EPFL

Thèse n° 9958

Beyond worst-case analysis, with or without predictions

Présentée le 15 septembre 2023

Faculté informatique et communications
Laboratoire de théorie des communications
Programme doctoral en informatique et communications

pour l'obtention du grade de Docteur ès Sciences

par

Andreas MAGGIORI

Acceptée sur proposition du jury

Prof. E. Telatar, président du jury
Prof. R. Urbanke, Prof. O. N. A. Svensson, directeurs de thèse
Prof. S. Banerjee, rapporteur
Dr S. Vassilvitskii, rapporteur
Prof. M. Gastpar, rapporteur

École
polytechnique
fédérale
de Lausanne

2023

Remembering that you are going to die is the best way I know
to avoid the trap of thinking you have something to lose.
You are already naked. There is no reason not to follow your heart.
— Steve Jobs

Στη Ζωζώ, για μια ζωή προσφοράς χωρίς να ζητήσει ποτέ τίποτα πίσω...

Acknowledgements

I would like to thank the members of my jury. The private defense took place in the beginning of August and while walking around Lausanne the Saturday before, I realized that we were probably the only ones there. Thus, thanks for accepting even if the date was more of a vacation date than an exam date and thanks for the interesting discussion we had during my defense.

A big thank you goes to my advisors, Rüdiger and Ola. They gave me freedom to choose the problems I wanted to work on and guided me whenever I asked for help. Their advising style is very different: Ola is energetic and fast - very fast. Working with him is something between a game and a competition to which I still lose but I am getting better at... Rüdiger listens silently, he asks few or no questions and usually understands more than everyone in the room.

I would like to thank all my collaborators and friends during these years at EPFL, Google and Berkeley. Exposition to different schools of thought is one of the very few ways I know to really advance both as a researcher and as a human being.

It would not be true if I said that all these years were fun and games (thank you COVID). The times were really challenging from all points of view...I would like to thank all my friends in Greece for the support. With most of them, we are together almost 20 years now; same school, same neighborhood, same square. If we were a paper we would already have gotten the test-of-time award (sorry for the joke!).

A special thanks goes to my family: my aunts and uncles, to pillo, to the marmotto and to papà. Having a failing pillow "no matter what, we love you", is what lets you fail until you succeed.

Last but not least, a paragraph goes to Alexandra. She is the real hero of the story. More than ten years, she still bears with me and I hope she will continue to do so for many more years to come.

Loutraki, 18 August 2023 Andreas Maggiori

Abstract

In this thesis we design online combinatorial optimization algorithms for beyond worst-case analysis settings.

In the first part, we discuss the online matching problem and prove that, in the edge arrival model, no online algorithm can achieve a competitive ratio better than $1/2 + c$ for any constant $c > 0$. This result serves as an illustrative example to showcase the limitations of worst-case analysis.

The second and third parts introduce the concept of learning-augmented algorithms, which leverage predictions about the input to enhance their performance. The learning-augmented algorithms developed in those parts exhibit improved performance when predictions are accurate, while also demonstrating robustness even in the presence of misleading predictions.

In the second part, we investigate the online speed scheduling problem for energy minimization. We design an algorithm that incorporates predictions about future requests in a black-box manner and surpasses known lower-bounds of classical online algorithms when the predictions are accurate, while still maintaining robustness when predictions are incorrect.

The third part extends the Primal-Dual method from the classical online algorithms setting to the learning-augmented setting. We apply this technique to various problems, including online set cover, ski rental, TCP acknowledgment, and Bahncard.

Finally, in the fourth part, we delve into the correlation clustering problem in the online with recourse model. While the classical online setting is too restrictive and strong impossibility results make the problem uninteresting, in the recourse model we present an algorithm that achieves a worst-case logarithmic recourse with constant competitive ratio.

Keywords: online algorithms, matching, competitive analysis, learning-augmented, primal-dual, scheduling, clustering

Riassunto

In questa tesi progettiamo algoritmi in linea di ottimizzazione combinatoria per contesti di analisi oltre il caso peggiore.

Nella prima parte, discutiamo il problema dell'accoppiamento in linea e dimostriamo che, nel modello di arrivo degli archi, nessun algoritmo in linea può ottenere un rapporto competitivo migliore di $1/2 + c$ per qualsiasi costante $c > 0$. Questo risultato serve come esempio illustrativo per mostrare le limitazioni dell'analisi nel caso peggiore.

La seconda e terza parte introducono il concetto di algoritmi potenziati dall'apprendimento, che sfruttano previsioni sui dati in ingresso per migliorare le loro prestazioni. Gli algoritmi potenziati dall'apprendimento sviluppati in queste parti mostrano prestazioni migliorate quando le previsioni sono accurate, dimostrando allo stesso tempo robustezza anche in presenza di previsioni fuorvianti.

Nella seconda parte, indaghiamo il problema della pianificazione in linea con un processore a velocità variabile per la minimizzazione dell'energia. Progettiamo un algoritmo che incorpora previsioni sulle future richieste senza alcuna ulteriore supposizione e supera i limiti inferiori noti degli algoritmi in linea classici quando le previsioni sono accurate, mantenendo comunque la robustezza quando le previsioni sono errate.

La terza parte estende il metodo Primal-Dual dal contesto degli algoritmi in linea classici al contesto potenziato dall'apprendimento. Applichiamo questa tecnica a vari problemi, tra cui la copertura degli insiemi in linea, il problema del noleggio degli sci, la conferma TCP e il problema della Bahncard.

Infine, nella quarta parte, approfondiamo il problema del raggrupamento di correlazione nel modello in linea con ripensamento. Mentre il contesto in linea classico è troppo restrittivo e dei risultati di impossibilità rendono il problema poco interessante, nel modello con ripensamento presentiamo un algoritmo che raggiunge un ripensamento logaritmico nel caso peggiore con un rapporto competitivo costante.

Parole chiave: algoritmi in linea, accoppiamento, analisi competitiva, potenziamento tramite l'apprendimento, primal-dual, pianificazione, raggruppamento

Contents

Acknowledgements i

Abstract (English/Italiano) iii

1 Introduction 1

2 Preliminaries 5
2.1 Online algorithms . . . 5
2.2 Learning-augmented online algorithms . . . 6
2.3 Problem definitions . . . 6

I Online matching under the edge arrival model 9

3 Overview and related work 11
3.1 Introduction . . . 11
3.2 Notation and problem definition . . . 11
3.3 Prior work and our results . . . 12

4 Counterexample 13

Introduction to learning-augmented algorithms 17

II Learning-augmented energy minimization via speed scaling 19

5 Overview and related work 21
5.1 Introduction . . . 21
5.2 Model and preliminaries . . . 21
5.3 Prior work and our results . . . 23

6 Main algorithm 25
6.1 A consistent and smooth algorithm . . . 25
6.2 Robustification . . . 26
6.3 Summary of the algorithm . . . 29

7 Experiments 31

8 Extension to evolving predictors 35

9 Further extensions 39

III The Primal-Dual method for learning-augmented algorithms 41

10 Overview and related work 43
10.1 The classical Primal-Dual method . . . 43
10.2 Our contributions . . . 43

11 Online set cover with predictions 45

12 PDLA in action 51
12.1 The ski rental problem . . . 51
12.2 Dynamic TCP acknowledgement . . . 56
12.3 The Bahncard problem . . . 60

13 Experiments 67

IV Online and consistent correlation clustering 71

14 Overview and related work 73
14.1 Introduction . . . 73
14.2 Our contributions . . . 74
14.3 Problem definition . . . 74

15 Main algorithm 77
15.1 The Agreement algorithm . . . 77
15.2 Online Agreement algorithm . . . 77
15.3 Proof sketch . . . 79
15.4 Lower bound . . . 82

16 Main proof 85
16.1 Properties of the Agreement algorithm . . . 85
16.2 Dynamic analysis of the clustering sequence $\widetilde{\mathcal{C}}_1, \widetilde{\mathcal{C}}_2, \ldots$ . . . 88
16.3 Bounding the competitive ratio . . . 90
16.4 Bounding the worst-case recourse . . . 95

17 Experiments 101
17.1 Datasets . . . 101
17.2 Baselines . . . 101
17.3 Setup . . . 102
17.4 Results . . . 102

18 Conclusion 107

A Extensions and deferred proofs of Part II 109
A.1 Pure online algorithms for uniform deadlines . . . 109
A.2 Impossibility results for learning-augmented speed scaling . . . 111
A.3 A shrinking lemma . . . 114
A.4 Making an algorithm noise tolerant . . . 117
A.4.1 Noise tolerant measure of error . . . 117
A.4.2 Noise tolerant procedure . . . 118

A.5 ROBUSTIFY for general deadlines 125

B Deferred proofs of Part III **131**

Bibliography **143**

Curriculum Vitae **145**

1 Introduction

In this thesis we focus on beyond worst-case analysis frameworks to design algorithms for combinatorial optimization problems where the input is revealed in an online manner and the algorithm has to take irrevocable decisions every time a new part of the input is revealed. In the following we give an overview of the problems which are tackled in this thesis and present our main contributions.

The first part of the thesis concentrates on the classical online matching problem and serves as a motivating example on why we should study beyond worst-case analysis frameworks. The matching problem is a classical problem in combinatorial optimization where one is given a graph and the goal is to find a maximum matching, i.e., a maximum cardinality set of edges that share no endpoint. The online maximum matching problem was introduced by Karp, Vazirani and Vazirani in their seminal paper [61]. In their model the input is a bipartite graph $G = (V, U, E)$ where V is known in advance, in every round i a new node $u_i \in U$ arrives revealing at the same time all the edges incident to it and the algorithm has to decide either to add one of these edges to the existing matching or discard them forever. The goal is to maintain a matching as large as possible. To measure the quality of an algorithm we use the notion of competitive ratio, i.e., the worst case ratio of the (expected) cost of our algorithm's solution divided by the (expected) cost of the optimal solution. Karp, Vazirani and Vazirani in their one-sided node arrival model designed the famous $(1 - 1/e)-$competitive *Ranking* algorithm. While in the one-sided node arrival model *Ranking* is optimal it has been a long-standing question whether one can achieve a competitive ratio better than $1/2$ for more general arrival models. We note that a competitive ratio of $1/2$ can be achieved by the trivial *Greedy* algorithm which adds a newly revealed edge to the existing matching whenever that is possible. For the less restrictive edge arrival model, i.e., in every round a new edge is revealed and the algorithm has to decide either to add it to the current matching or discard it forever we prove the following:

> No online algorithm has a competitive ratio of $1/2 + c$ for any constant $c > 0$

Thus, in the edge arrival model, at least in terms of competitive ratio, *Greedy* is the best we can hope for. The results of the first part appeared in FOCS 2019 [44] and are based on a joint work with Buddhima Gamlath, Michael Kapralov, Ola Svensson and David Wajc.

In the second and third part of the thesis, we deviate from the classical online algorithms setting and consider a model where the performance is still measured in terms of competitive ratio but we are also given a prediction regarding the true instance. At a high level the goal is to maintain a solution whose cost is close to the optimum offline if predictions are accurate and at the same

time when predictions are misleading to output a solution which is not much worse than what the best online algorithm with no access to predictions would have achieved. That setting has been first formalized by Lykouris and Vassilvitskii [72] (see also [73] for an earlier similar setting) and it is often referred to as *learning-augmented algorithms* or *algorithms with predictions*. The competitive ratio that a learning-augmented algorithm achieves when provided with a perfectly accurate prediction is called *consistency* while the competitive ratio which is achieved no matter the quality of the prediction is referred to as *robustness*.

In the second part we focus on the learning-augmented problem of online speed scheduling for energy minimization. In real-world systems, e.g., cloud applications [14, 62], computational power and resources are scaled dynamically to serve new request loads and at the same time minimize costs. Motivated by the latter scenario and the fact that requests usually follow patterns, we studied the classical problem of energy minimization speed scaling [88] in the learning-augmented setting. In the classical version of the problem, at every new point in time t, w_t requests arrive and we are required to serve them all within some fixed amount of time. In order to satisfy all the requests in time, the server can dynamically change its processing speed $s(t)$ at the price of increasing its power consumption which is usually modeled as a super-linear function of the speed, i.e., $s(t)^\alpha$. In the learning-augmented setting we consider the scenario where all jobs need to be served within the same amount of time T from their arrival and we have a prediction about future loads, i.e, $\tilde{w}_1, \tilde{w}_2, \ldots$. We design an algorithm with the following guarantees:

> For any given $\epsilon \in (0, 1)$, our algorithm is $(1 + \epsilon)$-consistent and $O\left(\frac{\alpha}{\epsilon}\right)^\alpha$-robust

The results of the second part appeared in NeurIPS 2020 [8] as a spotlight presentation and are based on a joint work with Etienne Bamas, Lars Rohwedder and Ola Svensson.

In the third part of the thesis we develop general techniques to design learning-augmented online algorithms. Indeed we extend the Primal-Dual method [17], which is a powerful technique used to design online algorithms, into the learning-augmented setting. Using our Primal-Dual learning-augmented (PDLA) method we design algorithms which incorporate predictions for problems like the online set cover problem, ski rental, TCP acknowledgment and Bahncard. The main technical contribution in this part is to show how to translate the advice provided by a predictor into rates of change of primal and dual variables. The algorithms designed using our technique overcome known lower bounds of the classical online algorithms literature when the prediction is accurate while maintaining robustness even when the prediction is misleading. The main conceptual contribution of this part is:

> We extend the Primal-Dual method into the learning-augmented setting for solving problems that can be formulated as covering problems

The results of the third part appeared in NeurIPS 2020 [9] as an oral presentation and are based on a joint work with Etienne Bamas and Ola Svensson.

In the fourth part of the thesis we consider the minimization objective of the online correlation clustering problem. In the offline setting the input is a signed graph where each edge $e = (u, v)$ has either a $'+'$ sign or a $'-'$ sign denoting respectively the similarity or dissimilarity of u and v. The goal is to partition the set of nodes so as to minimize the number of edges with a $'-'$ sign inside the same partition and the number of edges with a $'+'$ sign across different partitions. In

the online setting we consider the node arrival model where each time a new node arrives all its adjacent edges to previously arrived nodes are revealed. Upon the arrival of a node we need to irrevocably decide if that node will join one of the already formed clusters or if it will start a cluster on its own. Unfortunately, in the classical worst-case analysis setting Mathieu et al. [75] proved that there is no online algorithm with a competitive ratio better that $O(n)$, where n is the total number of nodes. Consequently, we turn our attention to the less restrictive model of online correlation clustering with recourse. In the latter model decisions are no longer irrevocable but come at a large cost, thus the goal is to have an algorithm with a constant competitive ratio while minimizing the maximum number of cluster re-assignments of a node u, i.e., minimizing the recourse of node u. We design an algorithm with the following guarantees:

In our algorithm the recourse of every node is $O(\log n)$ and its competitive ratio is $\Theta(1)$

The results of the fourth part appeared in ICML 2022 [29] and are based on a joint work with Vincent Cohen-Addad, Silvio Lattanzi and Nikos Parotsidis.

Outline of the thesis: In Chapter 2 we introduce common notation of the different problems that this thesis addresses and define the main computational problem that each chapter considers. We particularly focus on introducing the setting of online learning-augmented algorithms and the main desiderata of algorithms in that setting. In Part I we discuss related work on the online matching problem, present the graph construction and hardness proof on getting an algorithm with competitive ratio larger that $1/2 + c$ for any constant $c > 0$ and finally argue why worst-case analysis is sometimes too restrictive and unrealistic. In the beginning of Part II, that is Chapter 5, we discuss related work on classical energy minimization scheduling and learning-augmented scheduling. Subsequently in Chapter 6 we present our main algorithm for the problem of learning-augmented energy minimization scheduling via speed scaling and prove its properties. In Chapter 7 we experimentally validate our claims and finally in Chapter 8 and Chapter 9 we extend our techniques to more general versions of the problem. Part III is devoted to the Primal-Dual learning-augmented technique (PDLA). In Chapter 10 we introduce the reader to the Primal-Dual method when used in the classical online setting and discuss related work. Subsequently in Chapter 11 we apply PDLA to the fractional version of the online set cover problem. In Chapter 12 we apply our technique to the learning-augmented versions of the ski rental problem, where we recover the results of [80], the Bahncard problem which is a generalization of the ski rental problem, and the TCP acknowledgment problem. In Chapter 13 we conduct experiments to strengthen our claims. Part IV is divided in four chapters: Chapter 14 where we discuss related work on the correlation clustering problem, Chapter 15 where we present our main algorithm along with a proof sketch of its guarantees, Chapter 16 where we present the full proof of its guarantees, and Chapter 17 where we conduct experiments on both artificial and real-world datasets. In the final chapter of this thesis, Chapter 18, we present some interesting open directions in the area of beyond worst-case analysis.

2 Preliminaries

This chapter is devoted to introduce the classical and learning-augmented online algorithms settings and formally define the problems considered in this thesis. While the concepts and properties of online algorithms will not be reiterated, the formal definition of each problem will be restated in the corresponding chapter. Moreover, when describing the properties of classical and learning-augmented online algorithms, the notation is designed to be as abstract and general as possible. However, when focusing on a specific problem, the notation is simplified to ensure clarity and avoid unnecessary complexity.

We use $G = (V, E)$ to denote a graph where V denotes the set of nodes and E the set of edges. $\mathbb{N} = \{1, 2, \dots\}$ and $\mathbb{Z} = \{0, -1, 1, -2, 2, \dots\}$ denote the set of natural and integer numbers respectively. In addition we denote by $\mathbb{Z}_{\geq 0} = \{0, 1, 2, \dots\}$ the set containing all positive integers and 0.

2.1 Online algorithms

In the classical online algorithms setting the input is revealed in an online fashion and the algorithm is required to take irrevocable decisions without any assumption about the future. In order to measure performance we will use the competitive analysis framework. To facilitate description we denote by $\text{cost}_{\mathcal{ALG}}(\mathcal{I})$ the cost incurred by algorithm $\mathcal{ALG}$ on the online instance $\mathcal{I}$ and by $\text{OPT}(\mathcal{I})$ the cost of the offline optimum. For minimization problems we say that $\mathcal{ALG}$ has a *competitive ratio* of c, or equivalently is *c-competitive* if $\text{cost}_{\mathcal{ALG}}(\mathcal{I}) \leq c \cdot \text{OPT}(\mathcal{I})$ (2.1) for any input $\mathcal{I}$. We note that in the literature a *c-competitive* algorithm is sometimes defined using the equation $\text{cost}_{\mathcal{ALG}}(\mathcal{I}) \leq c \cdot \text{OPT}(\mathcal{I}) + \beta$, $\forall \mathcal{I}$ for an arbitrary but constant β. The results of this thesis do not change qualitatively using the latter definition, and for simplicity, we omit the additive constant. Along the same lines for maximization problems we say that $\mathcal{ALG}$ is *c-competitive* if $\text{cost}_{\mathcal{ALG}}(\mathcal{I}) \geq c \cdot \text{OPT}(\mathcal{I})$ for any input $\mathcal{I}$. We underline that $\mathcal{ALG}$ could be either deterministic or randomized, where in the latter case $\text{cost}_{\mathcal{ALG}}(\mathcal{I})$ denotes the expected cost over $\mathcal{ALG}$'s internal randomness.

As it is standard in the online algorithms literature on minimization/maximization problems, we use lower/upper bounds (on c) to refer to hardness results, and upper/lower bounds to refer to positive results.

2.2 Learning-augmented online algorithms

Although appealing, the *no-assumption* regime of classical online algorithms comes at a high cost: Because the algorithm has to be overly prudent and prepare for all possible future events, i.e., Equation (2.1) needs to be true for all $\mathcal{I}$, the guarantees are often poor. Due to the success story of machine learning (ML), learning-augmented algorithms suggest incorporating the predictions provided by ML algorithms in the design of online algorithms and obtain better guarantees. An obvious caveat of this approach is that ML predictors often come with no worst-case guarantees and so we would like our algorithm to be robust to misleading predictions without making any assumption on their quality. Following the terminology defined in [72, 80] we present the main desirable properties that a learning-augmented algorithm should have (for simplicity we focus on minimization problems). We denote by $\mathcal{I}^{pred}$ the prediction of the true instance $\mathcal{I}$. Since $\mathcal{I}^{pred}$ is a source of unreliable information the learning-augmented algorithms are parametrized by a robustness parameter $\epsilon \in (0, 1)$, the smaller ϵ is the more we trust the prediction. Thus, the cost of a learning-augmented algorithm $\mathcal{ALG}$ on instance $\mathcal{I}$ with prediction $\mathcal{I}^{pred}$ and robustness parameter ϵ is denoted by $\text{cost}_{\mathcal{ALG}}(\mathcal{I}, \mathcal{I}^{pred}, \epsilon)$. The main properties that an online learning-augmented algorithm should have are the following:

- **Consistency:** If the prediction is perfectly accurate, i.e., $\mathcal{I}^{pred} = \mathcal{I}$, then the provable guarantees should be better than what a pure online algorithm can achieve. Ideally, the algorithm produces an offline optimal solution or comes close to it. By close to optimal, we mean that the cost of the algorithm (when the prediction is perfectly accurate) should be at most $C(\epsilon) \cdot \text{OPT}(\mathcal{I})$, where $C(\epsilon)$ tends to 1 as ϵ approaches 0.
- **Robustness:** The competitive ratio of the algorithm should always be bounded even for arbitrarily bad (adversarial) predictions. Ideally, the competitive ratio is somewhat comparable to the competitive ratio of algorithms from literature of the pure online case. Formally, the cost of the algorithm should always be bounded by $R(\epsilon) \cdot \text{OPT}(\mathcal{I})$ for some function $R(\epsilon)$.
- **Smoothness:** A perfect prediction is a strong requirement. The consistency property should transition smoothly for all ranges of errors, that is, the algorithm's guarantees deteriorate smoothly as the prediction error, denoted by $\text{err}(\mathcal{I}, \mathcal{I}^{pred})$, increases. Formally, the cost of the algorithm should always be at most $C(\epsilon) \cdot \text{OPT}(\mathcal{I}) + F(\epsilon, \text{err}(\mathcal{I}, \mathcal{I}^{pred}))$ for some function F such that $F(\epsilon, 0) = 0$ for any ϵ.

2.3 Problem definitions

Online matching under the edge arrival model: In the matching problem we are given a graph $G = (V, E)$ and the goal is to find a maximum cardinality subset of the edges $M \subset E$ such that no two edges in M share a node. In the online maximum matching problem, the final graph is unknown, and under the edge arrival model, edges are revealed one by one. An online matching algorithm must decide immediately and irrevocably whether to match an edge on arrival, or whether to leave both endpoints free to be possibly matched later.

Learning-augmented energy minimization via speed scaling: We proceed defining the main algorithmic problem addressed in Part II, the Uniform Speed Scaling problem with

predictions. An instance of the problem can be formally described as a triple $\mathcal{I} = (w, D, T)$ where $[0, T]$ is a finite time horizon, each time $i \in \{0, \dots, T - D\}$ jobs with a total workload $w_i \in \mathbb{Z}_{\geq 0}$ arrive, which have to be completed by time $i + D$. To do so, we can adjust the speed $s_i(t)$ at which each workload w_i is processed for $t \in [i, i + D]$. To finish each job on time, we require that the amount of work dedicated to job i in the interval $[i, i + D]$ should be w_i. In other words, $\int_i^{i+D} s_i(t)\ dt = w_i$. The overall speed of our processing unit at time t is the sum $s(t) = \sum_i s_i(t)$, which yields an energy consumption of $\int s(t)^\alpha dt$, $\alpha > 1$, which we aim to minimize. While in the offline setting, the whole instance is known in advance, in the learning-augmented online setting only a prediction of the workload vector w^{pred} is given in advance and at each new time i, the i-th component of the true workload vector, w_i gets revealed. Consequently the predicted instance is $\mathcal{I}^{pred} = (w^{\text{pred}}, D, T)$ and we define the prediction error as $\text{err}(\mathcal{I}, \mathcal{I}^{pred}) = ||w - w^{\text{pred}}||_\alpha^\alpha = \sum_i |w_i - w_i^{\text{pred}}|^\alpha$. Since the prediction error only depends on the difference between the predicted and true workload vector we will denote it, for simplicity, as $\text{err}(w, w^{\text{pred}})$. The main goal of Part II is to design a consistent, robust and smooth algorithm under that prediction error.

The Primal-Dual method for learning-augmented algorithms: In Part III the focus is on extending the Primal-Dual method from the setting of classical online algorithms to the learning-augmented online setting. It is worth noting that since we do not focus on a specific problem, but rather on extending a technique and applying it to many different problems we avoid defining a prediction error, and consequently we focus only on designing consistent and robust learning-augmented algorithms. The algorithms we design in Part III of this thesis have guarantees of the form

$$\underset{\mathcal{ALG}}{\text{cost}}(\mathcal{I}, \mathcal{A}, \epsilon) \leq \min\{C(\epsilon) \cdot S(\mathcal{A}, \mathcal{I}), R(\epsilon) \cdot \text{OPT}(\mathcal{I})\}$$

where $S(\mathcal{A}, \mathcal{I})$ denotes the cost of the output solution on input $\mathcal{I}$ if the algorithm follows blindly the prediction $\mathcal{A}$. If $\mathcal{A}$ is accurate then $S(\mathcal{A}, \mathcal{I}) = \text{OPT}(\mathcal{I})$ and we recover the consistency bound of Section 2.2. Note that we change notation substituting $\mathcal{I}^{pred}$ with $\mathcal{A}$ to underline that the prediction might not be viewed as a prediction of the true instance $\mathcal{I}$ but rather as a form of advice.

Online and consistent correlation clustering: In the disagreements minimization version of the correlation clustering problem the input is a complete signed undirected graph $G = (V, E, s)$ where each edge $e = \{u, v\}$ is assigned a sign $s(e) \in \{+, -\}$ and the goal is to find a partition of the vertices such that the number of $'-'$ edges inside the same cluster and $'+'$ edges in between clusters is minimized. For simplicity we denote the set of $'+'$ and $'-'$ edges by E^+ and E^- respectively. It simplifies the notation to represent the solution computed by an algorithm $\mathcal{ALG}$ by an *assignment function* $f : V \to \mathbb{Z}$; Using that notation f induces a partition $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$ such that two nodes u, v belong to the same partition, i.e., cluster C_i, if and only if $f(u) = f(v)$, or in other words, they are *assigned* the same cluster id. The cost of a solution computed by $\mathcal{ALG}$, or equivalently f or the clustering induced by the latter is equal to:

$$\text{cost}(f) = \sum_{\substack{\{u,v\} \in E^+ \\ f(u) \neq f(v)}} 1 + \sum_{\substack{\{u,v\} \in E^- \\ f(u) = f(v)}} 1$$

In the online setting nodes arrive one at a time, revealing upon arrival all the edges to previously arrived nodes. An instance of the online correlation clustering problem can be described by a pair $\mathcal{I} = (G, \sigma)$ where G is the *final graph* and σ is an order on the vertices of G: $\sigma = \langle v_1, v_2, \ldots, v_{|V|} \rangle$. The solution of an algorithm $\mathcal{ALG}$ on an online instance $\mathcal{I}$ can be described as a sequence of assignment functions $f_1, f_2, \ldots, f_{|V|}$. The recourse of a node u, $r(u)$, that arrived at time t is the number of times the assignment function sequence changes the cluster id assigned to u. That is $r(u) = \sum_{t' > t} \mathbb{1}\{f_{t'-1}(u) \neq f_{t'}(u)\}$. The recourse of an algorithm is the worst case recourse over all instances $\mathcal{I}$ and nodes u. In the classical online setting, since decisions are irrevocable, the recourse of an algorithm should be 0, however it turns out that this requirement is too restrictive [75] and one cannot achieve a competitive ratio better than $\Theta(|V|)$ in the classical online setting. Thus, the goal of Part IV is to design a constant-factor approximation algorithm whose recourse is $O(\log(|V|))$, i.e., its clustering is as consistent as possible.

