q_U$ and l is the number of jobs completed by A^* but not by A at time t, the total time spent by A^* on processing jobs in $Q_A - Q_U$ is at most $\sum_{J_j\in P_A(t)} x_j^A(t)$ by non-laziness. It follows that

$$l + 1 \leq \frac{\sum_{J_j\in P_A(t)} x_j^A(t)}{p^*_{\min}} + 1 \leq \frac{c\cdot p^*_{\max}}{p^*_{\min}} + 1 = c\cdot P + 1 \leq (c+1)\cdot P$$

and therefore $\delta_A(t) \leq (c+1)\cdot P \cdot \delta_{A^*}(t)$.

Algorithm 7: Robust Online Scheduling

```
Data   : job size predictions $p_j$ and the total prediction error $\eta$
Result : schedule with the optimal consistency and $3P$ robustness
// maintain $x_j(t)$, $y_j(t)$ as processing time and remaining time estimate of job $J_j$ at time t, i.e.,
$y_j(t) = \frac{p_j}{\eta} - x_j(t)$.
1  Event Function JOBRELEASE() // job $J_j$ is released at time t
2      $y_j(t) \leftarrow \frac{p_j}{\eta}$.
3      if the machine is idle then
4          start processing $J_j$.
5      else
6          let $J_q$ be the job currently being processed.
7          if $y_j(t) \le y_q(t)$ then
8              preempt $J_q$ and start processing $J_j$.

9  Event Function JOBCOMPLETE() // job $J_j$ has been processed for $p_j^*$ time
10     if there is no unfinished job then
11         idle the machine.
12     else
13         let $J_q$ be the unfinished job with the minimum $y_q(t)$.
14         start (or resume) processing $J_q$.
```

Case 2 ($m \ge 1$). Since $l > m$, we have

$$\frac{\delta_A(t)}{\delta_{A^*}(t)} = \frac{l+o}{m+o} \le \frac{l}{m}$$

Observe that the total time spent by A^* on processing the l jobs in $Q_A - Q_U$ is at most the total time spent by A on the m jobs in $Q_{A^*} - Q_U$ plus the term $\sum_{J_j \in P_A(t)} x_j^A(t)$, which is bounded above by $m \cdot p^*_{\max} + \sum_{J_j \in P_A(t)} x_j^A(t)$. Thus, we obtain

$$\frac{l}{m} \le \frac{m \cdot p^*_{\max} + \sum_{J_j \in P_A(t)} x_j^A(t)}{p^*_{\min} \cdot m} \le P + \frac{c}{m} \cdot P \le (c+1) \cdot P \qquad \square$$

Theorem IV.1 shows that any algorithm is $O(P)$-competitive regardless of the value of η, which is also $O(P)$-robust if the processing time spent on partial jobs is bounded by a constant multiple of $p^*_{\max}$. Meanwhile, the algorithm must follow the Shortest Remaining Processing Time (SRPT) strategy for optimal consistency to hold. We then present a simple but effective algorithm that ensures both constant consistency and bounded robustness. The algorithm uses η to underestimate the job sizes intentionally. Note that knowing the error is also assumed in previous works [25]. To underestimate job sizes, the algorithm assigns $p_j^e \leftarrow \frac{p_j}{\eta}$ to job J_j and uses $p_j^e - x_j^A(t)$ to estimate the remaining size of J_j. The algorithm maintains the invariant of processing the unfinished job with the minimum remaining size estimate.

Algorithm 7 shows the pseudo-code. Here, we give an overview of the algorithm. When a job J_j arrives, we compute the job size estimate $p_j^e = \frac{p_j}{\eta}$. Process J_j immediately if there is no running job. Otherwise, let J_q be the running job. We preempt J_q and process J_j if $p_j^e \le y_q(t)$, or we change nothing otherwise.

When the machine completes a job, the job leaves the system, and the machine starts processing the unfinished job with the minimum job size estimate computed as $p_j^e - x_j(t)$. Idle the machine if there is no unfinished job. The algorithm always processes the unfinished job with the minimum remaining size estimate with the above operations.

We prove the performance bounds for Algorithm 7 in the following theorem.

Theorem IV.2 (Performance Bounds for Algorithm 7)**.** *Algorithm 7 is* 1*-consistent and* $3P$*-robust.*

Proof. The 1-consistency follows because $\frac{p_q}{\eta} = p_q = p_q^*$ when $\eta = 1$, which forces the algorithm strictly follow the SRPT strategy. Therefore, we focus on proving the $3P$-robustness below. Fix a time t. Define $P(t)$ as the set of unfinished jobs. We need to bound term $\sum_{J_j \in P(t)} x_j(t)$ to apply Theorem IV.1. Consider the unprocessed jobs in $P(t)$ with positive $x_j(t)$ and denote these jobs as $J_{i1}, ..., J_{ik}$ in the increasing order of the time it starts processing, i.e., J_{i1} is the first job that has been processed among the k jobs, followed by J_{i2}, $J_{i3}, ..., J_{ik}$. We make three observations for every J_{ir}. (1) For the first time J_{ir} starts being processed, jobs $J_{i1}, ..., J_{i(r-1)}$ must be pending in the system. (2) At the first time $t'(t' < t)$ when J_{ir} starts being processed, the term $y_{ir}(t') = \frac{p_{ir}}{\eta}$ must be the minimal one among $y_{i1}(t'), ..., y_{i(r-1)}(t')$. (3) After J_{ir} starts being processed at time t', jobs $J_{i1}, ..., J_{i(r-1)}$ will never be processed before t, i.e., $y_{il}(t) = y_{il}(t')$ for $l = 1, ..., r-1$. Combing the above observations, we obtain

$$\frac{p_{ir}}{\eta} \leq y_{i(r-1)}(t) = \frac{p_{i(r-1)}}{\eta} - x_{i(r-1)}(t)$$

for $r = 2, ..., k$. Summing these $k-1$ inequalities with telescoping, we have

$$x_{i1}(t) + ... + x_{i(k-1)}(t) \leq \frac{p_{i1}}{\eta} - \frac{p_{ik}}{\eta}$$

Since J_{ik} has not been completed at time t, it follows that $x_{ik}(t) \leq p_{ik}^*$ and we can bound term $\sum_{J_j \in P(t)} x_j(t)$ as follows.

$$\begin{aligned} \sum_{J_j \in P(t)} x_j(t) &= x_{i1}(t) + ... + x_{i(k-1)}(t) + x_{ik}(t) \\ &\leq \frac{p_{i1}}{\eta} - \frac{p_{ik}}{\eta} + p_{ik}^* \leq p_{i1}^* + p_{ik}^* \leq 2p_{\max}^* \end{aligned}$$

By Theorem IV.1, we have $\delta_A(t) \leq 3P \cdot \delta_{A^*}(t)$ for our algorithm A for every time t. By integrating δ over t, it follows that our algorithm is $3P$-competitive. This result holds for any error η; thus, the algorithm is $3P$-robust. □

Theorem IV.3 (Optimality of Consistency and Robustness for Algorithm 7)**.** *Algorithm 7 has asymptotically optimal consistency and asymptotically optimal robustness.*

Proof. The lower bound for the competitive ratio of any clairvoyant algorithm is $\Omega(1)$. The lower bound for the competitive ratio of any non-clairvoyant algorithm is $\Omega(P)$. By the Fundamental Theorems of Consistency (Theorem II.4) and the Fundamental Theorems of Robustness (Theorem II.6), Algorithm 7 has consistency and robustness both matching the corresponding lower bound. □

Algorithm 8: Greedy-bin Scheduling

```
Data   : job size predictions $p_j$
Result : schedule with an $O(\eta^2)$ competitive ratio
// initialize a set $F$ and a LIFO stack $P$.
1  Event Function JobRelease() // job $J_j$ is released
2      $F \leftarrow F \cup \{J_j\}$
3      if $\delta_F(t) > \delta_P(t)$ then
4          let $J_q$ be the job with the minimum job size prediction in set $F$.
5          $F \leftarrow F - \{J_q\}$.
6          push $J_q$ to the top of stack $P$.
7      start (or resume) processing the job at the top of stack $P$.
8  Event Function JobComplete() // job $J_j$ has been processed for $p_j^*$ time
9      if there is no unfinished job then
10         idle the machine.
11     else
12         if $\delta_F(t) > \delta_P(t)$ then
13             let $J_q$ be the job with the minimum job size prediction in set $F$.
14             $F \leftarrow F - \{J_q\}$.
15             push $J_q$ to the top of stack $P$.
16         start (or resume) processing the job at the top of stack $P$.
```

4.4 Conjecture: An $O(\eta^2)$-Competitive Algorithm

Another interesting question is how to use predictions if the error is reasonably small. We aim to bound the performance by the error metric η in this case. The existing work [25] studies this scenario and proposes an $O(\eta^4)$-competitive algorithm via binning technique. Later, the same authors [32] improve the competitive ratio to $O(\eta^2 \log \eta)$ and prove a lower bound of $\Omega(\eta^2)$ for the problem of response time scheduling with predictions on a single machine. This section presents an error-oblivious algorithm that we conjecture to achieve the optimal $O(\eta^2)$ competitive ratio (Conjecture IV.4). By *error-oblivious*, we mean that the algorithm does not require the value of η. Inspired by the existing techniques used in designing the $O(\eta^4)$-competitive algorithm [25], we also apply the binning technique to our algorithm. The enhancement is that our algorithm directly uses the job size predictions to decide the job to process. Intuitively, the algorithm relies on predictions more when making scheduling decisions.

Our algorithm maintains a set F of full jobs and a Last In First Out (LIFO) stack P of partial jobs. The algorithm always processes the job at the top of stack P. The full jobs are unprocessed jobs, while the partial jobs are those processed but not completed. Newly arrived jobs are inserted into set F. Denote $\delta_F(t)$ and $\delta_P(t)$ as the size of set F and stack P at time t, respectively. When $\delta_F(t) > \delta_P(t)$, we move the job with the minimum job size prediction from set F to the top of stack P. We name this algorithm **Greedy-bin** since the choice from set F follows predictions greedily. Algorithm 8 shows the pseudo-code.

We discuss the intuition behind the $O(\eta^2)$ competitive ratio. The algorithm keeps roughly half of the jobs as full jobs so that it never runs into a bad situation where most jobs are partial and nearly completed. Fix time t. Consider the worst-case scenario when the ratio $\delta_A(t)/\delta_{A^*}(t)$ is maximized, where A represents Algorithm 8. Let A complete no jobs by time t, and also it faces the worst-case situation where all jobs in stack P are almost complete, i.e., $p_j^* - x_j^A(t)$ is infinitely small for every job $J_j \in$ stack P. We define two types of jobs as "large" and "small" to classify the jobs to be processed. Roughly, the size of a "large" job is η^2 times of a "small" job. We force that, by time t, algorithm A spends all the time processing "large" jobs, while A^* completes all "small" jobs, leaving the "large" jobs as full jobs. Observe that A starts processing the job J_q with the minimum job size prediction in set F. Therefore, for any job J_j in stack F, we have $\frac{p_q^*}{\eta} \leq p_q \leq p_j \leq \eta \cdot p_j^*$, which implies that $p_q^* \leq \eta^2 \cdot p_j^*$ and that the time needed for completing J_q can complete at most η^2 jobs for A^*. Every unfinished job in the optimal schedule at time t corresponds to at most η^2 completed jobs in the schedule produced by A. Therefore, it follows that $\delta_A(t)/\delta_{A^*}(t) \in O(\eta^2)$ and Algorithm 8 is $O(\eta^2)$-competitive. We conjecture that this reasoning is core to proving the $O(\eta^2)$ competitiveness. The main difficulty of the formal proof is showing that the structural properties of the worst-case input lead to the above situation. However, showing this seems non-trivial, thus making the proof of Conjecture IV.4 our ongoing work.

Conjecture IV.4 (Performance Bound for Algorithm 8). *Algorithm 8 is $O(\eta^2)$-competitive.*

4.5 Conclusions and Future Work

This work presents the first algorithm for online non-clairvoyant scheduling on a single machine to minimize the mean response time with job size predictions, which simultaneously achieves optimal consistency and optimal robustness up to a constant multiplicative factor. Our proposed algorithm is 1-consistent and $3P$-robust, where P denotes the maximum ratio between any two job sizes. We derive a sufficient condition for $O(P)$ robustness in our analysis, which is helpful for algorithm design and analysis in the future. Finally, we conjecture that an optimal $O(\eta^2)$-competitive algorithm exists for the case where the prediction error is small, making it preferable to have performance depending purely on the error. Proving this conjecture is our ongoing work.

The most desirable algorithm has constant consistency, bounded robustness, and a good competitive ratio on η. It appears challenging to design such an algorithm or show that such one does not exist. The milestones towards this ambitious goal are sufficient conditions for constant consistency, bounded robustness, and a good competitive ratio on η. Then, one may attempt to design an algorithm to maintain the invariant satisfying the right conditions. Our work finds one sufficient condition for bounded robustness. Finding others is our future work.

CHAPTER 5

Real-Time Scheduling with Predictions

The recent revival in learning theory gives us improved capabilities for accurate predictions and increased opportunities for performance enhancement. This chapter extends the research agenda of online scheduling with predictions to one of the central problems — soft real-time scheduling on single and parallel machines to minimize the mean response time. We design an algorithm, PEDRMLF (Predictions Enhanced Dynamic Randomized MultiLevel Feedback), that incorporates job size predictions, achieving an optimal competitive ratio under perfect predictions and the best-known competitive ratio under any predictions. PEDRMLF is the first algorithm that simultaneously achieves optimal consistency and bounded robustness. Simulations show that the proposed algorithm performs close to the theoretically optimal bound while consistently outperforming state-of-the-art benchmarks.

5.1 Introduction

Guaranteed quality of service (QoS) is the focus of *soft real-time systems*. Typical applications include CPU scheduling in time-sharing operating systems, multimedia processing, and web browsing [34, 35]. In soft real-time systems, jobs do not have hard deadline requirements but prefer earlier completion times [34, 36]. A common metric to measure QoS in such systems is the *mean response time*, i.e., the average time a job spends in the system, including waiting time and service time. Minimizing this metric is good for soft real-time systems [36] and desirable in scheduling low-priority jobs under non-emergency conditions for *mixed-criticality systems* [37]. However, scheduling jobs to minimize mean response time is NP-hard on identical parallel machines, even if we allow preemption and know the exact job sizes upon job arrival [38].

Assuming *clairvoyance* where exact job sizes are known, *Shortest Remaining Processing Time* (SRPT) achieves an optimal competitive ratio of $O(\min\{\log \frac{n}{m}, \log P\})$ on parallel machines and optimal performance on a single machine [39, 40], where n is the number of jobs, m the number of machines, and P the maximum ratio between any two job sizes (see Figure V.1 for a problem illustration). At the other extreme, assuming *non-clairvoyance* where exact job sizes are unknown, *Randomized Multilevel Feedback* (RMLF) achieves the best-known competitive ratio of $O(\min\{\log n \log \frac{n}{m}, \log n \log P\})$ on parallel machines and an optimal $O(\log n)$ competitive ratio on a single machine [35, 41, 42]. Its deterministic version serves as the basis for the scheduling algorithms used in Windows NT and Unix [43]. SRPT is too optimistic about knowing exact job sizes, often unavailable in practice. RMLF is too

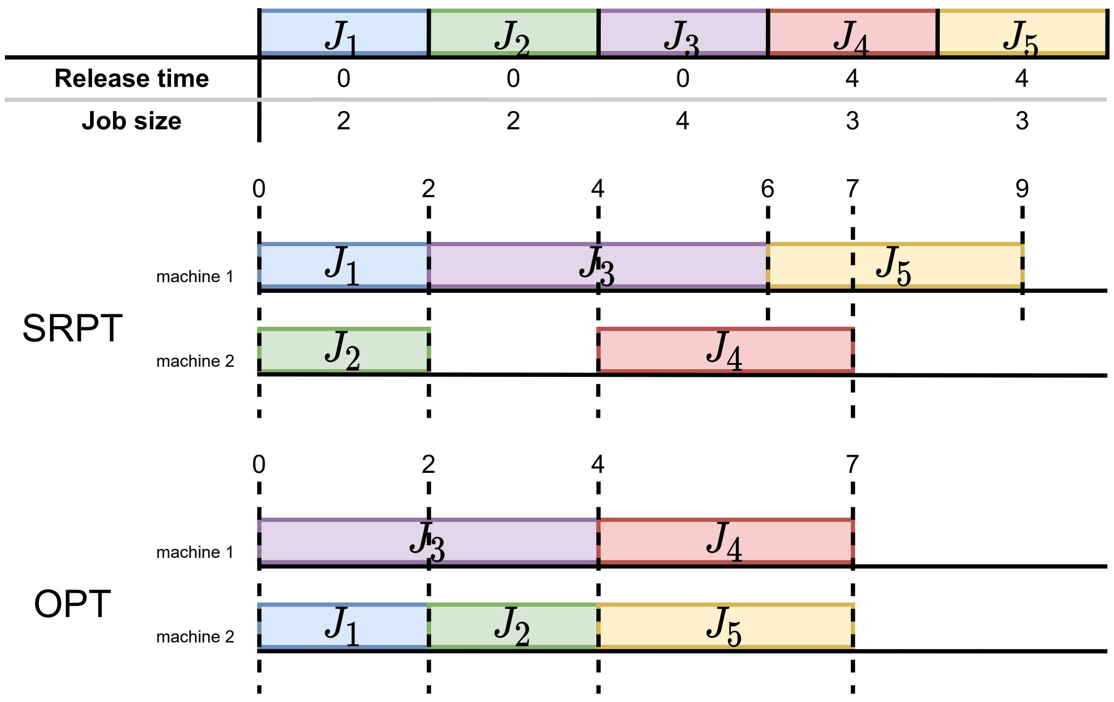


Figure V.1: Problem illustration with two identical parallel machines and five jobs. SRPT always processes the jobs with minimal remaining sizes. This example shows that SRPT is not optimal, as the mean response time achieved by SRPT is 3.6, while the optimal (OPT) is 3.2.

pessimistic about having no information on job sizes. Though the job sizes are almost impossible to know precisely in advance, they are not entirely invisible to us.

The recent revival in learning theory has shown improved capabilities for predicting job sizes [28–30] and advancing online decision-making [1, 3, 10, 11, 25, 26, 44]. However, the predictions are likely imperfect, which may degrade algorithm performance [44]. We expect the performance to improve as the predictions become more accurate but to stay bounded under arbitrarily bad predictions. The metric consistency and robustness reveal this desirable property. A good algorithm should obtain near-optimal consistency and maintain bounded robustness.

Existing learning-augmented algorithms have shown great success in online scheduling with objectives like minimizing makespan [1, 10] and total completion time [11, 26]. However, limited results have been found in mean response time minimization. A recent work [25] studies the weighted response time minimization with job size predictions on a single machine. The algorithm achieves good competitive ratios with good predictions, but the performance is unbounded under arbitrarily bad predictions. No results have yet been found for mean response time minimization on parallel machines.

There are two main challenges in designing algorithms to incorporate predictions. The first is quantifying the prediction error, which should be simple enough for learning models to optimize. Counterexamples are the error metrics strongly correlated with the target problem [26] or algorithm performance [10]. Such metrics, indeed, simplify algorithm design but complicate the training of the predictive model.

The error metric is thus expected to be problem- and algorithm-independent, often making algorithm design difficult. The second challenge is achieving near-optimal consistency and bounded robustness simultaneously [1, 3]. The intuition is that if an algorithm relies heavily on predictions, it may perform well under good predictions but likely goes arbitrarily bad against unbounded prediction errors. On the other hand, the algorithm may overlook the benefits of good predictions if the predictions are not trusted enough.

We define *total prediction error* η to be the largest multiplicative factor between the prediction and the exact value. This error metric is simple, intuitive, and also problem- and algorithm-independent. The worse the predictions become, the greater η grows. We design an algorithm in the spirit of RMLF, which creates a set of priority queues to schedule jobs. We use the ratio between job size prediction and the total error to intentionally underestimate the actual job size. This job size estimate is used to decide which priority queue the job enters, determining the job's priority over the others.

Contributions. In this work, we present the **first** scheduling algorithm with predictions on single and parallel machines to minimize mean response time. We define a suitable prediction error metric η and propose PEDRMLF (Predictions Enhanced Dynamic Randomized MultiLevel Feedback) that combines predictions with classic RMLF. We show that PEDRMLF achieves an optimal competitive ratio of $O(\min\{\log \frac{n}{m}, \log P\})$ under perfect predictions and a best-known competitive ratio of $O(\min\{\log n \log \frac{n}{m}, \log n \log P\})$ under any prediction error. We derive a sufficient condition for optimal consistency, useful for algorithm design and analysis in future. In addition, we show that PEDRMLF has a scalable time complexity of $O(\log n)$, suitable for large-scale applications. Finally, we perform extensive simulations to show the performance improvement achieved by PEDRMLF. The evaluation shows that PEDRMLF has similar performance to the optimal offline algorithm SRPT when the predictions are reasonably accurate while consistently outperforming RMLF even if the prediction error is unusually large. The solid performance guarantees pave the way for PEDRMLF to schedule soft real-time jobs or mixed-criticality applications on real-time systems.

5.2 Preliminaries

5.2.1 Problem Definition

We study *non-clairvoyant* scheduling to minimize the mean response time on single and parallel machines. There are m unit-speed machines and n jobs arriving over time. A job is denoted by J_j, $1 \leq j \leq n$, with a *release time* r_j and *job size* p_j^*. The release time r_j is unknown until J_j arrives, and the job size p_j^* is unknown until J_j is completed. This setting is known as *non-clairvoyant* in the literature. We let P denote the maximum ratio between any two job sizes. Jobs arrive over time, so they are time indexed by their release time: $j_1 < j_2$ implies $r_{j_1} \leq r_{j_2}$. Jobs are *dependency-free*, *preemptive*, and *non-parallelizable*, meaning they have no dependencies in between, any running job can be preempted and resumed on any machine, and any job can run on at most one machine at a time. Given a schedule, let C_j be the *completion time* of job J_j. The *response time* for job J_j is defined as

$F_j = C_j - r_j$. Our objective is to minimize the mean response time, i.e., $\frac{1}{n}\sum_{j=1}^{n} F_j$, or equivalently the total response time $F = \sum_{j=1}^{n} F_j$ given any problem instance I. The scheduling problem is defined as $Pm \mid$ *online-time-nclv*, *pmtn*, $r_j \mid \sum F_j$ (Graham notation [9]).

The scheduler can use the job size predictions from an oracle. However, the predictions are likely imperfect and may adversely affect algorithm performance. Denote by p_j the *job size prediction* for J_j. We define the prediction error of J_j as $\eta_j = \max\{\frac{p_j}{p_j^*}, \frac{p_j^*}{p_j}\}$ and the *total prediction error* η as

$$\eta = \max_{1\leq j\leq n} \eta_j = \max_{1\leq j\leq n} \max\{\frac{p_j}{p_j^*}, \frac{p_j^*}{p_j}\}$$

The predictions are perfect if and only if $\eta = 1$. Otherwise, η increases toward $+\infty$ as the predictions are getting worse. We argue that knowing η is a reasonable assumption since the prediction error measured during predictive model training will likely represent the prediction error in practice. Similar prediction error definitions can be found in [1, 3, 25].

5.2.2 Performance Evaluation

We evaluate the performance of our algorithm using the competitive framework while also examining its consistency and robustness to the prediction error [11]. Real-time systems strive to have worst-case performance guarantees at all times. The competitive framework provides solid support by showing algorithm behavior in the worst case and the distance from the optimum.

5.2.2.1 Competitive Framework

A randomized online algorithm A will compare against an *oblivious optimal adversary* A^* that decides a sequence of jobs along with their release times and sizes in advance without knowing the random choices used by A [14]. Let $F_\beta^A(I)$ denote the total response time obtained by A on problem instance I with a set of random choices β and $F^{A^*}(I)$ the optimal total response time. We say A has a competitive ratio c or is c-competitive if $E[F_\beta^A(I)] \leq c \cdot F^{A^*}(I)$ for any I, where c is a function of problem input and E denotes the expectation function. The expectation of $F_\beta^A(I)$ is taken over all possible β.

5.2.2.2 Consistency and Robustness

In addition to the competitive ratio, we evaluate the performance of the algorithms by the metrics of consistency and robustness introduced in Chapter 2.

5.2.3 Related Results

The work in [33] has been the first to study non-clairvoyant scheduling to minimize the mean response time. The authors prove an $\Omega(n^{1/3})$ lower bound on the competitive ratio for deterministic algorithms and an $\Omega(\log n)$ lower bound for randomized algorithms on a single machine. Later, [45] gives the first $o(n)$-competitive algorithm for the single machine case. The authors show that RMLF achieves an

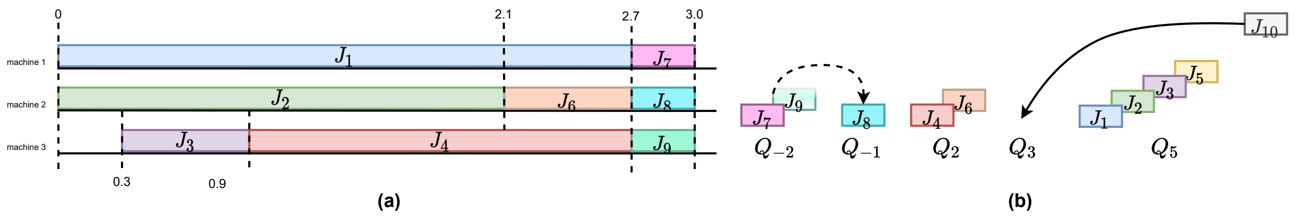


Figure V.2: PEDRMLF performs the workload given in Figure V.3 on three machines up to time 3.0. Subfigure (a) shows the schedule and (b) the status of the queues at time 3.0. PEDRMLF processes the jobs from the lowest to highest queues and within any queue in increasing order of job index. In this example, every job except for J_5 has been processed due to the earliest release time first policy within queues. At time 3.0, J_8 is promoted to Q_{-1}, J_9 is just completed, and a new job J_{10} arrives. The running jobs will become J_7, J_8, and J_4 right after time 3.0. Observe that the multilevel feedback queues are created on the fly with positive and non-positive queue indices.

$O(\log n \log\log n)$ competitive ratio. Extending RMLF to parallel machines, [41] conducts a strengthened analysis to show that RMLF achieves $O(\min\{\log n \log \frac{n}{m}, \log n \log P\})$ competitive ratio on parallel machines and an optimal $O(\log n)$ competitive ratio on a single machine. These results remain unaltered for decades until the recent [25] has studied non-clairvoyant scheduling with predictions. The authors propose an $O(\eta^4)$-competitive algorithm for the single machine case, while the same algorithm is extensible to the weighted version of the problem.

The problem of response time scheduling has been also tackled from other perspectives. In clairvoyant settings, the single machine case can be optimally solved by SRPT [40]. For parallel machines, [39] proves that SRPT is $O(\min\{\log \frac{n}{m}, \log P\})$-competitive, and this is optimal since the same work also shows an $\Omega(\log \frac{n}{m})$ and an $\Omega(\log P)$ lower bound on the competitive ratio for randomized algorithms, where P denotes the maximum ratio of any two job sizes. There are also scheduling algorithms under resource augmentation framework [46] or related objectives like minimizing stretch [47].

5.3 Robust Response Time Scheduling with Job Size Predictions

This section describes the design of our robust response time scheduling algorithm, PEDRMLF, for single and parallel machines. The elements of our algorithm are (1) real-time randomized estimation of job sizes boosted by predictions, (2) dynamic multilevel feedback queues, and (3) classic RMLF. PEDRMLF evolves RMLF [41, 45] via (1) using predictions to make more accurate estimations on job sizes, (2) dynamically creating multilevel feedback queues to discard the idealistic assumption of knowing the exact smallest job size, and (3) limiting probability distribution for the random variables to minimize the occurrence of "bad" jobs that are small in size but cause a long response time (see Section 5.4.2 for formal definition).

We fix any schedule for the below discussion. PEDRMLF schedules *active* jobs in the system. Formally, define $x_j(t)$ to be the processed size of job J_j by time t, and $y_j(t) = p_j^* - x_j(t)$ the *actual remaining size* of J_j at time t. Job J_j is said to be *active* at time t if $y_j(t) > 0$ and $r_j \leq t$.

	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9	J_{10}
Release time	0.0	0.0	0.3	0.9	1.5	2.1	2.7	2.7	2.7	3.0
Job size	40.0	36.0	120.0	10.0	39.0	6.0	0.5	0.4	0.3	9.0
Job size prediction	42.0	33.0	60.0	6.0	39.0	7.8	0.3	0.4	0.4	9.0
Initial queue index b	5	5	5	2	5	2	-2	-2	-2	3
Job size estimate $p^e_{j,b}$	40.0	56.0	37.0	4.5	62.0	7.0	0.4	0.3	0.4	9.0

Figure V.3: Workload parameters for Figure V.2. The total prediction error η is 2. During execution, PEDRMLF knows the job size predictions and η. The job size estimates $p^e_{j,b}$ depend on random choices β, so the last row presents one possible outcome in PEDRMLF's execution.

Active jobs are partitioned into a set of *priority queues* $Q_{k_1}, Q_{k_2}, ..., Q_{k_d}$, where the subscript k_i is an integer and may take a negative value. We say Q_i is lower than Q_k if $i < k$. A lower queue has a higher priority, i.e., the jobs in a lower queue will be processed before the ones from a higher queue. Within a queue, the earliest release time first policy determines the job priority. Formally, job J_{j_1} has a higher priority than J_{j_2} if they are within the same queue and $j_1 < j_2$. Recall that $j_1 < j_2$ implies $r_{j_1} \leq r_{j_2}$. Jobs thus have a fixed relative priority within a queue since their indices are fixed. The same policy was also used in [41].