Part I

Online matching under the edge arrival model

The limits of worst-case analysis

3 Overview and related work

3.1 Introduction

Matching theory has played a prominent role in the area of combinatorial optimization, with many applications [71, 82]. Moreover, many fundamental techniques and concepts in combinatorial optimization can trace their origins to its study, including the Primal-Dual framework [64], proofs of polytopes' integrality beyond total unimodularity [37], and even the equation of efficiency with polytime computability [38].

Given the prominence of matching theory in combinatorial optimization, it comes as little surprise that the maximum matching problem was one of the first problems studied from the point of view of online algorithms and competitive analysis. In 1990, Karp, Vazirani, and Vazirani [61] introduced the online matching problem, and studied it under one-sided bipartite arrivals. For such arrivals, Karp et al. noted that the trivial $1/2$-competitive GREEDY algorithm (which matches any arriving vertex to an arbitrary unmatched neighbor, if one exists) is optimal among deterministic algorithms for this problem. More interestingly, they provided an elegant randomized online algorithm for this problem, called RANKING, which achieves an optimal $(1 - 1/e)$ competitive ratio. (This bound has been re-proven many times over the years [15, 32, 36, 40, 46].) Online matching and many extensions of this problem under one-sided bipartite vertex arrivals were widely studied over the years, both under adversarial and stochastic arrival models. See recent work [24, 53, 54, 55] and the excellent survey of Mehta [77] for further references on this rich literature.

Despite our increasingly better understanding of one-sided online bipartite matching and its extensions, the problem of online matching under more general arrival models such as edge arrivals has remained staunchly defiant, resisting attacks. In particular, the basic question of whether the trivial $1/2$ competitive ratio is optimal for the adversarial edge-arrival model has remained a tantalizing open question in the online algorithms literature. In this part of the thesis, we answer this question affirmatively.

3.2 Notation and problem definition

In the matching problem we are given a graph $G = (V, E)$ and the goal is to find a maximum cardinality subset of the edges $M \subset E$ such that no two edges in M share a node. Note that a matching can be described as a vector belonging to the matching polytope: $\mathcal{P}_{int} = \{\vec{x} \mid x_e \in \{0, 1\}\ \forall e \in E \text{ and } \sum_{e \ni v} x_e \leq 1\ \forall v \in V\}$. Since randomized algorithms induce a fractional

solution for the maximum matching problem we similarly define the fractional matching polytope as $\mathcal{P} = \{\vec{x} \geq \vec{0} \mid \sum_{e \ni v} x_e \leq 1 \; \forall v \in V\}$.

In the online maximum matching problem, the final graph is unknown, and under the edge-arrival model, edges are revealed one by one. An online matching algorithm must decide immediately and irrevocably whether to match the edge on arrival, or whether to leave both endpoints free to be possibly matched later.

3.3 Prior work and our results

On the hardness front, the problem under the edge-arrival model is known to be strictly harder than the one-sided vertex arrival model of Karp et al. [61], which admits a competitive ratio of $1 - 1/e \approx 0.632$. In particular, Epstein et al. [39] gave an upper bound of $\frac{1}{1+\ln 2} \approx 0.591$ for this problem, recently improved by Huang et al. [55] to $2 - \sqrt{2} \approx 0.585$. (Both bounds apply even to online algorithms with preemption, i.e., allowing edges to be removed from the matching in favor of a newly-arrived edge.) On the positive side, as pointed out by Buchbinder et al. [19], the edge-arrival model has been proven challenging, and results beating the $1/2$ competitive ratio were only achieved under various relaxations, including: random order edge-arrival [51], bounded number of arrival batches [68], on trees, either with or without preemption [19, 84], and for bounded-degree graphs [19]. The above papers all asked whether there exists a randomized $(1/2 + \Omega(1))$-competitive algorithm for adversarial edge arrivals (see also Open Question 17 in Mehta's survey [77]).

In this part of the thesis, we answer this open question, providing it with a strong negative answer. In particular, we show that no online algorithm for *fractional* matching, i.e., an algorithm which immediately and irrevocably assigns values x_e to edge e upon arrival such that $\vec{x}$ is in the fractional matching polytope $\mathcal{P}$ is $1/2 + \Omega(1)$ competitive. As any randomized algorithm induces a fractional algorithm with the same competitive ratio, this rules out any randomized online matching algorithm which is better than deterministic algorithms.

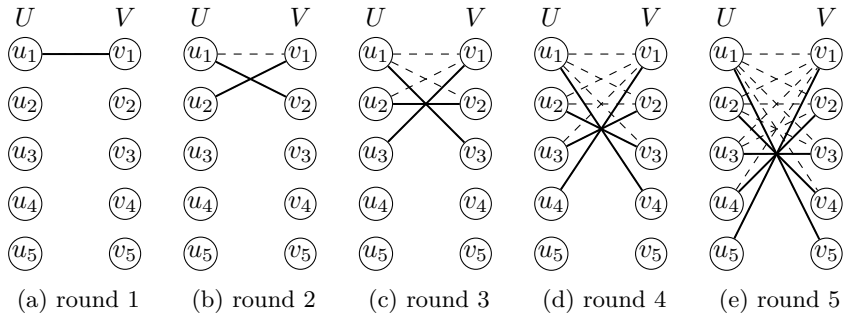
It is worth noting that in most prior upper bounds in the online literature [39, 40, 55, 61] the core difficulty of these hard instances is the uncertainty about the "identity" of vertices (in particular, which vertices will neighbor which vertices in the following arrivals). Our hardness instances rely on the uncertainty about the "time horizon". In particular, the underlying graph, vertex identifiers, and even arrival order are known to the algorithm, but the number of edges of the graph to be revealed (to arrive) is uncertain. Consequently, a c-competitive algorithm must accrue high enough value up to each arrival time to guarantee a high competitive ratio at all points in time. As we shall show, for competitive ratio $1/2 + \Omega(1)$, this goal is at odds with the fractional matching constraints, and so such a competitive ratio is impossible. In particular, we provide a family of hard instances and formulate their prefix-competitiveness and matching constraints as linear constraints to obtain a linear program whose objective value bounds the optimal competitive ratio. Solving the obtained LP's dual, we obtain by weak duality the claimed upper bound on the optimal competitive ratio. See [7, 25, 41] for more examples of the use of LP duality for proofs of hardness results for online problems, first advocated by [7].

4 Counterexample

In this chapter we prove the asymptotic optimality of the greedy algorithm for online matching under adversarial edge arrivals. As discussed briefly in Section 3.1, our main idea will be to provide a "prefix hardness" instance, where an underlying input and the arrival order is known to the online matching algorithm, but the prefix of the input to arrive (or "termination time") is not. Consequently, the algorithm must accrue high enough value up to each arrival time, to guarantee a high competitive ratio at all points in time. As we show, the fractional matching constraints rule out a competitive ratio of $1/2 + \Omega(1)$ even in this model where the underlying graph is known.

Theorem 4.1. *There exists an infinite family of bipartite graphs with maximum degree n and edge arrival order for which any online matching algorithm is at best $\left(\frac{1}{2} + \frac{1}{2n+2}\right)$-competitive.*

Proof. We will provide a family of graphs for which no fractional online matching algorithm has better competitive ratio. Since any randomized algorithm induces a fractional matching algorithm, this immediately implies our claim. The n^{th} graph of the family, $G_n = (U, V, E)$, consists of a bipartite graph with $|U| = |V| = n$ vertices on either side. We denote by $u_i \in U$ and $v_i \in V$ the i^{th} node on the left and right side of G_n, respectively. Edges are revealed in n discrete rounds. In round $i = 1, 2, \ldots, n$, the edges of a perfect matching between the first i left and right vertices arrive in some order, i.e., a matching of $u_1, u_2, \ldots, u_i$ and $v_1, v_2, \ldots, v_i$ is revealed. Specifically, edges (u_j, v_{i-j+1}) for all $i \geq j$ arrive. (See Figure 4.1 for example.) Intuitively, the difficulty for an algorithm attempting to assign high value to the edges of the optimum offline matching is that the (unique) maximum matching changes every round, and no edge ever re-enters an optimum matching of a previous round.



(a) round 1 (b) round 2 (c) round 3 (d) round 4 (e) round 5

Figure 4.1: G_5, together with arrival order. Edges of current (prior) round are solid (dashed).

Consider some c-competitive fractional algorithm $\mathcal{ALG}$. We call the edge of a vertex w in the (unique) maximum matching of the subgraph of G_n following round i the i^{th} edge of w. For $i \geq j$, denote by $x_{i,j}$ the value $\mathcal{ALG}$ assigns to the i^{th} edge of vertex u_j (and of v_{i-j+1}); i.e., to (u_j, v_{i-j+1}). By feasibility of the fractional matching output by $\mathcal{ALG}$, we immediately have that $x_{i,j} \geq 0$ for all i, j, as well as the following matching constraints for u_j and v_j. (For the latter, note that the i^{th} edge of v_{i-j+1} is assigned value $x_{i,j} = x_{i,i-(i-j+1)+1}$ and so the i^{th} edge of v_j is assigned value $x_{i,i-j+1}$).

$$\sum_{i=j}^{n} x_{i,j} \leq 1. \qquad (u_j \text{ matching constraint}) \tag{4.1}$$

$$\sum_{i=j}^{n} x_{i,i-j+1} \leq 1. \qquad (v_j \text{ matching constraint}) \tag{4.2}$$

On the other hand, as $\mathcal{ALG}$ is c-competitive, we have that after some k^{th} round – when the maximum matching has cardinality k – algorithm $\mathcal{ALG}$'s fractional matching must have value at least $c \cdot k$. (Else an adversary can stop the input after this round, leaving $\mathcal{ALG}$ with a worse than c-competitive matching.) Consequently, we have the following competitiveness constraints.

$$\sum_{i=1}^{k} \sum_{j=1}^{i} x_{i,j} \geq c \cdot k \qquad \forall k \in [n]. \tag{4.3}$$

Combining constraints (4.1), (4.2) and (4.3) together with the non-negativity of the $x_{i,j}$ yields the following linear program, LP(n), whose optimal value upper bounds any fractional online matching algorithm's competitiveness on G_n, by the above.

$$\begin{array}{rll} \text{maximize} & c & \\ \text{subject to:} & \sum_{i=j}^{n} x_{i,j} \leq 1 & \forall j \in [n] \\ & \sum_{i=j}^{n} x_{i,i-j+1} \leq 1 & \forall j \in [n] \\ & \sum_{i=1}^{k} \sum_{j=1}^{i} x_{i,j} \geq c \cdot k & \forall k \in [n] \\ & x_{i,j} \geq 0 & \forall i, j \in [n]. \end{array}$$

To bound the optimal value of LP(n), we provide a feasible solution to its LP dual, which we denote by Dual(n). By weak duality, any dual feasible solution's value upper bounds the optimal value of LP(n), which in turn upper bounds the optimal competitive ratio. Using the dual variables ℓ_j, r_j for the degree constraints of the j^{th} left and right vertices respectively (u_j and v_j) and dual variable α_k for the competitiveness constraint of the k^{th} round, we get the following dual linear program. Recall here again that $x_{i,i-j+1}$ appears in the matching constraint of v_j, with dual variable r_j, and so $x_{i,j} = x_{i,i-(i-j+1)+1}$ appears in the same constraint for v_{i-j+1}.)

$$\begin{array}{rll} \text{minimize} & \sum_{j=1}^{n} (\ell_j + r_j) & \\ \text{subject to:} & \sum_{k=1}^{n} k \cdot \alpha_k \geq 1 & \\ & \ell_j + r_{i-j+1} - \sum_{k=i}^{n} \alpha_k \geq 0 & \forall i \in [n], j \in [i] \\ & \ell_j, r_j, \alpha_k \geq 0 & \forall j, k \in [n]. \end{array}$$

We provide the following dual solution.

$$\alpha_k = \frac{2}{n(n+1)} \qquad \forall k \in [n]$$
$$\ell_j = r_j = \begin{cases} \frac{n-2(j-1)}{n(n+1)} & \text{if } j \leq n/2+1 \\ 0 & \text{if } n/2+1 < j \leq n. \end{cases}$$

We start by proving feasibility of this solution. The first constraint is satisfied with equality. For the second constraint, as $\sum_{k=i}^{n} \alpha_k = \frac{2(n-i+1)}{n(n+1)}$ it suffices to show that $\ell_j + r_{i-j+1} \geq \frac{2(n-i+1)}{n(n+1)}$ for all $i \in [n], j \in [i]$. Note that if $j > n/2+1$, then $\ell_j = r_j = 0 > \frac{n-2(j-1)}{n(n+1)}$. So, for all j we have $\ell_j = r_j \geq \frac{n-2(j-1)}{n(n+1)}$. Consequently, $\ell_j + r_{i-j+1} \geq \frac{n-2(j-1)}{n(n+1)} + \frac{n-2(i-j+1-1)}{n(n+1)} = \frac{2(n-i+1)}{n(n+1)}$ for all $i \in [n], j \in [i]$. Non-negativity of the ℓ_j, r_j, α_k variables is trivial, and so we conclude that the above is a feasible dual solution.

It remains to calculate this dual feasible solution's value. We do so for n even,[1] for which

$$\begin{aligned} \sum_{j=1}^{n} (\ell_j + r_j) &= 2 \cdot \sum_{j=1}^{n} \ell_j \\ &= 2 \cdot \sum_{j=1}^{n/2+1} \frac{n-2(j-1)}{n(n+1)} \\ &= \frac{1}{2} + \frac{1}{2n+2}, \end{aligned}$$

completing the proof. □

Remark Recall that Buchbinder et al. [19] and Lee and Singla [68] presented better-than-1/2-competitive algorithms for bounded-degree graphs and bounded number of arrival batches. Our upper bound above shows that a deterioration of the competitive guarantees as the maximum degree and number of arrival batches increase (as in the algorithms of [19, 68]) is inevitable.

Motivating beyond worst-case analysis The result of this chapter shows that the study of relaxed variants of online matching under edge arrivals [19, 51, 68, 84] is not only justified by the difficulty of beating the trivial bound for this problem, but rather by its impossibility. The classical worst-case analysis framework of online algorithms is sometimes overly pessimistic, indeed online algorithms in that setting need both to take irrevocable decisions and to prepare against all possible future events. In the rest of the thesis we concentrate on two different approaches to overcome the pessimistic nature of classical online algorithms: (1) in Part II and Part III we exploit the fact that for most problems past data is available and in most cases predictors help us prepare against future events; and (2) in Part IV we assume that decisions are no longer irrevocable but can be changed at a high cost.

[1] A similar bound and calculation for odd n holds. As it is unnecessary to establish the result of this theorem, we omit it.

Introduction to learning-augmented algorithms

The current chapter provides an overview of the nascent area of learning-augmented algorithms and serves as a shared introduction for both Part II and Part III of this thesis. As already mentioned in Chapter 2, classical online algorithms tend to be overly cautious which sometimes causes their performance in real-world situations to be far from what a machine learning (ML) algorithm would have achieved. Indeed in many practical applications future events follow patterns which are easily predictable using ML methods. In [72] Lykouris and Vassilvitskii formalized a general framework for incorporating (ML) predictions into online algorithms and designed an extension of the marking algorithm to solve the online caching problem when provided with predictions. While some related approaches were considered before (see e.g. Xu and Xu [87], Mahdian et al. [73] and Medina and Vassilvitskii [76]), the attention in this subject has increased substantially in the recent years and [72] was quickly followed by many other papers studying different learning-augmented online problems such as scheduling [66, 69], caching [5, 81], ski rental [47, 63, 80, 85], clustering [34] and other problems [52, 79]. The main challenge is to incorporate the prediction without knowing how the prediction was computed and in particular without making any assumption on the quality of the prediction. This setting is natural as in real-world situations, predictions are provided by ML algorithms that rarely come with worst-case guarantees on their accuracy. Thus, the difficulty in designing a learning-augmented algorithm is to find a good balance: on the one hand, following blindly the prediction might lead to a very bad solution if the prediction is misleading. On the other hand if the algorithm does not trust the prediction at all, it will simply never benefit from an excellent prediction.

Part II

Learning augmented energy minimization via speed scaling

Beyond worst-case analysis for energy minimization

5 Overview and related work

5.1 Introduction

The problem we are considering is motivated by the following scenario. Consider a server that receives requests in an online fashion. For each request some computational work has to be done and, as a measure of Quality-of-Service, we require that each request is answered within some fixed time. In order to satisfy all the requests in time the server can dynamically change its processor speed at any time. However, the power consumption can be a super-linear function of the processing speed (more precisely, we model the power consumption as s^α where s is the processing speed and $\alpha > 1$). Therefore, the problem of minimizing energy becomes non-trivial. This problem can be considered in the online model where the server has no information about the future tasks at all. However, this assumption seems unnecessarily restrictive as these requests tend to follow some patterns that can be predicted. For this reason a good algorithm should be able to incorporate some given predictions about the future. Similar scenarios appear in real-world systems as, for instance, in dynamic frequency scaling of CPUs or in autoscaling of cloud applications [14, 62]. In the case of autoscaling, ML advice is already being incorporated into online algorithms in practice [14]. However, on the theory side, while the above speed scaling problem was introduced by Yao et al. [88] in a seminal paper who studied it both in the online and offline settings (see also [12, 13]), it has not been considered in the learning-augmented setting.

5.2 Model and preliminaries

We define the Uniform Speed Scaling problem, a natural restricted version of the speed scaling problem [88], where predictions can be integrated naturally. While the restricted version is our main focus as it allows for cleaner exposition and prediction model, we also show that our techniques can be adapted to more complex algorithms yielding similar results for the general problem (see Chapter 9 for further extensions).

Problem definition. An instance of the problem can be formally described as a triple $\mathcal{I} = (w, D, T)$ where $[0, T]$ is a finite time horizon, each time $i \in \{0, \dots, T - D\}$ jobs with a total workload $w_i \in \mathbb{Z}_{\geq 0}$ arrive, which have to be completed by time $i + D$. To do so, we can adjust the speed $s_i(t)$ at which each workload w_i is processed for $t \in [i, i + D]$. Jobs may be processed in parallel. The overall speed of our processing unit at time t is the sum $s(t) = \sum_i s_i(t)$, which yields a power consumption of $s(t)^\alpha$, where $\alpha > 1$ is a problem specific constant. Since we want to finish each job on time, we require that the amount of work dedicated to job i in the interval

$[i, i+D]$ should be w_i. In other words, $\int_i^{i+D} s_i(t)\ dt = w_i$. In the offline setting, the whole instance is known in advance, i.e., the vector of workloads w is entirely accessible. In the online problem, at time i, the algorithm is only aware of all workloads w_j with $j \leq i$, i.e., the jobs that were released before time i. As noted by Bansal et al. [12], in the offline setting the problem can be formulated concisely as the following mathematical program:

Definition 5.1 (Uniform Speed Scaling problem)**.** On input $\mathcal{I} = (w, D, T)$ compute the optimal solution for

$$\min \int_0^T s(t)^\alpha\ dt \quad s.t.\ \forall i\ \int_i^{i+D} s_i(t)\ dt = w_i, \quad \forall t\ \sum_i s_i(t) = s(t), \quad \forall i \forall t\ s_i(t) \geq 0.$$

In contrast, we refer to the problem of Yao et al. [88] as the *General Speed Scaling* problem. The difference is that there the time that the processor is given to complete each job is not necessarily equal across jobs. More precisely, there we replace w and D by a set of jobs $J_j = (r_j, d_j, w_j)$, where r_j is the time the job becomes available, d_j is the deadline by which it must be completed, and w_j is the work to be completed. As a shorthand, we sometimes refer to these two problems as the *uniform deadlines* case and the *general deadlines* case. As mentioned before, Yao et al. [88] provide a simple optimal greedy algorithm that runs in polynomial time. As for the online setting, we emphasize that both the general and the uniform speed scaling problem are non-trivial. More specifically, we prove that no online algorithm can have a competitive ratio better than $\Omega((6/5)^\alpha)$ even in the uniform case (see Theorem A.1 in Appendix A.1).

In both problems the processor is allowed to run multiple jobs in parallel. However, we underline that restricting the problem to the case where the processor is only allowed to run at most one job at any given point in time is equivalent. Indeed, given a feasible solution $s(t) = \sum_i s_i(t)$ in the parallel setting, rescheduling jobs sequentially according to the earliest deadline first (EDF) policy creates a feasible solution of the same (energy) cost where at each point in time only one job is processed.

Prediction model and error measure. In the following, we present the model of prediction we are considering. Recall an instance of the problem is defined as a time horizon $[0, T]$, a duration D, and a vector of workloads w_i, $i = 1, \ldots, T - D$. A natural prediction is simply to give the algorithm a predicted instance $\mathcal{I}^{pred} = (w^{\text{pred}}, T, D)$ at time $t = 0$. From now on, we will refer to the ground truth work vector and predicted work vector as w^{real} and w^{pred} respectively. We define the error of the prediction as

$$\text{err}(\mathcal{I}^{real}, \mathcal{I}^{pred}) = ||w^{\text{real}} - w^{\text{pred}}||_\alpha^\alpha = \sum_i |w_i^{\text{real}} - w_i^{\text{pred}}|^\alpha.$$

We simply write $\text{err}(w^{\text{real}}, w^{\text{pred}})$ or err, when D, T, w^{real} and w^{pred} are clear from the context. The motivation for using α in the definition of err and not some other constant p comes from strong impossibility results. Clearly, guarantees for higher values p are weaker than for lower p. Therefore, we would like to set p as low as possible. However, we show that p needs to be at least α in order to make a sensible use of a prediction (see Theorem A.5 in Appendix A.2).

In the following, depending of which part of the input D, T, w is clear from the context we will use either $\text{OPT}(D, T, w)$ or $\text{OPT}(w)$ or simply OPT to denote the energy cost of the optimal

offline schedule.

As previously discussed in Chapter 2, learning-augmented algorithms receive as input a robustness parameter $\epsilon \in (0, 1)$ which denotes the trust towards the prediction. While for some problems (see [72]) the robustness and consistency functions may not depend on ϵ, in our problems we prove that such a dependency is unavoidable. Indeed, no algorithm can be perfectly consistent and robust at the same time (see Theorem A.4 in Appendix A.2). We also note that the properties of our algorithms are dependent on the problem parameter α and in Theorem A.1 we prove that this dependency is necessary.

5.3 Prior work and our results

On the one hand, learning-augmented algorithms is a relatively new research area [47, 52, 63, 66, 69, 72, 76, 80]. On the other hand, the speed scaling problem proposed by Yao et al. in [88] is well understood in both the offline and online setting. In its full generality, a set of tasks each with different arrival times, deadlines, and workloads needs to be completed in time while the speed is scaled in order to minimize energy. In the offline setting Yao et al. proved that the problem can be solved in polynomial time by a greedy algorithm. In the online setting, in which the jobs are revealed only at their release time, Yao et al. designed two different algorithms: (1) the Average Rate heuristic (AVR), for which they proved a bound of $2^{\alpha-1}\alpha^{\alpha}$ on the competitive ratio. This analysis was later proved to be asymptotically tight by Bansal et al. [13]. (2) The Optimal Available heuristic (OA), which was shown to be α^{α}-competitive in [12]. In the same paper, Bansal et al. proposed a third online algorithm named BKP for which they proved a competitive ratio asymptotically equivalent to e^{α}. While in these competitive ratios the exponential in α might not seem satisfying, Bansal et al. also proved that the exponential dependency cannot be better than e^{α}. A number of variants of the problem have also been considered in the offline setting (no preemption allowed, precedence constraints, nested jobs and more listed in a recent survey by Gerards et al. [45]) and under a stochastic optimization point of view (see for instance [4]). It is important to note that, while in theory the problem is interesting in the general case, i.e., when α is an input parameter, in practice we usually focus on small values of α such as 2 or 3 since they model certain physical laws (see e.g. Bansal et al. [12]). Although the BKP algorithm provides the best asymptotic guarantee, OA or AVR often lead to better solutions for small α and therefore remain relevant.

We initiate the study of the online speed scaling problem in the learning-augmented setting. We formalize an intuitive and well-founded prediction model for the uniform speed scaling problem. We show that our problem is non-trivial both in the learning-augmented and in the classical online setting. In the latter the AVR algorithm was proved to be $2^{\alpha-1} \cdot \alpha^{\alpha}$-competitive by Yao et al. [88] with a quite technical proof; we show, with a simple proof, that AVR is in fact 2^{α}-competitive in the uniform deadlines case and we provide an almost matching lower bound on the competitive ratio (see Theorem A.2 and Theorem A.3 in Appendix A.1). In the learning-augmented setting we provide an unconditional lower bound that demonstrates: An algorithm cannot be optimal, if the prediction is correct, and at the same time retain robustness. We then focus on our main contribution which is the design and analysis of a simple and efficient algorithm which incorporates any ML predictor as a black-box without making any further assumption. We achieve this in a modular way: First, we show that there is a consistent (but not robust) online algorithm. Then we develop a technique to make any online algorithm (which may use the prediction) robust at

a small cost. In addition to the theoretical analysis, we also provide an experimental analysis that supports our claims on both synthetic and real datasets. We further note that it may seem natural to consider a predictor that is able to renew its prediction over time, e.g., by providing our algorithm a new prediction at every integral time i. To this end, in Chapter 8, we show how to naturally extend all our results from the single prediction to the evolving prediction model.

Moreover, in Appendix A.4 we design general methods to allow algorithms to cope with small perturbations in the prediction.

Although in the main part of this thesis we focus on a restricted case of the speed scaling problem by Yao et al. [88], where predictions can be integrated naturally. In Appendix A.5 we show that with more sophisticated algorithms our techniques extend well to the general case.

6 Main algorithm

In this chapter we develop two modular building blocks to obtain a consistent, smooth, and robust algorithm. The first block is an algorithm which computes a schedule online taking into account the prediction for the future. This algorithm is consistent and smooth, but not robust. Then we describe a generic method on how to robustify an arbitrary online algorithm at a small cost.

6.1 A consistent and smooth algorithm

In the following we describe a learning-augmented online algorithm, which we call LAS-Trust.

Preparation. We compute an optimal schedule s^{pred} for the predicted jobs. An optimal schedule can always be normalized such that each workload w_i^{pred} is completely scheduled in an interval $[a_i, b_i]$ at a uniform speed c_i, that is,

$$s_i^{\text{pred}}(t) = \begin{cases} c_i & \text{if } t \in [a_i, b_i], \\ 0 & \text{otherwise.} \end{cases}$$

Furthermore, the intervals $[a_i, b_i]$ are non-overlapping. For details we refer the reader to the optimal offline algorithm by Yao et al. [88], which always creates such a schedule.

The online algorithm. At time i we first schedule w_i^{real} at uniform speed in $[a_i, b_i]$, but we cap the speed at c_i. If this does not complete the job, that is, $w_i^{\text{real}} > c_i(b_i - a_i) = w_i^{\text{pred}}$, we uniformly schedule the remaining work in the interval $[i, i + D]$

More formally,we define $s_i(t) = s_i'(t) + s_i''(t)$ where

$$s_i'(t) = \begin{cases} \min\left\{\frac{w_i^{\text{real}}}{b_i - a_i}, c_i\right\} & \text{if } t \in [a_i, b_i], \\ 0 & \text{otherwise.} \end{cases}$$

and

$$s_i''(t) = \begin{cases} \frac{1}{D}\max\{0, w_i^{\text{real}} - w_i^{\text{pred}}\} & \text{if } t \in [i, i + D], \\ 0 & \text{otherwise.} \end{cases}$$

Analysis. It is easy to see that the algorithm is consistent: If the prediction of w_i^{real} is perfect ($w_i^{\text{pred}} = w_i^{\text{real}}$), the job will be scheduled at speed c_i in the interval $[a_i, b_i]$. If all predictions are perfect, this is exactly the optimal schedule.

Theorem 6.1. *For every $0 < \delta \leq 1$, the cost of the schedule produced by the algorithm LAS-Trust is bounded by $(1+\delta)^\alpha \operatorname{OPT} + (24/\delta)^\alpha \cdot \operatorname{err}$.*

Proof. Define $w_i^+ = \max\{0, w_i^{\text{real}} - w_i^{\text{pred}}\}$ as the additional work at time i as compared to the predicted work. Likewise, define $w_i^- = \max\{0, w_i^{\text{pred}} - w_i^{\text{real}}\}$. We use $\operatorname{OPT}(w^+)$ and $\operatorname{OPT}(w^-)$ to denote the cost of optimal schedules of these workloads w^+ and w^-, respectively. We will first relate the energy of the schedule $s(t)$ to the optimal energy for the predicted instance, i.e., $\operatorname{OPT}(w^{\text{pred}})$. Then we will relate $\operatorname{OPT}(w^{\text{pred}})$ to $\operatorname{OPT}(w^{\text{real}})$.

For the former let s_i' and s_i'' be defined as in the algorithm. Observe that $s_i'(t) \leq s_i^{\text{pred}}(t)$ for all i and t. Hence, the energy for the partial schedule s' (by itself) is at most $\operatorname{OPT}(w^{\text{pred}})$. Furthermore, by definition we have that $s_i''(t) = w_i^+/D$. In other words, s_i'' is exactly the AVR schedule on instance (w^+, D, T). By analysis of AVR, we know that the total energy of s_i'' is at most $2^\alpha \operatorname{OPT}(w^+)$. Since the energy function is non-linear, we cannot simply add the energy of both speeds. Instead, we use the following inequality: For all $x, y \geq 0$ and $0 < \gamma \leq 1$, it holds that $(x+y)^\alpha \leq (1+\gamma)^\alpha x^\alpha + \left(\frac{2}{\gamma}\right)^\alpha y^\alpha$. This follows from a simple case distinction whether $y \leq \gamma x$. Thus, (substituting γ for $\delta/3$) the energy of the schedule s is bounded by

$$\begin{aligned}\int (s'(t) + s''(t))^\alpha dt &\leq (1+\delta/3)^\alpha \int s_i'(t)^\alpha dt + (6/\delta)^\alpha \int s_i''(t)^\alpha dt \\ &\leq (1+\delta/3)^\alpha \operatorname{OPT}(w^{\text{pred}}) + (12/\delta)^\alpha \operatorname{OPT}(w^+). \end{aligned} \tag{6.1}$$

For the last inequality we used that the competitive ratio of AVR is 2^α.

In order to relate $\operatorname{OPT}(w^{\text{pred}})$ and $\operatorname{OPT}(w^{\text{real}})$, we argue similarly. Notice that scheduling w^{real} optimally (by itself) and then scheduling w^- using AVR forms a valid solution for w^{pred}. Hence,

$$\operatorname{OPT}(w^{\text{pred}}) \leq (1+\delta/3)^\alpha \operatorname{OPT}(w^{\text{real}}) + (12/\delta)^\alpha \operatorname{OPT}(w^-).$$

Inserting this inequality into (6.1) we conclude that the energy of the schedule s is at most

$$\begin{aligned}&(1+\delta/3)^{2\alpha} \operatorname{OPT}(w^{\text{real}}) + (24/\delta)^\alpha (\operatorname{OPT}(w^+) + \operatorname{OPT}(w^-)) \\ &\leq (1+\delta)^\alpha \operatorname{OPT}(w^{\text{real}}) + (24/\delta)^\alpha \cdot \operatorname{err}.\end{aligned}$$

This inequality follows from the fact that the error function $\|\cdot\|_\alpha^\alpha$ is always an upper bound on the energy of the optimal schedule (by scheduling every job within the next time unit). □

6.2 Robustification

In this section, we describe a method Robustify that takes any online algorithm which guarantees to complete each job in $(1-\delta)D$ time, that is, with some slack to its deadline, and turns it into a robust algorithm without increasing the energy of the schedule produced. Here $\delta > 0$ can be chosen at will, but it impacts the robustness guarantee. We remark that the slack constraint is easy to achieve: In Appendix A.3 we prove that decreasing D to $(1-\delta)D$ increases the energy

of the optimum schedule only very mildly. Specifically, if we let $\mathrm{OPT}(w^{\mathrm{real}}, (1-\delta)D, T)$ and $\mathrm{OPT}(w^{\mathrm{real}}, D, T)$ denote the costs of optimal schedules of workload w^{real} with durations $(1-\delta)D$ and D, respectively, then:

Claim 6.1. *For any instance $(w^{\mathrm{real}}, D, T)$ we have that*

$$\mathrm{OPT}(w^{\mathrm{real}}, (1-\delta)D, T) \leq \frac{1}{(1-\delta)^{\alpha-1}} \mathrm{OPT}(w^{\mathrm{real}}, D, T).$$

Hence, running a consistent algorithm with $(1-\delta)D$ will not increase the cost significantly. Alternatively, we can run the online algorithm with D, but increase the generated speed function by $1/(1-\delta)$ and reschedule all jobs using EDF. This also results in a schedule where all jobs are completed in $(1-\delta)D$ time.

For a schedule s of $(w^{\mathrm{real}}, (1-\delta)D, T)$ we define the δ-convolution operator which returns the schedule $s^{(\delta)}$ of the original instance $(w^{\mathrm{real}}, D, T)$ by

$$s_i^{(\delta)}(t) = \frac{1}{\delta D} \int_{t-\delta D}^{t} s_i(r)\, dr$$

for each $i \in T$ (letting $s_i(r) = 0$ if $r < 0$). See Figure 6.1 for an illustration. The name comes from the fact that this operator is the convolution of $s_i(t)$ with the function $f(t)$ that takes value $1/(\delta D)$ if $0 \leq t \leq \delta D$ and value 0 otherwise.

Next we state three key properties of the convolution operator.

Claim 6.2. *If s is a feasible schedule for $(w^{\mathrm{real}}, (1-\delta)D, T)$ then $s^{(\delta)}$ is a feasible schedule for $(w^{\mathrm{real}}, D, T)$.*

Proof. Since s is a feasible schedule for $(w, (1-\delta D), T)$, we have that

$$\int_i^{i+D} s_i^{(\delta)}(t)dt = \int_i^{i+D} \frac{1}{\delta D} \left(\int_{t-\delta D}^{t} s_i(t')dt' \right) dt = \int_i^{i+(1-\delta)D} s_i(t') \left(\int_{t'}^{t'+\delta D} \frac{1}{\delta D} dt \right) dt' = w_i.$$

□

Claim 6.3. *The cost of schedule $s^{(\delta)}$ is not higher than that of s, that is,*

$$\int_0^T (s^{(\delta)}(t))^\alpha dt \leq \int_0^T (s(t))^\alpha dt.$$

Proof. The proof only uses Jensen's inequality in the second line and the statement can be

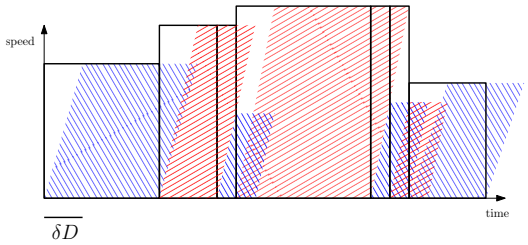


Figure 6.1: A schedule and its convolution.

calculated as follows.

$$\begin{aligned}\int_0^T \left(s^{(\delta)}(t)\right)^\alpha dt &= \int_0^T \left(\frac{1}{\delta D}\int_{t-\delta D}^{t} s(t')dt'\right)^\alpha dt \\ &\leq \int_0^T \frac{1}{\delta D}\left(\int_{t-\delta D}^{t} (s(t'))^\alpha dt'\right) dt \\ &= \int_0^T (s(t'))^\alpha \left(\int_{t'}^{t'+\delta D} \frac{1}{\delta D} dt\right) dt' \\ &= \int_0^T (s(t))^\alpha \, dt\end{aligned}$$

□

Let $s_i^{\mathrm{AVR}}(t)$ denote the speed of workload w_i^{real} of the Average Rate heuristic, that is, $s_i^{\mathrm{AVR}}(t) = w_i^{\mathrm{real}}/D$ if $i \leq t \leq i + D$ and $s_i^{\mathrm{AVR}}(t) = 0$ otherwise. We relate $s_i^{(\delta)}(t)$ to $s_i^{\mathrm{AVR}}(t)$.

Claim 6.4. *Let s be a feasible schedule for $(w^{\mathrm{real}}, (1-\delta)D, T)$. Then $s_i^{(\delta)}(t) \leq \frac{1}{\delta} s_i^{\mathrm{AVR}}(t)$.*

Proof. We have that

$$s_i^{(\delta)}(t) = \frac{1}{\delta D}\int_{t-\delta D}^{t} s_i(t')dt' \leq \frac{1}{\delta D}\int_{i}^{i+D} s_i(t')dt' = \frac{w_i}{\delta D} = \frac{s_i^{\mathrm{AVR}}(t)}{\delta}.$$

□

By using that the competitive ratio of Average Rate is at most 2^α (see Appendix A.1), we get

$$\int_0^T (s^{(\delta)}(t))^\alpha dt \leq \left(\frac{1}{\delta}\right)^\alpha \int_0^T (s^{\mathrm{AVR}}(t))^\alpha dt \leq \left(\frac{2}{\delta}\right)^\alpha \mathrm{OPT}\,.$$

We conclude with the following theorem, which follows immediately from the previous claims.

Theorem 6.2. *Given an online algorithm that produces a schedule s for $(w^{\mathrm{real}}, (1-\delta)D, T)$, we can compute online a schedule $s^{(\delta)}$ with*

$$\int_0^T (s^{(\delta)}(t))^\alpha dt \leq \min\left\{\int_0^T (s(t))^\alpha dt,\ \left(\frac{2}{\delta}\right)^\alpha \mathrm{OPT}\right\}.$$

Algorithm 1 Learning Augmented Scheduling (LAS)

Input: T, D, and w^{pred} initially and w^{real} in an online fashion
Output: A feasible schedule $(s_i)_{i=0}^{T-D}$
Let $\delta \in (0, 1/2)$ with $\left(\frac{1+\delta}{1-\delta}\right)^{\alpha} = 1 + \epsilon$.
Compute optimal offline schedule for $(w^{\text{pred}}, T, (1-\delta)D)$ where the jobs w_i^{pred} are run at uniform speeds c_i an disjoint intervals $[a_i, b_i]$ using [88].

while w_i^{real} **do**
Let $s_i'(t) = \begin{cases} \min\left\{\frac{w_i^{\text{real}}}{b_i - a_i}, c_i\right\} & \text{if } t \in [a_i, b_i], \\ 0 & \text{otherwise.} \end{cases}$
Let $s_i''(t) = \begin{cases} \frac{1}{D}\max\{0, w_i^{\text{real}} - w_i^{\text{pred}}\} & \text{if } t \in [i, i+D], \\ 0 & \text{otherwise.} \end{cases}$
Let $s_i(t) = \frac{1}{\delta D}\int_{t-\delta D}^{t} s_i'(r) + s_i''(r)\, dr$
end while

6.3 Summary of the algorithm

By combining LAS-Trust and Robustify, we obtain an algorithm LAS (see Algorithm 1) which has the following properties.

Theorem 6.3. *For any given $\epsilon \in (0, 1)$, algorithm LAS constructs a schedule of cost at most* $\min\left\{(1+\epsilon)\,\mathrm{OPT} + O\left(\frac{\alpha}{\epsilon}\right)^{\alpha}\mathrm{err},\ O\left(\frac{\alpha}{\epsilon}\right)^{\alpha}\mathrm{OPT}\right\}$.

Proof. We choose δ such that $(\frac{1+\delta}{1-\delta})^{\alpha} = 1 + \epsilon$. Note that $\delta \le \epsilon/\alpha$. By Claim 6.1 we know that

$$\mathrm{OPT}(w^{\text{real}}, (1-\delta)D, T) \le \left(\frac{1}{1-\delta}\right)^{\alpha}\mathrm{OPT}\,.$$

Hence, by Theorem 6.1 algorithm LAS-Trust constructs a schedule with cost at most

$$\left(\frac{1+\delta}{1-\delta}\right)^{\alpha}\mathrm{OPT} + O\left(\frac{1}{\delta}\right)^{\alpha}\mathrm{err}$$

Finally, we apply Robustify and with Theorem 6.2 obtain a bound of

$$\min\left\{\left(\frac{1+\delta}{1-\delta}\right)^{\alpha}\mathrm{OPT} + O\left(\frac{1}{\delta}\right)^{\alpha}\mathrm{err},\ O\left(\frac{1}{\delta}\right)^{\alpha}\mathrm{OPT}\right\} \le \min\left\{(1+\epsilon)\,\mathrm{OPT} + O\left(\frac{\alpha}{\epsilon}\right)^{\alpha}\mathrm{err},\ O\left(\frac{\alpha}{\epsilon}\right)^{\alpha}\mathrm{OPT}\right\}.$$

□

7 Experiments

In this chapter, we will test the LAS algorithm on both synthetic and real datasets. We will calculate the competitive ratios with respect to the offline optimum. In most of our experiments, we fix $\alpha = 3$ as this value models the power consumption of modern processors (see Bansal et al. [12]) and at the end of the chapter we explore the influence of α in the performance of various algorithms. For each experiment, we compare our LAS algorithm to the three main online algorithms that exist for this problem which are AVR and OA by Yao et al. [88] and BKP by Bansal et al. [12]. We note that the code is publicly available at https://github.com/andreasr27/LAS.

Artificial datasets. In the synthetic data case, we will mimic the request pattern of a typical data center application by simulating a bounded random walk. In the following we write $Z \sim \mathcal{U}\{m, M\}$ when sampling an integer uniformly at random in the range $[m, M]$. Subsequently, we fix three integers s, m, M where $[m, M]$ define the range in which the walk should stay. For each integral time i we sample $X_i \sim \mathcal{U}\{-s, s\}$. Then we set $w_0 \sim \mathcal{U}\{m, M\}$ and w_{i+1} to be the median value of the list $\{m, w_i + X_i, M\}$, that is, if the value $w_i + X_i$ remains in the predefined range we do not change it, otherwise we round it to the closest point in the range. For this type of ground truth instance we test our algorithm coupled with three different predictors. The **accurate** predictor for which we set $\tilde{w}_i \sim w_i + \mathcal{U}\{-s, s\}$, the **random** predictor where we set $\tilde{w}_i \sim \mathcal{U}\{m, M\}$ and the **misleading** predictor for which $\tilde{w}_i = (M - w_i) + m$. In each case we perform 20 experiment runs. The results are summarized in Table 7.1. In the first two cases (accurate and random predictors) we present the average competitive ratios of every algorithm over all runs. In contrast, for the last column (misleading predictor) we present the maximum competitive ratio of each algorithm taken over the 20 runs to highlight the worst case robustness of LAS. We note that in the first case, where the predictor is relatively accurate but still noisy, LAS is consistently better than any online algorithm achieving a competitive ratio close to 1 for small values of ϵ. In the second case, the predictor does not give us useful information about the future since it is completely uncorrelated with the ground truth instance. In such a case, LAS experiences a similar performance to the best online algorithms. In the third case, the predictor tries to mislead our algorithm by creating a prediction which constitutes a symmetric (around $(m + M)/2$) random walk with respect to the true instance. When coupled with such a predictor, as expected, LAS performs worse than the best online algorithm, but it still maintains an acceptable competitive ratio. Furthermore, augmenting the robustness parameter ϵ, and thereby trusting less the predictor, improves the competitive ratio in this case.

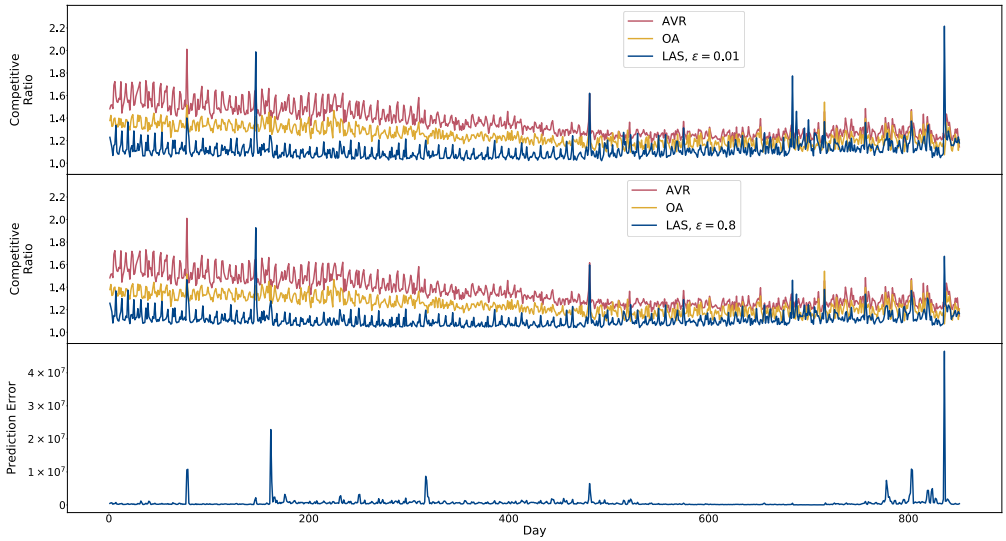


Figure 7.1: From top to bottom: The first two graphs show the performance of LAS when $\epsilon = 0.01$ and $\epsilon = 0.8$ with respect to the online algorithms AVR and OA. The bottom graph presents the prediction error. The timeline was discretized in chunks of ten minutes and D was set to 20.

Table 7.1: Artificial dataset results

Algorithm	Accurate	Random	Misleading
AVR	1.268	1.268	1.383
BKP	7.880	7.880	10.380
OA	1.199	1.199	1.361
LAS, $\epsilon = 0.8$	1.026	1.203	1.750
LAS, $\epsilon = 0.6$	1.022	1.207	1.758
LAS, $\epsilon = 0.4$	1.018	1.213	1.767
LAS, $\epsilon = 0.2$	1.013	1.224	1.769
LAS, $\epsilon = 0.01$	1.008	1.239	1.766

We used $m = 20$, $M = 80$, $s = 5$, $T = 220$ and $D = 20$.

Table 7.2: Real dataset results with different α values

Algorithm	$\alpha = 3$	$\alpha = 6$	$\alpha = 9$	$\alpha = 12$
AVR	1.365	2.942	7.481	21.029
OA	1.245	2.211	4.513	9.938
LAS, $\epsilon = 0.8$	1.113	1.576	2.806	7.204
LAS, $\epsilon = 0.01$	1.116	1.598	2.918	8.055

The timeline was discretized in chunks of ten minutes and D was set to 20.

Real dataset. We provide additional evidence that the LAS algorithm outperforms purely online algorithms by conducting experiments on the login requests to *BrightKite* [23], a no longer functioning social network. We note that this dataset was previously used in the context of learning-augmented algorithms by Lykouris and Vassilvitskii [72]. In order to emphasize the fact that even a very simple predictor can improve the scheduling performance drastically, we will use the arguably most simple predictor possible. We use the access patterns of the previous day as a prediction for the current day. In Figure 7.1 we compare the performance of the LAS algorithm for different values of the robustness parameter ϵ with respect to AVR and OA. We did not include BKP, since its performance is substantially worse than all other algorithms. Note that our algorithm shows a substantial improvement with respect to both AVR and OA, while maintaining a low competitive ratio even when the prediction error is high (for instance in the last days). The first 100 days, where the prediction error is low, by setting $\epsilon = 0.01$ (and trusting more the prediction) we obtain an average competitive ratio of 1.134, while with $\epsilon = 0.8$ the average competitive ratio slightly deteriorates to 1.146. However, when the prediction error is high, setting $\epsilon = 0.8$ is better. On average from the first to the last day of the timeline, the competitive ratio of AVR and OA is 1.36 and 1.24 respectively, while LAS obtains an average competitive ratio of 1.116 when $\epsilon = 0.01$ and 1.113 when $\epsilon = 0.8$, thus beating the online algorithms in both cases.

Finally, we further explore the performance of LAS algorithm for different values of the parameter α on *BrightKite*. The results are summarized in Table 7.2. In every column the average competitive ratios of each algorithm for a fixed α are presented. We note that, as expected, higher values of α penalize heavily wrong decisions deteriorating the competitive ratios of all algorithms. Nevertheless, LAS algorithm consistently outperforms AVR and OA for all different values of α.

8 Extension to evolving predictors

In this chapter, we extend our results to the case where the algorithm is provided several predictions over time. In particular, we assume that the algorithm is provided a new prediction at each integral time t. The setting is natural as for a very long timeline, it is intuitive that the predictor might renew its prediction over time. Since making a mistake in the prediction of a very far future seems also less hurtful than making a mistake in predicting an immediate future, we define a generalized error metric incorporating this idea.

Let $0 < \lambda < 1$ be a parameter that describes how fast the confidence in a prediction deteriorates with the time until the expected arrival of the predicted job. Define the prediction received at time t as a workload vector $w^{\text{pred}}(t)$. Recall we are still considering the uniform deadlines case hence an instance is defined as a triplet $\mathcal{I} = (w, D, T)$.

We then define the total error of a series of predictions as

$$\text{err}^{(\lambda)} = \sum_{t} \sum_{i=t+1}^{\infty} |w_i^{\text{real}} - w_i^{\text{pred}}(t)|^{\alpha} \cdot \lambda^{i-t}.$$

In the following we reduce the evolving predictions model to the single prediction one.

We would like to prove similar results as in the single prediction setting with respect to $\text{err}^{(\lambda)}$. In order to do so, we will split the instance into parts of bounded time horizon, solve each one independently with a single prediction, and show that this also gives a guarantee based on $\text{err}^{(\lambda)}$. In particular, we will use the algorithm for the single prediction model as a black-box.

The basic idea is as follows. If no job were to arrive for a duration of D, then the instance before this interval and afterwards can be solved independently. This is because any job in the earlier instance must finish before any job in the later instance can start. Hence, they cannot interfere. At random points, we ignore all jobs for a duration of D, thereby split the instance. The ignored jobs will be scheduled sub-optimally using AVR. If we only do this occasionally, i.e., after every intervals of length $\gg D$, the error we introduce is negligible.

We proceed by defining the splitting procedure formally. Consider the timeline as infinite in both directions. To split the instance, we define some interval length $2kD$, where $k \in \mathbb{N}$ will be specified later. We split the infinite timeline into contiguous intervals of length $2kD$. Moreover, we choose an offset $x \in \{0, \cdots, k-1\}$ uniformly at random. Using these values, we define intervals $I_i = [2((i-1)k - x)D, 2(ik - x)D)$. We will denote by $t_i = (2(i-1)k - x)D$ the start time of the interval I_i. Consequently, the end of I_i is t_{i+1}.

In each interval I_i, we solve the instance given by the jobs entirely contained in this interval using our algorithm with the most recent prediction as of time t_i, i.e., $w^{\text{pred}}(t_i)$, and schedule the jobs accordingly. We write $s^{\text{ALG}(i)}$ for this schedule. For the jobs that are overlapping with two contiguous intervals we schedule them independently using the Average Rate heuristic. The schedule for the jobs overlapping with intervals I_i and I_{i+1} will be referred to as $s^{\text{AVR}(i)}$.

It is easy to see that this algorithm is robust: The energy of the produced schedule is

$$\int \left(\sum_i \left[s^{\text{ALG}(i)}(t) + s^{\text{AVR}(i)}(t) \right] \right)^\alpha dt \\ \leq 2^\alpha \int \left(\sum_i s^{\text{ALG}(i)}(t) \right)^\alpha dt + 2^\alpha \int \left(\sum_i s^{\text{AVR}(i)}(t) \right)^\alpha dt.$$

Moreover, the first term can be bounded by $2^\alpha \cdot O(\alpha/\epsilon)^\alpha \operatorname{OPT}$ using Theorem 6.3 and the second term can be bounded by $2^\alpha \cdot 2^\alpha \operatorname{OPT}$ because of Theorem A.2. This gives an overall bound of $O(\alpha/\epsilon)^\alpha$ on the competitive ratio.

In the rest of the section we focus on the consistency/smoothness guarantee. We first bound the costs of $s^{\text{ALG}(i)}$ and $s^{\text{AVR}(i)}$ isolated (ignoring potential interferences). Using these bounds, we derive an overall guarantee for the algorithm's cost.

Lemma 8.1.

$$\mathbb{E}\left(\sum_i \int s^{\text{AVR}(i)}(t)^\alpha \right) \leq \frac{2^\alpha}{k} \operatorname{OPT}$$

Proof. Fix some $i > 0$ and let us call O_i the job instance consisting of jobs overlapping with both intervals I_i and I_{i+1}. By Theorem A.2 the energy used by AVR is at most a 2^α-factor from the optimum schedule. Hence,

$$\int s^{\text{AVR}(i)}(t)^\alpha dt \leq 2^\alpha \operatorname{OPT}(O_i).$$

Now denote by s^{OPT} the speed function of the optimum schedule over the whole instance. Then

$$\operatorname{OPT}(O_i) \leq \int_{t_i - D}^{t_i + D} s^{\text{OPT}}(t)^\alpha dt.$$

This holds because s^{OPT} processes some work during $[t_i - D, t_i + D]$ which has to include all of

O_i. Hence, we have that

$$\begin{aligned}
&\mathbb{E}\left(\sum_i \mathrm{OPT}(O_i)\right) \\
&\leq \frac{1}{k}\sum_{x=0}^{k-1}\sum_i \int_{2(ik-x)D-D}^{2(ik-x)D+D} s^{\mathrm{OPT}}(t)^\alpha dt \\
&\leq \frac{1}{k}\int s^{\mathrm{OPT}}(t)^\alpha dt = \frac{1}{k}\,\mathrm{OPT}
\end{aligned}$$

The second inequality holds, because the integrals are over disjoint ranges. Together, with the bound on $s^{\mathrm{AVR}(i)}$ we get the claimed inequality. □

Lemma 8.2.

$$\sum_i \int s^{\mathrm{ALG}(i)}(t)^\alpha dt \leq (1+\epsilon)\,\mathrm{OPT} + O\left(\frac{\alpha}{\epsilon}\right)^\alpha \cdot \lambda^{-2kD} \cdot \mathrm{err}^{(\lambda)}\,.$$

Proof. Note that for any i

$$\sum_{t=t_i+1}^{t_{i+1}} |w_t^{\mathrm{real}} - w_t^{\mathrm{pred}}(t_i)|^\alpha \leq \lambda^{-2kD} \sum_{t=t_i+1}^{t_{i+1}} |w_t^{\mathrm{real}} - w_t^{\mathrm{pred}}(t_i)|^\alpha \lambda^{t-t_i}.$$

Hence,

$$\sum_i \sum_{t=t_i}^{t_{i+1}} |w_t^{\mathrm{real}} - w_t^{\mathrm{pred}}(t_i)|^\alpha \leq \lambda^{-2kD}\,\mathrm{err}^{(\lambda)}\,.$$

Using Theorem 6.3 for each $\int s^{(i)}_{\mathrm{ALG}}(t)^\alpha dt$, we get a bound depending on $\sum_{t=t_i}^{t_{i+1}} |w_t^{\mathrm{real}} - w_t^{\mathrm{pred}}(t_i)|^\alpha$. Summing over i and using the inequality above finishes the proof of the lemma. □

We are ready to state the consistency/smoothness guarantee of the splitting algorithm.

Theorem 8.3. *With robustness parameter $O(\epsilon/\alpha)$ the splitting algorithm produces in expectation a schedule of cost at most*

$$(1+\epsilon)\,\mathrm{OPT} + O\left(\frac{\alpha}{\epsilon}\right)^\alpha \cdot \lambda^{-D/\epsilon \cdot O(\alpha/\epsilon)^\alpha} \cdot \mathrm{err}^{(\lambda)}\,.$$

In other words, we get the same guarantee as in the single prediction case, except that the dependency on the error is larger by a factor of $\lambda^{-D/\epsilon \cdot O(\alpha/\epsilon)^\alpha}$. The exponential dependency on D may seem unsatisfying, but (1) it cannot be avoided (see Theorem A.6) and (2) for moderate values of λ, e.g. $\lambda = 1 - 1/D$, this exponential dependency vanishes.