Every active job J_j is assigned with a job size estimate $p^e_{j,i}$ when it enters Q_i. This estimation is updated when J_j is promoted to a higher priority queue. We estimate the job size using the ratio of the job size prediction over the total prediction error, i.e., $\frac{p_j}{\eta}$. This is an underestimation of the actual job size, since $\frac{p_j}{\eta} \leq \frac{p_j}{\eta_j} \leq p^*_j$. Note that it is our intention to ensure that the job J_j enters a queue Q_b initially with $\frac{p_j}{\eta} \leq p^*_j \leq 2^b$, where b denotes the *initial queue index*.

For randomization used by PEDRMLF, define $\beta_{j,i}$ to be a random variable with distribution $P[\beta_{j,i} \leq x] = 1 - (j+1)^{-\lambda \cdot \frac{x}{2^i}}$, where λ is a symbolic constant and set to 4 in this work. This exponential distribution is crucial according to [41].

5.3.1 Algorithm PEDRMLF

The pseudo-code of PEDRMLF is given in Algorithm 9. The core of PEDRMLF is always processing the active jobs from the lowest to the highest queue, within any queue in increasing order of job index, on any available machine(s). At first, no queue is created. When any job J_j is released at time r_j, PEDRMLF takes the following actions:

- Compute the initial queue index $b = [\log_2 \frac{p_j}{\eta}] + 1$.
- Job J_j enters Q_b with $p^e_{j,b} = \max\{2^b, 2^{b+1} - \beta_{j,b}\}$. Create Q_b if it does not exist.
- If at least one available machine exists, start processing J_j on any available machine.

Algorithm 9: PEDRMLF

```
Data   : job size predictions $p_j$ and the total prediction error $\eta$
Result : schedule with the optimal consistency and best-known robustness
1  Event Function JobRelease() // job $J_j$ is released
2      set $b \leftarrow [\log_2 \frac{p_j}{\eta}] + 1$.
3      if $Q_b$ does not exist then
4          create $Q_b$ of jobs in increasing order of job index.
5      push $J_j$ into $Q_b$ and set $p^e_{j,b} \leftarrow \max\{2^b, 2^{b+1} - \beta_{j,b}\}$.
6      if there exists at least one available machine then
7          run $J_j$ on any available machine.
8      else
9          let $J_r$ be the running job from the highest queue with the largest job index in the queue.
10         let $Q_h$ be the queue that contains $J_r$.
11         if $b < h$ then
12             preempt $J_r$ and run $J_j$ on the same machine.

13 Event Function JobPromote() // job $J_j \in Q_{i-1}$ has been processed for $p^e_{j,i-1}$ time
14     remove $J_j$ from $Q_{i-1}$.
15     if $Q_i$ does not exist then
16         create $Q_i$ of jobs in increasing order of job index.
17     push $J_j$ into $Q_i$ and set $p^e_{j,i} \leftarrow \max\{2^i, 2^{i+1} - \beta_{j,i}\}$.

18 Event Function JobComplete() // job $J_j \in Q_i$ has been processed for $p^*_j$ time
19     remove $J_j$ from $Q_i$.
20     if there is at least one active non-running job then
21         let $J_s$ be the active non-running job from the lowest queue with the smallest job index within the
             queue.
22         run $J_s$ on the machine that has completed $J_j$.
23     else
24         idle the machine that has completed $J_j$.
```

- If there is at least one machine processing a job from a higher queue than Q_b when no available machine exists, preempt the running job from the highest queue (tie breaks by the larger job index) and process J_j.

When a job $J_j \in Q_{i-1}$ runs for $p^e_{j,i-1} < p^*_j$ time, PEDRMLF takes the following actions:

- Job J_j leaves Q_{i-1} and enters Q_i. Create Q_i if it does not exist.
- Set $p^e_{j,i} = \max\{2^i, 2^{i+1} - \beta_{j,i}\}$.

When a job $J_j \in Q_i$ has run for $p^*_j \leq p^e_{j,i}$ time, it is completed and leaves the system. Process the active non-running job from the lowest queue with the smallest job index within the queue on the machine completing J_j. Idle the machine if no such job exists. Figures V.2 and V.3 present a sample run of PEDRMLF.

5.4 Algorithm Analysis

This section proves the performance bounds for PEDRMLF on single and parallel machines. We show that PEDRMLF behaves optimally under perfect predictions while achieving the same competitive ratio as the best-known non-clairvoyant algorithm under any predictions. We prove the time complexity of PEDRMLF at the end. See Table V.1 for the table of notations.

S	Meaning	**S**	Meaning
n	number of jobs	m	number of machines
J_j	job with index j	r_j	release time of J_j
p_j^*	job size of J_j	C_j	completion time of J_j
η_j	prediction error of J_j	η	total prediction error
F_j	response time of J_j	F	total response time
A	algorithm PEDRMLF	A^*	the optimal scheduler
Q_i	queue with index i	$p_{j,i}^e$	job size estimate of J_j
$\beta_{j,i}$	a random choice of J_j	β	set of random choices
P	max job size ratio	b	the initial queue index
λ	constant ($\lambda = 4$)	I	a problem instance
x_j	processed size of J_j	y_j	remaining size of J_j
U	uncompleted jobs	c_j	class of J_j
$\underline{k}$	minimal job class	$\overline{k}$	near-maximal job class
T_j	a partition of timeline	$\lvert T_j\rvert$	sum of time in T_j
t_p	last partial time instant	t_i	a partition time instant
t_l, t_u	release time in $(t_l, t_u]$	c_l, c_u	class in $[c_l, c_u]$
V	volume of jobs	ΔV	volume difference
X_j^u	unlucky job indicator	X_j^l	lucky job indicator
X_j^b	big job indicator	X_j^a	active job indicator
U^u	unlucky active jobs	U^l	lucky active jobs
U^b	big active jobs	R	a trivial algorithm
$\overline{W}$	busy time of R	W_β	busy time of A
$\lvert\overline{W}\rvert$	sum of time in $\overline{W}$	$\lvert W_\beta\rvert$	sum of time in W_β

Table V.1: A table of notations.

Theorem V.1 (Consistency). *Algorithm PEDRMLF is $O(\min\{\log \frac{n}{m}, \log P\})$-competitive with $\eta = 1$ on single and parallel machines ($m \geq 1$).*

Theorem V.2 (Robustness). *Algorithm PEDRMLF is $O(\log n)$-competitive with any η on a single machine ($m = 1$). It is $O(\min\{\log n \log \frac{n}{m}, \log n \log P\})$-competitive with any η on parallel machines ($m > 1$).*

Theorem V.3 (Complexity). *Algorithm PEDRMLF has run-time complexity $O(\log_2 n)$ to make any online decision. The space complexity required by PEDRMLF is $O(n)$.*

These theorems will be proved via the following lemmas. We first prove the consistency results assuming $\eta = 1$ and then extend these results to any η. In proving the consistency, we first fix the random choices β to show several critical bounds (Lemma V.6 – V.12) holding in all deterministic executions. These results are also useful in analyzing robustness. To prove the bounds, we simplify and enhance the tools developed in the studies [41, 45] of response time scheduling. We fix a problem instance I and drop the symbol I when the context is clear. The subscript β represents the execution of PEDRMLF determined by given random choices $\beta = (\beta_{j,i})$.

A widely used formula for total response time is $F = \int_{t \geq 0} |U(t)| \, dt$, where $U(t)$ is the set of uncompleted active jobs at time t [48]. We denote $U_\beta(t)$ and $U^*(t)$ as the set of uncompleted active jobs at time t for PEDRMLF and the optimal scheduler, respectively. By linearity of expectation, it follows $E[F^A_\beta] = \int_{t \geq 0} E[|U_\beta(t)|] \, dt$, allowing us to bound F^A_β via bounding $E[|U_\beta(t)|]$.

Define *class* of job J_j to be c_j, where we let $c_j = [\log_2 p_j^*]$. By our definition, the jobs with sizes in the range $[2^i, 2^{i+1})$ will be considered in the same class i. Note that c_j can take a negative value. For job J_j, a sequence of $\beta_{j,i}$ is generated until the term $\beta_{j,k}$ reaches $p^e_{j,k} = \max\{2^k, 2^{k+1} - \beta_{j,k}\} \geq p_j^*$, where $k = c_j$ or $k = c_j + 1$. If $\eta = 1$, the first generated random variable $\beta_{j,b}$ already satisfies $p^e_{j,b} = \max\{2^b, 2^{b+1} - \beta_{j,b}\} \geq 2^b = 2^{[\log_2 \frac{p_j}{\eta}]+1} = 2^{[\log_2 p_j^*]+1} \geq p_j^*$, where $b = [\log_2 \frac{p_j}{\eta}] + 1 = [\log_2 p_j^*] + 1 = c_j + 1$. Thus, job J_j initially enters Q_{c_j+1} and is completed within the queue. We name such property as *good predictions property* and will show that it is a sufficient condition to achieve optimal consistency.

Property V.4 (Good Predictions Property)**.** *Every job J_j initially enters Q_{c_j+1} and is completed within the queue.*

The following basic result bounds the total job size by F^{A^*}.

Lemma V.5 (Bound on Total Job Size)**.**

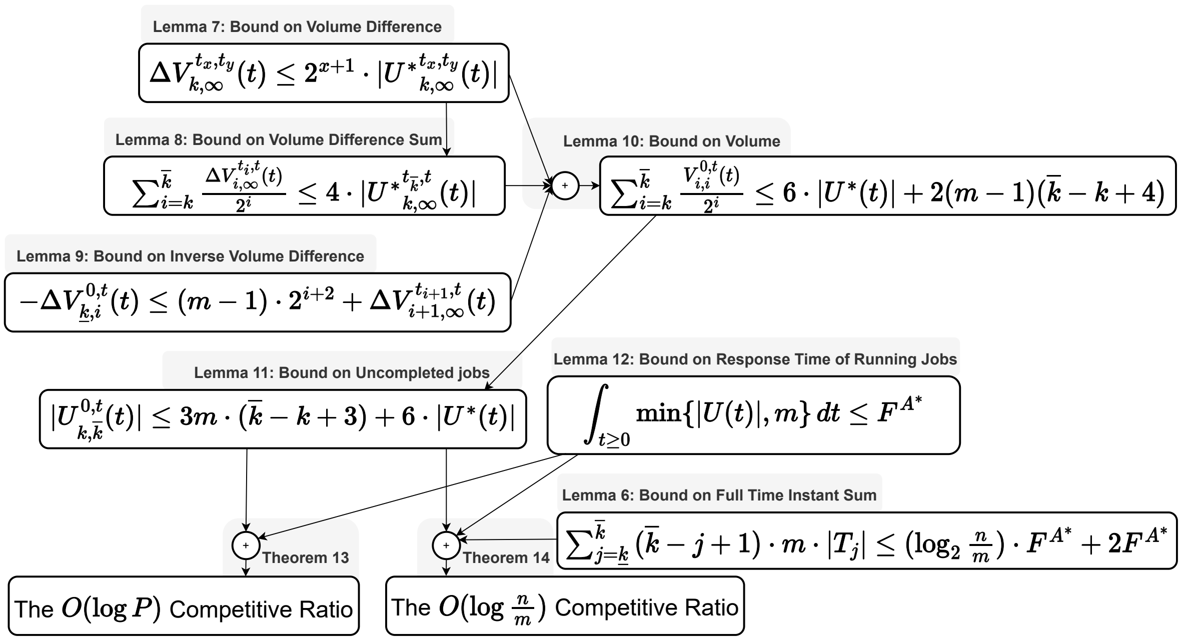
$$\sum_{j=1}^{n} p_j^* \leq F^{A^*}$$

Proof. Any job can only be processed on one machine at any time. Therefore, it follows that $p_j^* \leq C_j - r_j$ and $\sum_{j=1}^n p_j^* \leq \sum_{j=1}^n C_j - r_j \leq F^{A^*}$ for the optimal schedule. □

5.4.1 Deterministic Analysis

For the analysis below, we fix the random variable β, so the execution of PEDRMLF is deterministic. The purpose is to prove several results for any β. Thus, we discard the symbol β below and will use it again in later probabilistic analysis.

We begin with the intuitions and key ideas in our deterministic analysis. Our target is to relate the uncompleted jobs in PEDRMLF and those in the optimal scheduler, i.e., bounding $|U(t)|$ by $|U^*(t)|$ at any time t. No direct relationship can be found. To study $|U(t)|$, we expand and express it in terms of *volume* V that represents the total remaining size of jobs. This expansion is from the observation that every job of class i has its size of at least 2^i. Then, we bound V by $|U^*(t)|$. Observe that $V = V^* + \Delta V$, where V^* is the volume for the optimal scheduler and ΔV the volume difference between PEDRMLF and the optimal scheduler. Relating V^* with $|U^*(t)|$ is immediate. We bound ΔV by $|U^*(t)|$ instead.



Lemma 7: Bound on Volume Difference
$\Delta V_{k,\infty}^{t_x,t_y}(t) \leq 2^{x+1} \cdot |U^{*}{}_{k,\infty}^{t_x,t_y}(t)|$
Lemma 8: Bound on Volume Difference Sum
$\sum_{i=k}^{\overline{k}} \frac{\Delta V_{i,\infty}^{t_i,t}(t)}{2^i} \leq 4 \cdot |U^{*}{}_{k,\infty}^{t_{\overline{k}},t}(t)|$
Lemma 9: Bound on Inverse Volume Difference
$-\Delta V_{\underline{k},i}^{0,t}(t) \leq (m-1) \cdot 2^{i+2} + \Delta V_{i+1,\infty}^{t_{i+1},t}(t)$
Lemma 10: Bound on Volume
$\sum_{i=k}^{\overline{k}} \frac{V_{i,i}^{0,t}(t)}{2^i} \leq 6 \cdot |U^*(t)| + 2(m-1)(\overline{k}-k+4)$
Lemma 11: Bound on Uncompleted jobs
$|U_{k,\overline{k}}^{0,t}(t)| \leq 3m \cdot (\overline{k}-k+3) + 6 \cdot |U^*(t)|$
Lemma 12: Bound on Response Time of Running Jobs
$\int_{t\geq 0} \min\{|U(t)|, m\}\, dt \leq F^{A^*}$
Lemma 6: Bound on Full Time Instant Sum
$\sum_{j=\underline{k}}^{\overline{k}} (\overline{k}-j+1) \cdot m \cdot |T_j| \leq (\log_2 \frac{n}{m}) \cdot F^{A^*} + 2F^{A^*}$
Theorem 13
Theorem 14
The $O(\log P)$ Competitive Ratio
The $O(\log \frac{n}{m})$ Competitive Ratio

Figure V.4: Proof structure for deterministic analysis.

To study ΔV, we examine the behavioral difference between PEDRMLF and the optimal scheduler. To do so, we partition the whole timeline into intervals composed of time instances. The time instants are either *full* or *partial* ones according to the number of running machines. We first show that the partial instances are insignificant in our analysis. We further partition the full instants according to the highest queue where PEDRMLF is processing jobs at any given time. This partitioning allows us to bound ΔV by $|U^*(t)|$ individually on all intervals. We finally derive the relationship between $|U(t)|$ and $|U^*(t)|$ with the above results for bounding F^A by F^{A^*}. The proof structure for deterministic analysis is given in Figure V.4.

Define two terms $\underline{k}$ and $\overline{k}$ for two job classes of our primary interest. Define $\underline{k}$ to be the minimal class of all jobs, i.e., $\underline{k} = \min\{c_j | 1 \leq j \leq n\}$ and $\overline{k}$ to be the maximal class while having at least m jobs with the class equal to or higher than $\overline{k}$, i.e., $\overline{k} = \max\{c_j | 1 \leq j \leq n \wedge |\{J_k | c_k \geq c_j\}| \geq m\}$. Note that the definitions of $\underline{k}$ and $\overline{k}$ are asymmetric: $\overline{k}$ is not necessarily the maximal class of all jobs. It is easy to derive that at most $m-1$ jobs are of a class higher than $\overline{k}$. We prove several bounds related to the jobs with class in between $\underline{k}$ and $\overline{k}$. Here, we assume $n > m$ because PEDRMLF is trivially optimal otherwise. We say a time instant t is *full* if $|U(t)| \geq m$, or *partial* otherwise. We focus on full time instants since the contribution of partial time instants to the total response time is marginal, which is supported by Lemma V.12. Partition the full time instants into disjoint sets $T_{\underline{k}}, T_{\underline{k}+1}, ..., T_{\overline{k}}$. The set T_j, $\underline{k} \leq j < \overline{k}$, includes the full time instants when the largest class of the running jobs is j, and the set $T_{\overline{k}}$ includes the other full time instants. Let $|T_j|$ denote the sum of the time instants in T_j, i.e., $|T_j| = \int_{t \in T_j} dt$. Lemma V.6 relates $|T_j|$ with F^{A^*}.

Lemma V.6 (Bound on Full Time Instant Sum)**.**

$$\sum_{j=\underline{k}}^{\overline{k}} (\overline{k} - j + 1) \cdot m \cdot |T_j| \leq (\log_2 \frac{n}{m}) \cdot F^{A^*} + 2F^{A^*}$$

Proof. Define variable $S_j = \sum_{c_i \leq j} p_i^*$, $\underline{k} \leq j < \overline{k}$ that corresponds to set T_j. At any time instant $t \in \cup_{k=\underline{k}}^{j} T_k$, $\underline{k} \leq j < \overline{k}$, every one of the m machines is processing one job of class at most j, yielding $\sum_{k=\underline{k}}^{j} m \cdot |T_k| \leq S_j$. Similarly, we have $\sum_{k=\underline{k}}^{\overline{k}} m \cdot |T_k| \leq \sum_{j=1}^{n} p_j^*$. Then, it follows $\sum_{k=\underline{k}}^{\overline{k}} m \cdot |T_k| \leq F^{A^*}$ by Lemma V.5. We obtain

$$\begin{aligned}\sum_{j=\underline{k}}^{\overline{k}} (\overline{k} - j + 1) \cdot m \cdot |T_j| &= \sum_{j=\underline{k}}^{\overline{k}-1} \sum_{k=\underline{k}}^{j} m \cdot |T_k| + \sum_{k=\underline{k}}^{\overline{k}} m \cdot |T_k| \\ &\leq \sum_{j=\underline{k}}^{\overline{k}-1} S_j + F^{A^*}\end{aligned}$$

Define function $f(S_{\underline{k}}, ..., S_{\overline{k}-1}) = \sum_{j=\underline{k}}^{\overline{k}-1} S_j$. We then bound f by $O(\log \frac{n}{m}) \cdot F^{A^*}$. Observe the following two facts. First, the sequence $S_{\underline{k}}, ..., S_{\overline{k}-1}$ is non-decreasing and is bounded by F^{A^*}, i.e., $S_{\underline{k}} \leq ... \leq S_{\overline{k}-1} \leq \sum_{j=1}^{n} p_j^* \leq F^{A^*}$. Second, expression $S_j - S_{j-1}$, $\underline{k} \leq j < \overline{k}$, equals the total size of jobs of class j, with convention $S_{\underline{k}-1} = 0$. Since any job of class j has size bounded by 2^{j+1}, the number of jobs of class j is at least $\frac{S_j - S_{j-1}}{2^{j+1}}$. Then, it follows $n \geq \sum_{j=\underline{k}}^{\overline{k}-1} \frac{S_j - S_{j-1}}{2^{j+1}} = \sum_{j=\underline{k}}^{\overline{k}-2} \frac{S_j}{2^{j+2}} + \frac{S_{\overline{k}-1}}{2^{\overline{k}}}$. Consider the following optimization problem

$$\begin{aligned}\max_{S_{\underline{k}}, ..., S_{\overline{k}-1}} \quad & f(S_{\underline{k}}, ..., S_{\overline{k}-1}) = \sum_{j=\underline{k}}^{\overline{k}-1} S_j \\ \text{subject to} \quad & n \geq \sum_{j=\underline{k}}^{\overline{k}-2} \frac{S_j}{2^{j+2}} + \frac{S_{\overline{k}-1}}{2^{\overline{k}}} \\ & F^{A^*} \geq S_{\overline{k}-1} \geq ... \geq S_{\underline{k}}\end{aligned}$$

Function f increases with increasing S_j. Thus, the optimal solution must satisfy that $F^{A^*} = S_{\overline{k}-1} = ... = S_{\overline{k}-l} \geq S_{\overline{k}-l-1}$ and $S_{\overline{k}-l-2} = ... = S_{\underline{k}} = 0$, for an integer l. Indeed, otherwise, one can transform any optimal solution into this form via incrementing S_j and decrementing S_{j-1}, without violating the constraint or decreasing f. We bound l by solving $n = F^{A^*} \cdot (\sum_{j=\overline{k}-l-1}^{\overline{k}-2} \frac{1}{2^{j+2}} + \frac{1}{2^{\overline{k}}})$, yielding $l \leq \log_2 \frac{n \cdot 2^{\overline{k}}}{F^{A^*}}$. By definition of $\overline{k}$, at least m jobs are of class $\overline{k}$ or higher. Any of these jobs has a size of at least $2^{\overline{k}}$. Thus, $F^{A^*} \geq \sum_{j=1}^{n} p_j^* \geq m \cdot 2^{\overline{k}}$ and we have $l \leq \log_2 \frac{n \cdot 2^{\overline{k}}}{F^{A^*}} \leq \log_2 \frac{n}{m}$. It follows that

$$\sum_{j=\underline{k}}^{\overline{k}-1} S_j \leq (l+1) \cdot F^{A^*} \leq (\log_2 \frac{n}{m}) \cdot F^{A^*} + F^{A^*}$$

□

We bound $|U(t)|$. By definition, $|U(t)| < m$ for any partial time instant t. Therefore, we study bounding $|U(t)|$ for full time instants. Fix any full time instant t, and we drop the symbol t when the context is clear. Let t_p be the last (largest) partial time instant before t. Partition the (maximal) full time interval $(t_p, t]$ into disjoint subintervals $(t_{i+1}, t_i]$, $t_{i+1} \le t_i$, such that PEDRMLF is processing jobs only from Q_i or lower in $(t_i, t]$ and t_i is minimal. We have the convention $t_{\underline{k}-1} = t$. Equivalently, t_i is the last time instant before t when PEDRMLF is processing a job from Q_{i+1} or higher. Define the superscript and subscript notations to restrict a set of jobs. Formally, $U^{t_l,t_u}_{c_l,c_u}(t)$ represents a restrictive subset of $U(t)$: $U^{t_l,t_u}_{c_l,c_u}(t) = \{J_j | t_l < r_j \le t_u \wedge c_l \le c_j \le c_u \wedge J_j \in U(t)\}$. The superscript restricts the release time, and the subscript the class. Define *volume* V over a set of jobs to be the total remaining size: $V^{t_l,t_u}_{c_l,c_u}(t) = \sum_{J_j \in U^{t_l,t_u}_{c_l,c_u}(t)} y_j(t)$. We use $U^{t_l,t_u}_{c_l,c_u}(t)$, $U^{*t_l,t_u}_{c_l,c_u}(t)$ and $V^{t_l,t_u}_{c_l,c_u}$, $V^{*t_l,t_u}_{c_l,c_u}$ to denote the quantities for PEDRMLF and the optimal scheduler, respectively. For simplicity, we write $\Delta V^{t_l,t_u}_{c_l,c_u} = V^{*t_l,t_u}_{c_l,c_u} - V^{t_l,t_u}_{c_l,c_u}$. The following two lemmas bound $\Delta V^{t_i,t}_{i,\infty}(t)$ by $|U^*(t)|$.

Lemma V.7 (Bound on Volume Difference)**.** *Fix any full time instant t. For any $k \ge \underline{k}$, $x \ge y$, we have*

$$\Delta V^{t_x,t_y}_{k,\infty}(t) \le 2^{x+1} \cdot |U^{*t_x,t_y}_{k,\infty}(t)|$$

Proof. All quantities in this proof correspond to a fixed time t. Thus, the symbol t is dropped below. PEDRMLF processes jobs only from Q_x or lower in $(t_x, t_y]$, so every job released in $(t_x, t_y]$ receives at most 2^{x+1} time for processing. We have

$$V^{t_x,t_y}_{k,\infty} \ge \sum_{J_j \in U^{*t_x,t_y}_{k,\infty}} (p^*_j - 2^{x+1}) \ge V^{*t_x,t_y}_{k,\infty} - 2^{x+1} \cdot |U^{*t_x,t_y}_{k,\infty}| \qquad \square$$

Lemma V.8 (Bound on Volume Difference Sum)**.** *Fix any full time instant t. For any $\underline{k} \le k \le \overline{k}$, we have*

$$\sum_{i=k}^{\overline{k}} \frac{\Delta V^{t_i,t}_{i,\infty}(t)}{2^i} \le 4 \cdot |U^{*t_{\overline{k}},t}_{k,\infty}(t)|$$

Proof. The symbol t is dropped for the analysis below. By Lemma V.7, we have $\Delta V^{t_{j+1},t_j}_{k,\infty} \le 2^{j+2} \cdot |U^{*t_{j+1},t_j}_{k,\infty}|$ for any k and j. With $|U^{*t_{j+1},t_j}_{i,\infty}| \le |U^{*t_{j+1},t_j}_{k,\infty}|$ for any $i \ge k$, we obtain

$$\begin{aligned}
\sum_{i=k}^{\overline{k}} \frac{\Delta V^{t_i,t}_{i,\infty}}{2^i} &\le \sum_{i=k}^{\overline{k}} \sum_{j=\underline{k}-1}^{i-1} \frac{2^{j+2} \cdot |U^{*t_{j+1},t_j}_{i,\infty}|}{2^i} \\
&\le \sum_{j=\underline{k}-1}^{\overline{k}-1} \sum_{i=\max\{j+1,k\}}^{\overline{k}} \frac{|U^{*t_{j+1},t_j}_{k,\infty}|}{2^{i-j-2}} \le 4 \sum_{j=\underline{k}-1}^{\overline{k}-1} |U^{*t_{j+1},t_j}_{k,\infty}| \\
&= 4 \cdot |U^{*t_{\overline{k}},t}_{k,\infty}| \qquad \square
\end{aligned}$$

Lemma V.9 bounds the inverse volume difference.

Lemma V.9 (Bound on Inverse Volume Difference)**.** *Fix any full time instant t. For any $\underline{k} \le i \le \overline{k}$, we have*

$$-\Delta V^{0,t}_{\underline{k},i}(t) \le (m-1) \cdot 2^{i+2} + \Delta V^{t_{i+1},t}_{i+1,\infty}(t)$$

Proof. The symbol t is dropped for the analysis below. For any $i \geq \underline{k}$, we have $-\Delta V^{0,t}_{\underline{k},i} - \Delta V^{t_{i+1},t}_{i+1,\infty} = V^{0,t_{i+1}}_{\underline{k},i} - (V*^{0,t_{i+1}}_{\underline{k},i} + V*^{t_{i+1},t}_{\underline{k},i}) + V^{t_{i+1},t}_{\underline{k},\infty} - V*^{t_{i+1},t}_{i+1,\infty} \leq V^{0,t_{i+1}}_{\underline{k},i} + V^{t_{i+1},t}_{\underline{k},\infty} - V*^{t_{i+1},t}_{\underline{k},\infty} = V^{0,t_{i+1}}_{\underline{k},i} + (-\Delta V^{t_{i+1},t}_{\underline{k},\infty})$. We then bound $V^{0,t_{i+1}}_{\underline{k},i} + (-\Delta V^{t_{i+1},t}_{\underline{k},\infty})$.

There is at least one active job from Q_{i+2} or higher and at most $m-1$ active jobs from Q_{i+1} or lower at time t_{i+1}. By definition of t_{i+1}, the job(s) from Q_{i+2} or higher are not processed in $(t_{i+1}, t]$. For the rest of this proof, we refer to the active jobs from Q_{i+1} or lower at time t_{i+1} as remaining jobs. Let L denote the amount of time PEDRMLF spends in processing the remaining jobs in $(t_{i+1}, t]$. Since PEDRMLF does not idle any machine in $(t_{i+1}, t]$, it follows $V^{t_{i+1},t}_{\underline{k},\infty} = \sum_{t_{i+1} < r_j \leq t} p^*_j - m \cdot (t - t_{i+1}) + L$. With $V*^{t_{i+1},t}_{\underline{k},\infty} \geq \sum_{t_{i+1} < r_j \leq t} p^*_j - m \cdot (t - t_{i+1})$, term $-\Delta V^{t_{i+1},t}_{\underline{k},\infty} = V^{t_{i+1},t}_{\underline{k},\infty} - V*^{t_{i+1},t}_{\underline{k},\infty}$ is bounded above by L. Therefore, we have $V^{0,t_{i+1}}_{\underline{k},i} + (-\Delta V^{t_{i+1},t}_{\underline{k},\infty}) \leq V^{0,t_{i+1}}_{\underline{k},i} + L$. We then bound $V^{0,t_{i+1}}_{\underline{k},i} + L$. For any remaining job J_j, if $c_j \leq i$, then $p^*_j \leq 2^{i+1}$ and J_j contributes at most 2^{i+1} to $V^{0,t_{i+1}}_{\underline{k},i}$ and at most 2^{i+1} to L. Altogether, J_j contributes 2^{i+2}, if $c_j \leq i$. Otherwise, J_j contributes 0 to $V^{0,t_{i+1}}_{\underline{k},i}$ and at most 2^{i+2} to L, since any running job in $(t_{i+1}, t]$ is from Q_{i+1} or lower. With the number of remaining jobs bounded by $m-1$, it holds that

$$-\Delta V^{0,t}_{\underline{k},i} - \Delta V^{t_{i+1},t}_{i+1,\infty} \leq V^{0,t_{i+1}}_{\underline{k},i} + (-\Delta V^{t_{i+1},t}_{\underline{k},\infty}) \leq (m-1) \cdot 2^{i+2} \qquad \square$$

We combine Lemmas V.7, V.8, and V.9 to establish the following important lemma that bounds the volume in PEDRMLF.