Proof. We will make use of the following inequality: For all $a, b \geq 0$ and $0 < \delta \leq 1$, it holds that

$$(a+b)^\alpha \leq (1+\delta)a^\alpha + \left(\frac{3\alpha}{\delta}\right)^\alpha b^\alpha.$$

This follows from a simple case distinction whether $b \leq a \cdot \delta/(2\alpha)$. In expectation, the cost of the algorithm is bounded by

$$\begin{aligned}
&\mathbb{E}\left[\int \left(\sum_i [s^{\mathrm{ALG}(i)}(t) + s^{\mathrm{AVR}(i)}(t)]\right)^\alpha dt\right] \\
&\quad \leq (1+\epsilon)\mathbb{E}\left[\int \sum_i (s^{\mathrm{ALG}(i)}(t))^\alpha dt\right] \\
&\qquad + \left(\frac{3\alpha}{\epsilon}\right)^\alpha \mathbb{E}\left[\int \sum_i (s^{\mathrm{AVR}(i)}(t))^\alpha dt\right] \\
&\quad \leq (1+\epsilon)\int \sum_i s^{\mathrm{ALG}(i)}(t))^\alpha dt \\
&\qquad + \frac{1}{k}\left(\frac{6\alpha}{\epsilon}\right)^\alpha \mathrm{OPT}\,.
\end{aligned}$$

By choosing $k = 1/\epsilon(6\alpha/\epsilon)^\alpha$ the latter term becomes $\epsilon\,\mathrm{OPT}$. With Lemma 8.2 we can bound the term above by

$$(1+\epsilon)^3\,\mathrm{OPT} + O\left(\frac{\alpha}{\epsilon}\right)^\alpha \cdot \lambda^{-D/\epsilon \cdot O(\alpha/\epsilon)^\alpha} \cdot \mathrm{err}^{(\lambda)}\,.$$

Scaling ϵ by a constant yields the claimed guarantee. □

We also note that the dependency of Theorem 8.3 on the λ parameter is essentially tight and we prove the latter in Theorem A.6 of Appendix A.2.

9 Further extensions

As already mentioned in Section 5.3 we extend the results presented in this part of the thesis in two further ways:

1. In Appendix A.5 we consider the General Speed Scheduling problem and show that a more sophisticated method allows us to robustify any algorithm even in this more general setting. Hence, for this case we can also obtain an algorithm that is almost optimal in the consistency case and always robust.

2. In Appendix A.4 we also cope with small perturbations in the prediction. Indeed, the careful reader may have noted that one can craft instances so that the used error function err is very sensitive to small shifts in the prediction. An illustrative example is as follows: Consider a predicted workload w^{pred} defined by $w_i^{\text{pred}} = 1$ for those time steps i that are divisible by a large constant, say 1000, and let $w_i^{\text{pred}} = 0$ for all other time steps. If the real instance w^{real} is a small shift of w^{pred} say $w_{i+1}^{\text{real}} = w_i^{\text{pred}}$ then the prediction error $\text{err}(w^{\text{real}}, w^{\text{pred}})$ is large although w^{pred} intuitively forms a good prediction of w^{real}. To overcome this sensitivity, we first generalize the definition of err to err_η which is tolerant to small shifts in the workload. In particular, $\text{err}_\eta(w^{\text{real}}, w^{\text{pred}}) = 0$ for the example given above. We then give a generic method for transforming an algorithm so as to obtain guarantees with respect to err_η instead of err at a small loss.

Part III

The Primal-Dual method for learning-augmented algorithms

General techniques for beyond worst-case analysis

10 Overview and related work

The goal of this part is to extend the Primal-Dual method from the setting of classical online algorithms to the online learning-augmented setting. In the current chapter we will present an overview of the Primal-Dual method for online algorithms and highlight our main contributions.

10.1 The classical Primal-Dual method

The Primal-Dual (PD) method is a very powerful algorithmic technique to design online algorithms. It was first introduced by Alon et al. [3] to design an online algorithm for the classical online set cover problem and later extended to many other problems such as weighted caching [10], revenue maximization in ad-auctions, TCP acknowledgement and ski rental [18]. We mention the survey of Buchbinder and Naor [17] for more references about this technique. In a few words, the technique consists in formulating the online problem as a linear program P complemented by its dual D. Subsequently, the algorithm builds online a feasible fractional solution to both the primal P and dual D. Every time an update of the primal and dual variables is made, the cost of the primal increases by some amount ΔP while the cost of the dual increases by some amount ΔD. The competitive ratio of the fractional solution is then obtained by upper bounding the ratio $\frac{\Delta P}{\Delta D}$ and using weak duality. The integral solution is then obtained by an online rounding scheme of the fractional solution.

10.2 Our contributions

We show how to extend the Primal-Dual method (when predictions are provided) for solving problems that can be formulated as covering problems. The algorithms designed using this technique receive as input a robustness parameter ϵ and incorporate a prediction. If the prediction is accurate our algorithms can be arbitrarily close to the optimal offline (beating known lower bounds of the classical online algorithms) while being robust to failures of the predictor. We first apply our Primal-Dual Learning-Augmented (PDLA) technique to the online version of the weighted set cover problem, which constitutes the most canonical example of a covering Linear Program (LP). For that problem we show how we can easily modify the Primal-Dual algorithm to incorporate predictions. Even though in this case, prediction may not seem very natural, this result reveals that we can use PDLA to design learning-augmented algorithms for the large class of problems that can be formulated as a covering LP. We then continue by addressing problems in which the prediction model is much more natural. Using the PDLA technique, we first design an algorithm which recovers the results of Purohit et al. [80] for the ski

rental problem, and we also prove that the consistency-robustness trade-off of that algorithm is optimal. We additionally design a learning-augmented algorithm for a generalization of the ski rental, namely the Bahncard problem. Finally, we turn our attention to a problem which arises in network congestion control, the TCP acknowledgement problem. We design a learning-augmented algorithm for that problem and conduct experiments which confirm our claims. We note that the analysis of the algorithms designed using PDLA is (arguably) simple and boils down to (1) proving robustness with (essentially) the same proof as in the original Primal-Dual technique and (2) proving consistency using a simple charging argument that, without making use of the dual, relates the cost incurred by our algorithms to the prediction. In addition to that, using PDLA, the design of online learning-augmented algorithms is almost automatic. We emphasize that the preexisting online rounding schemes to obtain an integral solution from a fractional solution still apply to our learning-augmented algorithms. Hence we focus only on building a fractional solution and provide appropriate references for the rounding scheme.

11 Online set cover with predictions

In this chapter we apply PDLA to solve the online weighted set cover problem when provided with predictions. Set cover is arguably the most canonical example of a covering problem and the framework that we develop readily applies to other covering problems as the ski rental, Bahncard, and dynamic TCP acknowledgement, which are all problems that can be formulated as covering LPs.

The weighted set cover problem. In this problem, we are given a universe $\mathcal{U} = \{e_1, e_2, \ldots, e_n\}$ of n elements and a family $\mathcal{F}$ of m sets over this universe, each set $S \in \mathcal{F}$ has a weight w_S and each element e is covered by any set in $\mathcal{F}(e) = \{S \in \mathcal{F} \mid e \in S\}$. Let $d = \max_{e \in \mathcal{U}} |\mathcal{F}(e)|$ denote the maximum number of sets that cover one element. Our goal is to select sets so as to cover all elements while minimizing the total weight. In its online version, elements are given one by one and it is unknown to the algorithm which elements will arrive and in which order. When a new element arrives, it is required to cover it by adding a new set if necessary. Removing a set from the current solution to decrease its cost is not allowed. Alon et al. [3] first studied the online version designing an almost optimal $O(\log n \log d)$-competitive algorithm. We note that the $O(\log n)$ factor comes from the integrality gap of the linear program formulation of the problem (Figure 11.1) while the $O(\log d)$ is due to the online nature of the problem. Since Alon et al. [3] designed an online rounding scheme at a multiplicative cost of $O(\log n)$, we will focus on building an increasing fractional solution to the set cover problem (i.e. x_S can only increase over time for all S).

Primal
minimize $\sum_{S \in \mathcal{F}} w_S x_S$ subject to: $\sum_{S \in \mathcal{F}(e)} x_S \geq 1 \quad \forall e \in \mathcal{U}$ $x_S \geq 0 \quad \forall S \in \mathcal{F}$
Dual
maximize $\sum_{e \in \mathcal{U}} y_e$ subject to: $\sum_{e \in S} y_e \leq w_S \quad \forall S \in \mathcal{F}$ $y_e \geq 0 \quad \forall e \in \mathcal{U}$

Figure 11.1: Primal dual formulation of weighted set cover

PDLA for weighted set cover. Algorithm 3 takes as input a predicted covering $\mathcal{A} \subset \mathcal{F}$ and a robustness parameter $\epsilon \in (0, 1]$. While an instance $\mathcal{I}$ is revealed in an online fashion, an increasing fractional solution $\{x_S\}_{S \in \mathcal{F}} \in [0, 1]^{\mathcal{F}}$ is built. We do not assume that our prediction $\mathcal{A}$ forms a feasible solution, i.e., it may be that for some element e, $\mathcal{F}(e) \cap \mathcal{A} = \emptyset$, in that case we just use the same primal update as the purely online algorithm ($\epsilon = 1$). In addition, we can assume w.l.o.g. that $w_S \geq 1 \; \forall S \in \mathcal{F}$. Indeed, we can always scale up by the same amount all the weights of the sets in $\mathcal{F}$ so that the latter assumption holds and any solution will increase its cost by the same multiplicative factor.

Algorithm Intuition. We first turn our attention to the original online algorithm of Alon et. al. [3] described in Algorithm 2. To get an intuition assume that $w_S = 1$, $\forall S$ and consider the very first arrival of an element e. After the first execution of the while loop, e is covered and $x_S = \frac{1}{|\mathcal{F}(e)|}, \forall S \in \mathcal{F}(e)$. In other words, the online algorithm creates a uniform distribution over the sets in $\mathcal{F}(e)$, reflecting in such a way his unawareness about the future. On the contrary Algorithm 3 uses the prediction to adjust the increase rate of primal variables, augmenting more aggressively primal variables of sets which are predicted to be in the optimal offline solution. Indeed, assuming that $\mathcal{A}$ covers element e, after the first execution of the while loop, sets which belong to $\mathcal{A}$ get a value of $\frac{\epsilon}{|\mathcal{F}(e)|} + \frac{1-\epsilon}{|\mathcal{F}(e)\cap\mathcal{A}|}$ while sets which are not chosen by the prediction get $\frac{\epsilon}{|\mathcal{F}(e)|}$.

We continue by exposing our main conceptual contribution, and to that end, we introduce some auxiliary notation:

1. $S(\mathcal{A}, \mathcal{I})$ equals the cost of the (possibly partial) covering if prediction $\mathcal{A}$ is followed blindly.
2. C_{nc} equals the cost of optimally covering elements which are not covered by the prediction.
3. $\text{cost}_{\mathcal{PDLA}}(\mathcal{I}, \mathcal{A}, \epsilon)$ equals the cost of the covering solution calculated by Algorithm 3.

Theorem 11.1. *The cost of the fractional solution output by Algorithm 3 satisfies*

$$\underset{\mathcal{PDLA}}{\text{cost}}(\mathcal{I}, \mathcal{A}, \epsilon) \leq \min\left\{O\left(\frac{1}{1-\epsilon}\right) \cdot S(\mathcal{A}, \mathcal{I}) + O(\log(d)) \cdot C_{nc},\ O\left(\log\left(\frac{d}{\epsilon}\right)\right) \cdot \text{OPT}(\mathcal{I})\right\}$$

Theorem 11.1 states that Algorithm 3 has consistency $O\left(\frac{1}{1-\epsilon}\right)$ and robustness $O\left(\log\left(\frac{d}{\epsilon}\right)\right)$. We note that similar bounds could be achieved by simpler algorithms (see Theorem 1 in [72]). Thus, the main contribution of Algorithm 3 is conceptual, i.e., the Primal-Dual method for covering problems can be extended to the learning-augmenting setting. In Chapter 12 we substantiate our claim by applying the same ideas to more covering problems.

As already mentioned in Chapter 2, it is worth noting that in this part of the thesis we do not focus on defining a specific error metric for the prediction $\mathcal{A}$, consequently the smoothness of Algorithm 3 is implicit in the definition of $S(\mathcal{A}, \mathcal{I})$.

To prove the robustness of Algorithm 3 we start by arguing that the dual constraints are only violated by a multiplicative factor of $O\left(\log\left(\frac{d}{\epsilon}\right)\right)$. Thus, scaling down the dual solution of Algorithm 3 by $O\left(\log\left(\frac{d}{\epsilon}\right)\right)$ creates a feasible dual solution which will permit us to use weak duality.

Note that whenever $|\mathcal{F}(e) \cap \mathcal{A}| = 0$, Algorithm 3 updates the primal variables as the purely online Algorithm 2. With that in mind, we make the exposition more clear by defining $\frac{(1-\epsilon)\cdot\mathbb{1}\{S\in\mathcal{A}\}}{w_S \cdot |\mathcal{F}(e)\cap\mathcal{A}|} = 0$, if $|\mathcal{F}(e) \cap \mathcal{A}| = 0$ and $\mathbb{1}\{S \in \mathcal{A}\} = 0$.

Lemma 11.2. *Let y be the dual solution built by Algorithm 3. Then $\frac{y}{\Theta(\log(d/\epsilon))}$ is a feasible solution to the dual problem.*

Proof. The proof essentially follows the same path as in [17]. The only constraints that can be violated are of the form $\sum_{e\in S} y_e \leq w_S$ for some $S \in \mathcal{F}$. Consider one such constraint. At

Algorithm 2 PRIMAL DUAL METHOD FOR ONLINE WEIGHTED SET COVER [3].

Initialize: $x_S \leftarrow 0$, $y_e \leftarrow 0\ \forall S, e$
for all element e that just arrived **do**
  while $\sum_{S \in \mathcal{F}(e)} x_S < 1$ **do**
    / Primal Update */*
    for all $S \in \mathcal{F}(e)$ **do**
      $x_S \leftarrow x_S \left(1 + \frac{1}{w_S}\right) + \frac{1}{w_S |\mathcal{F}(e)|}$
    end for
    / Dual Update */*
    $y_e \leftarrow y_e + 1$
  end while
end for

⇒

Algorithm 3 PDLA FOR ONLINE WEIGHTED SET COVER.

Input: ϵ, $\mathcal{A}$
Initialize: $x_S \leftarrow 0$, $y_e \leftarrow 0\ \forall S, e$
for all element e that just arrived **do**
  while $\sum_{S \in \mathcal{F}(e)} x_S < 1$ **do**
    for all $S \in \mathcal{F}(e)$ **do**
      if $|\mathcal{F}(e) \cap \mathcal{A}| \geq 1$ **then**
        / Primal Update (more aggressive if $\mathbb{1}\{S \in \mathcal{A}\} = 1$) */*
        $x_S \leftarrow x_S \left(1 + \frac{1}{w_S}\right) + \frac{\epsilon}{w_S \cdot |\mathcal{F}(e)|} + \frac{(1-\epsilon) \cdot \mathbb{1}\{S \in \mathcal{A}\}}{w_S \cdot |\mathcal{F}(e) \cap \mathcal{A}|}$
      else
        / e is not covered by the prediction */*
        $x_S \leftarrow x_S \cdot \left(1 + \frac{1}{w_S}\right) + \frac{1}{w_S \cdot |\mathcal{F}(e)|}$
      end if
    end for
    / Dual Update */*
    $y_e \leftarrow y_e + 1$
  end while
end for

every update of the primal variable x_S the sum $\sum_{e \in S} y_e$ increases by 1, since the dual variable corresponding to the newly arrived element increases by 1 . We prove by induction on the number of such updates that at any point in time $x_S \geq \frac{\epsilon}{d}\left(\left(1 + \frac{1}{w_S}\right)^{\sum_{e \in S} y_e} - 1\right)$. Indeed, when no update concerning S is done we have that $x_S = 0$ and $\sum_{e \in S} y_e = 0$. Suppose this is true after k updates of the variable x_S, i.e. $\sum_{e \in S} y_e = k$. Now, assume that a newly arrived element $e^* \in S$ provokes a primal update from x_S^{old} to x_S^{new} and increases its dual value by one, i.e. $y_{e^*}^{new} = y_{e^*}^{old} + 1$. Then we always have:

$$\begin{aligned} x_S^{new} \geq & x_S^{old} \cdot \left(1 + \frac{1}{w_S}\right) + \min\left\{\frac{1}{|\mathcal{F}(e)| \cdot w_S}, \frac{\epsilon}{|\mathcal{F}(e)| \cdot w_S} + \frac{(1-\epsilon) \cdot \mathbb{1}\{S \in \mathcal{A}\}}{|\mathcal{F}(e) \cap \mathcal{A}| \cdot w_S}\right\} \geq \\ \geq & x_S^{old} \cdot \left(1 + \frac{1}{w_S}\right) + \frac{\epsilon}{d \cdot w_S} \end{aligned}$$

Thus, by the induction hypothesis

$$\begin{aligned} x_S^{new} \geq & \frac{\epsilon}{d}\left(\left(1 + \frac{1}{w_S}\right)^{\sum_{e \in S \setminus \{e^*\}} y_e + y_{e^*}^{old}} - 1\right) \cdot \left(1 + \frac{1}{w_S}\right) + \frac{\epsilon}{d \cdot w_S} \\ = & \frac{\epsilon}{d}\left(\left(1 + \frac{1}{w_S}\right)^{\sum_{e \in S \setminus \{e^*\}} y_e + y_{e^*}^{new}} - 1\right) = \frac{\epsilon}{d}\left(\left(1 + \frac{1}{w_S}\right)^{\sum_{e \in S} y_e} - 1\right) \end{aligned}$$

Moreover, since $w_S \geq 1$, we have that $(1 + 1/w_s)^{w_s} \geq 2$, thus:

$$x_S \geq \frac{\epsilon}{d}\left(\left(1 + \frac{1}{w_S}\right)^{w_S \cdot \frac{\sum_{e \in S} y_e}{w_S}} - 1\right) \geq \frac{\epsilon}{d}\left(2^{\frac{\sum_{e \in S} y_e}{w_S}} - 1\right)$$

We continue by upper bounding the value of x_S. Note that once $x_S \geq 1$, no more primal updates can happen, therefore whenever an update is made we have $x_S < 1$ just before the update. Thus:

$$\begin{aligned} x_S^{new} &\leq x_S^{old} \cdot \left(1 + \frac{1}{w_S}\right) + \max\left\{\frac{\epsilon}{w_S \cdot |\mathcal{F}(e)|} + \frac{(1-\epsilon)\cdot \mathbb{1}\{S \in \mathcal{A}\}}{w_S \cdot |\mathcal{F}(e) \cap \mathcal{A}|}, \frac{1}{w_S \cdot |\mathcal{F}(e)|}\right\} \\ &\leq x_S^{old} \cdot 2 + 1 \leq 3 \end{aligned}$$

Combining the lower and upper bound on x_S we get that:

$$\sum_{e \in S} y_e \leq \log\left(\frac{3d}{\epsilon} + 1\right) \cdot w_S = O(\log(d/\epsilon)) \cdot w_S$$

which concludes the proof. □

Lemma 11.3 (Robustness)**.** *The competitive ratio is always bounded by* $O\left(\log\left(\frac{d}{\epsilon}\right)\right)$

Proof. We denote as before by x_S^{old} and x_S^{new} the primal variables before and after the update respectively. Each time the while loop is executed we have that $\sum_{S \in \mathcal{F}(e)} x_S^{old} < 1$ and the increase in the dual is $\Delta D = 1$. Denote by $\delta x_S = x_S^{new} - x_S^{old}$ the increase of a variable for a specific set S. If an element is covered by the prediction then it holds that:

$$\begin{aligned} \Delta P &= \sum_{S \in \mathcal{F}(e)} w_S \cdot \delta x_S = \sum_{S \in \mathcal{F}(e) \cap \mathcal{A}} w_S \cdot \delta x_S + \sum_{S \in \mathcal{F}(e) \setminus \mathcal{F}(e) \cap \mathcal{A}} w_S \cdot \delta x_S = \\ &= \sum_{S \in \mathcal{F}(e)} \left(x_S^{old} + \frac{\epsilon}{|\mathcal{F}(e)|}\right) + \sum_{S \in \mathcal{F}(e) \cap \mathcal{A}} \frac{(1-\epsilon)}{|\mathcal{F}(e) \cap \mathcal{A}|} = \sum_{S \in \mathcal{F}(e)} x_S^{old} + \epsilon + 1 - \epsilon \leq 2 \end{aligned}$$

By repeating the same calculation we get that if an element is uncovered by the prediction then:

$$\Delta P = \sum_{S \in \mathcal{F}(e)} w_S \cdot \delta x_S = \sum_{S \in \mathcal{F}(e)} \left(x_S^{old} + \frac{1}{|\mathcal{F}(e)|}\right) = \sum_{S \in \mathcal{F}(e)} x_S^{old} + 1 \leq 2$$

Overall we have that:

1. At any iteration $\frac{\Delta P}{\Delta D} \leq 2$.
2. The final primal solution is feasible.
3. By Lemma 11.2, denoting y the final dual solution, $\frac{y}{\Theta(\log(d/\epsilon)}$ is feasible.

Thus, by weak duality we get that the competitive ratio of Algorithm 3 is upper bounded by $2 \cdot O(\log(d/\epsilon)) = O(\log(d/\epsilon)))$. □

Lemma 11.4 (Consistency)**.** $\text{cost}_{\mathcal{PDLA}}(\mathcal{I}, \mathcal{A}, \epsilon) \leq O\left(\frac{1}{1-\epsilon}\right) \cdot S(\mathcal{A}, \mathcal{I}) + O(\log(d)) \cdot C_{nc}$

Proof. We split the analysis in two parts. First, we look at the case when an element which is uncovered by the prediction arrives. In this case Algorithm 3 emulates the pure online algorithm ($\epsilon = 1$). More precisely, by the same calculations as before, we can show that y_{nc} the solution of the dual problem restricted to the uncovered elements satisfy the property that

$\frac{y_{nc}}{O(\log d)}$ is feasible. Therefore for those elements by Lemma 11.3 the cost of Algorithm 3 is upper bounded by $O(\log d) \cdot C_{nc}$. We turn our attention to the more interesting case where the prediction covers an element. In this case, after the execution of the while loop we decompose the primal increase into two parts. ΔP_c which denotes the increase due to sets S chosen by $\mathcal{A}$ ($\mathbb{1}\{S \in \mathcal{A}\} = 1$) and ΔP_u which denotes the increase due to sets S not chosen by the prediction ($\mathbb{1}\{S \in \mathcal{A}\} = 0$), thus we have $\Delta P = \Delta P_c + \Delta P_u$. Let $c = \{S \in \mathcal{F}(e) : \mathbb{1}\{S \in \mathcal{A}\} = 1\}$ and $u = \{S \in \mathcal{F}(e) : \mathbb{1}\{S \in \mathcal{A}\} = 0\}$. We then have:

$$\Delta P_c = \sum_{S \in c} x_S + \frac{\epsilon \cdot |c|}{|c| + |u|} + 1 - \epsilon \geq \frac{\epsilon}{d} + 1 - \epsilon$$

$$\Delta P_u = \sum_{S \in u} x_S + \frac{\epsilon \cdot |u|}{|c| + |u|} \leq 1 + \epsilon$$

since , $\frac{|c|}{|c|+|u|} \geq \frac{1}{d}$ and $\frac{|u|}{|c|+|u|} \leq 1$. Combining the two bounds we get that $\Delta P_u \leq \frac{1+\epsilon}{\frac{\epsilon}{d}+1-\epsilon} \cdot \Delta P_c$ and consequently:

$$\Delta P \leq \left(1 + \frac{1+\epsilon}{\frac{\epsilon}{d} + 1 - \epsilon}\right) \Delta P_c = O\left(\frac{1}{1-\epsilon}\right) \Delta P_c$$

Since the cost increase ΔP_c is caused by sets which are selected by the prediction, we can charge this cost to the corresponding increase of $S(\mathcal{A}, \mathcal{I})$ losing only a multiplicative $O(1)$ factor. By combining the two cases we conclude the proof. □

12 PDLA in action

12.1 The ski rental problem

As another application of PDLA we design a learning-augmented algorithm for one of the simplest and well studied online problems, the ski rental problem. In this problem, every new day, one has to decide whether to rent skis for this day, which costs 1 dollar or to buy skis for the rest of the vacation at a cost of B dollars. In its offline version the total number of vacation days, N, is known in advance and the problem becomes trivial. From the primal dual formulation of the problem (Figure 12.1) it is clear that if $B < N$, the optimal strategy is to buy the skis at day one while if $B \geq N$ the optimal strategy is to always rent. In the online setting the difficulty relies in the fact that we do not know N in advance. A deterministic 2-competitive online algorithm has been known for a long time [58] and a randomized $\frac{e}{e-1} \approx 1.58$-competitive algorithm was also designed later [59]. Both competitive ratios are known to be optimal for deterministic and randomized algorithms respectively. This problem was already studied in various learning-augmented settings [47, 63, 80, 85]. Our approach recovers, using the primal-dual method, the results of [80]. As in [80] our prediction $\mathcal{A}$ will be the total number of vacation days N^{pred}.

Primal
minimize $B \cdot x + \sum_{j \in [N]} f_j$
subject to: $x + f_j \geq 1 \quad \forall j \in [N]$
$x, f_j \geq 0 \quad \forall j \in [N]$
Dual
maximize $\sum_{j \in [N]} y_j$
subject to: $\sum_{j \in [N]} y_j \leq B$
$1 \geq y_j \geq 0 \quad \forall j \in [N]$

Figure 12.1: Primal dual formulation of the ski rental problem.

PDLA for ski rental: To simplify the description, we denote an instance of the problem as $\mathcal{I} = (N, B)$ and define the function $e(z) = (1 + 1/B)^{z \cdot B}$. Note that if $B \to \infty$, then $e(z)$ approaches e^z hence the choice of notation. In an integral solution, the variable x is 1 to indicate that the skis are bought and 0 otherwise. In the same spirit f_j indicates whether we rent on day j or not. Buchbinder et al. [18] showed how to easily turn a fractional monotone solution (i.e. it is not permitted to decrease a variable) to an online randomized algorithm of expected cost equal to the cost of the fractional solution. Hence we focus only on building online a fractional solution. Algorithm 4 is due to [18] and uses the Primal-Dual method to solve the problem. Each new day j a new constraint $x + f_j \geq 1$ is revealed. To satisfy this constraint, the algorithm updates the primal and dual variables while trying to maintain (1) the ratio $\Delta P / \Delta D$ as small as possible and (2) the primal and dual solutions feasible. As in the online weighted set cover problem, the key idea for extending Algorithm 4 to the learning-augmented Algorithm 5 is to use the prediction

N^{pred} in order to adjust the rate at which each variable is increased. Thus, when $N^{pred} > B$ we increase the buying variable more aggressively than the pure online algorithm. Here, the cost of following blindly the prediction N^{pred} is $S(N^{pred},\mathcal{I}) = B \cdot \mathbb{1}\{N^{pred} > B\} + N \cdot \mathbb{1}\{N^{pred} \leq B\}$.

Algorithm 4 Primal-Dual for Ski Rental [18].

Initialize: $x \leftarrow 0$, $f_j \leftarrow 0$, $\forall j$
$c \leftarrow e(1)$, $c' \leftarrow 1$
for each new day j s.t. $x + f_j < 1$ **do**
  / Primal Update */*
  $f_j \leftarrow 1 - x$
  $x \leftarrow (1 + \frac{1}{B})x + \frac{1}{(c-1)\cdot B}$
  / Dual Update */*
  $y_j \leftarrow c'$
end for

⇒

Algorithm 5 PDLA for Ski Rental.

Input: ϵ, N^{pred}
Initialize: $x \leftarrow 0$, $f_j \leftarrow 0$, $\forall j$
if $N^{pred} \geq B$ **then**
  / Prediction suggests buying */*
  $c \leftarrow e(\epsilon)$, $c' \leftarrow 1$
else
  / Prediction suggests renting */*
  $c \leftarrow e(1/\epsilon)$, $c' \leftarrow \epsilon$
end if
for each new day j s.t. $x + f_j < 1$ **do**
  / Primal Update */*
  $f_j \leftarrow 1 - x$
  $x \leftarrow (1 + \frac{1}{B})x + \frac{1}{(c-1)\cdot B}$
  / Dual Update */*
  $y_j \leftarrow c'$
end for

In the following we assume that either ϵB or B/ϵ is an integer (depending on whether c equals $e(\epsilon)$ or $e(1/\epsilon)$ respectively in Algorithm 5). Our results do not change qualitatively by rounding up to the closest integer.

Theorem 12.1 (PDLA for ski rental)**.** *For any $\epsilon \in (0, 1]$, the cost of PDLA for ski rental is bounded as follows*

$$\underset{\mathcal{PDLA}}{\text{cost}}(\mathcal{I}, N^{pred}, \epsilon) \leq \min\left\{\frac{\epsilon}{1 - e(-\epsilon)} \cdot S(N^{pred},\mathcal{I}), \frac{1}{1 - e(-\epsilon)} \cdot \text{OPT}(\mathcal{I})\right\}$$

Theorem 12.1 is proved using similar arguments to the proof of Theorem 11.1 for the weighted set cover problem. The robustness bound follows essentially using the same proof as for the original analysis of Algorithm 4 in [18] and for the consistency bound we just calculate the total primal increase. We first prove an easy lemma about the feasibility of the dual solution.

Lemma 12.2. *Let y be the dual solution built by Algorithm 5. Then y is a feasible solution (assuming $\frac{B}{\epsilon}$ is integral if the prediction suggests to rent).*

Proof. To see this, note that the only constraint that might by violated is the constraint $\sum_{j\in[N]} y_j \leq B$. Denote by S the value of the sum $\sum_{j\in[N]} y_j$. Note that once $x \geq 1$, the value of S will never change anymore. The value of S increases by 1 for every big update and by ϵ for every small update. In the case $N^{pred} > B$, the algorithm always does big updates (the prediction suggest to buy). We claim that at most $\lceil \epsilon B \rceil$ big updates can be made before $x \geq 1$. We denote $x(k)$ the value of x after k updates. We then prove by induction that $x(k) \geq \frac{e(k/B)-1}{e(\epsilon)-1}$ (recall that $e(z) = (1 + 1/B)^{z\cdot B} \approx e^z$). Clearly, if $k = 0$, we have $x(0) \geq 0$. Now assume this is

the case for k updates we then have

$$\begin{aligned} x(k+1) &= \left(1+\frac{1}{B}\right)\cdot x(k) + \frac{1}{(e(\epsilon)-1)\cdot B} \\ &\geq \left(1+\frac{1}{B}\right)\cdot \frac{e(k/B)-1}{e(\epsilon)-1} + \frac{1}{(e(\epsilon)-1)\cdot B} \\ &= \frac{(1+1/B)\cdot(e(k/B)-1)+1/B}{e(\epsilon)-1} \\ &= \frac{e((k+1)/B)-1}{e(\epsilon)-1} \end{aligned}$$

which ends the induction. Hence at most $\lceil \epsilon B \rceil \leq B$ big updates can be made before $x \geq 1$. This implies that $S \leq B$ at the end of the algorithm. In the case where $N^{pred} \leq B$, we prove in exactly the same way that at most $\lceil \frac{B}{\epsilon} \rceil$ updates are performed before $x \geq 1$. Hence we have that $S \leq \epsilon \cdot \lceil \frac{B}{\epsilon} \rceil$. By assumption, we have that B/ϵ is an integer hence $S \leq 1$ and y is again feasible. □

We can now prove Theorem 12.1

Proof. We prove first the robustness bound. By Lemma 12.2, we know that the dual solution is feasible. Hence what remains to prove is to upper bound the ratio $\frac{\Delta P}{\Delta D}$ and use weak duality. In the case of a big update we have

$$\frac{\Delta P}{\Delta D} = \Delta P = 1 + \frac{1}{e(\epsilon)-1} = \frac{1}{1-e(-\epsilon)}$$

In the case of a small update we have

$$\frac{\Delta P}{\Delta D} = \frac{\Delta P}{\epsilon} = \frac{1}{\epsilon}\cdot\frac{1}{1-e(-1/\epsilon)} \leq \frac{1}{1-e(-\epsilon)}$$

where the last inequality comes from Lemma B.1 inequality (B.1). By weak duality, we have the robustness bound.

To prove consistency, we have two cases. If $N^{pred} \leq B$, then Algorithm 5 does at most N updates, each of cost at most $\frac{1}{1-e(-1/\epsilon)}$ while the prediction $\mathcal{A}$ pays a cost of N. Noting again that, by Lemma B.1, $\frac{1}{1-e(-1/\epsilon)} \leq \frac{\epsilon}{1-e(-\epsilon)}$ ends the proof of consistency in this case. The other case is different. As in the proof of Lemma 12.2, we still have that $x(k) \geq \frac{e(k/B)-1}{e(\epsilon)-1}$ hence at most $\lceil \epsilon B \rceil \leq B$ updates are done by Algorithm 5, each of cost at most $\frac{1}{1-e(-\epsilon)}$ hence a total cost of at most

$$\frac{\lceil \epsilon B \rceil}{1-e(-\epsilon)}$$

Since we assume in this case that ϵB is integral and that the prediction $\mathcal{A}$ pays a cost of B, the competitive ratio is indeed $\frac{\epsilon}{1-e(-\epsilon)}$ □

In addition to recovering the positive results of [80], we additionally show in Lemma 12.3 that this consistency-robustness trade-off is optimal.

Lemma 12.3. *Any $\frac{\epsilon}{1-e^{-\epsilon}}$-consistent learning-augmented algorithm for ski rental has robustness $R(\epsilon) \geq \frac{1}{1-e^{-\epsilon}}$*

Proof. For simplicity, we will consider the ski rental problem in the continuous case which corresponds to the behaviour of the discrete version when $B \to \infty$. In this problem, the cost of buying is 1 and a randomized algorithm has to define a (buying) probability distribution $\{p_t\}_{t\geq 0}$. Moreover, consider the case where the true number of vacation days $t_{end} \in [0,1] \cup (2,\infty)$. In such a case we can assume w.l.o.g. that $p_t = 0, \forall t > 1$. Indeed moving buying probability mass from any $p_t, t > 1$ to p_1 does not increase the cost of the randomized algorithm. Assume now that the prediction suggests us that the end of vacations is at $\hat{t}_{end} > 2$, thus the optimal offline solution, if the prediction is correct, is to buy the skis in the beginning for a total cost of 1. Since the algorithm has to define a probability distribution in $[0,1]$, $\{p_t\}$ needs to satisfy the equality constraint $\int_0^1 p_t dt = 1$. Moreover, note that when the prediction is correct, i.e. $t_{end} > 2$, the LA algorithm suffers an expected cost of $\int_0^1 (t+1)p_t dt$ while the optimum offline has a cost of 1. Thus the consistency requirement forces the distribution to satisfy the inequality $\int_0^1 (t+1)p_t dt \leq \frac{\epsilon}{1-e^{-\epsilon}}$. Now assume that the best possible LA algorithm is c-robust. If $t_{end} \leq 1$ then the LA algorithm's cost is $\int_0^{t_{end}} (t+1)p_t dt + t_{end} \int_{t_{end}}^1 p_t dt$ while the optimum offline cost is t_{end}. Thus, due to c-robustness we have that for every $t' \in [0,1]$, $\int_0^{t'} (t+1)p_t dt + t' \int_{t'}^1 p_t dt \leq ct'$. We calculate the best possible robustness c with the following LP:

Figure 12.2: Primal Robustness for ski rental problem.

Primal
minimize c
subject to: $\int_0^1 p_t dt = 1$
$\int_0^1 (t+1)p_t dt \leq \frac{\epsilon}{1-e^{-\epsilon}}$
$\int_0^{t'} (t+1)p_t dt + t' \int_{t'}^1 p_t dt \leq ct' \quad \forall t' \in [0,1]$
$p_t \geq 0 \quad \forall t' \in [0,1]$

To lower bound the best possible robustness c we will present a feasible solution to the dual of Figure 12.2. The dual variables ϵ_d and ϵ_c correspond respectively to the first and second primal constraints in Figure 12.2. The dual variables ϵ_t, $\forall t \in [0,1]$ correspond to the robustness constraints described in the third line of the primal.

The corresponding dual is:

Figure 12.3: Dual Robustness for ski rental problem.

Dual
maximize $\epsilon_d - \epsilon_c \cdot \frac{\epsilon}{1-e^{-\epsilon}}$
subject to: $\int_0^1 t\epsilon_t dt \leq 1$
$\epsilon_d - (t'+1)\epsilon_c \leq \int_0^{t'} t\epsilon_t dt + (t'+1)\int_{t'}^1 \epsilon_t dt \quad \forall t' \in [0,1]$
$\epsilon_c, \epsilon_t \geq 0 \quad \forall t \in [0,1]$

Let $K = \frac{1}{1-\epsilon e^{-\epsilon} - e^{-\epsilon}}$. Then, $\epsilon_t = K \cdot e^{-t} \cdot \mathbb{1}\{t \leq \epsilon\}$, $\epsilon_d = K$ and $\epsilon_c = K \cdot e^{-\epsilon}$.

We first prove that this dual solution is feasible. For the first constraint notice that

$$\int_0^1 t\epsilon_t dt = K \cdot \int_0^\epsilon te^{-t}dt = K \cdot \left(1 - (\epsilon+1)e^{-\epsilon}\right) = 1$$

For the second type of constraint first in the case $t' > \epsilon$ we get

$$\int_0^{t'} t\epsilon_t dt + (t'+1)\int_{t'}^1 \epsilon_t dt = \int_0^\epsilon t\epsilon_t dt = 1$$

and we note that

$$\epsilon_d - (t'+1)\epsilon_c \leq \epsilon_d - (\epsilon+1)\epsilon_c = K \cdot \left(1 - (\epsilon+1)e^{-\epsilon}\right) = 1$$

hence these constraints are satisfied.

In the second case $t' \leq \epsilon$, we have that

$$\begin{aligned}
\int_0^{t'} t\epsilon_t dt + (t'+1)\int_{t'}^1 \epsilon_t dt &= K \cdot \left(\int_0^{t'} te^{-t}dt + (t'+1)\int_{t'}^\epsilon e^{-t}dt\right) \\
&= K \cdot \left(1 - (t'+1)e^{-t'} + (t'+1)(e^{-t'} - e^{-\epsilon})\right) \\
&= K \cdot \left(1 - (t'+1)e^{-\epsilon}\right) \\
&= \epsilon_d - (t'+1)\epsilon_c
\end{aligned}$$

which proves that these constraints are also satisfied. Hence this dual solution is feasible. Finally note that the cost of this dual solution is

$$\begin{aligned}
\epsilon_d - \epsilon_c \cdot \frac{\epsilon}{1-e^{-\epsilon}} &= K \cdot \left(1 - \frac{\epsilon}{1-e^{-\epsilon}} \cdot e^{-\epsilon}\right) \\
&= K \cdot \frac{1 - e^{-\epsilon} - \epsilon e^{-\epsilon}}{1-e^{-\epsilon}} = \frac{1}{1-e^{-\epsilon}}
\end{aligned}$$

By weak duality, we conclude that the best robustness cannot be better than $\frac{1}{1-e^{-\epsilon}}$ □

12.2 Dynamic TCP acknowledgement

In this section, we continue by applying PDLA to a classic network congestion problem of the Transmission Control Protocol (TCP). During a TCP interaction, a server receives a stream of packets and replies back to the sender acknowledging that each packet arrived correctly. Instead of sending an acknowledgement for each packet separately, the server can choose to delay its response and acknowledge multiple packets simultaneously via a single TCP response. Of course, in this scenario there is an additional cost incurred due to the delayed packets, which is the total latency incurred by those packets. Thus, on one hand sending too many acknowledgments (acks) overloads the network, on the other hand sending one ack for all the packets slows down the TCP interaction. Hence a good trade-off has to be achieved and the objective function which we aim to minimize will be the sum of the total number of acknowledgements plus the total latency. The problem was first modeled by Dooly et al. [35], where they showed how to solve the offline problem optimally in quadratic time along with a deterministic 2-competitive online algorithm. Karlin et al. [60] provided the first $\frac{e}{e-1}$-competitive randomized algorithm which was later shown to be optimal by Seiden in [83]. The problem was later solved using the primal-dual method by Buchbinder et al. [18] who also obtained an $\frac{e}{e-1}$-competitive algorithm. Figure 12.4 presents the primal-dual formulation of the problem. In this formulation each packet j arrives at time $t(j)$ and is acknowledged by the first ack sent after $t(j)$. Here, variable x_t corresponds to sending an ack at time t and f_{jt} is set to one (in the integral solution) if packet j was not acknowledged by time t. The time granularity is controlled by the parameter d and each additional time unit of latency comes at a cost of $1/d$. As in the ski rental problem, there is no integrality gap and a fractional monotone solution can be converted to a randomized algorithm in a lossless manner (see [18] for more details).

Primal
minimize $\sum_{t\in T} x_t + \sum_{j\in M}\sum_{t\mid t\geq t(j)} \frac{1}{d} f_{jt}$
subject to: $f_{jt} + \sum_{k=t(j)}^{t} x_k \geq 1 \quad \forall j, t \geq t(j)$
$f_{jt} \geq 0 \quad \forall j, t \geq t(j)$
$x_t \geq 0 \quad \forall t \in T$
Dual
maximize $\sum_{j\in M}\sum_{t\mid t\geq t(j)} y_{jt}$
subject to: $\sum_{j\mid t\geq t(j)}\sum_{t'\geq t} y_{jt} \leq 1 \quad \forall t \in T$
$0 \leq y_{jt} \leq \frac{1}{d} \quad \forall j, t \geq t(j)$

Figure 12.4: Primal dual formulation of the TCP acknowledgement problem

PDLA for TCP ack: Our prediction consists in a collection of times $\mathcal{A}$ in which the prediction suggests sending an ack. Let $\alpha(t)$ be the next time $t' \geq t$ when prediction sends an ack. With this definition each packet j, if the prediction is followed blindly, is acknowledged at time $\alpha(t(j))$ incurring a latency cost of $(\alpha(t(j)) - t(j)) \cdot \frac{1}{d}$. In the same spirit as for the ski rental problem we adapt the pure online Algorithm 6 into the learning-augmented Algorithm 7. Algorithm 7 adjusts the rate at which we increase the primal and dual variables according to the prediction $\mathcal{A}$. Thus if a packet j at time t is "uncovered" ($\sum_{k=t(j)}^{t} x_k + f_{jt} < 1$) by our fractional solution and "covered" by $\mathcal{A}$ ($\alpha(t(j)) \leq t$) we increase x_t at a faster rate. To simplify the description of Algorithm 7 we define $e(z) = (1 + \frac{1}{d})^{z \cdot d}$. To get to the continuous time case, we will take the limit $d \to \infty$ so the reader should think intuitively as $e(z) \approx e^z$. In addition, we will call big and small updates, the updates where z is set to ϵ and $1/\epsilon$ respectively.

We continue by presenting the guarantees of Algorithm 7 together with their proof. As before, $\mathcal{I}$ denotes the TCP ack problem instance which is revealed in an online fashion.

Algorithm 6 PRIMAL-DUAL METHOD FOR TCP ACKNOWLEDGEMENT [18].

Initialize: $x \leftarrow 0$, $y \leftarrow 0$
for all times t **do**
  for all packages j such that $\sum_{k=t(j)}^{t} x_k < 1$ **do**
    $c \leftarrow e(1)$, $c' \leftarrow 1/d$
    / Primal Update */*
    $f_{jt} \leftarrow 1 - \sum_{k=t(j)}^{t} x_k$
    $x_t \leftarrow x_t + \frac{1}{d} \cdot \left(\sum_{k=t(j)}^{t} x_k + \frac{1}{c-1} \right)$
    / Dual Update */*
    $y_{jt} \leftarrow c'$
  end for
end for

⇒

Algorithm 7 PDLA FOR TCP ACKNOWLEDGEMENT

Input: ϵ, $\mathcal{A}$
Initialize: $x \leftarrow 0$, $y \leftarrow 0$
for all times t **do**
  for all packages j such that $\sum_{k=t(j)}^{t} x_k < 1$ **do**
    if $t \geq \alpha(t(j))$ **then**
      / Prediction already acknowledged packet j */*
      $c \leftarrow e(\epsilon)$, $c' \leftarrow 1/d$
    else
      / Prediction did not acknowledge packet j yet */*
      $c \leftarrow e(1/\epsilon)$, $c' \leftarrow \epsilon/d$
    end if
    / Primal Update */*
    $f_{jt} \leftarrow 1 - \sum_{k=t(j)}^{t} x_k$
    $x_t \leftarrow x_t + \frac{1}{d} \cdot \left(\sum_{k=t(j)}^{t} x_k + \frac{1}{c-1} \right)$
    / Dual Update */*
    $y_{jt} \leftarrow c'$
  end for
end for

Theorem 12.4 (PDLA for TCP ack)**.** *For any prediction $\mathcal{A}$, any instance $\mathcal{I}$ of the TCP ack problem, any parameter $\epsilon \in (0,1]$, and $d \to \infty$: Algorithm 7 outputs a fractional solution of cost at most* $c_{\mathcal{PDLA}}(\mathcal{I}, \mathcal{A}, \epsilon) \leq \min\left\{ \frac{\epsilon}{1-e^{-\epsilon}} \cdot S(\mathcal{A}, \mathcal{I}), \frac{1}{1-e^{-\epsilon}} \cdot \mathrm{OPT}(\mathcal{I}) \right\}$

We first analyze the consistency of Algorithm 7. To this end, denote by $n_{\mathcal{A}}$ the number of acknowledgements sent by $\mathcal{A}$ and by $\mathit{latency}(\mathcal{A})$ the latency paid by the prediction $\mathcal{A}$.

Lemma 12.5. *For any $\epsilon \in (0,1]$, $d > 0$,*

$$\underset{\mathcal{PDLA}}{\mathrm{cost}}(\mathcal{I}, \mathcal{A}, \epsilon) \leq n_{\mathcal{A}} \cdot \frac{1}{d} \cdot \frac{\lceil \epsilon d \rceil}{1 - e(-\epsilon)} + \mathit{latency}(\mathcal{A}) \cdot \frac{1}{1 - e(-1/\epsilon)}$$

Proof. We will use a charging argument to analyze the performance of Algorithm 7. Note that for a small update, the increase in cost of the fractional solution is

$$\Delta P = \frac{1}{d} \left(1 - \sum_{k=t(j)}^{t} x_k \right) + \frac{1}{d} \cdot \left(\sum_{k=t(j)}^{t} x_k + \frac{1}{e(1/\epsilon) - 1} \right) = \frac{1}{d} \cdot \frac{1}{1 - e(-1/\epsilon)}$$

However, for every small update that is made, it must be that the prediction $\mathcal{A}$ pays a latency of at least $\frac{1}{d}$. Hence the total cost of small updates made by Algorithm 7 is at most $\mathit{latency}(\mathcal{A}) \cdot \frac{1}{1-e(-1/\epsilon)}$.

Secondly we bound the total cost of big updates of our algorithm. Let t_0 be a time at which $\mathcal{A}$ sends an acknowledgment. Let Y be the set of big updates made because of jobs j that are acknowledged at time t_0 by $\mathcal{A}$ (these big updates are hence made at some time $t \geq t_0$). We claim that $|Y| \leq \lceil \epsilon d \rceil$.

To prove this denote by $S(l)$ the value of $\sum_{k=t_0}^{+\infty} x_k$ after l such big updates (there might be small updates influencing this value but only to make it bigger). Notice that once $\sum_{k=t_0}^{+\infty} x_k \geq 1$ there is no remaining update in Y. We prove by induction that

$$S(l) \geq \frac{(1+1/d)^l - 1}{(1+1/d)^{\epsilon d} - 1}$$

This is clear for $l = 0$ as $S(0) \geq 0$. Now assume this is the case for some value l and apply a big update at time t for job j to get

$$\begin{aligned}
S(l+1) &= S(l) + \frac{1}{d} \cdot \left(\sum_{k=t(j)}^{t} x_k + \frac{1}{e(\epsilon) - 1} \right) \\
&\geq S(l) \cdot (1 + 1/d) + \frac{1}{d(e(\epsilon) - 1)} \\
&= \frac{(1+1/d)^{l+1} - 1 - 1/d}{(1+1/d)^{\epsilon d} - 1} + \frac{1/d}{(e(\epsilon) - 1)} \\
&= \frac{(1+1/d)^{l+1} - 1 - 1/d}{(1+1/d)^{\epsilon d} - 1} + \frac{1/d}{(1+1/d)^{\epsilon d} - 1} \\
&= \frac{(1+1/d)^{l+1} - 1}{(1+1/d)^{\epsilon d} - 1}
\end{aligned}$$

Where the second inequality comes from noting that since we are considering an update due to a request j acknowledged at time t_0 by the predicted solution, it must be that $t(j) \leq t_0$ and $\sum_{k=t(j)}^{t} x_k \geq \sum_{k=t_0}^{t} x_k$. Hence we get that $S(\lceil \epsilon d \rceil) \geq 1$ which implies that $|Y| \leq \lceil \epsilon d \rceil$.

By a similar calculation as for the small update case, we have that the cost of a big update is

$$\Delta P = \frac{1}{d} \cdot \frac{1}{1 - e(-\epsilon)}$$

Hence the total cost of these updates in Y is charged to the acknowledgement that $\mathcal{A}$ pays at time t_0 to finish the proof. □

Taking the limit $d \to +\infty$, noting that $S(\mathcal{A}, \mathcal{I}) = n_{\mathcal{A}} + \mathit{latency}(\mathcal{A})$ and using Equation (B.1) from Lemma B.1 we get the following corollary:

Corollary 12.1. *For any $\epsilon \in (0, 1]$ and taking $d \to +\infty$, we have that*

$$\underset{\mathcal{PDLA}}{\text{cost}}(\mathcal{I}, \mathcal{A}, \epsilon) \leq n_{\mathcal{A}} \cdot \frac{\epsilon}{1 - e^{-\epsilon}} + \mathit{latency}(\mathcal{A}) \cdot \frac{1}{1 - e^{-1/\epsilon}} \leq \frac{\epsilon}{1 - e^{-\epsilon}} \cdot S(\mathcal{A}, \mathcal{I})$$

We then prove robustness of Algorithm 7 with the following lemmas.

Lemma 12.6. *Let y be the dual solution produced by Algorithm 7. Then $\frac{y}{1+1/d}$ is feasible.*

Proof. Notice that the constraints of the second type (i.e. $0 \leq y_{jt} \leq 1/d$) are always satisfied since $0 < \epsilon \leq 1$. We now check that the second constraints are almost satisfied (within some factor $(1 + 1/d)$). Fix a time $t \in T$ and consider the corresponding constraint:

$$\sum_{j|t \geq t(j)} \sum_{t' \geq t} y_{jt} \leq 1$$

Note that for a small update for some job j such that $t(j) \leq t$ the sum above increases by ϵ/d while it increases by $1/d$ for a big update. Notice that once we have that $\sum_{t' \geq t} x_{t'} \geq 1$, no more such update will be performed. Denote by S the value of this sum.

Notice that for a big update, the sum S becomes $\left(1 + \frac{1}{d}\right) \cdot S + \frac{1}{d((1+1/d)^{\epsilon d} - 1)}$. Similarly, for a small updates it becomes $\left(1 + \frac{1}{d}\right) \cdot S + \frac{1}{d((1+1/d)^{d/\epsilon} - 1)}$.

Hence, if we denote by s the number of small updates in this sum and by b the number of big updates, by Lemma B.2 we have that if $\epsilon s + b \geq d$ then $\sum_{t' \geq t} x_{t'} \geq 1$. This directly implies that the value of $\sum_{j|t \geq t(j)} \sum_{t' \geq t} y_{jt}$ is at most $1 + 1/d$ at the end of the algorithm (each update in the dual is of value at most $1/d$).

Therefore scaling down all y_{jt} by a multiplicative factor of $1 + 1/d$ yields a feasible solution to the dual. □

Lemma 12.7. *For $d \to +\infty$, Algorithm 7 outputs a solution of cost at most $\frac{1}{1-e^{-\epsilon}} \cdot \mathrm{OPT}(\mathcal{I})$*

Proof. We first compare the increase ΔP in the primal value to the increase ΔD in the dual value at every update. We claim that for every update we have

$$\frac{\Delta P}{\Delta D} \leq \frac{1}{1 - e(-\epsilon)}$$

In the case of a big update we directly have $\Delta P = \frac{1}{d}\left(1 + \frac{1}{e(\epsilon)-1}\right) = \frac{1}{d} \cdot \frac{1}{1-e(-\epsilon)}$ and $\Delta D = \frac{1}{d}$. In the case of a small update we have $\Delta D = \frac{\epsilon}{d}$ and $\Delta P = \frac{1}{d}\left(1 + \frac{1}{e(1/\epsilon)-1}\right) = \frac{1}{d} \cdot \frac{1}{1-e(-1/\epsilon)}$ and we conclude applying Lemma B.1 (inequality (B.2)) that we always have

$$\frac{\Delta P}{\Delta D} \leq \frac{1}{1 - e(-\epsilon)}$$

By lemma 12.6, $\frac{y}{1+1/d}$ is a feasible solution. Hence taking $d \to +\infty$ together with the previous remark and weak duality we get the result. □

Combining Corollary 12.1 and Lemma 12.7 yields Theorem 12.4.

12.3 The Bahncard problem

To further emphasize the generality of PDLA , we apply the same ideas to a generalization of the ski rental problem, namely, the Bahncard problem [43]. This problem models a situation where a tourist travels every day multiple trips. Before any new trip, the tourist has two choices, either to buy a ticket for that particular trip at a cost of 1 or buy a discount card, at a cost of B, that allows to buy tickets at a cheaper price of $\beta < 1$. The discount card remains valid during T days. Note that ski rental is modeled by taking $\beta = 0$ and $T \to \infty$. Karlin et al. [60] designed an optimal randomized online algorithm of competitive ratio $\frac{e}{e-1+\beta}$ when $B \to \infty$.

PDLA for the Bahncard problem: In the learning-augmented version of the problem we are given a prediction $\mathcal{A}$ which consists in a collection of times where we are advised to acquire the discount card. Using PDLA, we design a learning-augmented algorithm for the Bahncard problem with the guarantees of Theorem 12.8. An interesting feature of our algorithm is that, as for the TCP ack problem, it does not need to be given the full prediction in advance. If Bahncards are bought by the prediction $\mathcal{A}$ at a set of times $\{t_1, t_2, \ldots, t_k\}$, the algorithm does not need to know before time t_i that the Bahncard i is bought. For instance we could think of the prediction of an employee of the station giving short-term advice to a traveller every time he shows up at the station.

Theorem 12.8. *[PDLA for the Bahncard problem] For any $\epsilon \in (0, 1]$, any $\beta \in [0, 1]$ and $\frac{B}{1-\beta} \to \infty$, we have the following guarantees on any instance $\mathcal{I}$ and prediction $\mathcal{A}$*

$$\underset{\mathcal{PDLA}}{\text{cost}}(\mathcal{I}, \mathcal{A}, \epsilon) \leq \min\left\{\frac{\epsilon}{1-\beta+\epsilon\beta} \cdot \frac{e^\epsilon - \beta}{e^\epsilon - 1} \cdot S(\mathcal{A}, I), \frac{e^\epsilon - \beta}{e^\epsilon - 1} \cdot \text{OPT}(\mathcal{I})\right\}$$

We now give the primal dual formulation of the Bahncard problem along with its corresponding learning-augmented algorithm. We mention that, to the best of our knowledge, no online algorithm using the Primal-Dual method was designed before. Hence the primal-dual formulation (Figure 12.5) of the problem is new. In an integral solution, we would have $x_t = 1$ if the solution buys a Bahncard at time t and $x_t = 0$ otherwise. Then f_j represents the fractional amount of trip j done at time $t(j)$ that is bought at full price and d_j the amount of the trip bought at discounted price. The first natural constraint is the one that says that each trip should be paid entirely either in discounted or full price, i.e. $d_j + f_j \geq 1$. We then have the constraint $\sum_{t=t(j)-T}^{t(j)} x_t \geq d_j$ that says that to be able to buy a ticket at discounted price, at least one Bahncard must have been bought in the last T time steps.