Lemma V.10 (Bound on Volume). *Fix any full time instant t. For any $\underline{k} \leq k \leq \overline{k}$, we have*

$$\sum_{i=k}^{\overline{k}} \frac{V^{0,t}_{i,i}(t)}{2^i} \leq 6 \cdot |U^*(t)| + 2(m-1)(\overline{k} - k + 4)$$

Proof. The symbol t is dropped for the analysis below. With $\frac{\Delta V^{0,t}_{c_l,c_u}}{2^{c_u+1}} \leq \frac{V*^{0,t}_{c_l,c_u}}{2^{c_u+1}} \leq |U*^{0,t}_{c_l,c_u}|$ for any $c_l \leq c_u$, we have

$$\begin{aligned}
\sum_{i=k}^{\overline{k}} \frac{V^{0,t}_{i,i}}{2^i} &= \sum_{i=k}^{\overline{k}} \frac{V*^{0,t}_{i,i} - \Delta V^{0,t}_{i,i}}{2^i} \\
&\leq 2\sum_{i=k}^{\overline{k}} |U*^{0,t}_{i,i}| - \sum_{i=k}^{\overline{k}} \frac{\Delta V^{0,t}_{\underline{k},i} - \Delta V^{0,t}_{\underline{k},i-1}}{2^i} \\
&= 2|U*^{0,t}_{k,\overline{k}}| + \frac{\Delta V^{0,t}_{\underline{k},k-1}}{2^k} - \frac{\Delta V^{0,t}_{\underline{k},\overline{k}}}{2^{\overline{k}}} - \sum_{i=k}^{\overline{k}-1} \frac{\Delta V^{0,t}_{\underline{k},i}}{2^{i+1}} \\
&\leq 2|U*^{0,t}_{k,\overline{k}}| + |U*^{0,t}_{\underline{k},k-1}| + \frac{(m-1) \cdot 2^{\overline{k}+2} + \Delta V^{t_{\overline{k}+1},t}_{\overline{k}+1,\infty}}{2^{\overline{k}}} \\
&\quad + \sum_{i=k}^{\overline{k}-1} \frac{(m-1) \cdot 2^{i+2} + \Delta V^{t_{i+1},t}_{i+1,\infty}}{2^{i+1}} \quad \textit{(By Lemma V.9)}
\end{aligned}$$

$$
\begin{aligned}
&\leq 2|U^{*0,t}_{\underline{k},\overline{k}}| + 4(m-1) + \frac{2^{\overline{k}+2} \cdot |U^{*t_{\overline{k}+1},t}_{\overline{k}+1,\infty}|}{2^{\overline{k}}} \\
&\quad + 2(m-1)(\overline{k}-k) + \sum_{i=k+1}^{\overline{k}} \frac{\Delta V^{t_i,t}_{i,\infty}}{2^i} \quad (\textit{By Lemma V.7}) \\
&\leq 2|U^{*0,t}_{\underline{k},\overline{k}}| + 2(m-1)(\overline{k}-k+2) + 4|U^{*t_{\overline{k}+1},t}_{\overline{k}+1,\infty}| \\
&\quad + 4|U^{*t_{\overline{k}},t}_{\overline{k}+1,\infty}| \quad (\textit{By Lemma V.8}) \\
&\leq 6|U^*| + 2(m-1)(\overline{k}-k+4)
\end{aligned}
$$

The last inequality holds since at most $m-1$ jobs are of a class higher than $\overline{k}$. □

Lemma V.10 will bound $|U(t)|$ under both the good predictions property (Property V.4) and a general prediction error. We begin with the former.

Lemma V.11 (Bound on Uncompleted Jobs under the Good Predictions Property). *Fix any full time instant t. For any $\underline{k} \leq k \leq \overline{k}$, we have*

$$|U^{0,t}_{k,\overline{k}}(t)| \leq 3m \cdot (\overline{k}-k+3) + 6 \cdot |U^*(t)|$$

under the good predictions property.

Proof. The symbol t is dropped for the analysis below. By the earliest release time policy, every $Q_{k+1}, ..., Q_{\overline{k}+1}$ contains at most m jobs that have received processing, i.e., $y_j(t) < p^*_j$, while the other jobs have received no processing, i.e., $y_j(t) = p^*_j$. Thus, at most $m \cdot (\overline{k}-k+1)$ jobs have a remaining size less than the job size, while at most $\frac{V^{0,t}_{i,i}}{2^i}$ jobs of class i have a remaining size equal to the job size. We obtain

$$
\begin{aligned}
|U^{0,t}_{k,\overline{k}}| &\leq m \cdot (\overline{k}-k+1) + \sum_{i=k}^{\overline{k}} \frac{V^{0,t}_{i,i}}{2^i} \\
&\leq m \cdot (\overline{k}-k+1) + 6 \cdot |U^*| + 2(m-1)(\overline{k}-k+4) \\
&\qquad (\textit{By Lemma V.10}) \\
&\leq 3m \cdot (\overline{k}-k+3) + 6 \cdot |U^*|
\end{aligned}
$$
□

Lemma V.12 shows that the contribution of the running jobs to the total response time is bounded by F^{A^*}.

Lemma V.12 (Bound on Response Time of Running Jobs).

$$\int_{t \geq 0} \min\{|U(t)|, m\}\, dt \leq F^{A^*}$$

Proof. At any time t when $|U(t)| < m$, the machine(s) are processing all the active jobs. At any time t when $|U(t)| \geq m$, every one of the m machines is processing a job. Meanwhile, any job is processed on at most one machine at any time. We have $\int_{t\geq 0} \min\{|U(t)|, m\}\, dt \leq \sum_{j=1}^{n} p^*_j \leq F^{A^*}$. □

With Lemma V.12, for any random choices β we write

$$\begin{aligned}
F^A_\beta &= \int_{t\geq 0} |U_\beta(t)|\,dt \\
&= \int_{t\geq 0} \min\{|U_\beta(t)|, m\}\,dt + \int_{t\geq 0, |U_\beta(t)|\geq m} |U_\beta(t)| - m\,dt \\
&\leq F^{A^*} + \int_{t\geq 0, |U_\beta(t)|\geq m} |U_{\beta}{}^{0,t}_{\underline{k},\overline{k}}(t)|\,dt \quad (\textit{By Lemma V.12})
\end{aligned}$$

The last inequality holds since at most $m-1$ jobs are of a class higher than $\overline{k}$. We are ready to prove our consistency results.

Theorem V.13 (The $O(\log P)$ Competitive Ratio under the Good Predictions Property)**.**

$$F^A_\beta(I) \leq 3\cdot(\log_2 P)\cdot F^{A^*}(I) + 19\cdot F^{A^*}(I)$$

for any problem instance I and random choices β under the good predictions property.

Proof. Fix a problem instance I and random choices β. The symbols I and β are dropped for the analysis below. With $\overline{k} - \underline{k} \leq 1 + \log_2 P$, we have

$$\begin{aligned}
F^A &\leq F^{A^*} + \int_{t\geq 0, |U(t)|\geq m} |U^{0,t}_{\underline{k},\overline{k}}(t)|\,dt \\
&\leq F^{A^*} + \int_{t\geq 0, |U(t)|\geq m} 3m\cdot(\overline{k} - \underline{k} + 3)\,dt \\
&\quad + \int_{t\geq 0, |U(t)|\geq m} 6\cdot|U^*(t)|\,dt \quad (\textit{By Lemma V.11}) \\
&\leq 7\cdot F^{A^*} + 3\cdot(\log_2 P + 4)\cdot \int_{t\geq 0, |U(t)|\geq m} m\,dt \\
&\leq 7\cdot F^{A^*} + 3\cdot(\log_2 P + 4)\cdot F^{A^*} \quad (\textit{By Lemma V.12}) \\
&= 3\cdot(\log_2 P)\cdot F^{A^*} + 19\cdot F^{A^*} = O(\log P)F^{A^*}
\end{aligned}$$

□

Theorem V.14 (The $O(\log \frac{n}{m})$ Competitive Ratio under the Good Predictions Property)**.**

$$F^A_\beta(I) \leq 3\cdot(\log_2 \frac{n}{m})\cdot F^{A^*}(I) + 20\cdot F^{A^*}(I)$$

for any problem instance I and random choices β under the good predictions property.

Proof. The symbols I and β are dropped for the analysis below. Partition the full time instants into $T_{\underline{k}}, ..., T_{\overline{k}}$. At most $m-1$ jobs are from Q_{j-1} or lower at $t \in T_j$. We obtain

$$\begin{aligned}
F^A &\leq F^{A^*} + \int_{t\geq 0, |U(t)|\geq m} |U^{0,t}_{\underline{k},\overline{k}}(t)|\,dt \\
&\leq F^{A^*} + \sum_{j=\underline{k}}^{\overline{k}} \int_{t\in T_j} 3m(\overline{k} - j + 3) + 6\cdot|U^*(t)| + m\,dt \\
&\quad (\textit{By Lemma V.11})
\end{aligned}$$

$$\leq F^{A^*} + 7\int_{t\geq 0, |U(t)|\geq m} m\,dt + 6\int_{t\geq 0} |U^*(t)|\,dt$$

$$+ 3 \cdot \sum_{j=\underline{k}}^{\overline{k}} (\overline{k} - j + 1) \cdot m \cdot |T_j|$$

$$\leq 14 \cdot F^{A^*} + 3 \cdot ((\log_2 \frac{n}{m}) \cdot F^{A^*} + 2F^{A^*})$$

(*By Lemma V.12 and Lemma V.6*)

$$= 3(\log_2 \frac{n}{m}) \cdot F^{A^*} + 20 \cdot F^{A^*} = O(\log \frac{n}{m})F^{A^*} \qquad \square$$

Remark V.15. *Theorem V.1 is proved by combining Theorems V.13 and V.14. The good predictions property holds when $\eta = 1$, and the above bounds hold for any β. It follows that $E[F^A_\beta(I)] \leq \min\{3 \cdot (\log_2 P) + 19, 3 \cdot (\log_2 \frac{n}{m}) + 20\} \cdot F^{A^*}(I) = O(\min\{\log \frac{n}{m}, \log P\}) \cdot F^{A^*}(I)$.*

Theorem V.16 (Optimality of Consistency for Algorithm 7)**.** *Algorithm* PEDRMLF *has asymptotically optimal consistency.*

Proof. Recall that the lower bound for the competitive ratio of any clairvoyant algorithm is $\Omega(\max\{\log \frac{n}{m}, \log P\})$. By the Fundamental Theorems of Consistency (Theorem II.4), Algorithm PEDRMLF has consistency matching the lower bound. $\square$

5.4.2 Probabilistic Analysis

This section considers the general case with $\eta > 1$, where randomization is critical in bounding the performance. We reintroduce subscript β to represent the execution of PEDRMLF determined by given random choices $\beta = (\beta_{j,i})$.

We begin with the intuitions and key ideas in our probabilistic analysis. Without good predictions (Property V.4), a job may travel multiple queues before its completion. We thus want PEDRMLF to minimize the jobs with small remaining sizes in wait or to keep significant remaining sizes when jobs enter their final queue. The jobs are classified into *unlucky*, *lucky*, and *big*. Unlucky jobs have small remaining sizes while entering the highest queue they can. Lucky jobs are those not unlucky. Big jobs are the lucky ones with the additional property of having a significant remaining size when entering their final queue. We first show that unlucky jobs are neglectable in the analysis. Then, we relate lucky jobs and big jobs. The key observation is a linear relationship between the expected number of lucky jobs and big jobs. This observation allows us to bound $|U(t)|$ via bounding big jobs only, which is explicit by expressing $|U(t)|$ in terms of the volume V. The proof structure for probabilistic analysis is given in Figure V.5.

We define a job J_j to be *unlucky* if $p^*_j < 2^{c_j} + 2^{c_j - 1}$ and J_j ends in Q_{c_j+1}. Otherwise, job J_j is said to be *lucky*. Define a job J_j to be *big* at time t if J_j is lucky and within Q_i at time t and has remaining size at least $\frac{p^*_j}{2\log_2(j+1)}$ when entering Q_i, i.e., if J_j enters Q_i at time $q \leq t$, $y_j(q) \geq \frac{p^*_j}{2\log_2(j+1)}$ for lucky job J_j. Define $U^u_\beta(t)$, $U^l_\beta(t)$, and $U^b_\beta(t)$ to be the set of unlucky active, lucky active, and big jobs at time t,

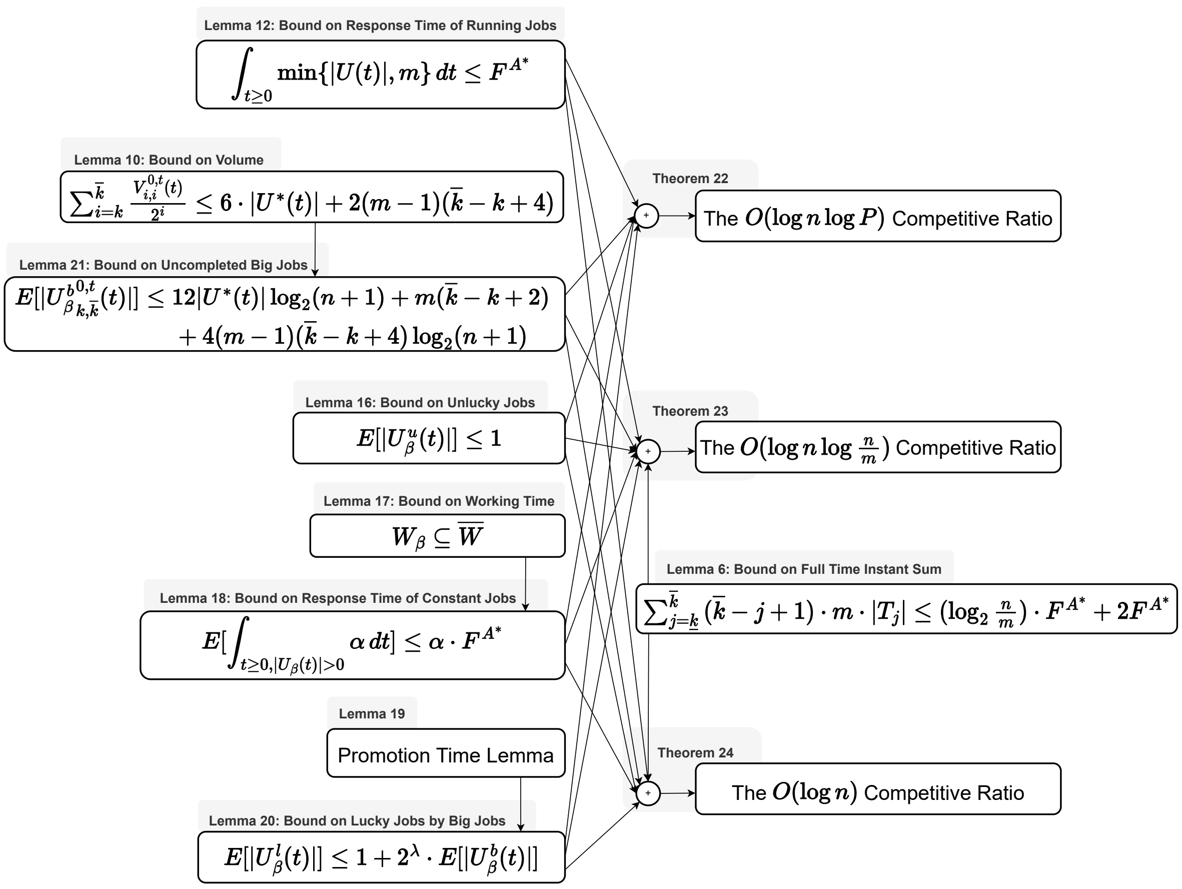

Lemma 12: Bound on Response Time of Running Jobs
$\int_{t\geq 0} \min\{|U(t)|, m\}\, dt \leq F^{A^*}$
Lemma 10: Bound on Volume
$\sum_{i=k}^{\bar{k}} \frac{V_{i,i}^{0,t}(t)}{2^i} \leq 6 \cdot |U^*(t)| + 2(m-1)(\bar{k}-k+4)$
Lemma 21: Bound on Uncompleted Big Jobs
$E[|U_{\beta_{k,\bar{k}}}^{b^{0,t}}(t)|] \leq 12|U^*(t)|\log_2(n+1) + m(\bar{k}-k+2) + 4(m-1)(\bar{k}-k+4)\log_2(n+1)$
Lemma 16: Bound on Unlucky Jobs
$E[|U_\beta^u(t)|] \leq 1$
Lemma 17: Bound on Working Time
$W_\beta \subseteq \overline{W}$
Lemma 18: Bound on Response Time of Constant Jobs
$E[\int_{t\geq 0, |U_\beta(t)|>0} \alpha\, dt] \leq \alpha \cdot F^{A^*}$
Lemma 19
Promotion Time Lemma
Lemma 20: Bound on Lucky Jobs by Big Jobs
$E[|U_\beta^l(t)|] \leq 1 + 2^\lambda \cdot E[|U_\beta^b(t)|]$
Theorem 22
The $O(\log n \log P)$ Competitive Ratio
Theorem 23
The $O(\log n \log \frac{n}{m})$ Competitive Ratio
Lemma 6: Bound on Full Time Instant Sum
$\sum_{j=\underline{k}}^{\bar{k}} (\bar{k}-j+1) \cdot m \cdot |T_j| \leq (\log_2 \frac{n}{m}) \cdot F^{A^*} + 2F^{A^*}$
Theorem 24
The $O(\log n)$ Competitive Ratio

Figure V.5: Proof structure for probabilistic analysis.

respectively. For every time t, define indicator variables $X_j^u(t)$, $X_j^l(t)$, $X_j^b(t)$, and $X_j^a(t)$, which takes value 1 if J_j is unlucky, lucky, big, and active at time t or 0 otherwise, respectively. Note that X_j^u and X_j^l are constants under fixed β, independent of time, while X_j^b and X_j^a depend on the time of interest. We first show that $E[|U_\beta^u(t)|]$ is small, so unlucky jobs are insignificant to the total response time.

Lemma V.17 (Bound on Unlucky Jobs)**.**

$$E[|U_\beta^u(t)|] \leq 1$$

for any time t.

Proof. We write $X_j^u(t) = X_j^u$, as it is constant with fixed β independent to t. By definition of unlucky jobs, we have

$$P[X_j^u = 1] = P[p_j^* < 2^{c_j} + 2^{c_j-1} \wedge p_{j,c_j}^e < p_j^*]$$
$$\leq P[\beta_{j,c_j} > 2^{c_j-1}] = (j+1)^{-\frac{\lambda}{2}} \leq \frac{1}{(j+1)^2}$$
$$(\textit{By definition of } p_{j,c_j}^e \textit{ and } \lambda \geq 4)$$

for any job J_j. Therefore, we obtain

$$E[|U^u_\beta(t)|] = E[\sum_{j=1}^{n} X^u_j \cdot X^a_j(t)] \leq E[\sum_{j=1}^{n} X^u_j]$$
$$= \sum_{j=1}^{n} E[X^u_j] = \sum_{j=1}^{n} P[X^u_j = 1] \leq \sum_{j=1}^{n} \frac{1}{(j+1)^2} \leq 1 \qquad \square$$

Lemma V.17 shows that our probability distribution for randomization bounds the expected number of unlucky jobs down to a constant 1. This compares to the previous works [41, 45], where the expected number of the unlucky jobs is in $O(\log n)$.

We then bound the time instants when PEDRMLF is processing job(s). Define set $W_\beta = \{t | t \geq 0 \wedge |U_\beta(t)| > 0\}$. Consider a trivial scheduling algorithm R which uses only one single machine and always runs the active job with the smallest job index to its completion. Define set $\overline{W}$ to be the time instants when R is processing a job. Denote $|W_\beta| = \int_{t\in W_\beta} dt$ and similarly $|\overline{W}| = \int_{t\in\overline{W}} dt$. Clearly, it holds $|\overline{W}| = \sum_{j=1}^{n} p^*_j \leq F^{A^*}$. Lemma V.18 bounds W_β by $\overline{W}$.

Lemma V.18 (Bound on Working Time)**.**

$$W_\beta \subseteq \overline{W}$$

for any random choices β.

Proof. Both PEDRMLF and R maintain the invariant that at least one machine is processing a job if there is an active job. Since R uses a single machine while PEDRMLF uses m machines ($m \geq 1$), it follows that $t \in \overline{W}$ for any $t \in W_\beta$. $\square$

We show that a constant number of jobs are insignificant to the total response time. This, together with Lemma V.17, implies that the unlucky jobs are insignificant.

Lemma V.19 (Bound on Response Time of Constant Jobs)**.**

$$E[\int_{t\geq 0, |U_\beta(t)|>0} \alpha \, dt] \leq \alpha \cdot F^{A^*}$$

for any constant $\alpha \geq 0$.

Proof.

$$E[\int_{t\geq 0, |U_\beta(t)|>0} \alpha \, dt] = E[\int_{t\in W_\beta} \alpha \, dt] \leq E[\int_{t\in\overline{W}} \alpha \, dt]$$
$$= \alpha \cdot |\overline{W}| \leq \alpha \cdot F^{A^*} \quad (\textit{By Lemma V.18}) \qquad \square$$

The following Promotion Time Lemma shows how random choice $\beta_{j,s}$ controls when job J_j leaves Q_{s+1}, with the other random choices fixed. We define *promotion time* of J_j from Q_{s+1} to be the time instant J_j leaves Q_{s+1} or the time instant when J_j leaves Q_s if J_j is completed when leaving Q_s. We show that the promotion time of J_j from Q_{s+1} is non-increasing with decreasing $\beta_{j,s}$.

Lemma V.20 (Promotion Time Lemma)**.** *For any job J_j and integer $s \leq c_j$, the promotion time of J_j from Q_{s+1} is non-increasing with decreasing $\beta_{j,s}$, with fixed random choices $\beta_{k,i}$, $k \neq j$ or $i \neq s$.*

Proof. If job J_j is completed when leaving Q_s, it must hold $p^e_{j,s} \geq p^*_j$. Then, every execution of PEDRMLF is identical to each other before time a, the first time instant when $x_j(a)$ increases to p^*_j. Therefore, the promotion time of J_j from Q_{s+1} remains at constant a, as long as $p^e_{j,s} \geq p^*_j$. In the following analysis, we consider $p^e_{j,s} \leq p^*_j$ and show that the promotion time of J_j from Q_{s+1} in the execution of PEDRMLF with $p^e_{j,s} = x$ is no less than with $p^e_{j,s} = y$, for any $x < y \leq p^*_j$.

Run PEDRMLF twice with fixed random choices $\beta_{k,i}$, $k \neq j$ or $i \neq s$. The first time we set $p^e_{j,s} = x$, and the second time $p^e_{j,s} = y$. We use superscripts (1) and (2) to denote the quantities in the first and second executions. Let $t^{(1)}$ and $t^{(2)}$ denote the promotion time of J_j from Q_{s+1} in the two executions. Let $\underline{t}$ be the promotion time of J_j from Q_s in execution 1. Two executions are identical to each other before $\underline{t}$. It follows $x^{(1)}_k(\underline{t}) = x^{(2)}_k(\underline{t})$ for every job J_k. Then, we show that $t^{(1)} \geq t^{(2)}$ using proof by contradiction.

Assume $t^{(1)} < t^{(2)}$. There exists a time instant $t^c \in (\underline{t}, t^{(1)}]$ such that $x^{(1)}_j(t^c) > x^{(2)}_j(t^c)$ and J_j is not running in execution 2 at time t^c. Define set $S(t) = \{J_k | k \neq j \wedge (J_k \textit{ is within } Q_s \textit{ or lower at time } t \vee J_k \textit{ is within } Q_{s+1} \textit{ and } k < j \textit{ at time } t)\}$. PEDRMLF maintains the invariant of processing the jobs with the highest priority of being within the lowest queue(s) and having the smallest job index(s) within a queue. For any time $t \in (\underline{t}, t^c]$, observe that (i) both executions must be processing the jobs with the highest priority from $S^{(1)}(t) \cup \{J_j\}$ and $S^{(2)}(t) \cup \{J_j\}$ respectively, (ii) job J_j is either within Q_s or Q_{s+1} in execution 2, while it stays within Q_{s+1} in execution 1, (iii) any job $J_k \in S^{(2)}(t)$ running in execution 2 is also running in execution 1 as long as $J_k \in S^{(1)}(t)$ by observation (ii) and the fixed relative priority between jobs within any queue, and (iv) if $J_k \in S^{(2)}(t)$ but $J_k \notin S^{(1)}(t)$, job J_k must have been promoted to a higher queue or completed by time t in execution 1. By combining these observations, it follows $x^{(1)}_k(t) \geq x^{(2)}_k(t)$ for any $J_k \in S^{(2)}(t)$, and $S^{(1)}(t) \subseteq S^{(2)}(t)$ for any $t \in (\underline{t}, t^c]$. By the invariant that PEDRMLF maintains with $S^{(1)}(t) \subseteq S^{(2)}(t)$, the total amount of work done to jobs in $S^{(2)}(t) \cup \{J_j\}$ in execution 2 is at least that in execution 1, for any $t \in (\underline{t}, t^c]$. However, we have shown $x^{(1)}_k(t^c) \geq x^{(2)}_k(t^c)$ for every $J_k \in S^{(2)}(t^c)$ and $x^{(1)}_j(t^c) > x^{(2)}_j(t^c)$, i.e., the total amount of work done to jobs in $S^{(2)}(t^c) \cup \{J_j\}$ in execution 1 is more than that in execution 2. This is a contradiction.

With $p^e_{j,s}$ non-decreasing with decreasing $\beta_{j,s}$, the promotion time of J_j from Q_{s+1} is non-increasing with decreasing $\beta_{j,s}$. □

Lemma V.21 bounds $E[|U^l_\beta(t)|]$ by $E[|U^b_\beta(t)|]$.

Lemma V.21 (Bound on Lucky Jobs by Big Jobs)**.**

$$E[|U^l_\beta(t)|] \leq 1 + 2^\lambda \cdot E[|U^b_\beta(t)|]$$

for any time t.

Proof. Fix any time t, and we drop the symbol t for the analysis below. The bound follows the inequality

$$P[X_j^b = 1] \geq \frac{1}{2^\lambda} \cdot P[X_j^l \cdot X_j^a = 1]$$

for any job J_j, $j \geq 2$. We first prove this inequality. Fix any lucky job J_j active at time t, and consider the following cases. If J_j is within Q_{c_j-1} or lower, it is always big: if J_j enters its current queue at time q, it follows that $x_j(q) \leq 2^{c_j-1}$ and $y_j(q) = p_j^* - x_j(q) \geq p_j^* - 2^{c_j-1} \geq \frac{p_j^*}{2} \geq \frac{p_j^*}{2\log_2(j+1)}$. Therefore, $P[X_j^b = 1] = P[X_j^l \cdot X_j^a = 1]$ if J_j is within Q_{c_j-1} or lower. Next, consider the case when J_j is within Q_{c_j} or Q_{c_j+1}. If J_j enters Q_{c_j} or Q_{c_j+1} at arrival, it immediately follows $P[X_j^b = 1] = P[X_j^l \cdot X_j^a = 1]$. Otherwise, for J_j within Q_{c_j} with fixed random choices $\beta_{k,i}$, $k \neq j, i \neq c_j - 1$, there exists a minimum value ρ such that J_j is active within Q_{c_j} at time t if $\beta_{j,c_j-1} \geq \rho$ by Lemma V.20. We obtain

$$\begin{aligned}
&P[X_j^b = 1 | X_j^l \cdot X_j^a = 1] \\
&= P[p_{j,c_j-1}^e \leq p_j^*(1 - \frac{1}{2\log_2(j+1)}) | \beta_{j,c_j-1} \geq \rho] \\
&\geq P[p_{j,c_j-1}^e \leq p_j^*(1 - \frac{1}{2\log_2(j+1)})] \\
&\qquad (\textit{By } p_{j,c_j-1}^e \textit{ non-increasing with increasing } \beta_{j,c_j-1}) \\
&= P[\beta_{j,c_j-1} \geq 2^{c_j} - p_j^*(1 - \frac{1}{2\log_2(j+1)})] \\
&= (j+1)^{\frac{-\lambda}{2^{c_j-1}} \cdot (2^{c_j} - p_j^*(1 - \frac{1}{2\log_2(j+1)}))} \geq 2^{-\lambda}
\end{aligned}$$

Similarly, for J_j within Q_{c_j+1} with fixed random choices $\beta_{k,i}$, $k \neq j, i \neq c_j$, there exists a minimum ρ' such that J_j is active within Q_{c_j+1} at time t if $\beta_{j,c_j} \geq \rho'$ by Lemma V.20. Lucky job J_j entering Q_{c_j+1} implies $2^{c_j} + 2^{c_j-1} \leq p_j^* < 2^{c_j+1}$ and $p_{j,c_j}^e < p_j^*$. Observe $2^{c_j} \leq p_j^*(1 - \frac{1}{2\log_2(j+1)})$ for any $j \geq 2$ and $p_j^* \geq 2^{c_j} + 2^{c_j-1}$. We obtain

$$\begin{aligned}
&P[X_j^b = 1 | X_j^l \cdot X_j^a = 1] \\
&= P[p_{j,c_j}^e \leq p_j^*(1 - \frac{1}{2\log_2(j+1)}) | \beta_{j,c_j} \geq \rho' \wedge p_{j,c_j}^e < p_j^*] \\
&\geq P[p_{j,c_j}^e \leq p_j^*(1 - \frac{1}{2\log_2(j+1)}) | \beta_{j,c_j} \geq 2^{c_j+1} - p_j^*] \\
&\qquad (\textit{By } p_{j,c_j}^e \textit{ non-increasing with increasing } \beta_{j,c_j}) \\
&= \frac{P[\beta_{j,c_j} \geq 2^{c_j+1} - p_j^*(1 - \frac{1}{2\log_2(j+1)})]}{P[\beta_{j,c_j} \geq 2^{c_j+1} - p_j^*]} = 2^{\frac{-\lambda \cdot p_j^*}{2^{c_j+1}}} \geq 2^{-\lambda}
\end{aligned}$$

We have $P[X_j^b = 1] = P[X_j^b = 1 | X_j^l \cdot X_j^a = 1] \cdot P[X_j^l \cdot X_j^a = 1] \geq \frac{1}{2^\lambda} \cdot P[X_j^l \cdot X_j^a = 1]$ for any job J_j, $j \geq 2$. We conclude

$$E[|U_\beta^l|] \leq 1 + \sum_{j=2}^{n} E[X_j^l \cdot X_j^a] = 1 + \sum_{j=2}^{n} P[X_j^l \cdot X_j^a = 1]$$

$$\le 1 + 2^{\lambda} \cdot \sum_{j=2}^{n} P[X_j^b = 1] = 1 + 2^{\lambda} \cdot E[|U_\beta^b|] \qquad \square$$

Lemma V.22 bounds $E[|U_\beta^b(t)|]$.