Figure 12.5: Primal dual formulation of the Bahncard problem.

Primal	**Dual**
minimize $B \cdot \sum_{t \in T} x_t + \sum_{j \in M} \beta d_j + f_j$	maximize $\sum_{j \in M} c_j$
subject to: $d_j + f_j \geq 1 \quad \forall j$	subject to: $c_j \leq 1 \quad \forall j$
$\sum_{t=t(j)-T}^{t(j)} x_t \geq d_j \quad \forall j$	$c_j - b_j \leq \beta \quad \forall j$
$x_t \geq 0 \quad \forall t \in \mathcal{T}$	$\sum_{j : t(j)-T \leq t \leq t(j)} b_j \leq B \quad \forall t \in \mathcal{T}$
$d_j, f_j \geq 0 \quad \forall j$	$c_j, b_j \geq 0 \quad \forall j$

Following the same idea as for the ski rental and the TCP ack problem, we will guide the

updates in the primal-dual algorithm with the advice provided. We define a function $e(z) = \left(1 + \frac{1-\beta}{B}\right)^{z\cdot(B/(1-\beta))}$. Again for $\frac{B}{1-\beta} \to \infty$, the reader should think intuitively of $e(z)$ as e^z. The parameter z will then take values either ϵ or $1/\epsilon$ depending on if we want to do a big or small update in the primal. As for ski rental, when we do a small update, we will need to scale down the dual update by a factor of ϵ to maintain feasibility of the dual solution.

The rule to decide if an update should be big or small is the following: if the prediction $\mathcal{A}$ bought a Bahncard less than T time steps in the past (i.e. if the predicted solution has currently a valid Bahncard) the update should be big. Otherwise the update should be cautious. In Algorithm 8, we denote by $l_{\mathcal{A}}(t)$ the latest time before time t at which the prediction $\mathcal{A}$ bought a Bahncard. We use the convention that $l_{\mathcal{A}}(t) = -\infty$ if no Bahncard was bought before time t. Of course in this problem it is possible that trips show up while the fractional solution already has a full Bahncard available (i.e. $\sum_{t=t(j)-T}^{t(j)} x_t \geq 1$). In this case there is no point in buying more fractions of a Bahncard and the algorithm will do what we call a *minimal* update.

Algorithm 8 PDLA for the Bahncard problem

Input: ϵ, $\mathcal{A}$
Initialize: $x, d, f \leftarrow 0$, $c, b \leftarrow 0$
for all trip j **do**
  if $\sum_{t=t(j)-T}^{t(j)} x_t \geq 1$ **then**
    $d_j \leftarrow 1$
    $c_j \leftarrow \beta$
  end if
  if $\sum_{t=t(j)-T}^{t(j)} x_t < 1$ **then**
    if $t(j) \leq l_{\mathcal{A}}(t(j)) + T$ **then**
      $d_j \leftarrow \sum_{t=t(j)-T}^{t(j)} x_t$
      $f_j \leftarrow 1 - d_j$
      $x_{t(j)} \leftarrow x_{t(j)} + \frac{1-\beta}{B} \cdot \left(\sum_{t=t(j)-T}^{t(j)} x_t + \frac{1}{e(\epsilon)-1}\right)$
      $b_j \leftarrow 1 - \beta$
      $c_j \leftarrow b_j + \beta$
    end if
    if $t(j) > l_{\mathcal{A}}(t(j)) + T$ **then**
      $d_j \leftarrow \sum_{t=t(j)-T}^{t(j)} x_t$
      $f_j \leftarrow 1 - d_j$
      $x_{t(j)} \leftarrow x_{t(j)} + \frac{1-\beta}{B} \cdot \left(\sum_{t=t(j)-T}^{t(j)} x_t + \frac{1}{e(1/\epsilon)-1}\right)$
      $b_j \leftarrow \epsilon(1 - \beta)$
      $c_j \leftarrow b_j + \beta$
    end if
  end if
end for

We first prove that the dual built by the algorithm is almost feasible.

Lemma 12.9. *Let (c, b) be the dual solution built by Algorithm 8, then $\frac{(c,b)}{1+(1-\beta)/B}$ is feasible.*

Proof. Note that the constraints $c_j \leq 1$ and $c_j - b_j \leq \beta$ are clearly maintained by Algorithm 8.

And scaling down both c and b by some factor bigger than 1 will not alter their feasibility. Hence we focus only on the constraints of the form $\sum_{j:t(j)-T\leq t\leq t(j)} b_j \leq B$ for a fixed time t. Note that during a minimal update, the value of b_j is not changed hence only small or big updates can alter the value of the sum $\sum_{j:t(j)-T\leq t\leq t(j)} b_j$. Similarly as for proofs in ski rental, denote by b the number of big updates that are counted in this sum and by s the number of small updates in this sum.

We first notice that once we have that $\sum_{t'=t}^{t+T} x_{t'} \geq 1$, no updates that alter the constraint $\sum_{j:t(j)-T\leq t\leq t(j)} b_j$ can happen. To see this, note that upon arrival of a trip j between time t and $t+T$, we have $\sum_{t=t(j)-T}^{t(j)} x_t \geq \sum_{t'=t}^{t(j)} x_{t'} = \sum_{t'=t}^{t+T} x_{t'}$.

Denote by S the value of the sum $\sum_{t'=t}^{t+T} x_{t'}$. Note that for a big update, we have that the value of the sum S is increased to at least $S \cdot \left(1+\frac{1-\beta}{B}\right) + \frac{1-\beta}{B} \cdot \frac{1}{e(\epsilon)-1}$. Similarly for a small update the new value of the sum is at least $S \cdot \left(1+\frac{1-\beta}{B}\right) + \frac{1-\beta}{B} \cdot \frac{1}{e(1/\epsilon)-1}$. Hence we can apply directly Lemma B.2 with $d = \frac{B}{1-\beta}$ to conclude that once $b+\epsilon s \geq \frac{B}{1-\beta}$, we have that $S \geq 1$.

Since for a big update, the sum $\sum_{j:t(j)-T\leq t\leq t(j)} b_j$ increases by $1-\beta$ and by $\epsilon(1-\beta)$ for a small update we can see that the first time the constraint $\sum_{j:t(j)-T\leq t\leq t(j)} b_j \leq B$ is violated, we have $S \geq 1$. Now since each update in the sum $\sum_{j:t(j)-T\leq t\leq t(j)} b_j$ is of value at most $1-\beta$ we can conclude that at the end of the algorithm, we have $\sum_{j:t(j)-T\leq t\leq t(j)} b_j \leq B+1-\beta$ hence the conclusion. □

We then prove robustness of Algorithm 8 by the following lemma.

Lemma 12.10 (Robustness)**.** *For any $\epsilon \in (0,1]$ and any $\beta \in [0,1]$, PDLA for the Bahncard problem is $\frac{(e(\epsilon)-\beta)\cdot(1+(1-\beta)/B)}{e(\epsilon)-1}$-robust.*

Proof. Algorithm 8 makes 3 possible types of updates. For a minimal update, we have $\Delta P = \Delta D = \beta$. For a small update we have

$$\begin{aligned}\Delta P &= (1-\beta)\cdot\left(\sum_{t=t(j)-T}^{t(j)} x_t + \frac{1}{e(1/\epsilon)-1}\right) + \beta\cdot\sum_{t=t(j)-T}^{t(j)} x_t + 1 - \sum_{t=t(j)-T}^{t(j)} x_t \\ &= 1 + \frac{1-\beta}{e(1/\epsilon)-1} = \frac{e(1/\epsilon)-\beta}{e(1/\epsilon)-1}\end{aligned}$$

and

$$\Delta D = \epsilon(1-\beta)+\beta = \beta(1-\epsilon)+\epsilon$$

hence the ratio is

$$\frac{\Delta P}{\Delta D} = \frac{1}{\beta(1-\epsilon)+\epsilon}\cdot\frac{e(1/\epsilon)-\beta}{e(1/\epsilon)-1}$$

Similarly in the case of a big update we have

$$\Delta P = \frac{e(\epsilon)-\beta}{e(\epsilon)-1}$$

and $\Delta D = 1$ which gives a ratio of

$$\frac{\Delta P}{\Delta D} = \frac{e(\epsilon) - \beta}{e(\epsilon) - 1}$$

We can conclude by Lemma B.1 (inequality (B.6)) that the ratio of primal cost increase vs dual cost increase is always bounded by

$$\frac{\Delta P}{\Delta D} \leq \frac{e(\epsilon) - \beta}{e(\epsilon) - 1}$$

Using Lemma 12.9 along with weak duality is enough to conclude that the cost of the fractional solution built by the algorithm is bounded as follows

$$\underset{\mathcal{PDLA}}{\text{cost}}(\mathcal{I}, \mathcal{A}, \epsilon) \leq \frac{(e(\epsilon) - \beta) \cdot (1 + (1 - \beta)/B)}{e(\epsilon) - 1} \cdot \text{OPT}(\mathcal{I})$$

which ends the proof. □

For consistency, we analyze the algorithm's cost in two parts. When the heuristic algorithm $\mathcal{A}$ buys its ith Bahncard at some time t_i, define the interval $I_i = [t_i, t_i + T]$ which represents the set of times during which this specific Bahncard is valid. This creates a family of intervals $I_1, \dots I_k$ if $\mathcal{A}$ buys k Bahncards. Note that we can assume that all these intervals are disjoint since if the prediction $\mathcal{A}$ suggests to buy a new Bahncard before the previous one expires, it is always better to postpone this buy to the end of the validity of the current Bahncard.

Lemma 12.11. *Denote by $(\Delta P)_{I_i}$ the increase in the primal cost of Algorithm 8 during interval I_i and by $cost(\mathcal{A})_{I_i}$ what prediction $\mathcal{A}$ pays during this same interval I_i (including the buy of the Bahncard at the beginning of the interval I_i). Then, for all i we have*

$$\frac{(\Delta P)_{I_i}}{cost(\mathcal{A})_{I_i}} \leq \frac{\left\lceil \epsilon \cdot \frac{B}{1-\beta} \right\rceil}{B + \beta \cdot \left\lceil \epsilon \cdot \frac{B}{1-\beta} \right\rceil} \cdot \frac{e(\epsilon) - \beta}{e(\epsilon) - 1}$$

Proof. Assume that m trips are requested during this interval I_i. Then we first have that $cost(\mathcal{A})_{I_i} = B + \beta m$ ($\mathcal{A}$ buys a Bahncard then pays a discounted price for every trip in the interval I_i).

As for Algorithm 8, for each trip j, we are possibly in the first two cases: either $\sum_{t=t(j)-T}^{t(j)} x_t \geq 1$ in which case the increase in the primal is $\Delta P = \beta$ or in the second case in which case the increase in the primal is

$$\Delta P = (1 - \beta) \cdot \left(\sum_{t=t(j)-T}^{t(j)} x_t + \frac{1}{e(\epsilon) - 1} \right) + \beta \cdot \sum_{t=t(j)-T}^{t(j)} x_t + 1 - \sum_{t=t(j)-T}^{t(j)} x_t = 1 + \frac{1 - \beta}{e(\epsilon) - 1}$$

We claim that the updates of the second case can happen at most $\left\lceil \epsilon \cdot \frac{B}{1-\beta} \right\rceil$ times during interval I_i. To see this, denote by $S(l)$ the value of $\sum_{t' \geq t_i} x_{t'}$ after l big updates in interval I_i. Note that once $\sum_{t' \geq t_i} x_{t'} \geq 1$, big updates cannot happen anymore. Hence all we need to prove is that $S\left(\left\lceil \epsilon \cdot \frac{B}{1-\beta} \right\rceil\right) \geq 1$.

We prove by induction that

$$S(k) \geq \frac{e(k \cdot (1-\beta)/B) - 1}{e(\epsilon) - 1}$$

This is indeed true for $k = 0$ as $S(0)$ is the value of $\sum_{t' \geq t_i} x_{t'}$ before any big update was made in I_i hence $S(0) \geq 0$. Now assume this is the case for some k and compute

$$\begin{aligned} S(k+1) &\geq \left(1 + \frac{1-\beta}{B}\right) \cdot S(k) + \frac{1-\beta}{B} \cdot \frac{1}{e(\epsilon) - 1} \\ &\geq \frac{\left(1 + \frac{1-\beta}{B}\right) \cdot (e(k \cdot (1-\beta)/B) - 1) + \frac{1-\beta}{B}}{e(\epsilon) - 1} \\ &\geq \frac{e((k+1) \cdot (1-\beta)/B) - 1}{e(\epsilon) - 1} \end{aligned}$$

which concludes the induction.

Hence on interval I_i, the total increase in the cost of the solution can be bounded as follows

$$(\Delta P)_{I_i} \leq \min\left\{\left\lceil \epsilon \cdot \frac{B}{1-\beta} \right\rceil, m\right\} \cdot \left(1 + \frac{1-\beta}{e(\epsilon) - 1}\right) + \max\left\{0, \left(m - \left\lceil \epsilon \cdot \frac{B}{1-\beta} \right\rceil\right)\right\} \cdot \beta$$

One can see that the worst case possible for the ratio $\frac{(\Delta P)_{I_i}}{\text{cost}(\mathcal{A})_{I_i}}$ is obtained for $m = \left\lceil \epsilon \cdot \frac{B}{1-\beta} \right\rceil$ and is bounded by

$$\frac{(\Delta P)_{I_i}}{\text{cost}(\mathcal{A})_{I_i}} \leq \frac{\left\lceil \epsilon \cdot \frac{B}{1-\beta} \right\rceil \cdot \left(1 + \frac{1-\beta}{e(\epsilon)-1}\right)}{B + \beta \cdot \left\lceil \epsilon \cdot \frac{B}{1-\beta} \right\rceil} = \frac{\left\lceil \epsilon \cdot \frac{B}{1-\beta} \right\rceil}{B + \beta \cdot \left\lceil \epsilon \cdot \frac{B}{1-\beta} \right\rceil} \cdot \frac{e(\epsilon) - \beta}{e(\epsilon) - 1}$$

□

We then consider times t that do not belong to any interval I_i. More precisely, we upper bound the value $(\Delta P)_j$ that is the increase in cost of the primal solution due to trip j such that $t(j)$ does not belong to any interval I_i. Note that in this case the prediction always pays a cost of 1.

Lemma 12.12. *For any trip j such that $t(j) \notin \bigcup_i I_i$, we have that*

$$(\Delta P)_j \leq \frac{e(1/\epsilon) - \beta}{e(1/\epsilon) - 1}$$

Proof. Note that Algorithm 8 pays either the cost of a small update which is $\frac{e(1/\epsilon)-\beta}{e(1/\epsilon)-1}$ or the cost of a minimal update which is β. □

For simplicity and better readability, the main theorem regarding the Bahncard problem is formulated for $\frac{B}{1-\beta} \to \infty$.

Theorem 12.8. *[PDLA for the Bahncard problem] For any $\epsilon \in (0,1]$, any $\beta \in [0,1]$ and $\frac{B}{1-\beta} \to \infty$, we have the following guarantees on any instance $\mathcal{I}$ and prediction $\mathcal{A}$*

$$\underset{\mathcal{PDLA}}{\text{cost}}(\mathcal{I}, \mathcal{A}, \epsilon) \leq \min\left\{ \frac{\epsilon}{1-\beta+\epsilon\beta} \cdot \frac{e^\epsilon - \beta}{e^\epsilon - 1} \cdot S(\mathcal{A}, I), \frac{e^\epsilon - \beta}{e^\epsilon - 1} \cdot \text{OPT}(\mathcal{I}) \right\}$$

Proof. By taking the limit in Lemma 12.10, we see that the cost of the solution output by Algorithm 8 is at most $\frac{e^\epsilon - \beta}{e^\epsilon - 1} \cdot \text{OPT}$ which proves the second bound in the theorem.

For the first bound, note that we can write the final cost of the solution as

$$cost_{\mathcal{PDLA}}(\mathcal{A}, \mathcal{I}, \epsilon) = \Delta P = \sum_i (\Delta P)_{I_i} + \sum_{j : t(j) \notin \bigcup_i I_i} (\Delta P)_j$$

By taking the limit in Lemma 12.11 we get that

$$\sum_i (\Delta P)_{I_i} \leq \frac{\epsilon}{1-\beta+\beta\epsilon} \cdot \frac{e^\epsilon - \beta}{e^\epsilon - 1} \cdot \sum_i cost(\mathcal{A})_{I_i}$$

and by taking the limit in Lemma 12.12, we get that

$$\sum_{j : t(j) \notin \bigcup_i I_i} (\Delta P)_j \leq \frac{e^{1/\epsilon} - \beta}{e^{1/\epsilon} - 1} \cdot \sum_{j : t(j) \notin \bigcup_i I_i} cost(\mathcal{A})_j$$

By using Lemma B.1 (inequality (B.5)), we see that

$$\max\left\{ \frac{\epsilon}{1-\beta+\beta\epsilon} \cdot \frac{e^\epsilon - \beta}{e^\epsilon - 1}, \frac{e^{1/\epsilon} - \beta}{e^{1/\epsilon} - 1} \right\} = \frac{\epsilon}{1-\beta+\beta\epsilon} \cdot \frac{e^\epsilon - \beta}{e^\epsilon - 1}$$

which ends the proof.

□

We finish this section by proving that a fractional solution can be rounded online into a randomized integral solution. The expected cost of the rounded instance will be equal to the cost of the fractional solution. Even if the rounding is very similar to the existing rounding of Buchbinder et al. [18] for ski rental or TCP acknowledgement, we still include it here for completeness as the Bahncard problem was never solved in a primal-dual way. The argument is summarized in the following lemma.

Lemma 12.13. *Given a fractional solution (x, d, f) to the Bahncard problem, it can be rounded online into an integral solution of expected cost equal to the fractional cost of (x, d, f).*

Proof. Choose some real number p uniformly at random in the interval $[0, 1]$. Then arrange the variables x_t on the real line (i.e. iteratively as follows, each time t takes an interval I_t of length x_t right after the interval taken by x_{t-1}). Then buy a Bahncard at every time t such that the interval corresponding to time t contains the real number $p + k$ for some integer k. We check first

that the expected buying cost is

$$B \cdot \sum_t \mathbb{E}\left(\mathbb{1}_{p+k \in I_t}\right) = B \cdot \sum_t x_t$$

Next, to compute the total expected price of the tickets, notice that if a ticket was bought in the previous T time steps, we can pay a discounted price, otherwise we need to pay the full price of 1. For a trip j, the probability that a ticket was bought in the previous T time steps is at least $\sum_{t=t(j)-T}^{t(j)} x_t$. Hence with probability at least $\sum_{t=t(j)-T}^{t(j)} x_t \geq d_j$ we pay a price of β and with probability $1 - d_j \leq f_j$ we pay a price of 1 which ends the proof. □

13 Experiments

In this chapter, we present experimental results that confirm the theoretical analysis of Algorithm 7 for the TCP acknowledgement problem. The code is publicly available at https://github.com/etienne4/PDLA. We experiment on various types of distribution for packet arrival inputs. Historically, the distribution of TCP packets was often assumed to follow some Poisson distribution ([74, 86]). However, it was later shown than this assumption was not always representative of the reality. In particular real-world distributions often exhibit a heavy tail (i.e. there is still a significant probability of seeing a huge amount of packets arriving at some time). To better integrate the latter property into models, heavy tailed distributions such as the Pareto distribution are often suggested (see for instance [48, 78]). This motivates our choice of distributions for random packet arrival instances. We will experiment on Poisson distribution, Pareto distribution and a custom distribution that we introduce and seems to generate the most challenging instances for our algorithms.

Input distributions. In all our instances, we set the subdivision parameter d to 100 which means that every second is split into 100 time units. Then we define an array of length 1000 where the i-th entry defines how many requests arrive at the i-th time step. Each entry in the array is drawn independently from the others from a distribution $\mathcal{D}$. In the case of a Poisson distribution, we set $\mathcal{D} = \mathcal{P}(1)$ (the Poisson distribution of mean 1). For the Pareto distribution, we choose $\mathcal{D}$ to be the Lomax distribution (which is a special case of Pareto distribution) with shape parameter set to 2 [30]. Finally, we define the *iterated* Poisson distribution as follows. Fix an integer $n > 0$ and $\mu > 0$. Draw $X_1 \sim \mathcal{P}(\mu)$. Then for i from 2 to n draw $X_i \sim \mathcal{P}(X_{i-1})$. The final value returned is X_n. This distribution, while still having an expectation of μ, appears to generate more spikes than the classical Poisson distribution. The interest of this distribution in our case is that it generates more challenging instances than the other two (i.e., the competitive ratios of the online algorithms are closer to the worst-case bounds). In our experiments, we choose $\mu = 1$ and $n = 10$. Note that for all these distributions, the expected value for each entry is 1. Plots of typical instances under these laws can be seen in Figure 13.1, Figure 13.2 and Figure 13.3.

Noisy prediction. The prediction $\mathcal{A}$ is produced as follows. We perturb the real instances with noise, then compute an optimal solution on this perturbed instance and use this as a prediction. More precisely, we introduce a *replacement* rate $p \in [0, 1]$. Then we go through the instance generated according to some distribution $\mathcal{D}$ and for each each entry at index $1 \leq i \leq 1000$, with probability p we set this entry to 0 (i.e. we delete this entry) and with probability p we add to this entry a random variable $Y \sim \mathcal{D}$. Both operations, adding and deleting, are performed

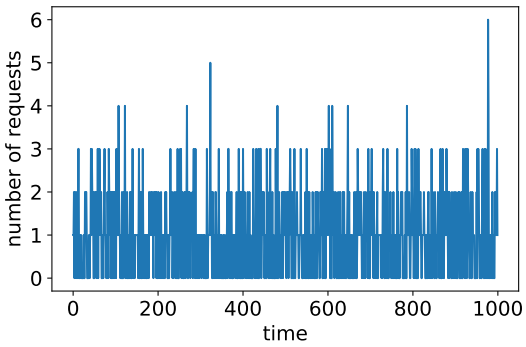


Figure 13.1: Typical instance under Poisson distribution

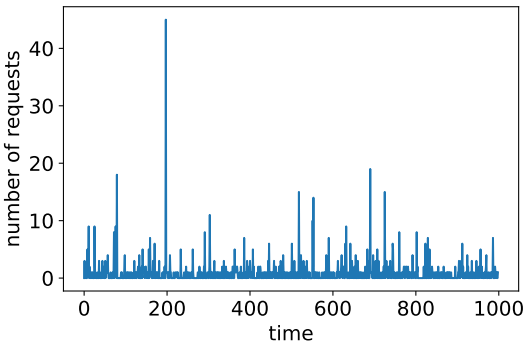


Figure 13.2: Typical instance under Pareto distribution

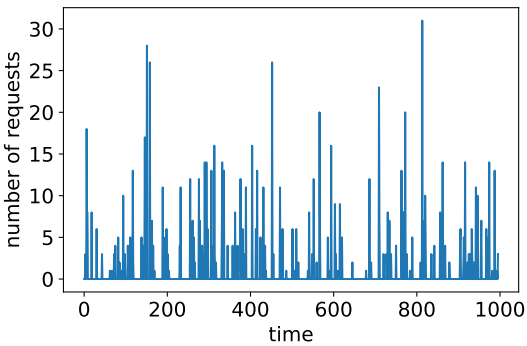
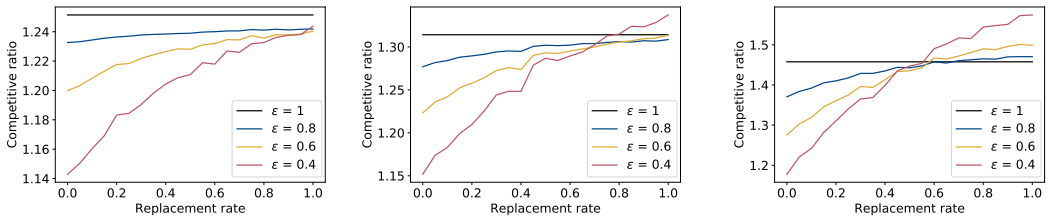


Figure 13.3: Typical instance under iterated Poisson distribution

independently of each other. We then test Algorithm 7 with 4 different values of the robustness parameter $\epsilon \in \{1, 0.8, 0.6, 0.4\}$.

Results. The plots in Figure 13.4 present the average competitive ratios of Algorithm 7 over 10 experiments for each distribution and each value of ϵ. As expected, with a perfect prediction, setting a lower ϵ will yield a much better solution while setting $\epsilon = 1$ simply means that we run the pure online algorithm of Buchbinder et al. [18] (that achieves the best possible competitive ratio for the pure online problem). On the most challenging instances generated by the iterated Poisson distribution (Figure 13.4c), even with a replacement rate of 1 where the prediction is simply an instance totally uncorrelated to the real instance, our algorithm maintains good guarantees for small values of ϵ. We note that in all the experiments the competitive ratios achieved by Algorithm 7 are better than the robustness guarantees of Theorem 12.4, which are $\{1.58, 1.68, 2.21, 3.03\}$ for $\epsilon \in \{1, 0.8, 0.6, 0.4\}$ respectively. In addition to that, all the competitive ratios degrade smoothly as the error increases which confirms our earlier discussion about smoothness.



(a) Poisson distribution (b) Pareto distribution (c) Iterated Poisson distribution

Figure 13.4: Competitive ratios under various distributions and replacement rates from 0 to 1

Part IV

Online and consistent correlation clustering

Beyond worst-case online clustering

14 Overview and related work

14.1 Introduction

Clustering is a fundamental problem in unsupervised learning. In clustering one is interested in partitioning the input elements so that similar elements are grouped together and different elements are assigned to different clusters. A natural way to capture this notion is the classic correlation clustering problem. Thanks to its simple and elegant formulation, the correlation clustering problem received a lot of attention from the theory and applied communities and it has found many practical applications including finding clustering ensembles [16], duplicate detection [6], community mining [22], disambiguation tasks [57], automated labelling [1, 20] and many more.

Formally, in the correlation clustering problem [11] we receive as input a weighted graph, where positive edges represent similarities between nodes and negative edges represent dissimilarities between them. The goal is to find a partitioning of the input graph so that the sum of the weights of the negative edges inside clusters and the positive edges between clusters is minimized. The problem is NP-hard and several approximation algorithms have been proposed for it.

For arbitrary weights a $O(\log n)$ approximation algorithm is known [31]. While, when we focus on the cases where all edges have weights in $\{-1, +1\}$ for a long time the best known algorithm [21] had an approximation guarantee of 2.06 until a recent breakthrough [26] where Cohen-Addad et al. designed a $1.94 + \epsilon$-approximation algorithm [1].

Since real world datasets continuously evolve in time, the design of approximation algorithms that maintain a good solution over time is becoming a central question in machine learning. Unfortunately, classic approximation algorithms often are either unpractical or they return very unstable solutions on dynamic datasets. In particular, the clustering returned by the algorithm may drastically change over time, potentially leading to inconsistent decisions. For this reason a lot of attention has been devoted in the past few years in designing efficient and consistent clustering algorithms for evolving datasets [27, 42, 49, 56, 65]. In this part of the thesis, we study the correlation clustering problem in the online setting. In this setting, nodes arrive one at the time and with them also their clustering preferences to the previously disclosed nodes are revealed, or in other words the set of positive and negative edges to the nodes that have already arrived. The algorithm has to assign a cluster to every node at its arrival and this assignment cannot change in the course of the algorithm.

[1] Note also that there is also a version of the problem [11] where the objective is to maximize the number of positive edges whose both endpoints are in the same cluster plus the number of negative edges across clusters.

Unfortunately this setting is too restrictive and in fact it has been shown [75] that any algorithm for the online correlation clustering problem has at least $\Omega(n)$ competitive ratio. Intuitively, this is true because if the first edge that is revealed is a positive edge then one cannot distinguish the case that this edge is part of a large clique or is the only bridge between two cliques. To get beyond this negative result, the problem has been studied in settings where the arrival order of vertices is not completely adversarial. When vertices arrive in a random order Ailon et al. [2] show that the Pivot algorithm is 3-competitive and in the semi-online model [67] where a random fraction of the instance is revealed in advance and Pivot still obtains a constant approximation. One main shortcoming of these results is that they make a strong assumption on the arrival order and for that reason they may not provide any guarantee in real world scenarios.

To overcome this limitation, in this thesis we consider the correlation clustering problem in the online model with recourse. In this model, nodes and edges are still revealed in an online fashion but the algorithm can re-assign nodes to different clusters after each arrival. The goal in this setting is to minimize the number of cluster re-assignments executed by the algorithm while still maintaining a constant factor approximation. From a practical perspective, this setting is interesting because it captures well the cases where changing the clustering assignment is possible but expensive. This is for example the case when the output clustering is used as input in a machine learning pipeline and so re-assigning points requires re-training. From a theoretical perspective, this setting allows us to study formally the trade-off between stability of the solution and quality of approximation. For these reasons other classic clustering problems were studied in this setting [27, 42, 49, 65].

14.2 Our contributions

Our first contribution is to design an algorithm that has logarithmic recourse per node and that maintains a constant approximation to the correlation clustering problem at any point in time. Our algorithm is different from previous algorithms for correlation clustering in the online setting and draws inspiration from a recent result in the setting of parallel algorithms for correlation clustering [28]. From a very high level perspective the main idea behind the algorithm is to track the dense structure in the graph in an approximate sense by carefully designing lazy updates of the clustering.

We then present a lower bound showing that any algorithm that maintains a constant approximation to the optimal solution has to incur at least logarithmic recourse per node.

Finally, we complement our theoretical results with an experimental analysis showing that our algorithm produces at the same time solutions that are stable and of high-quality. In particular, when compared with the Pivot algorithm, suggested by previous work [2, 67] in the classical online setting, we notice that our algorithm produces solutions that are substantially more stable and of higher quality.

14.3 Problem definition

For completeness we repeat the notation which was introduced in Chapter 2. The *disagreements minimization* version of the correlation clustering problem receives as input a complete signed

undirected graph $G = (V, E, s)$ where each edge $e = \{u, v\}$ is assigned a sign $s(e) \in \{+, -\}$ and the goal is to find a partition of the vertexes such that the number of $'-'$ edges inside the same cluster and $'+'$ edges in between clusters is minimized. For simplicity we denote the set of $'+'$ and $'-'$ edges by E^+ and E^- respectively. It is convenient to represent a solution to the problem, namely a clustering of the vertices of the graph, using an *assignment function* $f : V \to \mathbb{Z}$; Then, the *clustering* induced by f is a partition of the nodes $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$ such that two nodes u, v belong to the same partition, i.e., cluster C_i, if and only if $f(u) = f(v)$, or in other words, they are *assigned* the same cluster id. Given a partition function f and a clustering $\mathcal{C}$ induced by f we slightly abuse notation and denote by $f(C)$ the common cluster id of all nodes in cluster $C \in \mathcal{C}$. Hence, the cost of an algorithm $\mathcal{ALG}$ which computes an assignment function f or the clustering induced by the latter is equal to:

$$\text{cost}(f) = \sum_{\substack{\{u,v\}\in E^+ \\ f(u)\neq f(v)}} 1 + \sum_{\substack{\{u,v\}\in E^- \\ f(u)=f(v)}} 1$$

In the online setting nodes arrive one at a time, revealing upon arrival all the edges to previously arrived nodes. An instance of the online correlation clustering problem can be described by a pair $\mathcal{I} = (G, \sigma)$ where G is the *final graph* and σ is an order on the vertices of G: $\sigma = \langle v_1, v_2, \ldots, v_{|V|}\rangle$. For any $0 \leq t \leq |V|$, let $V_t = \langle v_1, v_2, \ldots, v_t\rangle$ be the set of the first t nodes in the order σ. We refer to these nodes as the nodes that have *arrived until time* t, and we refer to v_t as the node arriving at time t. We let G_t be the signed subgraph of G induced by V_t with the sign induced by s on the edges whose both endpoints are in V_t. We also denote by f_t^{OPT} an assignment function which induces an optimal correlation clustering solution for graph G_t. Note that the solution of an algorithm $\mathcal{ALG}$ on an online instance $\mathcal{I}$ can be described as a sequence of assignment functions $f_1, f_2, \ldots, f_{|V|}$, denoted for simplicity by $f_{1,2,\ldots,|V|}$. The recourse of a node u, $r(u)$, that arrived at time t is the number of times the assignment function sequence changes the cluster id assigned to u. That is $r(u) = \sum_{t'>t} \mathbb{1}\{f_{t'-1}(u) \neq f_{t'}(u)\}$. The recourse of an algorithm is the worst case recourse over all instances $\mathcal{I}$ and nodes u.

In the classical online setting decisions are irrevocable and the recourse of an algorithm should be 0, however due to strong impossibility results [75] it turns out that one cannot achieve at the same time a competitive ratio of $O(|V|)$ and 0 recourse. Thus, the goal of this part is to design an algorithm with $O(\log(|V|))$ recourse which maintains a constant factor approximate solution at all times, i.e., an algorithm which computes $f_{1,2,\ldots,|V|}$ such that $\text{cost}(f_t) \leq \Theta(1) \cdot \text{cost}(f_t^{OPT})$, $\forall t \in \{1, 2, \ldots, |V|\}$ for all instances . Despite the fact that we are not in the classical online setting we will slightly abuse notation and say that such an algorithm is a constant competitive algorithm.

15 Main algorithm

The goal of this chapter is to present our algorithm along with a proof sketch of its guarantees. The full proofs are quite technical and we present them in Chapter 16, hence the proof sketch of this chapter serves as a warm-up which will hopefully give the reader a clear intuition of our algorithm. At the end of the chapter we also prove that these guarantees are tight up to constant factors.

15.1 The Agreement algorithm

Our algorithm uses as a subroutine the static algorithm introduced in [28]; we refer to this algorithm as the Agreement algorithm. The latter algorithm achieves a constant factor approximation for the correlation clustering problem by partitioning the graph so that all non-trivial clusters are dense, i.e., any node u with positive degree d has at least $(1-\epsilon)d$ $'+'$ edges to nodes of the same cluster. At a high level, the Agreement algorithm uses a filtering procedure which ensures that two nodes with similar $'+'$ neighborhoods will end in the same cluster. As the Agreement algorithm is a central ingredient of our algorithm and our analysis depends on its properties, we present it in Algorithm 9. In the following, $N_{G'}(u)$ depending of whether G' is a signed or unsigned graph will either denote the positive neighborhood or just the neighborhood of node u respectively. Given a signed graph G_t, the Agreement algorithm is executed on the graph $G = (V(G_t), E^+(G_t))$ and makes use of the following definitions.

Definition 15.1. *Two nodes u, v are in ϵ-agreement in G if*

$$|N_G(u) \triangle N_G(v)| < \epsilon \max\{|N_G(u)|, |N_G(v)|\}$$

where the symbol $\triangle$ denotes the symmetric difference of two sets.

Definition 15.2. *A node u is light if it is in ϵ-agreement with less than an ϵ fraction of its neighborhood.*

15.2 Online Agreement algorithm

Before we describe our algorithm, we introduce some notation. Let $\mathcal{C}_t$ be the clustering of G_t induced by the assignment function f_t. A cluster is called singleton if it contains a single node. We denote by $S(\mathcal{C}_t)$ the set of nodes that belong to a singleton cluster in $\mathcal{C}_t$. Respectively, we use

Algorithm 9 AGREEMENT

Input: A signed graph G', and a parameter ϵ
Output: A clustering $\mathcal{C}$ of G'
Initialize: $G \leftarrow E^+(G'), \hat{G} \leftarrow E^+(G')$
Step 1: Remove all edges of $\hat{G}$ whose endpoints are not in ϵ-agreement in G.
Step 2: Remove all edges of $\hat{G}$ whose endpoints are both light.
Step 3: Compute the connected components $\tilde{\mathcal{C}}$ of $\hat{G}$.
return $\tilde{\mathcal{C}}$

$H(\mathcal{C}_t)$ to denote the set of nodes that belong to a non-singleton cluster in $\mathcal{C}_t$. At a high-level our algorithm performs lazy cluster-assignment updates. Namely, at each time t, our algorithm re-runs the Agreement algorithm (which produces an $O(1)$-approximate solution for graph G_t) and only updates the cluster ids of the vertices that joined a different (non-singleton) cluster at time t compared to the clustering at time $t-1$. A final rule changes the ids of clusters that have grown in size by a constant factor. We use $\widetilde{\mathcal{C}}_t$ to refer to the clustering produced by running the Agreement algorithm on G_t, and $\tilde{f}_t$ to refer to the assignment function that induces $\widetilde{\mathcal{C}}_t$. Similarly, we use $\tilde{f}_t(C)$ to refer to the unique cluster id of a cluster $C \in \widetilde{\mathcal{C}}_t$.

Evolving clusters: We keep track of clusters that evolve over time. Each evolving cluster is associated with a unique cluster id. The life-cycle of a cluster starts whenever a cluster id is seen for the first time, and it ends whenever no node uses the cluster id anymore. We monitor the growth of each evolving cluster using the following definition.

Definition 15.3 (Origin cluster)**.** *Let ID be a cluster id that is used by the assignment function f_t, and let t_{min} be the minimum t for which ID is used by $f_{t_{min}}$. The origin cluster of ID, denoted by $Origin(ID)$, is the cluster with id ID in the clustering induced by $f_{t_{min}}$.*

Once a cluster $C \in \mathcal{C}_t$ becomes a constant factor larger in size compared to the origin cluster with the same id, we assign a fresh id to $C \cap H(\widetilde{\mathcal{C}}_t)$ (hence making $C \cap H(\widetilde{\mathcal{C}}_t)$ the origin cluster of that fresh id). We do this so that we can later bound the competitive ratio of the clusters that we output.

Algorithm description: Our *Online Agreement (Agree-On)* algorithm is described in details in Algorithm 10. At a high-level, the algorithm has three main phases which are executed for every node-arrival.

- *Offline re-clustering phase:* We run the Agreement algorithm on G_t. Let $\widetilde{\mathcal{C}}_t$ be the resulting clustering and $\tilde{f}_t$ be the assignment function inducing $\widetilde{\mathcal{C}}_t$.
- *Initial assignment phase:* This phase combines $\widetilde{\mathcal{C}}_{t-1}, \widetilde{\mathcal{C}}_t$ as well as the previously maintained clustering $\mathcal{C}_{t-1}$ to produce $\mathcal{C}_t$. First, it assigns a new unique id to the newly arrived node v_t, if $\{v_t\}$ is a singleton cluster of $\widetilde{\mathcal{C}}_t$. Then it applies the following three rules to set the ids of all nodes (except possibly v_t):
  - **Rule 1**: For each non-singleton cluster $C \in \widetilde{\mathcal{C}}_t$ if there exists a non-singleton intersecting cluster $C' \in \widetilde{\mathcal{C}}_{t-1}$ then assign the cluster id of $f_{t-1}(C')$ to all nodes in C. This essentially

identifies C and C' to be part of the same evolving cluster. We exploit structural properties to show that any non-singleton cluster $C \in \widetilde{\mathcal{C}}_t$ intersects at most one non-singleton cluster $C' \in \widetilde{\mathcal{C}}_{t-1}$.

- **Rule 2**: For each non-singleton cluster $C \in \widetilde{\mathcal{C}}_t$ that consists entirely of nodes that were in singleton clusters of $\widetilde{\mathcal{C}}_{t-1}$ (but not necessarily in singleton clusters of $\mathcal{C}_{t-1}$), and potentially the newly arrived node, do the following. If there is a cluster $C' \in \mathcal{C}_{t-1}$ containing the majority of the nodes in C, then C gets the same cluster id as C'; otherwise C receives a new unique cluster id and C becomes the origin cluster of that id.
- **Rule 3**: ensures that each singleton cluster in $\widetilde{\mathcal{C}}_t$ retains its previous cluster id.

At the end of this phase, for each non-singleton cluster $C \in \widetilde{\mathcal{C}}_t$ we have assigned the id of a cluster $C' \in \mathcal{C}_t$ to all nodes in C. This assignment induces a mapping from ids of non-singleton clusters $C \in \widetilde{\mathcal{C}}_t$ to ids of non-singleton clusters $C' \in \mathcal{C}_t$; we represent this mapping using the assignment function ϕ_t.

- *Assignment refinement phase:* After forming an initial clustering of $\mathcal{C}_t$, this phase identifies clusters $C \in \widetilde{\mathcal{C}}_t$ that are significantly larger compared to the size of their origin cluster with id $\phi_t(\tilde{f}_t(C))$. Such clusters become new origin clusters and receive a new unique cluster id.

The crux of our algorithm is the combination of the notion of origin clusters together with $\tilde{f}_t$, $\tilde{f}_{t-1}$, and f_{t-1}. Before presenting some useful structural properties of our algorithm, we show that the algorithm is well defined; that is, we argue that for any time t, $\mathcal{C}_t$ indeed forms a partition of V_t. We start from the fact that $\widetilde{\mathcal{C}}_t$ is a partition of V_t. Each non-singleton cluster of $C \in \widetilde{\mathcal{C}}_t$ is assigned an id by either Rule 1 or Rule 2, since C consists of nodes that form singleton clusters of $\widetilde{\mathcal{C}}_{t-1}$ and at most one non-singleton cluster $C' \in \widetilde{\mathcal{C}}_{t-1}$ intersecting C. All singleton clusters of $\widetilde{\mathcal{C}}_t$ are assigned an id by Rule 3; if v_t is a singleton cluster in $\widetilde{\mathcal{C}}_t$, then it is assigned an id right before the main loop of the algorithm.

In Chapter 16 we prove several properties of our algorithm with respect to the assignment functions $\tilde{f}_t, f_t$. We mention these properties here so that the proof sketch is clear.

1. Let C be a non-trivial cluster of $\widetilde{\mathcal{C}}_t$. Then, at time t all nodes of C are assigned the same cluster id in f_t. This is ensured by **Rule 1** and **Rule 2**.
2. Let C, C' be two non-trivial clusters of $\widetilde{\mathcal{C}}_t$. Then, f_t assigns to all nodes in cluster C a cluster id that is distinct from the one assigned to the nodes in C'.
3. Let C be a non-trivial cluster of $\widetilde{\mathcal{C}}_t$, we denote by $S_t^C = \{v \in S(\widetilde{\mathcal{C}}_t) : f_t(v) = f_t(C)\}$ the set of nodes which do not belong to C but are clustered together with C in $\mathcal{C}_t$. Note that for every node u for which it holds $f_t(u) = f_t(C)$ belongs to $C \cup S_t^C$.

15.3 Proof sketch

To simplify our proof sketch we make the following simplifying assumption regarding the structure of $\widetilde{\mathcal{C}}_t$.

Assumption 15.1. *Let $C \in \widetilde{\mathcal{C}}_t$ be a non-trivial cluster and let $u \in C$. Node u has, in G_t, at least $(1-\epsilon)|C|$ positive edges to nodes in C and at most $\epsilon|C|$ positive edges to nodes outside C.*

Algorithm 10 ONLINE AGREEMENT (Agree-On)

Input: An online instance $\mathcal{I} = (G, \sigma)$, a parameter ϵ
Output: An assignment function sequence $f_{1,2,\ldots,|V|}$
Initialization: $G_{-1} \leftarrow \emptyset$, $ID_{next} \leftarrow 0$
on arrival of v_t **do**
  $G_t \leftarrow G_{t-1} \cup \{v_t\}$
  $\widetilde{\mathcal{C}}_t \leftarrow$ AGREEMENT(G_t, ϵ)
  if $v_t \in S(\widetilde{\mathcal{C}}_t)$ **then**
    $Origin(ID_{next}) \leftarrow \{v_t\}$
    $ID_{next} \leftarrow ID_{next} + 1$
  end if
  for all $C \in \widetilde{\mathcal{C}}_t$ s.t. $|C| > 1$ **do**
    //* ***Rule 1:***
    if $\exists\, C' \in \widetilde{\mathcal{C}}_{t-1}$ s.t. $C \cap C' \neq \emptyset, |C'| > 1$ **then**
      for $u \in C$ **do** $f_t(u) \leftarrow f_{t-1}(C')$ **endfor**
    end if
    //* ***Rule 2:***
    if $C \setminus v_t \subseteq S(\widetilde{\mathcal{C}}_{t-1})$ **then**
      if $\exists C' \in \mathcal{C}_{t-1}$ s.t. $|C' \cap C| > |C|/2$ **then**
        for $u \in C$ **do** $f_t(u) \leftarrow f_{t-1}(C')$ **endfor**
      else
        for $u \in C$ **do** $f_t(u) \leftarrow ID_{next}$ **endfor**
        $Origin(ID_{next}) \leftarrow C$
        $ID_{next} \leftarrow ID_{next} + 1$
      end if
    end if
  end for
  //* ***Rule 3:***
  for all $u \in S(\widetilde{\mathcal{C}}_t) \setminus v_t$ **do** $f_t(u) \leftarrow f_{t-1}(u)$ **endfor**
  //* ***Store*** ϕ_t
  for all $C \in \widetilde{\mathcal{C}}_t$ s.t. $|C| > 1$ **do**
    $\phi_t(\tilde{f}_t(C)) \leftarrow f_t(C)$
  end for
  //* ***Assignment refinement phase***
  Let $\mathcal{C}'$ be the current clustering induced by f_t.
  for all $C \in \widetilde{\mathcal{C}}_t$ s.t. $|C| > 1$ **do**
    Let $C' \in \mathcal{C}'$ be the cluster s.t. $C \subseteq C'$.
    if $|C| \geq (3/2)|Origin(f_t(C'))|$ **then**
      for $u \in S(\widetilde{\mathcal{C}}_t)$ **do** $f_t(u) \leftarrow ID_{next}$ **endfor**
      $ID_{next} \leftarrow ID_{next} + 1$
      $Origin(ID_{next}) \leftarrow C$
    end if
  end for
end on

Bounding the competitive ratio: We give an overview on how we prove that $\text{cost}(f_t) \leq \Theta(\text{cost}(f_t^{OPT}))$. We do this by proving $\text{cost}(f_t) \leq \Theta(\text{cost}(\tilde{f}_t))$, which combined with the fact that $\text{cost}(\tilde{f}_t) \leq \Theta(\text{cost}(f_t^{OPT}))$ implies the constant competitive ratio of our algorithm.

For convenience, we use $E^+(X, Y)$ (resp., $E^-(X, Y)$) to denote the edges in $E^+(G_t) \cap (X \times Y)$

(resp., $E^-(G_t) \cap (X \times Y)$).

The proof consists of two parts. First, we prove that the cost incurred on f_t by the edges on nodes in $S(\tilde{f}_t)$ is bounded by $\Theta(\text{cost}(f_t^{OPT}))$. Then, we move to showing that the cost incurred on f_t by the edges on nodes in $H(\tilde{f}_t)$ is also $\Theta(\text{cost}(f_t^{OPT}))$.

We first sketch the first part of the proof. Notice that the cost incurred on f_t for the edges of a node $u \in S(\tilde{f}_t)$ is only larger than the cost incurred on $\tilde{f}_t$ for the edges of u if f_t clusters u in a non-singleton cluster C. Fix a cluster $C \in \mathcal{C}_t$. We bound $|E^-(C, C)|$ by $\Theta(|E^+(S(\tilde{f}_t) \cap C, V_t)|)$. To do so, we prove that C is not much larger than its origin cluster and that further it holds that $|E^+(u, C)| \geq \Theta(|C|)$ for each $u \in S(\tilde{f}_t) \cap C$. The latter is proved by using properties of the clusters generated by the Agreement algorithm.

Now we move to sketching the second part of the proof. Let $C_1, C_2, \ldots, C_k$ be the set of non-singleton clusters of $\widetilde{\mathcal{C}}_t$, then f_t and $\tilde{f}_t$ disagree exactly on $S_t^{C_1}, S_t^{C_2}, \ldots, S_t^{C_k}$. Fix a specific pair C, S_t^C, and note that for this pair:

1. $\tilde{f}_t$ pays $|E^+(C \cup S_t^C, V_t \setminus C \cup S_t^C)| + |E^+(C, S_t^C)| + |E^-(C, C)|$.
2. f_t pays $|E^+(C \cup S_t^C, V_t \setminus C \cup S_t^C)| + |E^-(C, S_t^C)| + |E^-(S_t^C, S_t^C)|$

Combining this, with the first part of the proof, we get that $\text{cost}(f_t)$ is at most:

$$\left(\sum_{C \in \widetilde{\mathcal{C}}_t : |C| > 1} |E^-(C, S_t^C)| + |E^-(S_t^C, S_t^C)| \right) + \Theta(\text{cost}(\tilde{f}_t))$$
$$\leq \left(\sum_{C \in \widetilde{\mathcal{C}}_t : |C| > 1} |S_t^C||C| + |S_t^C|^2 \right) + \Theta(\text{cost}(\tilde{f}_t))$$

and,

$$\text{cost}(\tilde{f}_t) \geq \sum_{C \in \widetilde{\mathcal{C}}_t : |C| > 1} |E^+(C, S_t^C)|$$

To conclude, we are left to argue that $|S_t^C||C| + |S_t^C|^2 = \Theta(|E^+(C, S_t^C)|)$ for any non-trivial cluster $C \in \widetilde{\mathcal{C}}_t$.

Let $C \in \widetilde{\mathcal{C}}_t$ be a cluster whose nodes get assigned cluster id after $\tilde{f}_t(C)$.

Note that the assignment refinement phase, takes an initial version of $\mathcal{C}_t$ and if there is a cluster $C \in \widetilde{\mathcal{C}}_t$ that is significantly larger than $Origin(f_t(C))$, then all nodes in C receive a new cluster id in $\mathcal{C}_t$ and the origin cluster of the newly formed cluster is set to be C. At a high level the assignment refinement phase, forces a cluster of f_t to remain as similar as possible to its origin cluster. For simplicity here we make the following assumption.

Assumption 15.2. *Let $C' \in \widetilde{\mathcal{C}}_{t'}$ and $C \in \widetilde{\mathcal{C}}_t$ be two non-singleton clusters, for $t' < t$. If they are part of the same evolving cluster (i.e., if $\phi_{t'}(\tilde{f}_{t'}(C')) = \phi_t(\tilde{f}_t(C))$), then $|C' \cap C| \geq (1/2)\max\{|C'|, |C|\}$.*

We start by lower bounding the number of positive edges between nodes in S_t^C and nodes in C, i.e., $|E^+(C, S_t^C)|$. Note that any node $u \in S_t^C$ must have been part of a non-trivial cluster $C' \in \widetilde{\mathcal{C}}_{t'}$ s.t. $\phi_{t'}(\tilde{f}_{t'}(C')) = \phi_t(\tilde{f}_t(C))$, for some time $t' < t$. Hence, from Assumption 15.2 we get that $|C \cap C'| \geq (1/2)\max\{|C|, |C'|\}$. Moreover, from Assumption 15.1 we know that node u has at least $(1-\epsilon)|C'|$ positive edges inside C'. Combining the latter two facts we can conclude that node u has at least $(1/2-\epsilon)|C|$ edges shared with nodes in C. Thus, $|E^+(C, S_t^C)| \geq |S_t^C|(1/2-\epsilon)|C|$. Combining the latter with the trivial upper bound $|E^+(C, S_t^C)| \leq |S_t^C||C|$ we get that $|E^+(C, S_t^C)| = \Theta(1)|S_t^C||C|$.

To conclude the proof it suffices to show that $|S_t^C| = \Theta(1)|C|$. From Assumption 15.1 we have that positive edges between nodes in C and nodes outside C are at most $\epsilon|C|^2$. On the other hand, we just argued that $|E^+(C, S_t^C)| \leq \Theta(1)|S_t^C||C|$. By combining these two facts we get that $\Theta(1)|S_t^C||C| < \epsilon|C|^2$ and consequently $|S_t^C| \leq \Theta(1)|C|$.

Bounding the recourse: At a high level, the recourse is at most $O(\log|V|)$ because, roughly speaking, whenever a node changes cluster id then the size of the origin cluster corresponding to the new cluster id is a multiplicative factor larger than the size of the origin cluster which corresponds to the old cluster id. The proof of the latter is given in Section 16.4.

15.4 Lower bound

In this section we present an online correlation clustering instance where any algorithm that achieves constant competitive ratio requires $\Omega(\log|V|)$ recourse, in the worst case.

Let $c \geq 1$ be a constant. Initially our instance reveals two nodes u, v with a positive edge between them. The rest of the node arrival sequence works in logarithmically many phases. Our instance forces any c-competitive algorithm $\mathcal{ALG}$ to behave as follows. At odd phases $\mathcal{ALG}$ clusters u, v together while at even phases it clusters u, v separately. Let r be the total number of phases, then note that such a construction forces either u or v to have a recourse of at least $r/4$.

Lemma 15.3. *Let $c \geq 1$ be a constant. Any online algorithm for the correlation clustering problem with a competitive ratio smaller than c has a recourse of $\Omega(\log|V|)$.*

Proof. For a simpler description, we are using an unsigned graph to describe our construction. The actual signed instance is constructed by replacing every existing edge with a $'+'$ edge and every missing edge between two already revealed nodes with a $'-'$ edge.

We use two main gadgets: 1) cliques $C_{u,v}^s$ of size s whose nodes are connected to both u and v, and 2) cliques C_u^s (resp., C_v^s) of size s whose nodes are all connected to u (resp., v). We note that different cliques share no edge between them.

Initially, our adversarial node arrival sequence reveals the two base nodes. At that point any algorithm with a bounded competitive ratio connects the two nodes since the optimum solution has cost 0.

In phase 0, our adversarial sequence reveals a pair of cliques C_u^{2c}, C_v^{2c}. Note that the optimal solution clusters u with C_u^{2c} in a single cluster and v with C_v^{2c} in a second cluster, achieving a

total cost of 1 (the positive edge between the base nodes). Any solution that clusters u, v together costs at least $2c$ as any splitting of the two clusters leaves at least $2c - 1$ inter-clusters edges while merging the two clusters costs $\Omega(c^2)$. Thus, any algorithm with a competitive ratio less than c splits the nodes u, v to two clusters.

In phase 1, our sequence reveals a clique $C_{u,v}^{(2c)^3}$. The optimal solution costs $2 \cdot 2c$ and is achieved by clustering $u, v, C_{u,v}^{(2c)^3}$ in the same cluster and each other clique into a separate cluster. Notice that any splitting of the cluster $C_{u,v}^{(2c)^3}$ leaves at least $(2c)^3 - 1$ inter-clusters edges. Hence, any c-competitive online algorithm clusters u, v together with clique $C_{u,v}^{(2c)^3}$, otherwise it costs at least $(2c)^3 - 1 > c \cdot 4c = c \cdot \text{cost}(f^{OPT})$.

Following the same pattern, at even phases i our sequence reveals the pair of cliques $C_u^{(2c)^{3i}}$ and $C_v^{(2c)^{3i}}$ forcing any c-competitive algorithm to cluster u and v separately, while at odd phases i our sequence reveals the clique $C_{u,v}^{(2c)^{3i}}$ forcing any c-competitive algorithm to cluster u and v together.

We next prove the claim for even and odd phases separately. First, assume i is even. At the end of phase i, the optimum solution clusters v with $C_v^{(2c)^{3i}}$ in a single cluster, u with $C_u^{(2c)^{3i}}$ into a second cluster, and every other clique into a separate cluster on its own. The above clustering has a cost of $2 \cdot \left((2c)^3 + (2c)^6 + \cdots + (2c)^{3(i-1)}\right)$. Any algorithm where u, v are in the same cluster has a cost of $(2c)^{3i} > c \cdot 2 \cdot \left((2c)^3 + (2c)^6 + \cdots + (2c)^{3(i-1)}\right) \geq c \cdot \text{cost}(f^{OPT})$. Thus, any c-competitive online algorithm assigns u and v to different clusters.