Lemma V.22 (Bound on Uncompleted Big Jobs under $\eta > 1$)**.** *Fix any full time instant* t*, and for any* $\underline{k} \le k \le \overline{k}$*, we have*

$$E[|U^{b\,0,t}_{\beta_{k,\overline{k}}}(t)|] \le 12|U^*(t)|\log_2(n+1) + m(\overline{k} - k + 2)$$
$$+ 4(m-1)(\overline{k} - k + 4)\log_2(n+1)$$

for any $\eta > 1$.

Proof. The symbol t is dropped for the analysis below. The bound is established by bounding the number of big jobs, $|U^{b\,0,t}_{\beta_{k,\overline{k}}}|$, by $|U^*|$ for any β. Fix random choices β, and we drop the symbol β below. We focus on $Q_{\overline{k}+1}$ and lower, as the jobs of class at least k occur only in them. Any job J_j with $c_j \ge k$ is always big in Q_{k-1} or lower. Every $Q_k, ..., Q_{\overline{k}+1}$ contains at most m jobs that have received processing after entering the queue due to the earliest release time first policy. Any other big job J_j has remaining size at least $\frac{p_j^*}{2\log_2(j+1)} \ge \frac{2^{c_j}}{2\log_2(n+1)}$. It follows $|U^{b\,0,t}_{k,\overline{k}}| \le m \cdot (\overline{k} - k + 2) + 2 \cdot \log_2(n+1) \cdot \sum_{i=k}^{\overline{k}} \frac{V_{i,i}^{0,t}}{2^i} \le 2 \cdot \log_2(n+1) \cdot (6 \cdot |U^*| + 2(m-1)(\overline{k} - k + 4)) + m \cdot (\overline{k} - k + 2)$ by Lemma V.10. $\square$

Observe $|U_{\beta}{}^{0,t}_{\underline{k},\overline{k}}(t)| \le |U_\beta^u(t)| + |U^{l\,0,t}_{\beta_{\underline{k},\overline{k}}}(t)|$ for any t and random choices β. We write

$$E[F_\beta^A] \le F^{A^*} + \int_{t \ge 0, |U_\beta(t)| \ge m} E[|U_{\beta}{}^{0,t}_{\underline{k},\overline{k}}(t)|]\, dt$$
$$\le F^{A^*} + \int_{t \ge 0, |U_\beta(t)| \ge m} E[|U_\beta^u(t)|] + E[|U^{l\,0,t}_{\beta_{\underline{k},\overline{k}}}(t)|]\, dt$$
$$\le 2^{\lambda} \int_{t \ge 0, |U_\beta(t)| \ge m} E[|U^{b\,0,t}_{\beta_{\underline{k},\overline{k}}}(t)|]\, dt + E[\int_{t \ge 0, |U_\beta(t)| > 0} 2\, dt]$$
$$+ F^{A^*} \quad (\textit{By Lemma V.17 and Lemma V.21})$$
$$\le 3F^{A^*} + 2^{\lambda} \int_{t \ge 0, |U_\beta(t)| \ge m} E[|U^{b\,0,t}_{\beta_{\underline{k},\overline{k}}}(t)|]\, dt \quad (\textit{By Lemma V.19})$$

from which we prove our robustness results.

Theorem V.23 (The $O(\log n \log P)$ Competitive Ratio under $\eta > 1$ on Parallel Machines)**.**

$$E[F_\beta^A(I)] \le 2^{\lambda+2} \cdot (\log_2(n+1) + 1) \cdot (\log_2 P + 8) \cdot F^{A^*}(I)$$

for any problem instance I *and* $m > 1$.

Proof. Fix a problem instance I. The symbol I is dropped for the analysis below. We have

$$E[F_\beta^A] \le 3F^{A^*} + 2^{\lambda} \int_{t \ge 0, |U_\beta(t)| \ge m} E[|U^{b\,0,t}_{\beta_{\underline{k},\overline{k}}}(t)|]\, dt$$

$$\leq 2^{\lambda+2}\log_2(n+1)\int_{t\geq 0,|U_\beta(t)|\geq m}(m-1)(\overline{k}-\underline{k}+4)\,dt$$

$$+\,2^{\lambda+2}\cdot 3\cdot\log_2(n+1)\int_{t\geq 0,|U_\beta(t)|\geq m}|U^*(t)|\,dt$$

$$+\,2^{\lambda}\int_{t\geq 0,|U_\beta(t)|\geq m}m(\overline{k}-\underline{k}+2)\,dt+3F^{A^*}$$

(*By Lemma V.22*)

$$\leq 2^{\lambda+2}\log_2(n+1)(\log_2 P+5)F^{A^*}+2^{\lambda}(\log_2 P+3)F^{A^*}$$

$$+\,3\cdot 2^{\lambda+2}\cdot\log_2(n+1)\cdot F^{A^*}+3\cdot F^{A^*}$$

(*By* $\overline{k}-\underline{k}\leq 1+\log_2 P$ *and Lemma V.12*)

$$\leq 2^{\lambda+2}(\log_2(n+1)+1)(\log_2 P+8)F^{A^*}\quad (\textit{By } \lambda\geq 4)$$

$$=O(\log n\log P)\cdot F^{A^*}$$ □

Theorem V.24 (The $O(\log n\log\frac{n}{m})$ Competitive Ratio under $\eta>1$ on Parallel Machines)**.**

$$E[F_\beta^A(I)]\leq 2^{\lambda+2}\cdot\log_2(n+1)\cdot(\log_2\frac{n}{m}+9)\cdot F^{A^*}(I)$$

for any problem instance I and $m>1$.

Proof. The symbol I is dropped for the analysis below. Partition the full time instants into $T_{\underline{k}},\ldots,T_{\overline{k}}$. At most $m-1$ jobs are from Q_{j-1} or lower at time $t\in T_j$. We have

$$E[F_\beta^A]\leq 3F^{A^*}+2^{\lambda}\int_{t\geq 0,|U_\beta(t)|\geq m}E[|U^{b0,t}_{\beta_{\underline{k},\overline{k}}}(t)|]\,dt$$

$$\leq 3F^{A^*}+2^{\lambda}\sum_{j=\underline{k}}^{\overline{k}}\int_{t\in T_j}E[|U^{b0,t}_{\beta_{j,\overline{k}}}|]+m\,dt$$

$$\leq 3F^{A^*}+2^{\lambda+2}\cdot 3\cdot\log_2(n+1)\sum_{j=\underline{k}}^{\overline{k}}\int_{t\in T_j}|U^*(t)|\,dt$$

$$+\,2^{\lambda+2}\log_2(n+1)\sum_{j=\underline{k}}^{\overline{k}}\int_{t\in T_j}(m-1)(\overline{k}-j+4)\,dt$$

$$+\,2^{\lambda}\sum_{j=\underline{k}}^{\overline{k}}\int_{t\in T_j}m(\overline{k}-j+3)\,dt\quad(\textit{By Lemma V.22})$$

$$\leq 3F^{A^*}+2^{\lambda+2}\cdot 3\cdot\log_2(n+1)\cdot\int_{t\geq 0}|U^*(t)|\,dt$$

$$+\,(2^{\lambda+2}\log_2(n+1)+2^{\lambda})\sum_{j=\underline{k}}^{\overline{k}}(\overline{k}-j+1)\cdot m\cdot|T_j|$$

$$+\,(2^{\lambda+2}\cdot 3\cdot\log_2(n+1)+2^{\lambda+1})\int_{t\geq 0,|U_\beta(t)|\geq m}m\,dt$$

$$
\begin{aligned}
&\le 3F^{A^*} + 2^{\lambda+2} \cdot 3 \cdot \log_2(n+1) \cdot F^{A^*} \\
&\quad + (2^{\lambda+2} \log_2(n+1) + 2^{\lambda}) \cdot (\log_2 \frac{n}{m} + 2) \cdot F^{A^*} \\
&\quad + (2^{\lambda+2} \cdot 3 \cdot \log_2(n+1) + 2^{\lambda+1}) \cdot F^{A^*} \\
&\quad (\textit{By Lemma V.6 and Lemma V.12}) \\
&\le 2^{\lambda+2} \log_2(n+1)(\log_2 \frac{n}{m} + 9) F^{A^*} = O(\log n \log \frac{n}{m}) F^{A^*} \\
&\quad (\textit{By } \lambda \ge 4 \textit{ and } n > m > 1)
\end{aligned}
$$ □

Theorem V.25 (The $O(\log n)$ Competitive Ratio under $\eta > 1$ on a Single Machine)**.**

$$E[F_\beta^A(I)] \le 13 \cdot 2^{\lambda} \cdot (\log_2(n+1) + 1) \cdot F^{A^*}(I)$$

for any problem instance I and $m = 1$.

Proof. Following the same proof strategy used in Theorem V.24 with $m = 1$, we have

$$
\begin{aligned}
E[F_\beta^A] &\le 3F^{A^*} + 2^{\lambda} \sum_{j=\underline{k}}^{\overline{k}} \int_{t \in T_j} E[|U_{\beta j,\overline{k}}^{b0,t}|] + m \, dt \\
&\le 3F^{A^*} + 2^{\lambda+2} \cdot 3 \cdot \log_2(n+1) \sum_{j=\underline{k}}^{\overline{k}} \int_{t \in T_j} |U^*(t)| \, dt \\
&\quad + 2^{\lambda} \sum_{j=\underline{k}}^{\overline{k}} \int_{t \in T_j} m(\overline{k} - j + 3) \, dt \quad (\textit{By Lemma V.22}) \\
&\le 13 \cdot 2^{\lambda} \cdot (\log_2(n+1) + 1) \cdot F^{A^*} = O(\log n) \cdot F^{A^*}
\end{aligned}
$$ □

Remark V.26. *Theorem V.2 is proved by combining Theorems V.23 – V.25. It follows* $E[F_\beta^A(I)] \le 13 \cdot 2^{\lambda} \cdot (\log_2(n+1)+1) \cdot F^{A^*}(I) = O(\log n) \cdot F^{A^*}(I)$ *on a single machine, and* $E[F_\beta^A(I)] \le \min\{2^{\lambda+2} \cdot (\log_2(n+1)+1)(\log_2 P + 8), 2^{\lambda+2} \cdot \log_2(n+1)(\log_2 \frac{n}{m} + 9)\} \cdot F^{A^*}(I) = O(\min\{\log n \log \frac{n}{m}, \log n \log P\}) \cdot F^{A^*}(I)$ *on parallel machines, matching the best-known competitive ratio under the non-clairvoyant case.*

5.4.3 Complexity Analysis

Theorem V.27 (Time and Space Complexity)**.** *Algorithm* PEDRMLF *admits an implementation with $O(\log n)$ time complexity for each online decision and $O(n)$ space complexity.*

Proof. We present an implementation of PEDRMLF achieving the claimed complexities. For every job J_j, maintain its processed size, the queue it resides in, and the time it completes running within its queue for the running job J_j. We maintain these data in list-like structures, so access and update take $O(1)$ time per operation. Maintain a single priority queue of active non-running jobs stored as ordered pairs (*queue index*, *job index*). A job with a lower queue index has higher priority, with tie-breaking by a smaller job index. Maintain the upcoming events when job complete their processing as a balanced binary search tree, in which an event is stored as an ordered pair (*completion time*, *job index*).

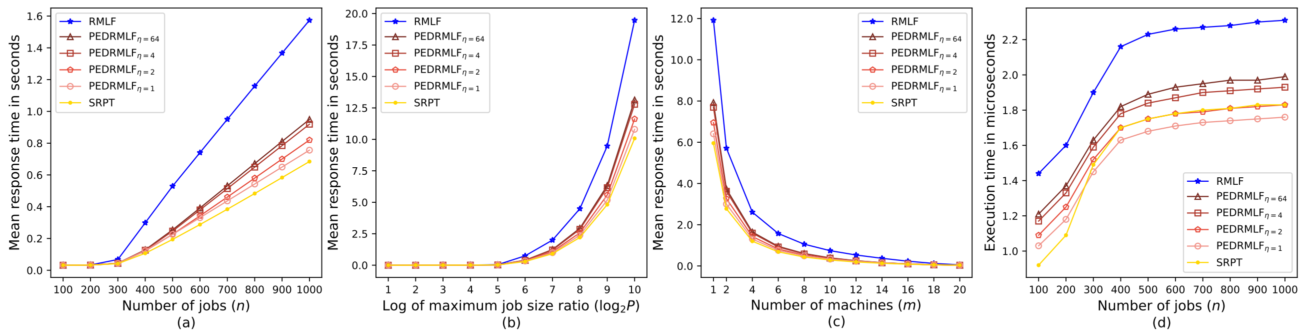


Figure V.6: Performance comparison between PEDRMLF, RMLF, and SRPT. PEDRMLF is executed four times with different prediction error $\eta = \{1, 2, 4, 64\}$.

Finally, maintain the set of running jobs as a balanced binary search tree, in which a job is stored as an ordered pair $(\textit{queue index}, \textit{job index})$. Accessing and updating a priority queue or balanced binary search tree takes $O(\log n)$ time per operation. Each online decision of PEDRMLF requires a constant number of times of accessing and updating these data structures, yielding $O(\log n)$ run-time complexity. Meanwhile, the space complexity for maintaining these data is $O(n)$. □

Remark V.28. *Theorem V.27 shows the complexity of each online decision made by PEDRMLF. At most n events can happen at any single time instant. Therefore, the real-time complexity for PEDRMLF is bounded by $O(n \log n)$ at any time.*

5.5 Experimental Evaluation

This section evaluates the performance of PEDRMLF on randomly-generated synthetic workloads by comparing its mean response time and the execution time against those from SRPT [39] and RMLF [41]. SRPT and RMLF are the best-known algorithms for mean response time scheduling applicable to single and parallel machines [35, 39, 41, 42]. The former holds an optimal competitive ratio for the clairvoyant case. The latter represents the state-of-the-art competitive ratio for the non-clairvoyant case.

5.5.1 Workload Generation

We generate 10^5 workloads to cover a rich set of scenarios. Table V.2 lists the ranges and default values of the parameters used for workload generation. The parameters n, m, and P are set to default values unless otherwise noted, while the others are randomly generated from the given ranges. We assume the prediction error follows the uniform distribution, so p_j is randomly selected from the interval $[\frac{p_j^*}{\eta}, \eta \cdot p_j^*]$. The uniform distribution of prediction error allows us to evaluate the performance under an adversary setting, where the prediction quality is less than under other favorable settings like normal distributions centered at the exact values.

Parameter	Range	Default value
number of jobs (n)	$[100, 1000]$	600
number of machines (m)	$[1, 20]$	10
maximum job size ratio (P)	$[2, 1024]$	64
job size (p_j^*)	$[1, P]$	N/A
release time (r_j)	$[0, 1000]$	N/A
prediction error (η)	$[1, 64]$	N/A
job size prediction (p_j)	$[\frac{p_j^*}{\eta}, \eta \cdot p_j^*]$	N/A

Table V.2: The workload parameters, with time-related ones in milliseconds.

5.5.2 Experiments

We run PEDRMLF, RMLF, and SRPT on each workload. To study the effects of prediction error, we test PEDRMLF under different prediction errors. We record the mean response time and execution time per test. Each test result for PEDRMLF and RMLF is computed as the average of 500 independent executions as the expected performance. We ensure that the algorithms run under the optimum conditions for a fair comparison. That means during the run, SRPT knows the exact job sizes, RMLF knows the exact minimum job size, and PEDRMLF knows the job size predictions and the total prediction error η only.

5.5.3 Experimental Results

Figure V.6 shows the mean response time (Figure V.6 (a) – (c)) and the execution time per test (Figure V.6 (d)) for the three algorithms. As shown in Figure V.6 (a) – (c), the mean response time increases with increasing jobs (n) and increasing maximum job size ratio (P), while it decreases with increasing machines (m). With increasing jobs, the number of uncompleted jobs at any time, $|U(t)|$, increases accordingly, causing an increased mean response time. With an increasing maximum job size ratio, the mean job size increases and thus requires more time to complete. In contrast, jobs can be completed in a shorter time if the system has more machines, which decreases $|U(t)|$ and the mean response time. Observe that the competitive ratio of PEDRMLF ($O(\min\{\log \frac{n}{m}, \log P\})$ consistency and $O(\min\{\log n \log \frac{n}{m}, \log n \log P\})$ robustness) shares the same concern with these parameters, which increases with increasing n or increasing P or decreasing m.

PEDRMLF steadily outperforms RMLF in mean response time. The mean response time of PEDRMLF with any prediction error is half of RMLF when jobs increase to 1000. Our algorithm provides performance at least twice as good in the tests. We also see the performance of PEDRMLF improves as the prediction error reduces, while it stays not far from SRPT even if the prediction error is as unusually large as 64.

Besides having a shorter mean response time than RMLF, PEDRMLF also shows a short execution time (Figure V.6 (d)). It is interesting to note a positive correlation between mean response time and execution time. The increasing mean response time suggests that increasing uncompleted jobs cause more data to

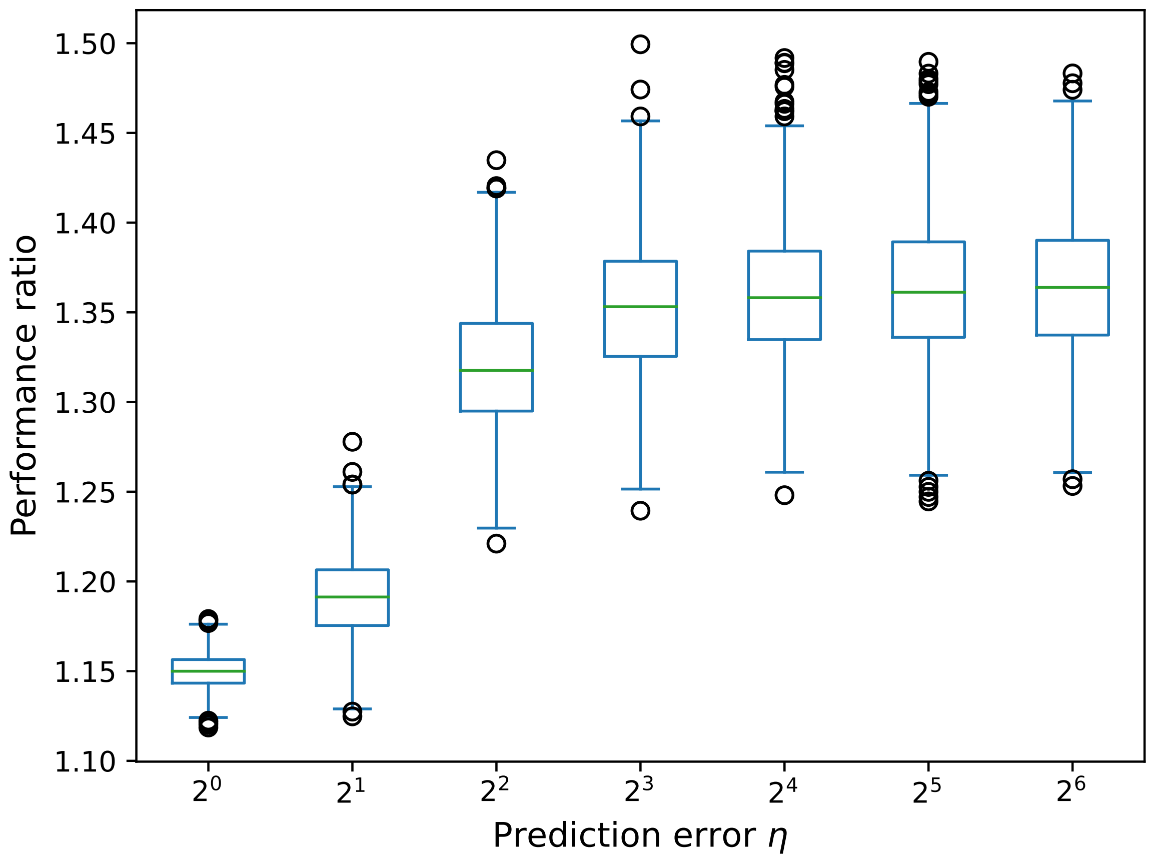


Figure V.7: Performance ratio of PEDRMLF with varying η. The performance ratio represents the mean response time of PEDRMLF over that of SRPT. The horizontal bars across the middle of the boxes represent the median of each boxplot. The box covers the middle 50% of the data points, while the lower and upper boundary of the box represents the 25th (75th) percentile.

manage in data structures and cost a higher time in decision-making. Therefore, PEDRMLF performs better when predictions are accurate. Again, the execution time of PEDRMLF stays close to SRPT even under large prediction errors. Surprisingly, PEDRMLF has a shorter execution time than SRPT when $1 \leq \eta \leq 2$ and $n \geq 400$, showing that processing the jobs from the lowest queues with the smallest index(es) within queues (PEDRMLF) is more time-saving than maintaining the remaining sizes for all jobs (SRPT). This result provides us with more options when the execution time matters.

Figure V.7 shows the performance ratio of PEDRMLF under increasing prediction error η. These data provide strong evidence supporting our theoretical results. The performance ratio is approximately 1.15 (close to 1) given perfect predictions, verifying our consistency results, and is bounded by approximately 1.35 as the predictions go arbitrarily bad, verifying our robustness results. The guaranteed robustness is crucial in dealing with the oracle providing predictions with unbounded errors. Meanwhile, the reward for an accurate oracle is an improved performance ratio and a reduced variance.

5.6 Conclusions and Future Work

This work gives the first real-time scheduling algorithm with job size predictions on single and parallel machines to minimize the mean response time. We first introduce a simple problem- and algorithm-independent prediction error metric η to quantify prediction quality. Then, we propose PEDRMLF achieving the optimal consistency of $O(\min\{\log \frac{n}{m}, \log P\})$ and the best-known robustness of $O(\min\{\log n \log \frac{n}{m}, \log n \log P\})$ with a highly scalable logarithmic run-time complexity. We prove these performance bounds and conduct extensive simulations to verify these results. The experimental evaluation shows that PEDRMLF stays close to SRPT when η is small while consistently outperforming RMLF even if η is unusually large.

We observe from the experimental results a clear positive correlation between the performance ratio of PEDRMLF and η when η is reasonably small. Analyzing how the performance changes with η, i.e., the smoothness of PEDRMLF, is our future work.

CHAPTER 6

Pulsed Power Load Scheduling with Predictions

Many mission- and time-critical cyber-physical systems deploy an isolated power system for their power supply. Under extreme conditions, the power system must process critical missions by maximizing the Pulsed Power Load (PPL) utility while maintaining the normal loads in the cyber-physical system. Optimal operation requires careful coordination of PPL deployment and power supply processes. In this chapter, we formulate the coordination problem for maximizing PPL utility under available resources, capacity, and demand constraints with normal load predictions. The coordination problem has two scenarios for different use cases: fixed and general normal loads. We develop an exact pseudo-polynomial time dynamic programming algorithm for each scenario with a proven guarantee to produce an optimal coordination schedule under the exact normal load prediction, i.e., the algorithm has optimal consistency. The performance of the algorithms is also experimentally evaluated, and the results agree with our theoretical analysis, showing the practicality of the solutions.

6.1 Introduction

There are increasing numbers of cyber-physical systems (CPS) deployed in harsh and hostile environments where decisions are often time- and mission-critical [49–52]. In this work, mission-critical means that operational failure on critical missions can seriously impact system performance and even cause catastrophes. Time-critical means that decision-making must be carried out in real time. A representative of such systems is an Isolated Power System (IPS), which refers to a power system operated in island mode to ensure carrier operations in deployments such as shipboard, aircraft, and space station. In particular, an IPS needs to handle the extremes of *Pulsed Power Load* (PPL) [53] that consume a substantial amount of energy within a very short period of time for a mission. Deploying PPL is the primary operation for an IPS to carry out and accomplish critical missions. Many IPSs operate PPL using a computer program. Thus, it is critical to model the IPS with PPL and study how to maximize the PPL utility via algorithms.

The system voltage and system frequency will be greatly affected if the PPL directly draws power from the IPS [54], causing the outage of other critical functions or system instability. To mitigate the negative impact, an Energy Storage System (ESS), such as a supercapacitor [55] and flywheel [56], is employed in the IPS as the energy source to support PPLs. The operator knows the operations of the PPL but cannot foresee the critical missions. We assume that maximizing the PPL utility in any circumstance is the best outcome the scheduler can guarantee to maximize the likelihood of succeeding in the critical mission completion. At any time, the ESS can be charged by a Power Source (PS) or disconnected from it to provide the necessary power for the PPL. As such, missions are accomplished within the energy constraints of the ESS while coordinating with charging processes to maximize the overall output utility. An energy distribution scheme between multiple PPLs needs to be determined under different conditions during mission executions. Thus, each PPL is required to maximize its output utility with the available energy from the PS within a given period. This work studies the coordination of PS charging and PPL deployment, called PPL scheduling, to allow the optimal coordination of a single PPL and PS in an IPS.

The PPL scheduling problem has attracted increasing attention recently for its extended use in many real-world applications [57–60]. Some works have focused on the integration of ESSs in IPSs [61–70]. They focus on studying the impact of PPL on the IPS and developing control strategies for the charging process of the ESS. The coordination of PPL deployment and charging by the PS has not yet been thoroughly investigated. Only a few attempts [71–73] have been made recently with heuristic algorithms with no provable performance guarantee. However, such coordination needs to have worst-case performance guarantees since it could become vital when an IPS operates under extreme conditions, e.g., severe weather. Also, the coordination schedule needs to maximize the PPL's output utility and be responsive when decision-making is adaptive and contingent upon the task and context. These requirements prevent the scheduling algorithms from being designed as heuristics [71, 73] without a performance guarantee to consuming extensive computation time in the worst-case scenarios [72]. Instead, we must look for solutions with performance guarantees in both mission and time. In our work, the worst-case performance guarantee means that an algorithm must produce the schedule with the maximum number of deployed pulses in the studied time period for any problem instances in real time. The existing solutions or machine learning-based approaches face the common problem of lacking theoretical performance guarantees. These algorithms are unacceptable to the target mission- and time-critical systems since a failure in one mission may cause a breakdown of the whole system. Our algorithms provide the theoretical guarantee of maximizing the PPL utility, outperforming the above approaches.

This work presents and proves the first optimal pseudo-polynomial algorithms for mission- and time-critical CPS scheduling with normal load predictions via dynamic programming. Our work observes a dominance rule in the set of (sub-)schedules, leading us to an optimal substructure property of the problem. Then, the problem is solved via dynamic programming. We prove the dominance rule and the correctness and optimality of our algorithms. More precisely, we develop schedulers to produce coordination solutions, ensuring (1) the energy of an ESS is constrained to lie within upper and lower bounds at all times, (2) the entangled parameters of charging processes are considered altogether towards producing an optimal coordination schedule, and (3) charging the ESS does not affect the execution of Normal Loads (NL) in the IPS. Such a schedule requires sequential coordination of PPL deployment and ESS charging, not just in terms of timing but also duration, intensity, and priority. The problem is solved under constant and variable NL functions, which have been separately studied using heuristic algorithms in the earlier works [73] and [71], respectively. This work presents the first provably optimal and computationally efficient method for the PPL scheduling problem. Our contributions are summarized below.

(1) We formulate the scheduling problem of single PPL deployment with ESS charging. To maximize the PPL's output utility under multiple constraints over time, we considered two endmember scenarios: the power consumption of normal loads is (1) constant and (2) variable.
(2) We propose optimal Dynamic Programming-based algorithms for the scheduling problems in the two above scenarios. We prove the correctness of the algorithms and present their time complexity results. Our solutions guarantee to produce, in both cases, the optimal sequence of PPL deployment and ESS charging that maximizes the PPL's utility. Our work is the first to give pseudo-polynomial algorithms for the optimal solution.
(3) We evaluate our approaches through experimental studies. The result shows that the proposed algorithms outperform the existing solutions in terms of both energy output and run time, meeting practical needs.

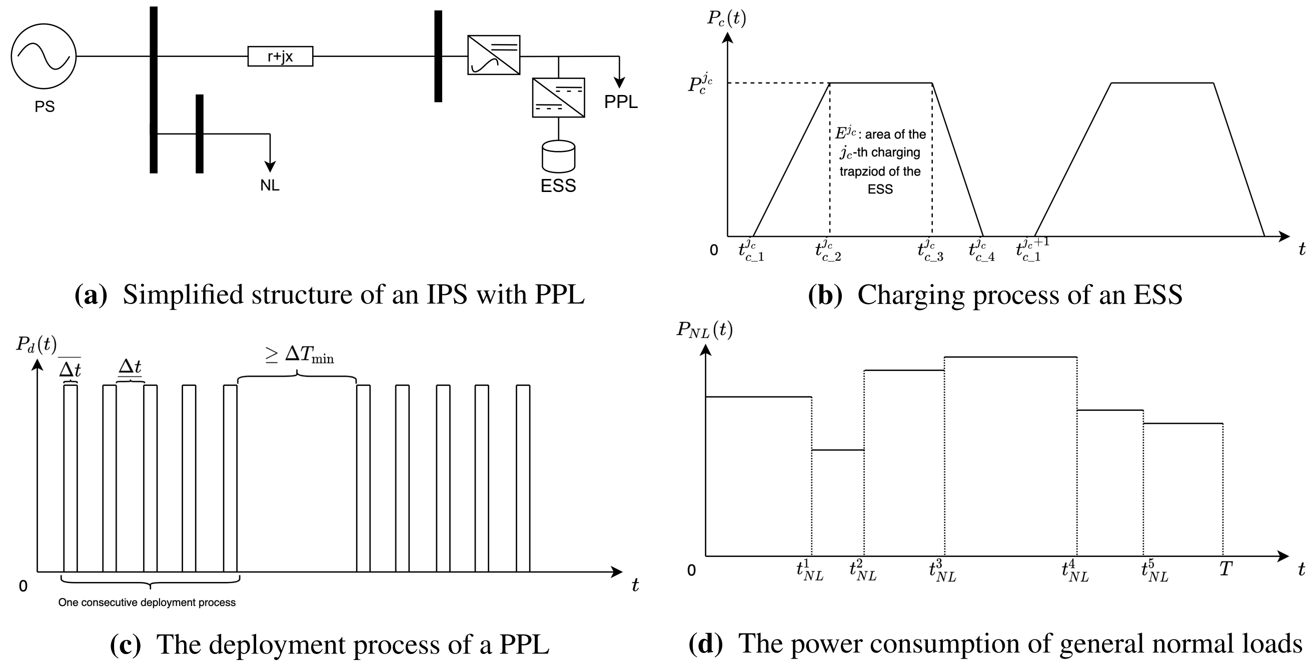
This chapter has the following structure. In Section 6.2, we model the coordination of PPL deployment and ESS charging. In Section 6.3, we formally define our optimization problem. Section 6.4 targets the coordination problem under constant NL functions. We present an optimal algorithm and give its analysis. Section 6.5 extends the algorithm for the constant NL functions to deal with the general NL functions. We present an optimal algorithm, for general NL functions, and give its analysis. Section 6.6 presents the simulation that evaluates the proposed algorithms. Section 6.7 discusses some related work. Conclusions and future work are given in Section 6.8.

6.2 Modeling of the Coordination Problem

This section introduces the overall structure of IPSs, the charging process of ESSs by PSs, and PPL deployment. We will use the terms ESS charging and PS charging interchangeably to refer to the same process that the PS is charging the ESS.

A simplified structure of an IPS supporting PPL [73] is depicted in Figure VI.1a. The loads in IPSs have two types, Normal Loads (NL) and PPL. For example, in an all-electric ship [74], the NL could include propulsion loads, lighting loads, and other workloads that consume power from the PS. An ESS is installed to mitigate the negative impact of PPL deployment in the regular operations of the IPS. The charging process of the ESS comprises trapezoidal-shaped pulses [75], as shown in Figure VI.1b. At the start and end of a charging process, the charging power changes linearly, subject to the ramp rates of the converter's output power. Once completed, the ESS will be disconnected from the IPS and provide energy for PPL deployment.

We argue the importance of modeling a trapezoid charging process instead of the rectangle one used in [72]. The Power Source (PS) cannot always reach a given power level instantly, making the practical charging process a trapezoid one. Rectangles could approximate trapezoids but are associated with a positive or negative error in energy representation. Such errors accumulated over time can cause the energy in an ESS to exceed its maximum limit or be below its minimum limit without being noticed by the scheduler when producing the schedule. Another concern is about the instantaneous power



(a) Simplified structure of an IPS with PPL **(b)** Charging process of an ESS

(c) The deployment process of a PPL **(d)** The power consumption of general normal loads

Figure VI.1: System modeling.

consumption limit. A rectangle cannot capture as much detail in charging as a trapezoid, which may cause the produced schedule to violate the instantaneous power consumption limit without being noticed again. When the above situations occur, the IPS may fail on critical missions. Thus, it is essential to model a trapezoid charging process to ensure precise energy calculation.

The PPL consumes a large amount of energy within a very short period. Thus the power consumption of a PPL can be modeled as a narrow rectangular-shaped current pulse, as shown in Figure VI.1c. Multiple pulses could occur in one consecutive PPL deployment process. Each pulse lasts for time $\overline{\Delta t}$, and its maximum power is $\bar{P}_d$. The time difference between two adjacent pulses inside one deployment process is $\underline{\Delta t}$. A pulse, therefore, consumes $\bar{P}_d \cdot \overline{\Delta t}$ amount of energy from the ESS. The number of pulses deployed in the j_d-th deployment process is denoted by $S_d^{j_d}$. The time interval between two consecutive PPL deployment processes is constrained by a minimum time interval $\Delta T_{\min}$. The PPL deployment becomes unavailable in this time interval so that the PS can charge the ESS.

In this model, the optimal operation of an IPS requires designing both the charging processes of the ESS and the PPL deployment processes. The parameters for a charging process are $t_{c\_1}^{j_c}$, $t_{c\_2}^{j_c}$, $t_{c\_3}^{j_c}$, $t_{c\_4}^{j_c}$, and the maximum charging power of the trapezoid $P_c^{j_c}$, for $1 \leq j_c \leq C$, where C is the total number of charging processes. The parameters for a PPL deployment process are the starting time of the process $t_d^{j_d}$, the number of pulses in the j_d-th process $S_d^{j_d}$, for $1 \leq j_d \leq D$, where D is the total number of deployment processes. As the PPL deployment is strongly correlated with the charging processes, these parameters must be carefully chosen, so the PPL and PS can be optimally coordinated within the given period.

6.3 Problem Formulation

This section formulates the coordination problem of PPL deployment and ESS charging. When the IPS is operating, the processes of PPL deployment and ESS charging take control in turn. The scheduler needs to decide the upcoming processes for IPS operations. We assume that all system parameters are known within a so-called studied time T window. The studied time T is a single user-defined parameter representing how long the system guarantees IPS stability. We present the objective function, problem constraints, and the overall optimization problem. A table of modeling parameters is provided in Table VI.1.

This section is organized as follows. Subsection 6.3.1 presents the objective function of our optimization problem. Then, every subsection (Subsections 6.3.2 — 6.3.7) introduces a type of constraint. In particular, Subsection 6.3.3 differentiates the case of constant NL functions (Subsection 6.3.3.1) and general NL functions (Subsection 6.3.3.2). The overall optimization problem is formulated in Subsection 6.3.8.

Symbol	Meaning
C	number of ESS charging processes
D	number of PPL deployment processes
T	studied time length
$t^{j_c}_{c\_1}$, $t^{j_c}_{c\_2}$, $t^{j_c}_{c\_3}$, $t^{j_c}_{c\_4}$	critical time instants of the j_c-th trapezoidal-shaped charging process
$P^{j_c}_c$	maximum charging power of the j_c-th trapezoid charging process
E^{j_c}	area of the j_c-th charging trapezoid of the ESS
$\overline{\Delta t}$	time that a PPL lasts
$\underline{\Delta t}$	time difference between two adjacent pulses inside one deployment process
$\Delta T_{\min}$	minimum time distance between two consecutive PPL deployment processes
$\bar{P}_d$	maximum power of a PPL deployment process
$t^{j_d}_d$	starting time of the j_d-th deployment process
$S^{j_d}_d$	number of pulses deployed in the j_d-th deployment process
P_{NL}	power consumption of NL (if the NL function is constant)
$P_{NL}(t)$	power consumption of NL over time
M_{NL}	number of the non-differentiable points in the NL function
t^1_{NL}, ..., $t^{M_{NL}}_{NL}$	time instants when the non-differentiable points occur in the NL function
E_0	initial energy stored in the ESS
$E_{\min}$, $E_{\max}$	minimum and maximum energy storage limits in the ESS
$P_c(t)$	power used by charging processes over time
$P_{G\max}$	upper limit of the PS's output power
$\Omega^{j_c}_c$, $\Omega^{j_d}_d$	the j_c-th charging time interval; the j_d-th deployment time interval
R^u, R^d	ramp-up and ramp-down rate limits in the ESS
P_{cmax}	maximum charging power limit of the PS

Table VI.1: A table of modeling parameters.

6.3.1 Objective Function

The utility of the PPL in a given period is the total energy output, which is proportional to the number of pulses deployed. The objective function of the coordination problem is defined as

$$\max \bar{P}_d \cdot \overline{\Delta t} \cdot \sum_{j_d=1}^{D} S_d^{j_d} \tag{6.1}$$

With constant $\bar{P}_d$ and $\overline{\Delta t}$, the actual term to optimize is $\sum_{j_d=1}^{D} S_d^{j_d}$. For simplicity, we will call term $\sum_{j_d=1}^{D} S_d^{j_d}$ the objective value.

6.3.2 Lower and Upper Energy Bounds of an ESS

Considering the capacity of an ESS, the stored energy should not exceed its maximum limit $E_{\max}$. The energy stored in the ESS should not exceed $E_{\max}$ by the end of any charging process, which is

$$E_0 + \sum_{j_c=1}^{a} E^{j_c} - \bar{P}_d \cdot \overline{\Delta t} \cdot \sum_{j_d=1}^{b} S_d^{j_d} \leq E_{\max}, \forall a \in [1, C] \tag{6.2}$$

where E_0 is the initial energy stored in the ESS, C is the total number of ESS charging processes, and b is the number of deployment processes before the a-th charging process. According to the charging process depicted in Figure VI.1b, we define E^{j_c} as below.

$$E^{j_c} = P_c^{j_c} \cdot \left(\frac{t_{c\_2}^{j_c} - t_{c\_1}^{j_c}}{2} + t_{c\_3}^{j_c} - t_{c\_2}^{j_c} + \frac{t_{c\_4}^{j_c} - t_{c\_3}^{j_c}}{2}\right) = \frac{1}{2} \cdot P_c^{j_c} \cdot (t_{c\_4}^{j_c} + t_{c\_3}^{j_c} - t_{c\_2}^{j_c} - t_{c\_1}^{j_c})$$

where $P_c^{j_c}$ is the PS's maximum charging power of the j_c-th charging process, $t_{c\_1}^{j_c}$, $t_{c\_2}^{j_c}$, $t_{c\_3}^{j_c}$, and $t_{c\_4}^{j_c}$ are the key time points of the charging process as shown in Figure VI.1b.

To further deal with some extreme conditions, there is a minimum stored energy $E_{\min}$ required. This constraint is equivalent to the idea that the energy stored in ESS should not go below $E_{\min}$ at the end of any deployment process, which is

$$E_0 + \sum_{j_c=1}^{a} E^{j_c} - \bar{P}_d \cdot \overline{\Delta t} \cdot \sum_{j_d=1}^{b} S_d^{j_d} \geq E_{\min}, \forall b \in [1, D] \tag{6.3}$$

where a is the number of charging processes before the b-th deployment process.

Our model assumes that the lower and upper energy limits ($E_{\min}$ and $E_{\max}$) cover the ESS State-of-Charge (SoC) requirement for simplicity. More precisely, the energy of the ESS cannot go below $E_{\min}$ or exceed $E_{\max}$ at any time. Meanwhile, the ESS is chargeable when the stored energy is below $E_{\max}$ and is dischargeable if the energy is above $E_{\min}$. With this assumption, the satisfaction of SoC is equivalent to the energy in ESS at any time within $[E_{\min}, E_{\max}]$ expressed by Constraints 6.2 and 6.3.

6.3.3 Available Charging Power Constraint

The IPS must maintain a power balance between ESS charging and the NL. The instantaneous power consumption of the NL and ESS charging cannot exceed the maximum availability of the PS, which is

$$P_{NL}(t) + P_c(t) \leq P_{G\max} \tag{6.4}$$

where $P_{NL}(t)$ is the power consumed by the NL, $P_c(t)$ is the power for the charging processes, and $P_{G\max}$ is the upper limit of the PS's output power. The stability of the DC (direct current) bus is critical in guaranteeing reliable operations of the IPS. Our work assumes that Constraint 6.4 ensures the DC bus's stability and the IPS's reliable operation. When coordinating the PPL deployment and the PS charging in real time, a prediction about the NL's required power in the studied period, $P_{NL}(t)$, is accessible by the algorithm. In this work, we assume the prediction is perfect and study the optimal algorithms under this setting. The results can construct consistent algorithms under general predictions. We consider two scenarios for $P_{NL}(t)$:

6.3.3.1 Constant NL Function

The power consumption of the NL remains constant in the studied period. In this case, we let $P_{NL}(t) = P_{NL}$.

6.3.3.2 General NL Function

The power consumption varies in the studied time period. $P_{NL}(t)$ is defined as a piecewise constant function as shown in Figure VI.1d. In this case, we assume M_{NL} non-differentiable points are included in $P_{NL}(t)$, denoted by t_{NL}^1, ..., $t_{NL}^{M_{NL}}$.

We treat these two cases as separate problems since each addresses a different need. The constant NL scenario has been discussed in [73], and the general NL scenario in [71]. In non-critical cases, an IPS can see the NL as a constant for routine operations. Otherwise, the NL could be electively sacrificed to achieve the best possible outcome to complete the mission when the IPS must maximize PPL utility. It is worth noting that our solution to the case of constant NL functions extends well into the case of general NL functions. More precisely, any NL functions can be approximated by piecewise constant functions by discretizing them into fine-granularity time slots.

6.3.4 Asynchronism of PS Charging and PPL Deployment

The ESS must be disconnected from IPS to support PPL deployment. Therefore, the charging period does not overlap the deployment period. That is, $\forall j_c = 1, ..., C$, and $\forall j_d = 1, ..., D$, the following holds.

$$\Omega_c^{j_c} \cap \Omega_d^{j_d} = \emptyset$$

where $\Omega_c^{j_c} = [t_{c\_1}^{j_c}, t_{c\_4}^{j_c})$ is the j_c-th charging duration, and $\Omega_d^{j_d} = [t_d^{j_d}, t_d^{j_d} + S_d^{j_d} \cdot (\overline{\Delta t} + \underline{\Delta t}) - \underline{\Delta t})$ is the j_d-th deployment duration. We thus only need to ensure: (i) the start or the end time of each charging process must not overlap with any deployment process, and (ii) vice versa. Condition (i) can be formulated as: $\forall j_d = 1, ..., D$, $j_c = 1, ..., C$,

$$\begin{cases} t_{c\_1}^{j_c} \geq t_d^{j_d} + S_d^{j_d} \cdot (\overline{\Delta t} + \underline{\Delta t}) - \underline{\Delta t} \text{ or } t_{c\_1}^{j_c} < t_d^{j_d} \\ t_{c\_4}^{j_c} > t_d^{j_d} + S_d^{j_d} \cdot (\overline{\Delta t} + \underline{\Delta t}) - \underline{\Delta t} \text{ or } t_{c\_4}^{j_c} \leq t_d^{j_d} \end{cases} \tag{6.5}$$

Condition (ii) can be formulated as: $\forall j_c = 1, ..., C$, $j_d = 1, ..., D$,

$$\begin{cases} t_d^{j_d} < t_{c\_1}^{j_c} \text{ or } t_d^{j_d} \geq t_{c\_4}^{j_c} \\ t_d^{j_d} + S_d^{j_d} \cdot (\overline{\Delta t} + \underline{\Delta t}) - \underline{\Delta t} > t_{c\_4}^{j_c} \\ \quad \text{or } t_d^{j_d} + S_d^{j_d} \cdot (\overline{\Delta t} + \underline{\Delta t}) - \underline{\Delta t} \leq t_{c\_1}^{j_c} \end{cases} \tag{6.6}$$

6.3.5 Ramp Rate Constraints and Converter Power Limit

The employment of ESS must limit the power ramp rates and apply power smoothing. Let R^u be the ramp-up rate limit of the ESS, and R^d be the ramp-down rate limit. Then, the following constraints must hold during ESS charging.

$$\begin{cases} P_c^{j_c} \leq R^u \cdot (t_{c\_2}^{j_c} - t_{c\_1}^{j_c}) \\ P_c^{j_c} \leq R^d \cdot (t_{c\_4}^{j_c} - t_{c\_3}^{j_c}) \end{cases} \tag{6.7}$$

The PS's maximum charging power of the j_c-th charging process, $P_c^{j_c}$, is also limited by the parameters of the interfacing converter, which is

$$P_c^{j_c} \leq P_{cmax} \tag{6.8}$$

where P_{cmax} is the power charging limit of the PS.

6.3.6 Time Sequence Constraints

The ESS charging time and the PPL deployment time must satisfy the following time sequence constraints. First, the last PPL deployment and the last ESS charging should not exceed the studied time T, that is

$$\begin{cases} t_d^D + S_d^D \cdot (\overline{\Delta t} + \underline{\Delta t}) - \underline{\Delta t} \leq T \\ t_{c\_4}^C \leq T \end{cases} \tag{6.9}$$

Second, the time instants within each charging process must satisfy the following conditions.

$$\begin{cases} t_{c\_1}^{j_c} < t_{c\_2}^{j_c} \leq t_{c\_3}^{j_c} < t_{c\_4}^{j_c} \\ t_{c\_4}^{j_c} \leq t_{c\_1}^{j_c+1} \end{cases} \tag{6.10}$$

Third, the time interval between two consecutive PPL deployment processes should satisfy the following conditions.

$$t_d^{j_d+1} - [t_d^{j_d} + S_d^{j_d} \cdot (\overline{\Delta t} + \underline{\Delta t}) - \underline{\Delta t}] \geq \Delta T_{\min} \tag{6.11}$$

where $\Delta T_{\min}$ is the minimum time interval of two consecutive PPL deployments. The following condition always holds between $\Delta T_{\min}$ and $\underline{\Delta t}$: $\Delta T_{\min} \geq \underline{\Delta t}$.

6.3.7 The Range of Decision Variables

Other constraints of the decision variables are given below.

$$\begin{cases} t_{c\_1}^{j_c}, t_{c\_2}^{j_c}, t_{c\_3}^{j_c}, t_{c\_4}^{j_c}, P_c^{j_c}, t_d^{j_d} \geq 0 \\ t_{c\_1}^{j_c}, t_{c\_4}^{j_c}, t_d^{j_d} \in \mathbb{N} \\ C, D, S_d^{j_d} \in \mathbb{N} \end{cases} \tag{6.12}$$

We also assume the integral representations: $\overline{\Delta t}, \underline{\Delta t}, \Delta T_{\min}, T \in \mathbb{N}$.

6.3.8 Overall Optimization Problem

The proposed coordination problem of the PPL and the PS can be formulated as the following optimization problem.

$$\begin{aligned} \text{Objective:} \quad & (6.1) \\ \text{Constraints:} \quad & (6.2) - (6.12) \end{aligned} \tag{6.13}$$

The optimization variables are C, D, $t_{c\_1}^{j_c}, t_{c\_2}^{j_c}, t_{c\_3}^{j_c}, t_{c\_4}^{j_c}, P_c^{j_c}$ $(1 \leq j_c \leq C)$, $t_d^{j_d}$, and $S_d^{j_d}$ $(1 \leq j_d \leq D)$. The main differences between our optimization problem and the counterpart in [71] and [73] are the number of PPLs considered and the assumption of periodic PPL deployment in the studied period (both [71] and [73] assume the coordination is periodic while we do not). Our optimization problem is difficult to solve due to the variety of constraints and non-linearity. First, the number of constraints (Constraint (6.2), (6.3), (6.5), (6.6), (6.7), (6.8), (6.10), and (6.11)) and the number of the decision variables associated with the symbol j_c or j_d increases with the values of the decision variables C and D. Second, Constraints (6.2), (6.3), and (6.4) are non-linear, and Constraint (6.4) has infinite dimensions for satisfying all real numbers $t \in [0, T)$. We solve this problem in two scenarios: one is for constant NL functions, and the other is for general NL functions. Both solutions use *Dynamic Programming* (DP), a widely used method in CPS scheduling [44, 76–78].

The scenario of general NL functions covers constant NL functions. We consider the problem in two separate scenarios since each has a different focus. The challenge in the general NL scenario is to design an algorithm that guarantees optimal PPL utility when NL is a piecewise constant function. Otherwise, the challenge is more on developing a computationally efficient algorithm by utilizing the assumption of a constant NL.

6.4 Optimal Solution: Case of Constant NL Functions

This section targets the coordination problem under constant NL functions. In Subsection 6.4.1, we present the high-level idea and the intuition in the algorithm design for both constant and NL functions. Subsection 6.4.2 presents the optimal strategy CtLwLE for ESS charging. Subsection 6.4.3 presents the DP algorithm solving the coordination problem under constant NL functions. It also analyzes the optimality of the algorithm.

6.4.1 High-level Idea of the Algorithms

Our objective is to produce the maximal number of pulses under all the system constraints. The energy for pulse deployment mainly comes from charging processes, which requires ESS charging to be efficient. The core idea of producing an optimal schedule is to find maximal trapezoids under the NL curve. For constant NL, our strategy is to let the charging process have a maximal ramp rate to reach the maximum charging power to complete the ESS charging. Section 6.4.2 provides the details. For general NL, the strategy of finding an optimal solution is much less obvious as it requires searching in an infinite solution space. Our approach is first to identify the structural properties of an optimal charging process that can reduce the search space down to finite. Then, we develop a DP for determining the optimal charging strategies while coping with the changing NL in Section 6.5.1. With the optimal charging strategies, we use another DP to search for the optimal schedule that determines when the system should perform charging or deploy PPL and the amount of energy to charge or deploy. We state and prove the important dominance rule (Theorem VI.5) that will allow us to search only the dominating sub-schedules efficiently. The idea is as follows. For the sub-schedules that deploy S pulses in a given period $[0, t)$, the one with the maximal remaining energy dominates the others. We prove that these dominating sub-schedules possess an optimal substructure property, and thus can be searched using DP. Combining all these, we finally develop two DP-based algorithms (Algorithm 10 and 11) that solve the problem under different NLs. Both algorithms can be viewed as computing the partial schedules sequentially following the topological order of the dependency graph of the DP states. Note that the latter algorithm is a generalization of the former.

6.4.2 Optimal Charging Strategy

Suppose at some time $t_{c\_1}^{j_c}$, the system starts a charging process with duration Δt^{j_c}, so that the ESS charging lasts for period $[t_{c\_1}^{j_c}, t_{c\_1}^{j_c} + \Delta t^{j_c})$. The optimal charging process is the one that charges as much energy as possible to the ESS under Constraints (6.2), (6.4), (6.7), and (6.8). Any charging trapezoid should have the maximum ramp rates at both the beginning and end of the process. It should attempt to make the maximum power reach the limit P_{climit}, defined as $P_{climit} = \min\{P_{cmax}, P_{G\max} - P_{NL}\}$, an upper bound of the maximum charging power. For now, we ignore Constraint (6.2) and formulate the above discussion as *Charging to the Limit* (CtL) strategy as follows.

Definition VI.1 (CtL). *Fix, for the j_c-th charging process, the charging time duration Δt^{j_c} and the start charging time instant $t^{j_c}_{c\_1}$. CtL strategy sets the parameters for the j_c-th charging process $(t^{j_c}_{c\_1}, t^{j_c}_{c\_2}, t^{j_c}_{c\_3}, t^{j_c}_{c\_4}, P^{j_c}_{c})$ as follows.*

If $\Delta t^{j_c} \geq P_{climit} \cdot (\frac{R^u+R^d}{R^u R^d})$, set $P^{j_c}_c = P_{climit}$ and

$$\begin{cases} t^{j_c}_{c\_4} = t^{j_c}_{c\_1} + \Delta t^{j_c} \\ t^{j_c}_{c\_2} = t^{j_c}_{c\_1} + \frac{P^{j_c}_c}{R^u} \\ t^{j_c}_{c\_3} = t^{j_c}_{c\_4} - \frac{P^{j_c}_c}{R^d} \end{cases}$$

We have $t^{j_c}_{c\_1} < t^{j_c}_{c\_2} \leq t^{j_c}_{c\_3} < t^{j_c}_{c\_4}$, and $P^{j_c}_c = P_{climit} = \min\{P_{climit}, \frac{\Delta t^{j_c} R^u R^d}{R^u+R^d}\}$.

If $\Delta t^{j_c} < P_{climit} \cdot (\frac{R^u+R^d}{R^u R^d})$, set $P^{j_c}_c = \frac{\Delta t^{j_c} R^u R^d}{R^u+R^d}$ and

$$\begin{cases} t^{j_c}_{c\_4} = t^{j_c}_{c\_1} + \Delta t^{j_c} \\ t^{j_c}_{c\_2} = t^{j_c}_{c\_3} = t^{j_c}_{c\_1} + \frac{P^{j_c}_c}{R^u} \end{cases}$$

We have $t^{j_c}_{c\_1} < t^{j_c}_{c\_2} = t^{j_c}_{c\_3} < t^{j_c}_{c\_4}$ and $P^{j_c}_c = \frac{\Delta t^{j_c} R^u R^d}{R^u+R^d} = \min\{P_{climit}, \frac{\Delta t^{j_c} R^u R^d}{R^u+R^d}\}$.

Note that when the charging duration is too short ($\Delta t^{j_c} < P_{climit} \cdot (\frac{R^u+R^d}{R^u R^d})$), the charging trapezoid degenerates to a triangle. With CtL, the amount of energy charged depends only on the time duration, Δt^{j_c}, so we define $E_{CtL}(\Delta t^{j_c})$ in Definition VI.2 to be the energy charged in one CtL charging process with the time duration Δt^{j_c}. Observe that $E_{CtL}(\Delta t^{j_c})$ is a continuous monotonically increasing function and thus has its inverse function on $[0, \infty)$ (Definition VI.3).

Definition VI.2. *We define $E_{CtL}(\Delta t^{j_c}) = \frac{(\Delta t^{j_c})^2 R^u R^d}{2(R^u+R^d)}$ for $\Delta t^{j_c} < P_{climit} \cdot (\frac{R^u+R^d}{R^u R^d})$ and $E_{CtL}(\Delta t^{j_c}) = P_{climit}\Delta t^{j_c} - \frac{1}{2}P^2_{climit} \cdot (\frac{R^u+R^d}{R^u R^d})$ for $\Delta t^{j_c} \geq P_{climit} \cdot (\frac{R^u+R^d}{R^u R^d})$.*

Definition VI.3. *We define the inverse function of $E_{CtL}(\Delta t^{j_c})$ as $E^{-1}_{CtL}(E)$ on $[0, \infty)$, where $E^{-1}_{CtL}(E) = \sqrt{\frac{2(R^u+R^d)E}{R^u R^d}}$ for $E < \frac{P^2_{climit}(R^u+R^d)}{2R^u R^d}$ and $E^{-1}_{CtL}(E) = \frac{E}{P_{climit}} + \frac{1}{2}P_{climit} \cdot (\frac{R^u+R^d}{R^u R^d})$ for $E \geq \frac{P^2_{climit}(R^u+R^d)}{2R^u R^d}$.*

Now we take Constraint (6.2) back into consideration. With the integer variables $t^{j_c}_{c_1}$s and $t^{j_c}_{c_4}$s, we may face the situation that CtL strategy cannot produce a feasible charging process. To address this issue, we introduce an enhanced CtL strategy, *Charging to the Limit with Limited Energy* strategy, or CtLwLE strategy. Before giving the details of CtLwLE, we first discuss how CtL strategy does not function as intended. Suppose the scheduler decides to charge the ESS until time $t^{j_c}_{c\_4}$ when ESS has remaining energy $E^{j_c}_R$ at time $t^{j_c}_{c\_1}$. It is feasible and optimal to apply CtL strategy for this charging process when $E_{CtL}(t^{j_c}_{c\_4} - t^{j_c}_{c\_1}) \leq E_{\max} - E^{j_c}_R$, but it is no longer true if CtL strategy would make the ESS charging exceeds the upper bound $E_{\max}$. In this case, the optimal solution is to charge the ESS to the upper bound, which is formalized in Definition VI.4.

Definition VI.4 (CtLwLE). *Fix, for the j_c-th charging process, the charging time duration $\Delta t^{j_c} \in \mathbb{N}$, the start charging time $t^{j_c}_{c\_1} \in \mathbb{N}$, and an additional parameter ΔE_m indicating the maximum energy that* *can be charged during the process. CtLwLE strategy sets the parameters of the j_c-th charging process $(t^{j_c}_{c\_1}, t^{j_c}_{c\_2}, t^{j_c}_{c\_3}, t^{j_c}_{c\_4}, P^{j_c}_c)$ as follows.*

If $E_{CtL}(\Delta t^{j_c}) \leq \Delta E_m$, variables $t^{j_c}_{c\_2}, t^{j_c}_{c\_3}, t^{j_c}_{c\_4}$, and $P^{j_c}_c$ are set according to the normal CtL strategy (Definition VI.1).

If $E_{CtL}(\Delta t^{j_c}) > \Delta E_m$, set $P^{j_c}_c = (\Delta t^{j_c} - \sqrt{(\Delta t^{j_c})^2 - 2 \cdot (\frac{R^u+R^d}{R^u R^d}) \cdot \Delta E_m}) \cdot (\frac{R^u R^d}{R^u+R^d})$ and

$$\begin{cases} t^{j_c}_{c\_4} = t^{j_c}_{c\_1} + \Delta t^{j_c} \\ t^{j_c}_{c\_2} = t^{j_c}_{c\_1} + \frac{P^{j_c}_c}{R^u} \\ t^{j_c}_{c\_3} = t^{j_c}_{c\_4} - \frac{P^{j_c}_c}{R^d} \end{cases}$$

With CtLwLE strategy, we can develop a dynamic programming algorithm to solve Problem (6.13) under constant NL functions.

6.4.3 The DP-based Solution

This subsection proposes a DP-based solution for finding the optimal schedule for PPL deployment and ESS charging. The core of the algorithm observes the following dominance rule. Define *partial schedule* to be a sub-schedule over time $[0, t)$ $(0 \leq t \leq T)$. We say that a partial schedule A *dominates* another partial schedule B, if for every schedule B^W over time $[0, T)$ containing B as its sub-schedule, there exists another schedule A^W over time $[0, T)$ containing A as its sub-schedule achieving an objective function value no less than the schedule B^W does.

Theorem VI.5 (Dominance Rule for DP). *Among the partial schedules that deploy S pulses during $[0, t)$ and have an additional pulse deploy at $[t, t + \overline{\Delta t})$, the one(s) with the maximal remaining energy at t dominates the others. This also holds for the partial schedules that deploy S pulses during $[0, t)$ and have a charging process ending at t.*

Proof. Let A_o be any partial schedule with the maximal remaining energy at time t, which deploys S pulses during the time interval $[0, t)$ and has an additional pulse to be deployed at period $[t, t + \overline{\Delta t})$. Consider any schedule A over the time interval $[0, T)$, which deploys S pulses during the time interval $[0, t)$ and has an additional pulse to be deployed at period $[t, t + \overline{\Delta t})$. We can replace A's partial schedule over $[0, t)$ with A_o, and for any j_c-th charging process of A, $t^{j_c}_{c\_1} \geq t$, in increasing j_c's order, we replace it with CtLwLE strategy with time duration $t^{j_c}_{c\_4} - t^{j_c}_{c\_1}$ and the difference between $E_{\max}$ and the remaining energy at time $t^{j_c}_{c\_1}$ as the maximum chargeable energy bound. This process produces another schedule, A', over $[0, T)$. Schedule A' has the same objective function value as schedule A does. It is valid since (1) the remaining energy at time t in schedule A' is no less than that in schedule A, and (2) for each j_c-th charging process, $t^{j_c}_{c\_1} \geq t$, the remaining energy at time $t^{j_c}_{c\_4}$ in schedule A' is no less than that in schedule A. Claim (1) is due to the definition of A_o, and Claim (2) can be proved by induction on j_c for all $t^{j_c}_{c\_1} \geq t$. The partial schedule A_o, therefore, dominates the others. With similar arguments, we can prove that among the partial schedules that deploy S pulses during period $[0, t)$ and have a charging process ending at time t, the one(s) with the maximal remaining energy at time t dominates the others. □

Theorem VI.5 states that only the dominating partial schedules need to be considered. To represent them compactly, we record, for each partial schedule, the time length t, the number of deployed pulses in $[0, t)$ (named S), and the remaining energy at time t. With the dominance rule and relations between these dominating partial schedules, we formulate our DP in Definition VI.6.

Definition VI.6 (Optimal DP Formulation). *Let $F_1(t, S)$ denote the maximum remaining energy at time t, given that the schedule deploys S pulses during the time interval $[0, t)$ and has an additional deployed pulse at time t. Similarly, let $F_2(t, S)$ denote the maximum remaining energy at time t, given that the schedule deploys S pulses during the time interval $[0, t)$ and has a charging process ending at time t. We have the following DP recurrences.*

$$F_1(t, S) = \max\{F_1(t - \overline{\Delta t} - \underline{\Delta t}, S - 1) - \bar{P}_d \cdot \overline{\Delta t}, F_2(t, S)\} \tag{6.14}$$

for all $0 \le t \le \max\{0, T - \overline{\Delta t}\}, 1 \le S \le \lfloor \frac{T+\Delta t}{\overline{\Delta t}+\underline{\Delta t}} \rfloor$.

$$F_2(t, S) = \max\{\min\{E_{\max}, F_1(t - \Delta t_c - \overline{\Delta t}, S - 1) - \bar{P}_d \cdot \overline{\Delta t} + E_{CtL}(\Delta t)\}\} \tag{6.15}$$

for all $0 \le t \le T$*,* $1 \le S \le \lfloor \frac{T+\Delta t}{\overline{\Delta t}+\underline{\Delta t}} \rfloor$*,* $1 \le \Delta t \le \lceil E_{CtL}^{-1}(E_{\max} - E_{\min})\rceil$*, and* $\Delta t_c = \max\{\Delta t, \Delta T_{\min}\}$ *if* $t < T$ *or* Δt *if* $t = T$*. Additionally, whenever term* $F_1(t-\overline{\Delta t}-\underline{\Delta t}, S-1)$ *or term* $F_1(t-\Delta t_c-\overline{\Delta t}, S-1)$ *or term* $F_2(t, S)$ *has invalid arguments or values, i.e., the value is less than* $E_{\min} + \bar{P}_d \cdot \overline{\Delta t}$ *for the first two terms or less than* $E_{\min}$ *for the term* $F_2(t, S)$*, the value is set to* $-\infty$*. We have the following base cases.*

$$F_2(t, 0) = \min\{E_0 + E_{CtL}(t), E_{\max}\}, \forall\, 0 \le t \le T \tag{6.16}$$

$$F_1(t, 0) = \begin{cases} F_2(t, 0), & F_2(t, 0) \ge E_{\min} + \bar{P}_d \cdot \overline{\Delta t} \\ -\infty, & F_2(t, 0) < E_{\min} + \bar{P}_d \cdot \overline{\Delta t} \end{cases}, \forall\, 0 \le t \le \max\{0, T - \overline{\Delta t}\} \tag{6.17}$$

The maximum number of pulses deployed by an optimal schedule is

$$\max\{\, \max\{1 + S' | F_1(T - \overline{\Delta t}, S') - \bar{P}_d \cdot \overline{\Delta t} \ge E_{\min}\}, \max\{S' | F_2(T, S') \ge E_{\min}\}, 0\} \tag{6.18}$$

Theorem VI.7 (Correctness and Optimality of the DP). *The DP in Definition VI.6 produces an optimal solution.*

Proof. The theorem is proved by showing the following claims.

(1) The state space(s) for $F_1(t, S)$ and $F_2(t, S)$ include all dominating partial schedules.
(2) The solution to the DP equals the optimal value of the objective function.
(3) The base cases are correctly set.
(4) The recurrence relation(s) are correct.

Let $\sigma_1(t, S)$, $\sigma_2(t, S)$ denote the partial schedules corresponding to DP states $F_1(t, S)$ and $F_2(t, S)$.

Algorithm 10: DP (Constant NL case)

```
Data   : problem instance with a constant NL function prediction
Result : an optimal schedule
1  initialize spaces for $F_1(t,S)$s, $F_2(t,S)$s.
2  for $t \leftarrow 0$ to $T$ do
3      $F_2(t,0) \leftarrow \min\{E_0 + E_{CtL}(t), E_{\max}\}$.
4      if $F_2(t,0) \geq E_{\min} + \bar{P}_d \cdot \overline{\Delta t}$ then
5          $F_1(t,0) \leftarrow F_2(t,0)$.
6      else
7          $F_1(t,0) \leftarrow -\infty$.
8  for $t \leftarrow 0$ to $T$ do
9      for $S \leftarrow 1$ to $\lfloor \frac{T+\Delta t}{\overline{\Delta t}+\underline{\Delta t}} \rfloor$ do
10         $F_2(t,S) \leftarrow \max\{\min\{E_{\max}, F_1(t-\Delta t_c-\overline{\Delta t}, S-1) - \bar{P}_d \cdot \overline{\Delta t} + E_{CtL}(\Delta t)\} \mid$
           $1 \leq \Delta t \leq \lceil E_{CtL}^{-1}(E_{\max}-E_{\min}) \rceil,\quad \Delta t_c = \begin{cases} \max\{\Delta t, \Delta T_{\min}\}, & t < T \\ \Delta t, & t = T \end{cases}\}$.
11         if $t \leq \max\{0, T-\overline{\Delta t}\}$ then
12             $F_1(t,S) \leftarrow \max\{F_1(t-\overline{\Delta t}-\underline{\Delta t}, S-1) - \bar{P}_d \cdot \overline{\Delta t}, F_2(t,S)\}$.
13             if $F_1(t,S) < E_{\min} + \bar{P}_d \cdot \overline{\Delta t}$ then
14                 $F_1(t,S) \leftarrow -\infty$.
15 $S_{\text{MAX}} \leftarrow \max\{\max\{1+S' | F_1(T-\overline{\Delta t}, S') - \bar{P}_d \cdot \overline{\Delta t} \geq E_{\min}\}, \max\{S' | F_2(T,S') \geq E_{\min}\}, 0\}$.
16 return TraceSchedule($F_1, F_2, S_{MAX}$)
```

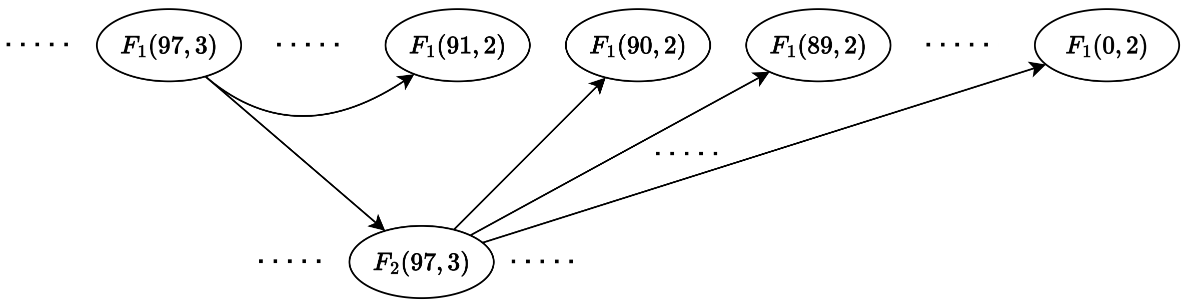
Proof for Claim 1. The second parameter, S, in either F_1 or F_2, takes values in range $[0, \lfloor \frac{T+\Delta t}{\overline{\Delta t}+\underline{\Delta t}} \rfloor]$, since the maximum amount of pulses that can be deployed in $[0,T)$ is $\lfloor \frac{T+\Delta t}{\overline{\Delta t}+\underline{\Delta t}} \rfloor$, which is achieved by one long deployment process spanning $[0,T)$. The first parameter of $F_1(t,S)$ takes values in range $[0, \max\{0, T-\overline{\Delta t}\}]$, since σ_1 requires an additional pulse starting at time t. Meanwhile, the first parameter of $F_2(t,S)$ takes values in the entire interval $[0,T]$.

Proof for Claim 2. Observe that there exists an optimal schedule with either a pulse ending at time T or a charging process ending at time T. Otherwise, the optimal schedule would perform no operation during time interval $[T-t', T)$ where t' denotes the maximal number satisfying this condition. In such a case, we can modify this schedule so that it ends exactly at time T by shifting the whole schedule t' unit to the right while retaining the optimality. If an optimal schedule ends with a pulse in time interval $[T-\overline{\Delta t}, T)$, the objective function value equals $\max\{1+S' | F_1(T-\overline{\Delta t}, S') - \bar{P}_d \cdot \overline{\Delta t} \geq E_{\min}\}$, in the corresponding dominating schedules. If, on the other hand, an optimal schedule has a charging process ending at time T, the objective function value equals $\max\{S' | F_2(T,S') \geq E_{\min}\}$ by similar arguments. Combining the two cases, the optimal objective value equals $\max\{\max\{1+S' | F_1(T-\overline{\Delta t}, S') - \bar{P}_d \cdot \overline{\Delta t} \geq E_{\min}\}, \max\{S' | F_2(T,S') \geq E_{\min}\}, 0\}$.

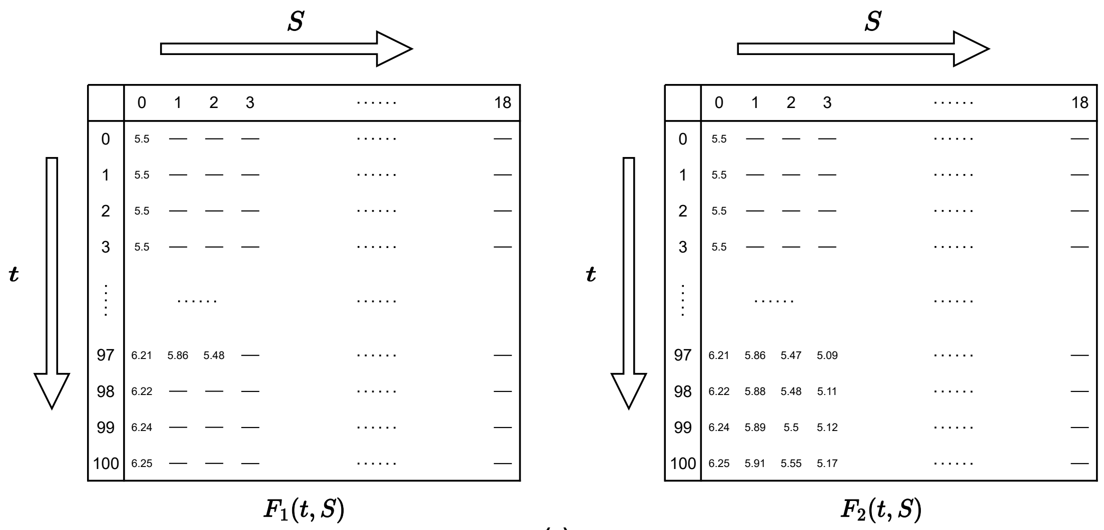
Proof for Claim 3. Consider $\sigma_2(t,0)$ which deploys 0 pulses in $[0,t)$. The maximum energy at time t is the amount of energy charged by CtLwLE strategy (with time duration t and maximum energy bound $E_{\max} - E_0$) plus the initial energy E_0, which gives Equation (6.16). Consider $\sigma_1(t,0)$. Since it also deploys 0 pulses in $[0,t)$, $F_1(t,0)$ equals $F_2(t,0)$. The only exception occurs if $\sigma_1(t,0)$ is invalid when

Parameter	$\bar{P}_d$	$\overline{\Delta t}$	$\underline{\Delta t}$	$E_{\min}$	$E_{\max}$	E_0	P_{NL}	$P_{G\max}$	R^u	R^d	P_{cmax}	$\Delta T_{\min}$	T
Value	0.1	3	3	5.0	25.0	5.5	0.15	0.25	0.0003	0.0003	0.04	4	100

(a)



(b)



	0	1	2	3	......	18
0	5.5	—	—	—	......	—
1	5.5	—	—	—	......	—
2	5.5	—	—	—	......	—
3	5.5	—	—	—	......	—
⋮	......				......	
97	6.21	5.86	5.48	—	......	—
98	6.22	—	—	—	......	—
99	6.24	—	—	—	......	—
100	6.25	—	—	—	......	—

$F_1(t,S)$

	0	1	2	3	......	18
0	5.5	—	—	—	......	—
1	5.5	—	—	—	......	—
2	5.5	—	—	—	......	—
3	5.5	—	—	—	......	—
⋮	......				......	
97	6.21	5.86	5.47	5.09	......	—
98	6.22	5.88	5.48	5.11	......	—
99	6.24	5.89	5.5	5.12	......	—
100	6.25	5.91	5.55	5.17	......	—

$F_2(t,S)$

(c)

Figure VI.2: A simple numerical example. Figure (a) gives the parameter values in the example. For simplicity, the units are omitted, and the values are normalized. Figure (b) visualizes (a segment of) the dependency graph of DP states. Computing $F_1(97,3)$ requires the values of $F_1(91,2)$ and $F_2(97,3)$; computing $F_2(97,3)$ requires $F_1(90,2)$, $F_1(89,2)$, ..., $F_1(0,2)$. The computation must therefore follow the topological order of the dependency graph. Figure (c) shows the order of computation. The algorithm computes F_1 and F_2 simultaneously row by row (increasing t) and column by column (increasing S) inside a row. The table represents the computed DP values, where a dash indicates an invalid DP state. The maximum number of pulses deployable in this instance is 3.

$F_2(t,0)$ does not charge enough energy to support one pulse, i.e., $F_2(t,0) < E_{\min} + \bar{P}_d \cdot \overline{\Delta t}$. If $\sigma_1(t,0)$ is invalid, $F_1(t,0)$ is set to $-\infty$. This gives Equation (6.17).

Proof for Claim 4. Partial schedule $\sigma_1(t,S)$ either has a deployment process crossing time t or has a charging process ending at time t. Otherwise, we can shift the partial (sub-)schedule before time t to the right to satisfy the condition while retaining the optimality. If $\sigma_1(t,S)$ has a deployment process crossing time t, it must deploy a pulse in $[t-\overline{\Delta t}-\underline{\Delta t}, t-\underline{\Delta t})$, so the maximum remaining energy at time t equals

$F_1(t - \overline{\Delta t} - \underline{\Delta t}, S-1) - \bar{P}_d \cdot \overline{\Delta t}$. Note that this case is not valid if term $F_1(t - \overline{\Delta t} - \underline{\Delta t}, S-1)$ has invalid arguments or has values less than $E_{\min} + \bar{P}_d \cdot \overline{\Delta t}$. If, on the other hand, $\sigma_1(t, S)$ has a charging process ending at time t, the maximum remaining energy at time t equals $F_2(t, S)$. This case is not valid if the value of $F_2(t, S)$ is less than $E_{\min} + \bar{P}_d \cdot \overline{\Delta t}$ when PPL is incapable of supporting an additional pulse at time t. This proves the recurrence $F_1(t, S) = \max\{F_1(t - \overline{\Delta t} - \underline{\Delta t}, S-1) - \bar{P}_d \cdot \overline{\Delta t}, F_2(t, S)\}$.

Partial schedule $\sigma_2(t, S)$ has a charging process ending at time t, and we let Δt denote the length of this last charging process. We assume that, without loss of generality, the last charging process uses CtLwLE strategy due to its optimality. We first show that we can force $1 \leq \Delta t \leq \lceil E_{CtL}^{-1}(E_{\max} - E_{\min}) \rceil$. If the charging process is longer than $\lceil E_{CtL}^{-1}(E_{\max} - E_{\min}) \rceil$, we can reduce the charging time to $\lceil E_{CtL}^{-1}(E_{\max} - E_{\min}) \rceil$ using CtLwLE strategy and charge the ESS to its upper bound $E_{\max}$. Suppose there is some idle time between processes in $\sigma_2(t, S)$ or idle time introduced by applying CtLwLE strategy to each charging process in the previous argument. In that case, we can remove these idle times by firstly shifting them all to the very beginning of the schedule to compact the rest of the schedule, and secondly combining these idle times at the front together with the first charging process, if there is one, into one charging process using CtLwLE strategy. Note that DP implements this operation by appropriately setting $F_2(t, 0)$. Next, since there is a deployment process before the last charging process, the last deployment process must end at time $t - \Delta t_c$, where $\Delta t_c = \max\{\Delta t, \Delta T_{\min}\}$ for $t < T$ or $\Delta t_c = \Delta t$ for $t = T$ (we need to enforce the minimum time distance constraint between consecutive deployment processes, but there is no such constraint for the last charging process ending at time T). Therefore, the remaining energy for $\sigma_2(t, S)$ at time t is $\min\{E_{\max}, F_1(t - \Delta t_c - \overline{\Delta t}, S-1) - \bar{P}_d \cdot \overline{\Delta t} + E_{CtL}(\Delta t)\}$, which is obtained by applying CtLwLE strategy for the last charging process. Since we do not know the value of Δt in advance, the recurrence (6.15) considers all possible values for Δt, which guarantees to find the correct value for term $F_2(t, S)$. Note that, for each Δt, we also need to check the lower energy bound constraint at time $t - \Delta t_c$, i.e., $F_1(t - \Delta t_c - \overline{\Delta t}, S-1) - \bar{P}_d \cdot \overline{\Delta t} \geq E_{\min}$, and if there is no such valid Δt, set $F_2(t, S)$ to $-\infty$. □

We provide the pseudo-code of the DP in Algorithm 10. Lines 1 to 15 compute the values for all DP states. Let $S_{\max}$ denote $\lfloor \frac{T + \underline{\Delta t}}{\overline{\Delta t} + \underline{\Delta t}} \rfloor$ and $\Delta t_{d\max}$ denote $\min\{T, \lceil E_{CtL}^{-1}(E_{\max} - E_{\min}) \rceil\}$. Figure VI.2 presents a sample run of Algorithm 10. The state space has size $O(T \cdot S_{\max})$ and state transition requires $O(\Delta t_{d\max})$ time. Overall, Lines 1 to 15 have complexity $O(T \cdot S_{\max} \cdot \Delta t_{d\max})$. Line 16 uses the computed DP states and the maximum objective function value to trace the corresponding optimal schedule in $O(S_{\max} \cdot \Delta t_{d\max})$ time. Tracing the optimal solution from the computed DP states is a standard procedure, so we omit it. The overall computational complexity of the DP in Definition VI.6 is $O(T \cdot S_{\max} \cdot \Delta t_{d\max})$, and the space complexity is $O(T \cdot S_{\max})$.

Theorem VI.8 (Optimality of Consistency for Algorithm 10)**.** *Algorithm 10 has optimal consistency.*

Proof. The lower bound for the competitive ratio of any clairvoyant algorithm is 1. By Theorem VI.7, Algorithm 10 has a competitive ratio 1 under perfect NL prediction. By the Fundamental Theorems of Consistency (Theorems II.4 and II.5), Algorithm 10 has consistency matching the lower bound. □

6.5 Optimal Solution: Case of General NL Functions

This section considers the case of general NL functions. Our solution is supported by the prediction of the variable NL beforehand. This condition is reasonable since mission- and time-critical CPSs often operate under pre-programmed NL, making the power consumption predictable during the run.

Extending the techniques in Subsection 6.4.2, Subsection 6.5.1 presents the optimal strategy GCtLwLE (Generalized Charging to the Limit with Limited Energy) for ESS charging under general NL functions. Extending the techniques in Subsection 6.4.3, Subsection 6.5.2 presents the (extended) DP algorithm solving the coordination problem under general NL functions.

6.5.1 Optimal Charging Strategy

With the variable NL, we cannot directly apply CtL strategy since the maximum chargeable energy in a period can no longer be determined by the function E_{CtL}. Fortunately, we can use the same idea of CtL strategy and measure the maximum chargeable energy for every possible time interval. This idea will extend the DP to the general NL case. We propose the dynamic program below (Definition VI.9) to compute the maximum chargeable energy for every time interval. Note that, when determining $C_1(i,j)$ or $C(i,j)$ in the DP, we again first ignore the upper energy bound $E_{\max}$ (Constraint (6.2)) but still enforce the others (Constraints (6.7) and (6.8)).

Definition VI.9 (DP for Maximum Chargeable Energy)**.** *We let $C(i,j)$ denote the maximum chargeable energy in period $[i,j)$. We let $C_1(i,j)$ denote the maximum chargeable energy in a single charging process starting at instant i and ending at instant j. We have the following DP recurrence.*

$$C(i,j) = \max\{C(i,k) + C_1(k,j) \mid i \le k \le j-1\} \tag{6.19}$$

for $0 \le i < j \le T$. The base cases are

$$C(i,i) = 0, 0 \le i \le T$$

Lemma VI.10. *The DP in Definition VI.9 computes the maximum chargeable energy for every time interval while ensuring all the constraints except for the upper energy bound constraint.*

Proof. Consider an optimal strategy that charges the ESS in the time interval $[i,j)$, ignoring the upper energy bound constraint. The strategy must be a sequence of charging processes providing the maximum amount of energy. Suppose that the last charging process starts at the time instant k' (which must end at the time instant j). It follows that the maximum amount of energy charged in time interval $[k',j)$ is $C_1(k',j)$, and the maximum amount of energy charged in time interval $[i,k')$ is $C(i,k')$. The optimal strategy, therefore, can charge up to $C(i,k') + C_1(k',j)$ amount of energy for some k'. Equation (6.19) considers the expression $C(i,k) + C_1(k,j)$ for all possible k, and thus the DP recurrence is correct. □

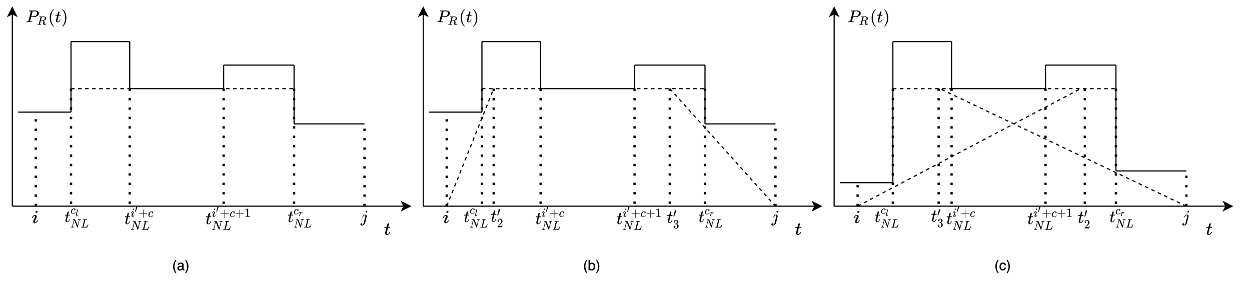


Figure VI.3: Critical points in Procedure VI.11.

We now present the procedure for computing $C_1(i, j)$ and its correctness proof. Essentially, we are computing, for a charging process starting at the time instant i and ending at j, two time instants $t_2, t_3 \in (i, j)$ and a maximum charging power P, such that the trapezoidal shape, determined by parameters i, t_2, t_3, j, P, is under the curve $P_R(t)$ for $i \leq t < j$ (we define $P_R(t) = P_{G\max} - P_{NL}(t)$) and has the maximal area. Since t_2 and t_3 are real numbers, enumeration of all combinations of t_2, t_3 is infeasible. Computing $C_1(i, j)$ thus requires extra work. We first introduce some notations for the function $P_R(t)$. Let the non-differentiable points in function $P_R(t)$ be t_{NL}^1, t_{NL}^2, ..., $t_{NL}^{M_{NL}}$ in increasing order, and let $t_{NL}^1 = 0$ and $t_{NL}^{M_{NL}} = T$. The non-differentiable points partition the entire time interval into sub-intervals $[t_{NL}^i, t_{NL}^{i+1})$, $1 \leq i < M_{NL}$, where $P_R(t)$ remains constant over each sub-interval, i.e., $P_R(t) = P_R(t_{NL}^i)$ for $t_{NL}^i \leq t < t_{NL}^{i+1}$.

Procedure VI.11. *Consider the time interval $[i, j)$ and let the non-differentiable points in $P_R(t)$ over (i, j) be $t_{NL}^{i'+1}$, $t_{NL}^{i'+2}$, ..., $t_{NL}^{i'+k}$, with $i < t_{NL}^{i'+1} < ... < t_{NL}^{i'+k} < j$. With $t_{NL}^{i'} = i$ and $t_{NL}^{i'+k+1} = j$, interval $[i, j)$ is then partitioned into sub-intervals $[t_{NL}^{i'+c}, t_{NL}^{i'+c+1})$, $0 \leq c \leq k$. The procedure that computes $C_1(i, j)$ works as follows.*

Enumerate sub-intervals $[t_{NL}^{i'+c}, t_{NL}^{i'+c+1})$ for $0 \leq c \leq k$, and for each sub-interval perform the following. Let $P' = \min(P_R(t_{NL}^{i'+c}), P_{cmax})$. Extend the interval to the left and to the right to form another interval $[t_{NL}^{c_l}, t_{NL}^{c_r})$ such that $t_{NL}^{c_l} \leq t_{NL}^{i'+c} < t_{NL}^{i'+c+1} \leq t_{NL}^{c_r}$ and $P_R(t) \geq P'$ for $t \in (t_{NL}^{c_l}, t_{NL}^{c_r})$ and the interval is maximal, as demonstrated in Figure VI.3 (a). Then, determine two values t_2' and t_3' (the procedure to compute these two values will be given later):

- *t_2' is the minimal value such that (1) $t_2' \geq t_{NL}^{c_l}$ and (2) $P_R(t) \geq \frac{P'}{t_2'-i} \cdot (t - i)$ for $t \in [i, t_{NL}^{c_l})$.*
- *t_3' is the maximal value such that (1) $t_3' \leq t_{NL}^{c_r}$ and (2) $P_R(t) \geq \frac{P'}{t_3'-j} \cdot (t - j)$ for $t \in [t_{NL}^{c_r}, j)$.*

As the end of an iteration, if $t_2' < t_3'$, we consider a charging process with parameters $t_2 = t_2'$, $t_3 = t_3'$, $P = P'$, which charges energy $E_c = \frac{1}{2} \cdot (t_3' - t_2' + j - i) \cdot P'$, referring to Figure VI.3 (b); otherwise, we consider a charging process with $t_2 = t_3 = \frac{it_3' - jt_2'}{t_3' - t_2' + i - j}$, $P = \frac{P' \cdot (i-j)}{t_3' - t_2' + i - j}$, which charges energy $E_c = \frac{1}{2} \cdot P \cdot (j - i)$, referring to Figure VI.3 (c).

Finally, $C_1(i, j)$ is the maximum E_c in all iterations.

Lemma VI.12. *Procedure VI.11 computes $C_1(i, j)$.*

Proof. We claim that one of the iterations in the procedure must contain the optimal charging scheme. Let t_2^o, t_3^o, P^o denote the parameters for the optimal charging scheme. There are two cases: (1) when $t_2^o < t_3^o$ or the charging process has a trapezoidal shape and (2) when $t_2^o = t_3^o$ or the charging process has a triangular shape.

Case 1: $t_2^o < t_3^o$. We have the following observations on optimal charging.

(1) P^o is either P_{cmax} or $P_R(t_{NL}^{i'+c})$ for some $0 \leq c \leq k$, since otherwise we can increase P^o by a sufficiently small amount without changing ramp rates. This creates another strategy that charges more energy to the ESS.
(2) Extend interval $[t_2^o, t_3^o)$ to the left and to the right to form the interval $[t_{NL}^{c_l}, t_{NL}^{c_r})$ such that $t_{NL}^{c_l} \leq t_2^o < t_3^o \leq t_{NL}^{c_r}$ and $P_R(t) \geq P^o$ for $t \in (t_{NL}^{c_l}, t_{NL}^{c_r})$ and the interval is maximal. One of the iterations encounters this interval $[t_{NL}^{c_l}, t_{NL}^{c_r})$ with P' set to P^o.
(3) With the formula for the energy charged in this charging process, $E_c = \frac{1}{2} \cdot (t_3^o - t_2^o + j - i) \cdot P^o$, and the optimality of the charging scheme, we must have t_3^o maximal, t_2^o minimal, $P_R(t) \geq \frac{P'}{t_2^o - i} \cdot (t - i)$ for $t \in [i, t_{NL}^{c_l})$, and $P_R(t) \geq \frac{P'}{t_3^o - j} \cdot (t - j)$ for $t \in [t_{NL}^{c_r}, j)$.

Therefore, there must be one iteration that contains this optimal charging scheme with the parameters t_2^o, t_3^o, and P^o.

Case 2: $t_2^o = t_3^o$. Consider functions $y_u(t) = \frac{P^o}{t_2^o - i} \cdot (t - i)$ and $y_d(t) = \frac{P^o}{t_3^o - j} \cdot (t - j)$. Let $t_2^o \in [t_{NL}^{i'+c}, t_{NL}^{i'+c+1})$ for some c, and consider the iteration with this interval. Let the parameters used in this iteration be $P' = \min(P_R(t_{NL}^{i'+c}), P_{cmax})$ and the extended interval be $[t_{NL}^{c_l}, t_{NL}^{c_r})$ (note that $t_{NL}^{c_l} \leq t_{NL}^{i'+c} \leq t_2^o = t_3^o \leq t_{NL}^{i'+c+1} \leq t_{NL}^{c_r}$), and we let $t_2'^o$ and $t_3'^o$ be the time instants such that $y_u(t_2'^o) = y_d(t_3'^o) = P'$. We have the following observations on optimal charging.

(1) We must have (i) $P' \geq P^o$ and $t_2'^o \geq t_2^o = t_3^o \geq t_3'^o$, (ii) $P_R(t) \geq \frac{P'}{t_2'^o - i} \cdot (t - i)$ for $t \in [i, t_{NL}^{c_l})$, and (iii) $P_R(t) \geq \frac{P'}{t_3'^o - j} \cdot (t - j)$ for $t \in [t_{NL}^{c_r}, j)$ so that the charging strategy is valid.
(2) Functions y_u and y_d can be rewritten as $y_u(t) = \frac{P'}{t_2'^o - i} \cdot (t - i)$ and $y_d(t) = \frac{P'}{t_3'^o - j} \cdot (t - j)$. Since $y_u(t_2^o) = y_d(t_2^o)$, we have $t_2^o = \frac{it_3'^o - jt_2'^o}{t_3'^o - t_2'^o + i - j}$ and $P_o = y_u(t_2^o) = \frac{P' \cdot (i-j)}{t_3'^o - t_2'^o + i - j}$.
(3) The energy charged in this charging process can be expressed by $E_c = \frac{1}{2} \cdot y_u(t_2^o) \cdot (j - i) = \frac{1}{2} \cdot \frac{P' \cdot (i-j)^2}{t_2'^o - t_3'^o + j - i}$, and, due to the optimality of the charging strategy, we must have $t_2'^o$ minimal and $t_3'^o$ maximal.

Again, one iteration must contain the optimal charging scheme with the parameters t_2^o, t_3^o, and P^o. Since $C_1(i, j)$ is the maximum E_c in all iterations, it is the maximum energy chargeable in one charging process that starts at the time instant i and ends at instant j. □

Procedure VI.13 below details how t_2' and t_3' are computed.

Procedure VI.13. *In the context of computing $C_1(i,j)$, given an interval $(t_{NL}^{c_l}, t_{NL}^{c_r})$, we can compute t_2' and t_3', with definitions given in Procedure VI.11, as follows. Let*

$$k_1 = \min\{\frac{P_R(t_{NL}^{i'+j})}{t_{NL}^{i'+j+1} - i} | i' \le i'+j < c_l\},\ k_2 = \max\{\frac{P_R(t_{NL}^{i'+j})}{t_{NL}^{i'+j} - j} | c_r \le i'+j \le i'+k\}$$

and we compute $t_2' = i + \frac{P'}{\min(k_1, R^u)}$ and $t_3' = j + \frac{P'}{\max(k_2, -R^d)}$. Note that we set $k_1 = \infty$ if $c_l = i'$, and we set $k_2 = -\infty$ if $c_r = i'+k+1$.

Lemma VI.14. *Procedure VI.13 computes t_2' and t_3' as defined in Procedure VI.11.*

Proof. We will prove the correctness of the formula for t_2', and the correctness for t_3' will hold by symmetry. Observe that condition $P_R(t) \ge \frac{P'}{t_2'-i} \cdot (t-i)$ for $t \in [i, t_{NL}^{c_l})$ is equivalent to $P_R(t_{NL}^{i'+j}) \ge \frac{P'}{t_2'-i} \cdot (t_{NL}^{i'+j+1} - i)$ for $i' \le i'+j < c_l$, which gives

$$t_2' \ge i + \frac{P'}{\frac{P_R(t_{NL}^{i'+j})}{t_{NL}^{i'+j+1}-i}} \text{ for } i' \le i'+j < c_l$$

which is equivalent to $t_2' \ge i + \frac{P'}{k_1}$. By Constraint (6.7), we also need to enforce $t_2' \ge i + \frac{P'}{R^u}$. Combining these inequalities, we have a lower bound for t_2'

$$t_2' \ge i + \frac{P'}{\min(k_1, R^u)}$$

Observe that when $c_l > i'$, we have $P_R(t_{NL}^{c_l-1}) < P'$ and

$$i + \frac{P'}{\min(k_1, R^u)} \ge i + \frac{P'}{k_1} \ge i + \frac{P'}{\frac{P_R(t_{NL}^{c_l-1})}{t_{NL}^{i'+c_l-1+1}-i}} > t_{NL}^{c_l}$$

and when $c_l = i'$, we have

$$i + \frac{P'}{\min(k_1, R^u)} = i + \frac{P'}{R^u} = t_{NL}^{i'} + \frac{P'}{R^u} \ge t_{NL}^{c_l}$$

Therefore, $i + \frac{P'}{\min(k_1,R^u)} \ge t_{NL}^{c_l}$, and the minimal value of t_2' such that (1) $t_2' \ge t_{NL}^{c_l}$ and (2) $P_R(t) \ge \frac{P'}{t_2'-i} \cdot (t-i)$ for $t \in [i, t_{NL}^{c_l})$ is exactly $i + \frac{P'}{\min(k_1,R^u)}$. □

Note that the current $C(i,j)$ ignores the upper energy bound constraint, and now we take it back into consideration. Observe that any charging strategy that charges the ESS with E energy can be adjusted to charge E' energy for $E' \le E$. Below we detail the procedure for adjusting any optimal charging strategy over the time interval $[i,j)$ to satisfy the upper energy bound constraint. This procedure generalizes *Charging to the Limit with Limited Energy*, so we name it *Generalized Charging to the Limit with Limited Energy* (GCtLwLE).

Definition VI.15 (GCtLwLE). *Fix a time interval $[i,j)$ and an additional parameter ΔE_m indicating the maximum amount of energy chargeable during the time interval. GCtLwLE strategy defines an optimal charging scheme in the time interval $[i,j)$ while ensuring all the constraints, which proceeds as* *follows. Let CS be an optimal charging scheme, ignoring the upper energy bound constraint, in the time interval $[i, j)$ that charges $C(i, j)$ amount of energy.*

If $C(i, j) \le \Delta E_m$, scheme CS is already an optimal scheme satisfying all constraints.

If $C(i, j) > \Delta E_m$, we adjust CS to make it charge exactly ΔE_m energy to the ESS. Let CS be a sequence of m consecutive charging processes at the time intervals $[i_k, i_{k+1})$, where $1 \le k \le m$, $i_1 = i$, and $i_{m+1} = j$. We have $\sum_{k=1}^{m} C_1(i_k, i_{k+1}) = C(i, j)$. Let q be the smallest index such that $\sum_{k=1}^{q} C_1(i_k, i_{k+1}) > \Delta E_m$, and let the parameters for the q-th charging process be $t_{c\_1}^{q} = i_q$, $t_{c\_2}^{q}$, $t_{c\_3}^{q}$, $t_{c\_4}^{q} = i_{q+1}$, and P_c^q. The adjusted scheme CS' is constructed from CS by keeping the first q charging processes, removing the charging processes after the q-th one, and adjusting the q-th charging process as follows. Define $R_q^u = \frac{P_c^q}{t_{c\_2}^{q} - t_{c\_1}^{q}}$, $R_q^d = \frac{P_c^q}{t_{c\_4}^{q} - t_{c\_3}^{q}}$, $\Delta t^q = t_{c\_4}^{q} - t_{c\_1}^{q}$. Let $t_{c\_1}^{'q} = i_q$, $t_{c\_2}^{'q}$, $t_{c\_3}^{'q}$, $t_{c\_4}^{'q} = i_{q+1}$, and $P_c^{'q}$ be the parameters of the charging process after the adjustment, we set $P_c^{'q} = (\Delta t^q - \sqrt{(\Delta t^q)^2 - 2 \cdot (\frac{R_q^u + R_q^d}{R_q^u R_q^d}) \cdot (\Delta E_m - \sum_{k=1}^{q-1} C_1(i_k, i_{k+1}))}) \cdot (\frac{R_q^u R_q^d}{R_q^u + R_q^d})$, $t_{c\_2}^{'q} = t_{c\_1}^{'q} + \frac{P_c^{'q}}{R_q^u}$, and $t_{c\_3}^{'q} = t_{c\_4}^{'q} - \frac{P_c^{'q}}{R_q^d}$.

Lemma VI.16. *GCtLwLE computes the optimal charging scheme for any time interval while ensuring all the constraints.*

Proof. We only need to show that when $C(i, j) > \Delta E_m$, the adjusted scheme CS' charges exactly ΔE_m energy to the ESS. Since we keep the first $q - 1$ charging processes, the total energy charged in these processes is $\sum_{k=1}^{q-1} C_1(i_k, i_{k+1})$. As for the q-th charging process, the way we set the parameters is the same as that in CtLwLE: without changing the ramp rates, by decreasing the maximum power $P_c^{'p}$, we make sure the charging process charges $\Delta E_m - \sum_{k=1}^{q-1} C_1(i_k, i_{k+1})$ energy (this amount is non-negative by the definition of the index q). Thus, in total, the adjusted charging scheme charges the ESS with exactly ΔE_m energy. □

6.5.2 The DP-based Solution

This subsection proposes the DP for finding the optimal coordination schedule of PPL deployment and ESS charging. It is based on the same dominance rule in Theorem VI.5.

Definition VI.17 (Generalized Optimal DP Formulation)**.** *Let $P_1(t, S)$ denote the maximum remaining energy at time t, given that the schedule deploys S pulses during the time interval $[0, t)$ and has an additional pulse to be deployed at period $[t, t + \overline{\Delta t})$. Similarly, let $P_2(t, S)$ denote the maximum remaining energy at time t, given that the schedule deploys S pulses during the time interval $[0, t)$ and has a charging process ending at time t. We have the following DP recurrences.*

$$P_1(t, S) = \max\{P_1(t - \overline{\Delta t} - \underline{\Delta t}, S - 1) - \bar{P}_d \cdot \overline{\Delta t}, P_2(t, S)\} \tag{6.20}$$

for all $0 \le t \le \max\{0, T - \overline{\Delta t}\}, 1 \le S \le \lfloor \frac{T + \Delta t}{\overline{\Delta t} + \underline{\Delta t}} \rfloor$.

$$P_2(t, S) = \max\{\min\{E_{\max}, P_1(t - \Delta t_c - \overline{\Delta t}, S - 1) - \bar{P}_d \cdot \overline{\Delta t} + C(t - \Delta t_c, t)\}\} \tag{6.21}$$

Algorithm 11: DP (General NL case)

```
Data   : problem instance with a general NL function prediction
Result : an optimal schedule
1  initialize spaces for $C_1(i,j)$s, $C(i,j)$s, $P_1(t,S)$s, $P_2(t,S)$s.
2  $C_1 \leftarrow$ ComputeAll$C_1()$.
3  $C \leftarrow$ ComputeAll$C()$.
4  for $t \leftarrow 0$ to $T$ do
5      $P_2(t,0) \leftarrow \min\{E_0 + C(0,t), E_{\max}\}$.
6      if $P_2(t,0) \geq E_{\min} + \bar{P}_d \cdot \overline{\Delta t}$ then
7          $P_1(t,0) \leftarrow P_2(t,0)$.
8      else
9          $P_1(t,0) \leftarrow -\infty$.
10 for $t \leftarrow 0$ to $T$ do
11     for $S \leftarrow 1$ to $\lfloor \frac{T+\Delta t}{\overline{\Delta t}+\underline{\Delta t}} \rfloor$ do
12         $P_2(t,S) \leftarrow \max\{\min\{E_{\max}, P_1(t-\Delta t_c-\overline{\Delta t}, S-1) - \bar{P}_d \cdot \overline{\Delta t} + C(t-\Delta t_c, t)\} \mid$
           $1 \leq \Delta t \leq t-\overline{\Delta t}, \Delta t_c = \begin{cases} \max\{\Delta t, \Delta T_{\min}\}, & t<T \\ \Delta t, & t=T \end{cases}\}$.
13         if $t \leq \max\{0, T-\overline{\Delta t}\}$ then
14             $P_1(t,S) \leftarrow \max\{P_1(t-\overline{\Delta t}-\underline{\Delta t}, S-1) - \bar{P}_d \cdot \overline{\Delta t}, P_2(t,S)\}$.
15             if $P_1(t,S) < E_{\min} + \bar{P}_d \cdot \overline{\Delta t}$ then
16                 $P_1(t,S) \leftarrow -\infty$.
17 $S_{\text{MAX}} \leftarrow \max\{\max\{1+S' | P_1(T-\overline{\Delta t}, S') - \bar{P}_d \cdot \overline{\Delta t} \geq E_{\min}\}, \max\{S' | P_2(T,S') \geq E_{\min}\}, 0\}$.
18 return TraceSchedule$(C_1, C, P_1, P_2, S_{MAX})$
```

for all $0 \leq t \leq T$, $1 \leq S \leq \lfloor \frac{T+\Delta t}{\overline{\Delta t}+\underline{\Delta t}} \rfloor$, $1 \leq \Delta t \leq t - \overline{\Delta t}$, *and* $\Delta t_c = \max\{\Delta t, \Delta T_{\min}\}$ *if* $t < T$ *or* Δt *if* $t = T$. *Similarly, as in the DP formulation in Definition VI.6, whenever term* $P_1(t - \overline{\Delta t} - \underline{\Delta t}, S-1)$, *term* $P_1(t - \Delta t_c - \overline{\Delta t}, S-1)$, *or term* $P_2(t,S)$ *has invalid arguments or values, i.e., the value is less than* $E_{\min} + \bar{P}_d \cdot \overline{\Delta t}$ *for the first two terms or less than* $E_{\min}$ *for the term* $P_2(t,S)$*, its value is set to* $-\infty$. *We have the following base cases.*

$$P_2(t,0) = \min\{E_0 + C(0,t), E_{\max}\}, \forall\, 0 \leq t \leq T \tag{6.22}$$

$$P_1(t,0) = \begin{cases} P_2(t,0), & P_2(t,0) \geq E_{\min} + \bar{P}_d \cdot \overline{\Delta t} \\ -\infty, & P_2(t,0) < E_{\min} + \bar{P}_d \cdot \overline{\Delta t} \end{cases}, \forall\, 0 \leq t \leq \max\{0, T - \overline{\Delta t}\} \tag{6.23}$$

The maximum number of pulses deployed by an optimal schedule is

$$\max\{\max\{1 + S' | P_1(T - \overline{\Delta t}, S') - \bar{P}_d \cdot \overline{\Delta t} \geq E_{\min}\}, \max\{S' | P_2(T, S') \geq E_{\min}\}, 0\} \tag{6.24}$$

The correctness proof is by similar arguments used for Theorem VI.7. We provide the pseudo-code of the DP in Algorithm 11. Let M_{NL} denote the number of non-differentiable points in function $P_R(t)$. Line 2 computes $C_1(i,j)$s by implementing Procedure VI.11 and VI.13, incurring a complexity of $O(T^2 \cdot M_{NL})$. Line 3 computes $C(i,j)$s using the DP in Definition VI.9, incurring a complexity of $O(T^3)$. Lines 4 to 17 compute the values for all DP states defined in Definition VI.17 with the computed

$C(i,j)$s. With $O(T \cdot S_{\max}) \subseteq O(T^2)$ state space and $O(T)$ transition time, computing all DP states incurs a cost of $O(T^3)$ time. Finally, Line 18 traces and returns the corresponding optimal schedule from all computed DP states in $O(T^2)$ time. We again omit the tracing. The computational complexity of the DP in Definition VI.17 is $O(T^3 + T^2 \cdot M_{NL})$, and the space complexity is $O(T^2)$.

Theorem VI.18 (Optimality of Consistency for Algorithm 11)**.** *Algorithm 11 has optimal consistency.*

Proof. The theorem holds by the same arguments for Theorem VI.8 and the Fundamental Theorems of Consistency (Theorems II.4 and II.5). □

6.6 Simulation

In this section, we evaluate the performance of the proposed DP algorithms under both scenarios of constant and general NL functions via extensive simulations. The proposed algorithm, in both constant and general NL cases, will be compared against the existing *Particle Swarm Optimization* (PSO) solution [71] and an improved greedy heuristic. In the constant NL case, we also compare our algorithm with the heuristic algorithm in [73], which we call *Heuristic*. The simulations are not meant to be exhaustive but are designed to validate the theoretical results and show performance improvement over the existing algorithms. Code for the proposed DP algorithm and the improved greedy heuristic under constant NL functions and general NL functions have been open-sourced[1].

This section is organized as follows. Subsection 6.6.1 gives the experimental setup. Subsection 6.6.2 presents the evaluation results. In particular, Subsection 6.6.2.1 shows the sample execution of the proposed algorithm; Subsection 6.6.2.2 compares the performance for the energy output between the proposed algorithms and the existing solutions; Subsection 6.6.2.3 compares the run time between the proposed algorithms and the existing solutions.

6.6.1 Experimental Setup

The parameters used in the experiments are set to the actual values used in real-world applications [73, 79–81], and the key parameter ranges are listed in Table 2. The studied time T reaches up to 20,000 ms as the practical time scale for such systems to predict the NL and operate under critical-mission modes is from milliseconds to thousands of milliseconds.

Parameter	$\bar{P}_d$	$\overline{\Delta t}$	$\underline{\Delta t}$	$E_{\min}$	$E_{\max}$	E_0	P_{NL}	$P_{G\max}$	R^u, R^d	P_{cmax}	$\Delta T_{\min}$	T
Range	$[4,10]$	$[1,33]$	$[1,33]$	$(0,5]$	$[5,40]$	$[E_{\min},E_{\max}]$	$(0,0.85P_{G\max}]$	$[10,30]$	$[0.1,3]$	$(0,10]$	$[\underline{\Delta t},T]$	$(0,20000]$
Unit	MW	ms	ms	MJ	MJ	MJ	MW	MW	MW/s	MW	ms	ms

Table VI.2: The range of key parameters used in the experiments.

[1]View source code on GitHub.

Parameter	$\bar{P}_d$	$\overline{\Delta t}$	$\underline{\Delta t}$	E_0	$E_{\min}$	$E_{\max}$	P_{NL}	$P_{G\max}$	R^u	R^d	P_{cmax}	$\Delta T_{\min}$	T
(a)	9.0	30	30	6.02	5.0	25.5	14.0	24.0	0.3	0.9	4.0	14000	20000
(b)	9.0	30	30	6.02	5.0	25.5	14.0	24.0	0.3	0.9	4.0	40	20000
(c)	9.0	30	30	5.35	5.0	25.5	–	–	3.0	2.8	4.0	9000	20000
(d)	9.0	30	30	5.35	5.0	25.5	–	–	3.0	2.8	4.0	7000	20000
(e)	9.0	30	30	5.35	5.0	25.5	–	–	3.0	2.8	4.0	5000	20000

Table VI.3: Parameters for the sample executions.

The parameter P_{NL} is a function in time, which is generated as follows. Iterate through time 1 to T, and at every time instant t, we generate, with the probability of r, a non-differentiable point that takes the function $P_{NL}(t)$ to some value randomly generated from the interval $(0, 0.85P_{G\max}]$. We set $r = \frac{1}{1000}$ so changes happen every few seconds, which is consistent with practical applications.

The existing algorithms do not guarantee feasible solutions. For this reason and to test the practical effectiveness of our proposed DP, we design two improved greedy algorithms. They guarantee to produce feasible solutions and aim to outperform the existing solutions in all cases. All the derived optimal charging processes are re-used in designing the greedy algorithms, i.e., the greedy algorithms exploit the structural properties of optimal charging as DP does. The greedy algorithms for both scenarios share the same idea: deploying as many pulses as possible when the ESS has enough energy to output one more pulse without violating any constraints. Otherwise, the system performs a charging process to ensure the energy in ESS reaches the threshold E_{up} defined as $E_{\min} + \lfloor \frac{E_{\max}-E_{\min}}{\bar{P}_d \cdot \overline{\Delta t}} \rfloor \cdot \bar{P}_d \cdot \overline{\Delta t}$. Observe that the energy above E_{up} and below $E_{\max}$ cannot be used by IPS, so this strategy makes full use of charging. The idea behind the greedy algorithms is that efficient charging processes support more pulses as the energy for pulse deployment comes from the charging processes. Observe that a charging process receives a high incentive per unit of time if $P_c^{j_c}$ is large. Grouping charging processes altogether is mostly beneficial. To determine the parameters for charging processes, the optimal CtLwLE strategy (Definition VI.4) is used in the constant NL scenario, and the optimal GCtLwLE (Definition VI.15) is used in the general NL scenario. We generate 5000 independent problem instances for various r ($r = 0$ indicates the constant NL case) with T increasing to 20,000 ms at a fixed step. For each instance, PSO, greedy, and DP algorithms have been executed in turn. The Heuristic also has also been executed for the instances with constant NL. The energy output and the run time for the algorithms are recorded and compared. For accurate measurement of execution time, each reported entry is an average of 1000 independent executions. The simulations are implemented in C++ and conducted on computers with Intel(R) Xeon(R) CPU E5-2697 v4 @ 2.60GHz and 32GB Memory. Finally, we note that the time complexities of greedy algorithms are $O(T)$ and $O(T^3 + T^2 \cdot M_{NL})$ for constant and general NL cases, respectively.

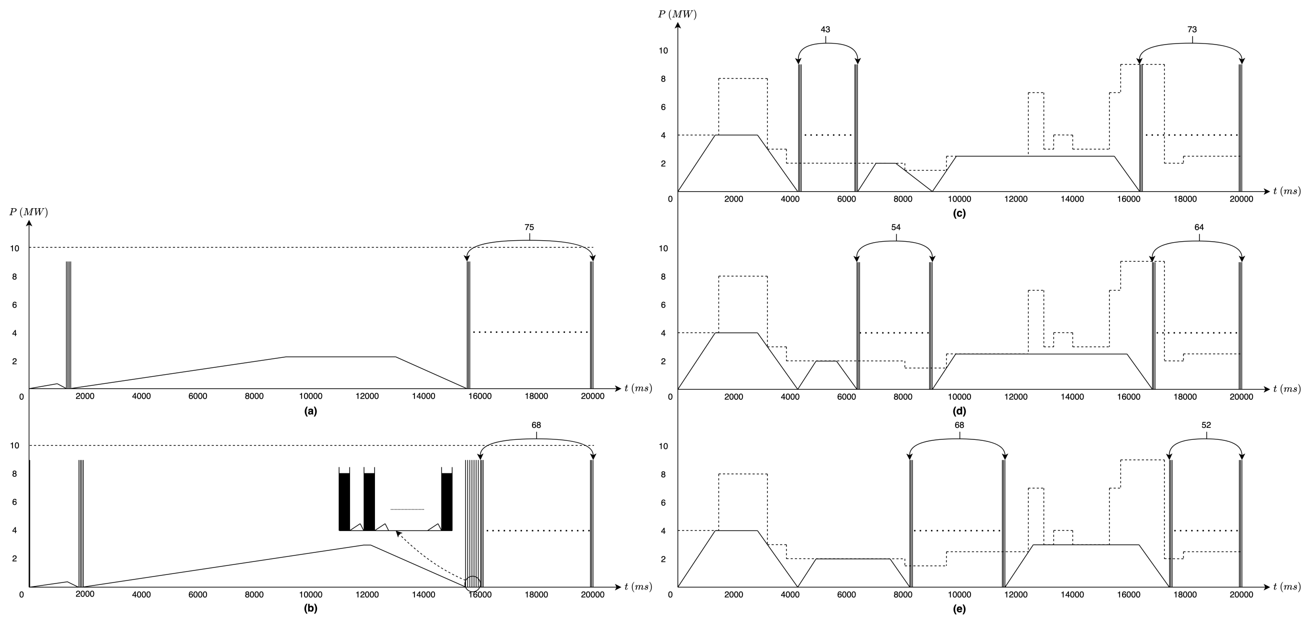


Figure VI.4: Sample executions.

6.6.2 Evaluation Results

6.6.2.1 Sample Execution

This section presents the results of five sample algorithm executions in Figure VI.4 with the parameters listed in Table VI.3. Executions (a) and (b) are for constant NL functions, and Executions (c), (d), and (e) are for general NL functions. Since $P_R(t) = P_{G\max} - P_{NL}(t)$ is a function in time for the last three executions, its values are omitted in the table but identifiable from the figures. The dashed line on top indicates the function $P_R(t)$; a thin rectangle indicates a deployed pulse; a trapezoid or triangle indicates a charging process. We intend to make the problem instances in the same scenario share similar parameters to observe the relative changes in the metric. The deployed pulses are 79, 80, 116, 118, and 120 in Executions (a) to (e).

The difference in parameter setting between Executions (a) and (b), or between Executions (c), (d), and (e), is the value of rest time of deployments, $\Delta T_{\min}$. Since the parameter $\Delta T_{\min}$ sets additional time restrictions on deployment processes, the larger the value is, the fewer pulses the IPS can deploy. This pattern holds in both cases and can be observed in the sample executions. Execution (a) has 79 deployed pulses with $\Delta T_{\min} = 14000$ while Execution (b) has 80 deployed pulses with $\Delta T_{\min} = 40$. Executions (c), (d), and (e) have a decreasing $\Delta T_{\min}$, from 9000 to 7000 to 5000 but an increasing number of deployed pulses, from 116 to 118 to 120. We also observe two interesting trends. When $\Delta T_{\min}$ is getting close to $\underline{\Delta t}$, as shown in Execution (b), the algorithm uses the time between adjacent pulses for performing tiny charging processes within some periods. Although the waiting time between adjacent pulses becomes longer than otherwise, the charging process receives more time in return. In the case of general NL functions, as shown in Executions (c), (d), and (e), the optimal strategy selects the best period for ESS charging. Instead of directly using one single charging process, it creates multiple

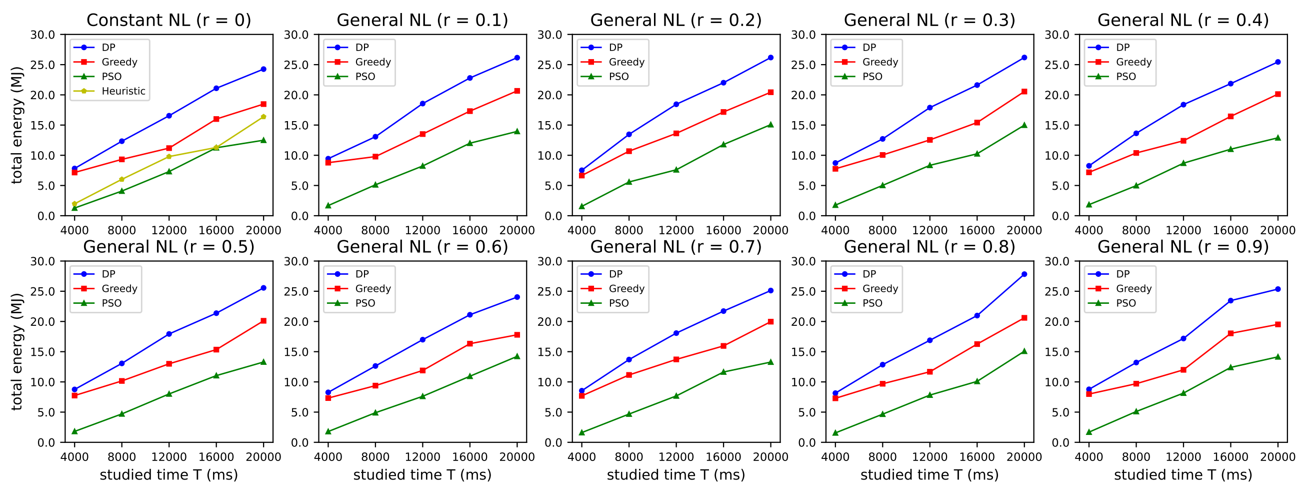


Figure VI.5: Energy output comparison between PSO, Heuristic, Greedy, and DP with varying r and increasing T.

charging processes in a given period. These observations suggest that our proposed algorithm utilizes different sets of IPS parameters to obtain the optimal objective values.

6.6.2.2 Energy Output Results

Figure VI.5 shows the energy output comparison between PSO, Heuristic, Greedy, and DP under two scenarios. The first plot corresponds to the constant NL case, while the rest corresponds to the general NL case with increasing power profiles. All figures indicate that DP consistently outperforms Greedy, which consistently outperforms PSO. In the case of constant NL, the Heuristic outperforms PSO but still runs below Greedy. On average, DP has a 175% performance improvement over PSO, 110% over Heuristic, and 70% over Greedy. The results remain the same no matter how r is changed.

In our experiments, we observe that the feasible solution space for PSO is narrow. Thus the execution relies heavily on the random initialization that wishes to start with some feasible solutions. In most cases, PSO finds trivial schedules that produce little pulses. In contrast, the greedy heuristics can find feasible solutions with the optimal charging strategies (Definitions VI.4 and VI.15). The high-quality charging processes lead the greedy heuristics to produce more pulses than Heuristic and PSO do.

Nevertheless, the ESS could be fully charged in a charging process under the control of the greedy algorithm. This operation may restrict the time for PPL deployment and thus lead to mission failure. DP, in contrast, can make an optimal decision in time allocation for deployment and charging. DP can also take advantage of overlapping the rest time between adjacent pulse deployment with tiny charging processes, as shown in the sample Execution (b). This trick is beneficial when $\Delta T_{\min}$ is close enough to $\underline{\Delta t}$. All these factors contribute to DP outperforming Greedy in energy utilization, revealing the essence of the proposed DP. The simulation confirms the theoretical results about the optimality of the proposed DP algorithms. They also show performance improvement when they are deployed to the real IPS.

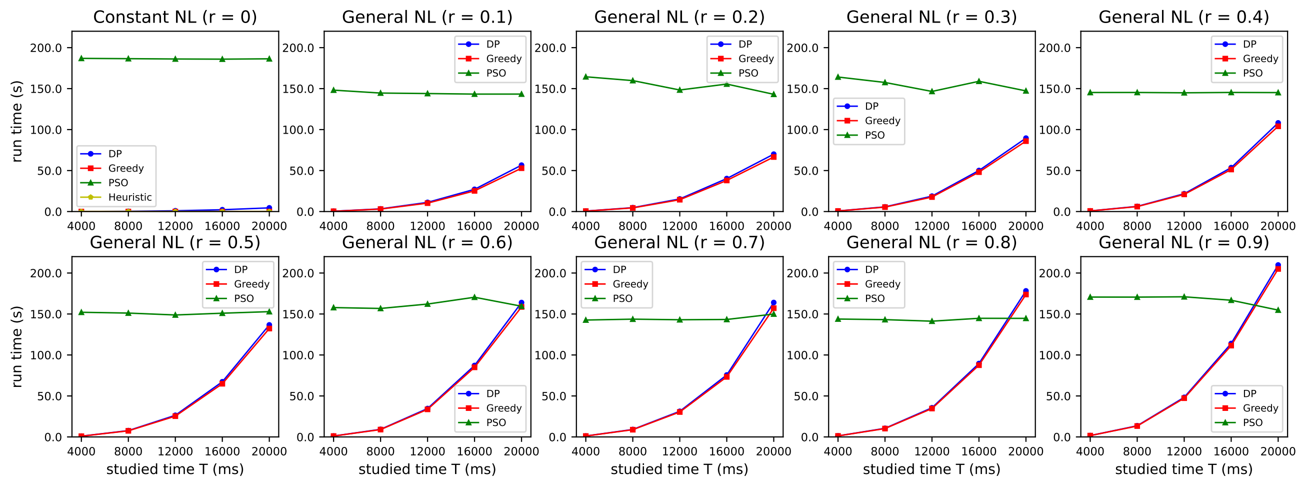


Figure VI.6: Run time comparison between PSO, Heuristic, Greedy, and DP with varying r and increasing T.

6.6.2.3 Execution Time Results

Figure VI.6 shows the run time comparison between PSO, Heuristic, Greedy, and DP under two scenarios. Heuristic and Greedy share similar run time results in the case of constant NL. Greedy is slightly faster than DP, and both are much faster than PSO. The same trend holds under the general NL case. The exceptions are when both T and r become very large in the last three figures, where PSO becomes the fastest, leaving Greedy second and DP the last. All the algorithms can finish in a reasonable time. It is interesting to observe that the run time for PSO stays about the same regardless of the values of T and r. The consistent run time performance is because PSO's run time depends on the number of iterations and particles in the swarm, and these hyper-parameters are constants in our simulations. In contrast, the run time of both Greedy and DP increases with the studied time T and the number of NL power profiles M_{NL}. In the constant NL case, the run time increases linearly with T, while it increases cubically in the general NL case with the other variables fixed. The slopes of the curves reflect these complexity results. Both the run times for Greedy and DP increase as the number of NL power profiles increases. The run time for Greedy and DP become closer to each other as r increases since both algorithms use the DP in Definition VI.9 to compute $C(i, j)$, incurring a computational cost of $O(T^3 + T^2 \cdot M_{NL})$ that dominates the complexities. Increasing r attains stronger dominance, thus leading to much closer run-time results.

As a concluding remark, DP satisfies the real-time requirements and is optimal in energy output and run time compared with PSO, Heuristic, and Greedy. Greedy has only a slight run time improvement over DP, but the negligible additional computational cost in DP returns an average of 70% performance improvement in PPL deployment.

6.7 Related Results

The coordination problem of maximizing a PPL's output utility is an emerging research problem. As a complex non-linear optimization, the problem has not yet been extensively studied. To our knowledge, no theoretical analysis has been developed for this problem. We review related works [71–73] on this topic and outline our differences. These works are all heuristic approaches and have exponential run times in the worst case.

The PPL scheduling problem has been first introduced and studied in [71]. The formulation in [71] assumes that both charging and deployment processes are periodic over time and relaxes the energy bounds constraints (Constraints (6.2) – (6.3)) to ensure their proposed approach finds solutions. The proposed solution uses PSO to search for sub-optimal solutions. The PSO algorithm requires a long processing time to find solutions and thus may not be suitable for practical use. If the number of iterations is reduced, the algorithm can terminate within a reasonable amount of time, but it will no longer guarantee feasible solutions.

The work in [72] is built on the intuition that more available energy can support more PPL deployment. Thus, the authors study a morphic equivalence problem of maximally allocating rectangles inside a region with part of the region marked as pre-allocated NL. Mixed-Integer Linear Programming is used for finding the optimal allocation. Although the solution is guaranteed optimal, it suffers from the explicit assumption of $R^u = R^d = \infty$ and an exponential computational cost. More importantly, this solution cannot apply directly to the PPL coordination problem; it requires further post-processing.

Recent work [73] proposes a solution to the PPL coordination problem under the following assumptions: (1) charging processes and deployment processes are periodic in time, (2) only the average power bound is considered, not the instantaneous power bound, and (3) the NL is assumed to be a constant. Their solution decomposes the problem into two sub-problems and then solves them separately to form the final solution. The sub-problems are heuristic, making the solution less likely to optimize the original problem.

6.8 Conclusions and Future Work

This work studies the PPL scheduling problem with NL prediction in mission- and time-critical CPS. The problem is modeled under two scenarios for different use cases: the constant NL and general NL functions. Then, we propose and prove the first exact pseudo-polynomial time algorithms based on Dynamic Programming, outperforming the existing solutions in both energy output and run time. We show that the algorithms achieve optimal consistency. The experimental results further confirm the validity and practicality of our approaches. Both algorithms guarantee that an IPS can perform optimally in completing critical missions, even in the worst case. In the future, we will study how our proposed solutions could be extended to solve the coordination of multiple and simultaneous PPLs in IPS. We also plan to study how to enable the scheduler to make reliable decisions under an inaccurate prediction of the NL. For now, our proposed algorithm for general NL functions relies on the assumption of predicting the accurate variable NL. Solving PPL scheduling without this assumption requires a formal definition of the prediction error metric and designing new algorithms using the possibly inaccurate prediction to yield guaranteed performance which depends on the prediction quality.

CHAPTER 7

Conclusion

In this thesis, we present a framework named online scheduling with predictions that addresses the challenges classic online scheduling faces. Our framework uses additional predictions about unknowns as inputs to improve scheduling performance. We introduce a metric for quantifying the prediction error and demonstrate its integration into the design and analysis of algorithms. We also present three performance metrics — consistency, robustness, and smoothness — which are used to evaluate the performance under the framework. We present solutions to central online scheduling problems and applications in cyber-physical systems scheduling, including uniform machine scheduling with job size predictions to minimize makespan, single and parallel machine scheduling with job size predictions to minimize mean response time, and pulsed power load scheduling in cyber-physical systems. Our solutions achieve near-optimal performance when predictions are accurate and maintain bounded performance even when predictions are poor. Through analysis and extensive simulations, we show the effectiveness of our proposed framework. Our algorithms consistently outperform the state-of-the-art methods by leveraging predictions.

We highlight the connections between online scheduling with predictions and online clairvoyant and non-clairvoyant scheduling. A best-performing learning-augmented algorithm should function like an optimal online clairvoyant scheduler with quality predictions and an optimal online non-clairvoyant scheduler with poor predictions. Our algorithms demonstrate how to manage uncertainty in online scheduling, with the prediction error as a proxy for the degree of uncertainty. The framework of online scheduling with predictions provides general solutions for decision-making under uncertainty.

The framework represents a breakthrough by moving beyond traditional assumptions of perfect or no information and surpassing traditional worst-case algorithm performance analysis by considering the practical setting of having potentially imperfect information about unknowns in the design and analysis of algorithms. The fundamental theorems of consistency, robustness, and smoothness reveal the potential and limitations of information in decision-making. The algorithms under this framework demonstrate how to use information while being aware of its imperfection. Together, they provide new insights into the role of information in online decision-making.

Future Work

Much remains to be done to extend the theory and applications of online scheduling with predictions.

Lower Bounds on Smoothness

The fundamental theorems of consistency and robustness provide tight lower bounds for consistency and robustness, but determining lower bounds on smoothness is more complex. It depends on the specific problem and the assumptions of the adversary. To establish such bounds, one must construct worst-case problem instances with predictions that prove the limitations of predictions. Systematic approaches to lower bound smoothness are an important area for further research, as the bounds represent the return of prediction quality in algorithm performance.

Leveraging the Distribution of Prediction Error

Analysis in this thesis considers the worst-case prediction errors. However, in many real-world scenarios, the prediction errors follow known distributions (e.g., normal distribution). Exploring the potential of knowing the error distribution and how algorithms can leverage this information is an important area for further research. Since stochastic scheduling requires known problem parameter distributions, we expect that online scheduling with predictions can also leverage the assumptions about the prediction error distributions.

Trade-off of Prediction Quality and Performance

While near-perfect predictions are desirable, achieving them can be computationally expensive and may not always result in matching performance improvement. The additional computation may serve better if they are used in decision-making, e.g., running more iterations for the algorithms. Therefore, the trade-off between prediction quality and performance must be considered to find the equilibrium between allocating computational resources to prediction quality and decision-making. A model or objective quantifying the quality of the trade-off is needed. This trade-off will be a key consideration in the application of online scheduling with predictions.

Bibliography

[1] T. Zhao, W. Li, and A. Y. Zomaya, "Uniform machine scheduling with predictions," *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 32, pp. 413–422, Jun. 2022.

[2] T. Zhao, W. Li, and A. Y. Zomaya, "Learning-augmented scheduling," *IEEE Transactions on Computers*, 2023. Submitted.

[3] T. Zhao, C. Li, W. Li, and A. Y. Zomaya, "Brief announcement: Towards a more robust algorithm for flow time scheduling with predictions," in *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '22, (New York, NY, USA), p. 385–388, Association for Computing Machinery, 2022.

[4] T. Zhao, W. Li, and A. Y. Zomaya, "Real-time scheduling with predictions," in *2022 IEEE Real-Time Systems Symposium (RTSS)*, pp. 331–343, 2022.

[5] T. Zhao, W. Li, B. Qin, L. Wang, and A. Y. Zomaya, "Pulsed power load coordination in mission and time critical cyber-physical systems," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, Dec. 2022.

[6] T. Zhao, W. Si, W. Li, and A. Y. Zomaya, "Optimizing the maximum vertex coverage attacks under knapsack constraint," *IEEE/ACM Transactions on Networking*, vol. 29, no. 3, pp. 1088–1104, 2021.

[7] T. Zhao, W. Si, W. Li, and A. Y. Zomaya, "Towards minimizing the r metric for measuring network robustness," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 4, pp. 3290–3302, 2021.

[8] "Algorithms with predictions." `https://algorithms-with-predictions.github.io/`. Accessed: 2023-02-28.

[9] R. Graham, E. Lawler, J. Lenstra, and A. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," in *Discrete Optimization II* (P. Hammer, E. Johnson, and B. Korte, eds.), vol. 5 of *Annals of Discrete Mathematics*, pp. 287–326, Elsevier, 1979.

[10] S. Lattanzi, T. Lavastida, B. Moseley, and S. Vassilvitskii, "Online scheduling via learned weights," in *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '20, (USA), p. 1859–1877, Society for Industrial and Applied Mathematics, 2020.

[11] M. Purohit, Z. Svitkina, and R. Kumar, "Improving online algorithms via ml predictions," in *Advances in Neural Information Processing Systems* (S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), vol. 31, Curran Associates, Inc., 2018.

[12] A. Wei and F. Zhang, "Optimal robustness-consistency trade-offs for learning-augmented online algorithms," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, (Red Hook, NY, USA), Curran Associates Inc., 2020.

[13] E. Bampis, K. Dogeas, A. Kononov, G. Lucarelli, and F. Pascual, “Scheduling with untrusted predictions,” in *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22* (L. D. Raedt, ed.), pp. 4581–4587, International Joint Conferences on Artificial Intelligence Organization, 7 2022. Main Track.

[14] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*. USA: Cambridge University Press, 1998.

[15] J. Leung, L. Kelly, and J. H. Anderson, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. USA: CRC Press, Inc., 2004.

[16] R. M. Karp, “On-line algorithms versus off-line algorithms: How much is it worth to know the future?,” in *Proceedings of the IFIP 12th World Computer Congress on Algorithms, Software, Architecture - Information Processing ’92, Volume 1 - Volume I*, (NLD), p. 416–429, North-Holland Publishing Co., 1992.

[17] B. Awerbuch, S. Kutten, and D. Peleg, “Competitive distributed job scheduling (extended abstract),” in *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, STOC ’92, (New York, NY, USA), p. 571–580, Association for Computing Machinery, 1992.

[18] J. Boyar, L. M. Favrholdt, C. Kudahl, K. S. Larsen, and J. W. Mikkelsen, “Online algorithms with advice: A survey,” *SIGACT News*, vol. 47, p. 93–129, Aug. 2016.

[19] A. Antoniadis, C. Coester, M. Elias, A. Polak, and B. Simon, “Online metric algorithms with untrusted predictions,” in *Proceedings of the 37th International Conference on Machine Learning* (H. D. III and A. Singh, eds.), vol. 119 of *Proceedings of Machine Learning Research*, pp. 345–355, PMLR, 13–18 Jul 2020.

[20] S. Gollapudi and D. Panigrahi, “Online algorithms for rent-or-buy with expert advice,” in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, pp. 2319–2327, PMLR, 09–15 Jun 2019.

[21] M. Mitzenmacher, “Scheduling with predictions and the price of misprediction,” in *ITCS*, 2020.

[22] P. Dütting, S. Lattanzi, R. Paes Leme, and S. Vassilvitskii, “Secretaries with advice,” in *Proceedings of the 22nd ACM Conference on Economics and Computation*, EC ’21, (New York, NY, USA), p. 409–429, Association for Computing Machinery, 2021.

[23] M. Frye, D. Gyulai, J. Bergmann, and R. H. Schmitt, “Adaptive scheduling through machine learning-based process parameter prediction,” *MM Science journal*, vol. 2019, pp. HSM2019–023, Oct 2019.

[24] K. Anand, R. Ge, and D. Panigrahi, “Customizing ML predictions for online algorithms,” in *Proceedings of the 37th International Conference on Machine Learning* (H. D. III and A. Singh, eds.), vol. 119 of *Proceedings of Machine Learning Research*, pp. 303–313, PMLR, 13–18 Jul 2020.

[25] Y. Azar, S. Leonardi, and N. Touitou, “Flow time scheduling with uncertain processing time,” in *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2021, (New York, NY, USA), p. 1070–1080, Association for Computing Machinery, 2021.

[26] S. Im, R. Kumar, M. Montazer Qaem, and M. Purohit, “Non-clairvoyant scheduling with predictions,” in *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*,

SPAA '21, (New York, NY, USA), p. 285–294, Association for Computing Machinery, 2021.

[27] D. B. Shmoys, J. Wein, and D. P. Williamson, "Scheduling parallel machines on-line," *SIAM Journal on Computing*, vol. 24, no. 6, pp. 1313–1331, 1995.

[28] M. Amiri and L. Mohammad-Khanli, "Survey on prediction models of applications for resources provisioning in cloud," *J. Netw. Comput. Appl.*, vol. 82, p. 93–113, Mar. 2017.

[29] N. Peyravi and A. Moeini, "Estimating runtime of a job in hadoop mapreduce," *Journal of Big Data*, vol. 7, no. 1, p. 44, 2020.

[30] H. Yamashiro and H. Nonaka, "Estimation of processing time using machine learning and real factory data for optimization of parallel machine scheduling problem," *Operations Research Perspectives*, vol. 8, p. 100196, 2021.

[31] M. R. Garey and D. S. Johnson, *Computers and intractability: a guide to the theory of NP-completeness*. W.H. Freeman and Company, 2009.

[32] Y. Azar, S. Leonardi, and N. Touitou, "Distortion-oblivious algorithms for minimizing flow time," Sep 2021.

[33] R. Motwani, S. Phillips, and E. Torng, "Non-clairvoyant scheduling," in *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '93, (USA), p. 422–431, Society for Industrial and Applied Mathematics, 1993.

[34] G. Buttazzo, G. Lipari, L. Abeni, and M. Caccamo, *Soft Real-Time Systems: Predictability vs. Efficiency (Series in Computer Science)*. Plenum Publishing Co., 2005.

[35] K. Pruhs, J. Sgall, and E. Torng, "Online scheduling," in *Handbook of Scheduling: Algorithms, Models, and Performance Analysis* (J. Y.-T. Leung, ed.), USA: CRC Press, Inc., 2004.

[36] S. Bozhko, G. von der Brüggen, and B. B. Brandenburg, "Monte carlo response-time analysis," in *2021 IEEE Real-Time Systems Symposium (RTSS)*, pp. 342–355, 2021.

[37] A. a. Bhuiyan, K. Yang, S. Arefin, A. Saifullah, N. Guan, and Z. Guo, "Mixed-criticality multicore scheduling of real-time gang task systems," in *2019 IEEE Real-Time Systems Symposium (RTSS)*, pp. 469–480, 2019.

[38] O. Bellenguez-Morineau, M. Chrobak, C. Dürr, and D. Prot, "A Note on NP-Hardness of Preemptive Mean Flow-Time Scheduling for Parallel Machines," *Journal of Scheduling*, vol. 18, no. 3, pp. 299–304, 2015.

[39] S. Leonardi and D. Raz, "Approximating total flow time on parallel machines," *Journal of Computer and System Sciences*, vol. 73, no. 6, pp. 875–891, 2007.

[40] K. R. Baker, *Introduction to sequencing and scheduling*. Wiley, 1974.

[41] L. Becchetti and S. Leonardi, "Nonclairvoyant scheduling to minimize the total flow time on single and parallel machines," *J. ACM*, vol. 51, p. 517–539, jul 2004.

[42] Z. Scully, M. Harchol-Balter, and A. Scheller-Wolf, "Simple near-optimal scheduling for the m/g/1," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 4, may 2020.

[43] L. Becchetti, S. Leonardi, A. Marchetti-Spaccamela, G. Schafer, and T. Vredeveld, "Average case and smoothed competitive analysis of the multi-level feedback algorithm," in *44th Annual IEEE Symposium on Foundations of Computer Science, 2003. Proceedings.*, pp. 462–471, 2003.

[44] S. Liu, S. Yao, X. Fu, R. Tabish, S. Yu, A. Bansal, H. Yun, L. Sha, and T. Abdelzaher, "On removing algorithmic priority inversion from mission-critical machine inference pipelines," in *2020 IEEE*

Real-Time Systems Symposium (RTSS), pp. 319–332, 2020.

[45] B. Kalyanasundaram and K. R. Pruhs, “Minimizing flow time nonclairvoyantly,” *J. ACM*, vol. 50, p. 551–567, jul 2003.

[46] B. Kalyanasundaram and K. Pruhs, “Speed is as powerful as clairvoyance,” *J. ACM*, vol. 47, p. 617–643, jul 2000.

[47] L. Becchetti, S. Leonardi, and S. Muthukrishnan, “Scheduling to minimize average stretch without migration,” in *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA ’00, (USA), p. 548–557, Society for Industrial and Applied Mathematics, 2000.

[48] E. G. Coffman and P. J. Denning, *Operating Systems Theory*. Prentice Hall Professional Technical Reference, 1973.

[49] B. Wang, X. Li, L. P. de Aguiar, D. S. Menasche, and Z. Shafiq, “Characterizing and modeling patching practices of industrial control systems,” *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 1, June 2017.

[50] Q. Zhang, D. K. Hong, Z. Zhang, Q. A. Chen, S. Mahlke, and Z. M. Mao, “A systematic framework to identify violations of scenario-dependent driving rules in autonomous vehicle software,” *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 5, June 2021.

[51] N. Périvier, C. Hssaine, S. Samaranayake, and S. Banerjee, “Real-time approximate routing for smart transit systems,” *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 5, no. 2, p. 1–30, 2021.

[52] G. Pettet, A. Mukhopadhyay, M. J. Kochenderfer, and A. Dubey, “Hierarchical planning for resource allocation in emergency response systems,” in *Proceedings of the ACM/IEEE 12th International Conference on Cyber-Physical Systems*, (New York, NY, USA), pp. 155–166, Association for Computing Machinery, 2021.

[53] X. Feng, K. L. Butler-Purry, and T. Zourntos, “A multi-agent system framework for real-time electric load management in mvac all-electric ship power systems,” *IEEE Transactions on Power Systems*, vol. 30, no. 3, p. 1327–1336, 2015.

[54] S. Kulkarni and S. Santoso, “Impact of pulse loads on electric ship power system: With and without flywheel energy storage systems,” *2009 IEEE Electric Ship Technologies Symposium*, 2009.

[55] J. Boudjadar and M. H. Khooban, “A cost-effective scheduling control for a safety critical hybrid power system,” in *2020 IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pp. 1–4, IEEE, 2020.

[56] F. Geth, C. Coffrin, and D. Fobes, “A flexible storage model for power network optimization,” in *Proceedings of the Eleventh ACM International Conference on Future Energy Systems*, e-Energy ’20, (New York, NY, USA), p. 503–508, Association for Computing Machinery, 2020.

[57] B. Sun, A. Zeynali, T. Li, M. Hajiesmaili, A. Wierman, and D. H. Tsang, “Competitive algorithms for the online multiple knapsack problem with application to electric vehicle charging,” *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 4, no. 3, p. 1–32, 2020.

[58] H. Pandžić and V. Bobanac, “An accurate charging model of battery energy storage,” *IEEE Transactions on Power Systems*, vol. 34, no. 2, pp. 1416–1426, 2019.

[59] G. Wang, Y. Zhang, Z. Fang, S. Wang, F. Zhang, and D. Zhang, “Faircharge: A data-driven fairness-aware charging recommendation system for large-scale electric taxi fleets,” *Proc. ACM*

Interact. Mob. Wearable Ubiquitous Technol., vol. 4, Mar. 2020.

[60] K. Vatanparvar and M. A. A. Faruque, "Electric vehicle optimized charge and drive management," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 23, Aug. 2017.

[61] K. K. Tafanidis, K. D. Taxeidis, G. J. Tsekouras, and F. D. Kanellos, "Optimal operation of war-ship electric power system equipped with energy storage system," *Journal of Computations & Modelling*, vol. 3, no. 4, p. 41–60, 2013.

[62] J. Hou, J. Sun, and H. F. Hofmann, "Mitigating power fluctuations in electric ship propulsion with hybrid energy storage system: Design and analysis," *IEEE Journal of Oceanic Engineering*, vol. 43, no. 1, p. 93–107, 2018.

[63] C. R. Lashway, A. T. Elsayed, and O. A. Mohammed, "Hybrid energy storage management in ship power systems with multiple pulsed loads," *Electric Power Systems Research*, vol. 141, p. 50–62, 2016.

[64] S. Samineni, B. Johnson, H. Hess, and J. Law, "Modeling and analysis of a flywheel energy storage system for voltage sag correction," *IEEE Transactions on Industry Applications*, vol. 42, no. 1, p. 42–52, 2006.

[65] F. Scuiller, "Simulation of an energy storage system to compensate pulsed loads on shipboard electric power system," *2011 IEEE Electric Ship Technologies Symposium*, 2011.

[66] H. Smolleck, S. Ranade, N. Prasad, and R. Velasco, "Effects of pulsed-power loads upon an electric power grid," *IEEE Transactions on Power Delivery*, vol. 6, no. 4, p. 1629–1640, 1991.

[67] B. Cassimere, C. Valdez, S. Sudhoff, S. Pekarek, B. Kuhn, D. Delisle, and E. Zivi, "System impact of pulsed power loads on a laboratory scale integrated fight through power (iftp) system," *IEEE Electric Ship Technologies Symposium, 2005.*, 2005.

[68] J. M. Crider and S. D. Sudhoff, "Reducing impact of pulsed power loads on microgrid power systems," *IEEE Transactions on Smart Grid*, vol. 1, no. 3, p. 270–277, 2010.

[69] W.-S. Im, C. Wang, L. Tan, W. Liu, and L. Liu, "Cooperative controls for pulsed power load accommodation in a shipboard power system," *IEEE Transactions on Power Systems*, vol. 31, no. 6, p. 5181–5189, 2016.

[70] L. Farrier, C. Savage, and R. Bucknall, "Simulating pulsed power load compensation using lithium-ion battery systems," in *2019 IEEE Electric Ship Technologies Symposium (ESTS)*, pp. 45–51, IEEE, 2019.

[71] F. Li, Y. Chen, R. Xie, C. Shen, L. Zhang, and B. Qin, "Optimal operation planning for orchestrating multiple pulsed loads with transient stability constraints in isolated power systems," *IEEE Access*, vol. 6, p. 18685–18693, 2018.

[72] T. Ding, J. Bai, P. Du, B. Qin, F. Li, J. Ma, and Z. Dong, "Rectangle packing problem for battery charging dispatch considering uninterrupted discrete charging rate," *IEEE Transactions on Power Systems*, vol. 34, no. 3, pp. 2472–2475, 2019.

[73] R. Xie, Y. Chen, Z. Wang, S. Mei, and F. Li, "Online periodic coordination of multiple pulsed loads on all-electric ships," *IEEE Transactions on Power Systems*, vol. 35, no. 4, pp. 2658–2669, 2020.

[74] J. F. Hansen and F. Wendt, "History and state of the art in commercial electric ship propulsion, integrated power systems, and future trends," *Proceedings of the IEEE*, vol. 103, no. 12, pp. 2229–2242, 2015.

[75] M. Ibrahim, S. Jemei, G. Wimmer, and D. Hissel, “Nonlinear autoregressive neural network in an energy management strategy for battery/ultra-capacitor hybrid electrical vehicles,” *Electric Power Systems Research*, vol. 136, pp. 262–269, 2016.

[76] J. Li, B. Xia, X. Geng, H. Ming, S. Shakkottai, V. Subramanian, and L. Xie, “Mean field games in nudge systems for societal networks,” *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 3, Aug. 2018.

[77] H. Chen, Y. Zhang, M. C. Caramanis, and A. K. Coskun, “Energyqare: Qos-aware data center participation in smart grid regulation service reserve provision,” *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 4, Jan. 2019.

[78] S. K. Mandal, U. Y. Ogras, J. Rao Doppa, R. Z. Ayoub, M. Kishinevsky, and P. P. Pande, “Online adaptive learning for runtime resource management of heterogeneous socs,” in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2020.

[79] Z. Dong, X. Cong, Z. Xiao, X. Zheng, and N. Tai, “A study of hybrid energy storage system to suppress power fluctuations of pulse load in shipboard power system,” in *2020 International Conference on Smart Grids and Energy Systems (SGES)*, pp. 437–441, 2020.

[80] R. Hebner, J. Beno, and A. Walls, “Flywheel batteries come around again,” *IEEE Spectrum*, vol. 39, no. 4, pp. 46–51, 2002.

[81] A. T. Elsayed and O. A. Mohammed, “Distributed flywheel energy storage systems for mitigating the effects of pulsed loads,” in *2014 IEEE PES General Meeting | Conference Exposition*, pp. 1–5, 2014.