Next, assume i is odd. At the end of phase i, the optimum solution is formed by clustering u, v, and $C_{u,v}^{(2c)^{3i}}$ together and every other clique into a separate cluster on its own. The above clustering has a cost of $2 \cdot \left((2c)^3 + (2c)^6 + \cdots + (2c)^{3(i-1)}\right)$. Any algorithm where u, v are in different clusters has a cost of $(2c)^{3i} > c \cdot 2 \cdot \left((2c)^3 + (2c)^6 + \cdots + (2c)^{3(i-1)}\right) \geq c \cdot \text{cost}(f^{OPT})$. Thus, any c-competitive online algorithm clusters u and v together.

Let r be the number of phases . Given that every 2 phases, u, v are clustered together and then separately again, at least one of u, v needs to increase its recourse by at least 1. Thus, after r rounds, one of u, v incurs a recourse of at least $r/4$.

To end the proof let the last phase r be an odd round, then the total number of nodes is $|V| = 2 + 2 \cdot 2c + (2c)^3 + 2 \cdot (2c)^6 + (2c)^9 + ... + (2c)^{3r} < 2 \cdot (2c)^{3r+1} < (2c)^{3r+2}$. Thus, $r = \Omega(\log|V|/\log c)$ which concludes the proof. □

16 Main proof

This chapter is devoted in proving that the Online Agreement algorithm has a constant competitive ratio and achieves a logarithmic (in the number of nodes) recourse. In all the subsequent sections we will convert, for simplicity, a complete undirected signed graph $G' = (V, E', s)$ into a non-signed undirected graph $G = (V, E)$ where for each pair of nodes $\{u, v\}$ there is an edge between them in G if and only if $s(\{u, v\}) =' +'$. Thus the absence of an edge between two nodes corresponds to a negative edge in the original signed graph and the presence of an edge to a positive edge in the original graph. Note that the correlation clustering cost of an assignment function f and the clustering $\mathcal{C}$ induced by f becomes:

$$\text{cost}(f) = \sum_{\substack{\{u,v\}\in E \\ f(u)\neq f(v)}} 1 + \sum_{\substack{\{u,v\}\notin E \\ f(u)=f(v)}} 1$$

16.1 Properties of the Agreement algorithm

In this section we redefine the Agreement algorithm 9 presented in Chapter 15 in order to take as input a non-signed graph instead of a signed one. Thus, when we refer to the Agreement algorithm we will refer to Algorithm 11.

Algorithm 11 AGREEMENT

1: **Input:** A graph G, and a parameter ϵ
2: **Output:** A clustering $\mathcal{C}$ of G
3: **Initialize:** $\hat{G} \leftarrow G$
4: **Step 1:** Remove all edges of $\hat{G}$ whose endpoints are not in ϵ-agreement in G.
5: **Step 2:** Remove all edges of $\hat{G}$ whose endpoints are both light.
6: **Step 3:** Compute the connected components $\mathcal{C}$ of $\hat{G}$.
7: **return** $\mathcal{C}$

In addition, we will restate some useful definitions and lemmas from [28] regarding the Agreement algorithm and prove some additional structural properties of the clustering produced by the latter. For simplicity, we set both parameters λ, β of the Agreement algorithm to be $\lambda = \beta = \epsilon$. We will denote the clustering computed by the Agreement algorithm on a graph G as $\tilde{\mathcal{C}}$ and by $\tilde{f}$ the assignment function that induces the latter clustering[1].

[1]Note that a clustering may be induced by many different assignment functions, but two different clusterings cannot be induced by the same assignment function.

A cluster $C \in \tilde{C}$ will be called non-trivial/non-singleton if $|C| \geq 2$ and singleton/trivial otherwise. In addition, we will denote by f^{OPT} the assignment function that induces the optimal correlation clustering solution OPT of a graph G.

As a direct consequence of Definition 15.1 and Definition 15.2 we have that:

Proposition 16.1. *If u, v are in ϵ-agreement then $|N_G(u) \cap N_G(v)| \geq (1-\epsilon) max\{|N_G(u)|, |N_G(v)|\}$*

Note that the latter proposition also implies that nodes which are in ϵ-agreement have similar degrees.

Proposition 16.2. *If u, v are in ϵ-agreement then $(1-\epsilon)|N_G(u)| \leq |N_G(v)| \leq \frac{|N_G(u)|}{1-\epsilon}$*

An important property of clustering $\tilde{\mathcal{C}}$ which will permit us to bound the approximation ratio of our solution, is that its cost constitutes a constant factor approximation to the cost of the optimal correlation clustering. That is:

Lemma 16.1 (rephrased from [28])**.** *Let $\tilde{\mathcal{C}} = \text{Agreement}(G, \epsilon)$ then for ϵ small enough* $\text{cost}(\tilde{f}) \leq \Theta(1)\,\text{cost}(f^{\mathrm{OPT}})$.

The following lemmas prove that the non-trivial clusters of $\tilde{\mathcal{C}}$ form dense subgraphs in the initial graph G with a small number of outgoing edges.

Lemma 16.3 (rephrased from [28])**.** *Let C be a non-trivial cluster of $\tilde{\mathcal{C}}$ and u a node which belongs to C. Then $|N_G(u) \cap C| \geq (1-9\epsilon)|C|$.*

Lemma 16.4 (rephrased from [28])**.** *Let C be a non-trivial cluster of $\tilde{\mathcal{C}}$ and u, v two nodes which belong to C. Then u and v are in an 5ϵ-agreement. If, in addition, there exists a node $w \in C$ such that both u, v are in ϵ-agreement with w then u and v are in an 3ϵ-agreement.*

Note that by combining the latter Lemma 16.4 with Propositions 16.1 and 16.2 we can conclude that nodes in the same cluster $C \in \tilde{\mathcal{C}}$ have similar neighborhoods.

While Lemma 16.3 proves that the size of the intersection of a node's neighborhood $N_G(u)$ with the cluster C to which it belongs is lower bounded by $(1-\Theta(\epsilon))|C|$ the following lemma also proves that it is lower bounded by $(1-\Theta(\epsilon))|N_G(u)|$.

Lemma 16.2. *Let C be a non-trivial cluster of $\tilde{\mathcal{C}}$ then for a small enough constant ϵ it holds that:*

1. *If $|C| \geq 10$ then every node in C has at least a $(1-3\epsilon)$-fraction of its edges in G to nodes in C.*
2. *If $|C| < 10$ then nodes in C form a clique with no outgoing edges in G.*

Proof. In this proof, we will heavily rely on the structure of $\hat{G}$, the intermediate graph used by the Agreement algorithm. To that end, note that any cluster $C \in \tilde{\mathcal{C}}$ corresponds to a connected component of $\hat{G}$. In addition, note that to derive $\hat{G}$ from G the Agreement algorithm deletes edges between light nodes, thus any non-trivial cluster of $\tilde{\mathcal{C}}$ (which corresponds to a connected

component of $\hat{G}$) contains at least one heavy node and any light node is connected in $\hat{G}$ (and consequently in ϵ-agreement) with at least one heavy node of the same cluster. We prove the first part of the lemma by distinguishing between two cases, i.e. if u is heavy or if u is a light node.
If u is a heavy node of C then by definition u is in ϵ-agreement with at least an $(1-\epsilon)$ fraction of its neighborhood. Consequently, since edges between a heavy and a light node are not deleted, $|N_G(u) \cap C| \geq (1-\epsilon)|N_G(u)|$.
If u is light then let $v \in N_{\hat{G}}(u) \cap C$ be a heavy neighboring node with whom u is in ϵ-agreement. Since u and v are in an ϵ-agreement from Proposition 16.1 we get that:

$$|N_G(u) \cap N_G(v)| \geq (1-\epsilon)\max\{|N_G(u)|, |N_G(v)|\} \Rightarrow \tag{1}$$
$$|(N_G(u) \cap C) \cap (N_G(v) \cap C)| + |(N_G(u) \setminus C) \cap (N_G(v) \setminus C)| \geq (1-\epsilon)\max\{|N_G(u)|, |N_G(v)|\} \Rightarrow \tag{2}$$
$$|(N_G(u) \cap C) \cap (N_G(v) \cap C)| \geq (1-\epsilon)\max\{|N_G(u)|, |N_G(v)|\} - |(N_G(u) \setminus C) \cap (N_G(v) \setminus C)| \Rightarrow \tag{3}$$
$$|N_G(u) \cap C| \geq (1-\epsilon)\max\{|N_G(u)|, |N_G(v)|\} - |N_G(v) \setminus C| \Rightarrow \tag{4}$$
$$|N_G(u) \cap C| \geq (1-\epsilon)\max\{|N_G(u)|, |N_G(v)|\} - \epsilon|N_G(v)| \Rightarrow \tag{5}$$
$$|N_G(u) \cap C| \geq (1-\epsilon)|N_G(u)| - \frac{\epsilon}{1-\epsilon}|N_G(u)| \Rightarrow \tag{6}$$
$$|N_G(u) \cap C| \geq (1-3\epsilon)|N_G(u)| \tag{7}$$

Where:

1. from line (4) to line (5) we used the fact that node v is heavy , hence the set of nodes with which v is not in ϵ-agreement (a) is a superset of $N_G(v) \cap C$; and (b) is at most $\epsilon|N_G(v)|$;
2. from line (5) to line (6) we used that since u, v are in ϵ-agreement, from proposition 16.2 we get that $N_G(v) \leq \frac{N_G(u)}{1-\epsilon}$; and
3. from line (6) to line (7) we use that $(1-\epsilon-\frac{\epsilon}{1-\epsilon}) > (1-3\epsilon)$ holds for ϵ small enough.

We proceed into proving the second part of the lemma. Note that from the first part of the lemma we get that for every node $u \in C$, $|N_G(u)| \leq \frac{|C|}{1-3\epsilon}$. In order to prove that C forms a clique, assume towards a contradiction that there exist two nodes $u, v \in C$ such that there is no edge (u, v) in G. W.l.o.g. we can assume that u, v have a common neighbor in $\hat{G}$ (otherwise no pair of nodes which belong to C would have a common neighbor and C would be empty) and consequently from lemma 16.4 they are in an 3ϵ-agreement. Thus, we get:

$$|N_G(u) \triangle N_G(v)| \leq 3\epsilon \max\{|N_G(u)|, |N_G(v)|\} \leq 3\epsilon\frac{|C|}{1-3\epsilon} < 1 \tag{16.1}$$

which is a contradiction for all $\epsilon < 1/31$ whenever $|C| \leq 9$.
Now, it remains to prove that whenever $|C| \leq 9$ there is no outgoing edge in G from a node in C to a node which is not in C. By using $N_G(u) \leq \frac{|C|}{1-3\epsilon}$ we can deduce that whenever $|C| \leq 9$ and $\epsilon < 1/31$ we have $|N_G(u)| = |C|$. The latter together with the fact that any two nodes $u, v \in C$ have an edge between them is enough to deduce that $N_G(u) = C$ and C has no outgoing edges. □

At that point it is important to recapitulate all the lemmas concerning non-trivial clusters of $\tilde{\mathcal{C}}$ and their consequences. To that end, let u, v be two nodes which belong to the same non-trivial cluster C of $\tilde{\mathcal{C}}$, then for ϵ small enough the following properties hold.

Property 1 $|N_G(u) \cap C| \geq (1-3\epsilon)|N_G(u)|$

Property 2 $|N_G(u) \setminus C| < 3\epsilon|N_G(u)|$

Property 3 $|C| \geq (1-3\epsilon)|N_G(u)|$

Property 4 $|N_G(u) \cap C| \geq (1-9\epsilon)|C|$

Property 5 $|C \setminus N_G(u)| < 9\epsilon|C|$

Property 6 $|N_G(u)| \geq (1-9\epsilon)|C|$

Property 7 $|N_G(u) \cap N_G(v)| \geq (1-5\epsilon)\max\{|N_G(u)|, |N_G(v)|\}$

Property 8 $|N_G(v)|(1-5\epsilon) \leq |N_G(u)| \leq \frac{|N_G(v)|}{1-5\epsilon}$

Property 9 $|C \setminus N_G(u)| < 9\epsilon|C| < \frac{9\epsilon}{1-9\epsilon}|N_G(u)|$

Property 10 $|N_G(u) \setminus C| < 3\epsilon|N_G(u)| < \frac{3\epsilon}{1-3\epsilon}|C|$

Property 11 $N_G(u) \cap N_G(v) \neq \emptyset$

Note that Properties 2, 3, 5, 6, 9, 10, 11 are straightforward consequences of Properties 1, 4, 7, 8 which are stated in the lemmas and propositions of this section.

16.2 Dynamic analysis of the clustering sequence $\widetilde{\mathcal{C}}_1, \widetilde{\mathcal{C}}_2, \ldots$

The Agreement algorithm computes at any new arrival of a node u_t a clustering $\widetilde{\mathcal{C}}_t$ of graph G_t. In the subsequent sections we will bound the competitive ratio of our solution and the worst case recourse of any node. To bound the competitive ratio we prove that our solution at time t, induced by the assignment function f_t, is close to the solution $\widetilde{\mathcal{C}}_t$ induced by $\tilde{f}_t$. To bound the recourse we will ensure that for any node u most of the time it holds that $f_t(u) = f_{t-1}(u)$. Thus it is clear that for the latter two facts to be approximately true we need $\tilde{f}_t \approx \tilde{f}_{t-1}$ which, at a high level, is equivalent to prove that the structure of $\widetilde{\mathcal{C}}_t$ is similar to the structure of $\widetilde{\mathcal{C}}_{t-1}$. The current section is devoted to prove the latter. At an intuitive level the lemmas presented in this section prove that:

1. A node which belongs to a non-trivial cluster of $\widetilde{\mathcal{C}}_{t-1}$ will either belong to the same non-trivial cluster in $\widetilde{\mathcal{C}}_t$ or it will form a trivial cluster in $\widetilde{\mathcal{C}}_t$. Thus, a node cannot change non-trivial clusters in consecutive rounds.
2. Two non-trivial clusters of $\widetilde{\mathcal{C}}_{t-1}$ cannot merge in $\widetilde{\mathcal{C}}_t$.
3. A cluster in $\widetilde{\mathcal{C}}_{t-1}$ cannot split in two or more non-trivial clusters in $\widetilde{\mathcal{C}}_t$.

Lemma 16.3. *Let ϵ be a small enough constant and C and C' be non-trivial clusters of $\widetilde{\mathcal{C}}_{t-1}$ and u, v two nodes in C, C' respectively. In $\widetilde{\mathcal{C}}_t$, u and v cannot belong to the same cluster.*

Proof. Assume towards a contradiction that u, v belong to the same cluster of $\widetilde{\mathcal{C}}_t$. Then both C and C' should have been large clusters with more than 9 nodes, as otherwise from Lemma 16.2 we have that in G_t, u and v have at most 1 common neighbor (the newly arrived node). This, contradicts

Property 7 since $|N_{G_t}(u) \cap N_{G_t}(v)| \geq (1-5\epsilon)\max\{|N_{G_t}(u)|, |N_{G_t}(v)|\} \geq (1-5\epsilon)\cdot 2 > 1$ for ϵ small enough.
Thus, both C and C' have at least 10 nodes each. Given that $C \cap C' = \emptyset$, we can upper bound $|N_{G_{t-1}}(u) \cap N_{G_{t-1}}(v)|$ by $|N_{G_{t-1}}(u) \setminus C| + |N_{G_{t-1}}(v) \setminus C'|$ which (by Property 2) is less than $3\epsilon(|N_{G_{t-1}}(u)| + |N_{G_{t-1}}(v)|) < 6\epsilon \max\{|N_{G_{t-1}}(u)|, |N_{G_{t-1}}(v)|\}$. In addition, we have $|N_{G_t}(u) \cap N_{G_t}(v)| \leq |N_{G_{t-1}}(u) \cap N_{G_{t-1}}(v)| + 1$ and by Property 7 $|N_{G_t}(u) \cap N_{G_t}(v)| \geq (1-5\epsilon)\max\{|N_{G_t}(u)|, |N_{G_t}(v)|\}$, By combining the upper and lower bounds on $|N_{G_t}(u) \cap N_{G_t}(v)|$ we get that $(1-11\epsilon)\max\{|N_{G_t}(u)|, |N_{G_t}(v)|\} < 1$ which is false for ϵ small enough. □

Lemma 16.4. *Let ϵ be a small enough constant and let C be a cluster in $\widetilde{\mathcal{C}}_{t-1}$ and v_t the newly arrived node at time t, then for any two distinct non-trivial clusters C_1 and C_2 in $\widetilde{\mathcal{C}}_t$ either $C_1 \cap C = \emptyset$ or $C_2 \cap C = \emptyset$.*

Proof. If C is singleton, the statement is trivially true as $C_1 \cap C_2 = \emptyset$. Next, we assume the contrary. Towards a contradiction assume that $C_1 \cap C \neq \emptyset$ and $C_2 \cap C \neq \emptyset$. Let $u \in C \cap C_1$ and $v \in C \cap C_2$. Then, by Property 11 u and v have a common neighbor in G_{t-1} and, hence, both clusters C_1 and C_2 of $\widetilde{\mathcal{C}}_t$ have an outgoing edge from C_1 and C_2, respectively, in G_t. By Lemma 16.2, we conclude that $|C_1| \geq 10, |C_2| \geq 10$. Since $C_1 \cap C_2 = \emptyset$ we can bound $|N_{G_t}(u) \cap N_{G_t}(v)|$ by $|N_{G_t}(u) \setminus C_1| + |N_{G_t}(v) \setminus C_2|$ which (by Property 2) is at most $3\epsilon(|N_{G_t}(u)| + |N_{G_t}(v)|) < 6\epsilon \max\{|N_{G_t}(u)|, |N_{G_t}(v)|\} \leq 6\epsilon \max\{|N_{G_{t-1}}(u)|, |N_{G_{t-1}}(v)|\} + 6\epsilon$.

At the same time $|N_{G_{t-1}}(u) \cap N_{G_{t-1}}(v)| \geq (1-5\epsilon)\max\{|N_{G_{t-1}}(u)|, |N_{G_{t-1}}(v)|\}$ from Property 7. By noting that $|N_{G_t}(u) \cap N_{G_t}(v)| \geq |N_{G_{t-1}}(u) \cap N_{G_{t-1}}(v)|$ we get:

$$6\epsilon \geq (1-11\epsilon)\max\{|N_{G_{t-1}}(u)|, |N_{G_{t-1}}(v)|\}$$

which is false for ϵ small enough. □

Lemma 16.5. *Let ϵ be a small enough constant and let C', C be two clusters in $\widetilde{\mathcal{C}}_{t-1}$, $\widetilde{\mathcal{C}}_t$ respectively. If $C \cap C' \neq \emptyset$ then $|C| < 2|C'|$.*

Proof. Let $u \in C \cap C'$. Note that $|N_{G_t}(u)| \leq |N_{G_{t-1}}(u)| + 1$. In addition, from Property 3 we get that $|N_{G_{t-1}}(u)| \leq |C'|/(1-3\epsilon)$ and from Property 6 we get that $|N_{G_t}(u)]| \geq (1-9\epsilon)|C|$. By combining the latter inequalities we get that $(1-9\epsilon)|C| \leq |C'|/(1-3\epsilon) + 1$, from which we can conclude that $|C| < 2|C'|$ for ϵ small enough. □

Proposition 16.5. *Let ϵ be a small enough constant, let C', C be two non-trivial clusters in $\widetilde{\mathcal{C}}_{t-1}$ and $\widetilde{\mathcal{C}}_t$, respectively, and v_t the newly arrived node at time t. Then, if $C \cap C' \neq \emptyset$ we have that all nodes in $C \setminus (C' \cup \{v_t\})$ form trivial clusters in $\widetilde{\mathcal{C}}_{t-1}$ and $|C \setminus C'| \leq |C|/2$.*

Proof. Note that if $|C|$ has size 2 the lemma follow trivially, so we assume that $|C| \geq 3$.

The fact that all nodes in $C \setminus (C' \cup \{v_t\})$ form trivial clusters in $\widetilde{\mathcal{C}}_t$ is a direct consequence of Lemma 16.3, since nodes from two different non-trivial clusters of $\widetilde{\mathcal{C}}_{t-1}$ cannot be in agreement in the same cluster of $\widetilde{\mathcal{C}}_t$. To prove that $|C \setminus C'| \leq |C|/2$ we argue that the intersection of C, C' must be large, i.e., it suffices to argue that $|C \cap C'| > |C|/2$.

Let $u \in C \cap C'$. Then from Property 1 we have that $|N_{G_{t-1}}(u) \cap C'| \geq (1-3\epsilon)|N_{G_{t-1}}(u)|$ and $|N_{G_t}(u) \cap C| \geq (1-3\epsilon)|N_{G_t}(u)|$. In addition, note that $N_{G_{t-1}}(u) \cap C' = N_{G_t}(u) \cap C'$, hence

$|N_{G_t}(u) \cap C'| \geq (1-3\epsilon)|N_{G_{t-1}}(u)| \geq (1-3\epsilon)(|N_{G_t}(u)|-1)$. By combing these inequalities we get: $|N_{G_t}(u) \cap C \cap C'| \geq (1-6\epsilon)|N_{G_t}(u)| - 1 + 3\epsilon \geq (1-9\epsilon)(1-6\epsilon)|C| - 1 + 3\epsilon > |C|/2$ where in the second inequality we used Property 6 and in the last inequality the fact that $|C| \geq 3$ and that the inequality $(1-9\epsilon)(1-6\epsilon)x - 1 + 3\epsilon > x/2$ is true for $x \geq 3$ and ϵ small enough. □

16.3 Bounding the competitive ratio

In this section we prove that the Online Agreement algorithm is a constant competitive algorithm. Before proceeding to the proof, we repeat some auxiliary notation and definitions. We denote by $\widetilde{\mathcal{C}}_t$ the clustering computed by the Agreement algorithm and by $\mathcal{C}_t$ the clustering solution computed by the Online Agreement algorithm at time t. In addition, we denote by $f_t, \tilde{f}_t$ the assignment functions which induce the clusterings $\mathcal{C}_t, \widetilde{\mathcal{C}}_t$, respectively. With a slight abuse of notation we denote by $f_t(S), \tilde{f}_t(S)$ the common cluster id assigned to a set of nodes S by the respective assignment functions. Let $\mathcal{C}$ be a clustering. We use $S(\mathcal{C}), H(\mathcal{C})$ to denote the set of nodes that belong to trivial and non-trivial, respectively, clusters of $\mathcal{C}$.

As underlined in Chapter 15, a crucial definition which permits us to keep track of the growth of an evolving cluster is the *origin cluster* definition which we repeat for completeness:

Definition 15.3 (Origin cluster)**.** *Let ID be a cluster id that is used by the assignment function f_t, and let t_{min} be the minimum t for which ID is used by $f_{t_{min}}$. The origin cluster of ID, denoted by $Origin(ID)$, is the cluster with id ID in the clustering induced by $f_{t_{min}}$.*

An important observation is that, by construction, for any cluster $C' \in \widetilde{\mathcal{C}}_t$ the function f_t[2] assigns the same cluster id to all nodes in C'. In addition, note that f_t may assign the latter cluster id, i.e., $f_t(C')$, also to nodes which form trivial clusters in $\widetilde{\mathcal{C}}_t$, i.e., nodes in $S(\widetilde{\mathcal{C}}_t)$. For that reason, for a cluster $C' \in \widetilde{\mathcal{C}}_t$ we denote the set of nodes in $S(\widetilde{\mathcal{C}}_t)$ to which f_t assigns cluster id $f_t(C')$ as $S_t^{C'}$, that is:

$$S_t^{C'} = \{u \in S(\widetilde{\mathcal{C}}_t) : f_t(u) = f_t(C')\}$$

We mention the following trivial proposition, which is implied by the *Assignment refinement phase* of Online Agreement, as it is heavily used in all of our lemmas.

Proposition 16.6. *Let C' be a cluster of $\widetilde{\mathcal{C}}_t$. Then $|C'| < (3/2)|Origin(f_t(C'))|$.*

In the following, we prove that any non-singleton cluster contains most of the nodes of its corresponding origin cluster.

Lemma 16.6. *Let ϵ be a small enough constant and C' a non-trivial cluster of $\widetilde{\mathcal{C}}_t$ such that $Origin(f_t(C')) \cap C' \neq \emptyset$. Then $|Origin(f_t(C')) \cap C'| > (1-20\epsilon)|Origin(f_t(C'))|$.*

Proof. For simplicity, let $C = Origin(f_t(C'))$ and assume w.l.o.g. that C was formed at some time $t_0 < t$ (otherwise the lemma follows trivially since $C = C'$). Let $x \in C \cap C'$. Since $x \in C$, Property 4 gives that $|N_{G_{t_0}}(x) \cap C| \geq (1-9\epsilon)|C|$. Thus, at time t the edges of x outside C', i.e.,

[2] note that also the function $\tilde{f}_t$ assigns the same cluster id to all nodes in C'.

$|N_{G_t}(x) \setminus C'|$ are at least $(1-9\epsilon)|C| - |C \cap C'|$. From Property 2 we get $|N_{G_t}(x) \setminus C'| < 3\epsilon |N_{G_t}(x)|$. By combining the lower and upper bound we get the following inequality:

$$(1-9\epsilon)|C| - |C \cap C'| < 3\epsilon |N_{G_t}(x)| \Rightarrow$$
$$\frac{(1-9\epsilon)|C| - |C \cap C'|}{3\epsilon} < |N_{G_t}(x)|$$

From Proposition 16.6 we know that $|C'| < 3/2|C|$ and from Property 3 we get that $(1-3\epsilon)|N_{G_t}(x)| \leq |C'|$. Thus:

$$(1-3\epsilon)|N_{G_t}(x)| < 3/2|C| \Rightarrow$$
$$|N_{G_t}(x)| < \frac{3|C|}{2(1-3\epsilon)}$$

Combining the lower and upper bound on $N_{G_t}(x)$ we get:

$$|C \cap C'| > |C| \left(1 - 9\epsilon - \frac{9\epsilon}{2(1-3\epsilon)}\right) > (1-20\epsilon)|C|$$

for ϵ small enough. □

Lemma 16.7. *Le ϵ be a small enough constant and C' a non-trivial cluster of $\widetilde{\mathcal{C}}_t$. Then $Origin(f_t(C')) \cap C' \neq \emptyset$.*

Proof. For simplicity, let $C = Origin(f_t(C'))$ and assume w.l.o.g. that C was formed at some time $t_0 < t$ (otherwise the lemma follows trivially since $C = C'$). Assume towards contradiction that C' is the first such cluster which does not intersect with $C = Origin(f_t(C'))$. Since $f_t(C') = f_{t_0}(C)$ there exists a node $u \in C'$ and a cluster C'' of $\widetilde{\mathcal{C}}_{t'}$ for $t' \in [t_0, t)$ such that $C'' \cap C \neq \emptyset$, $u \in C''$ and $f_{t'}(C'') = f_{t_0}(C)$. Indeed if such a node u and cluster C'' do not exist then $f_t(C') \neq f_{t_0}(C)$ because we assumed that C' is the first cluster s.t. $C' \cap C = \emptyset$ and $f_t(C') = f_{t_0}(C)$, and moreover, in order for C' to be assigned the same cluster id as C it should intersect a non-singleton cluster of $\widetilde{\mathcal{C}}_{t-1}$ which has the same cluster id as C.

We lower bound the number of edges u has towards nodes of C. To that end, note that at time t' from Property 4 it holds that $|N_{G_{t'}}(u) \cap C''| \geq (1-9\epsilon)|C''|$. In addition to that, due to the Lemma 16.6 we have that $|C \cap C''| \geq (1-20\epsilon)|C| > (1-20\epsilon)\frac{2}{3}|C''| > \frac{|C''|}{2}$, where the second inequality comes from the fact that $|C''| < 3/2|C|$ and the third inequality is true for ϵ small enough. By combining the last two inequalities we get:

$$\begin{aligned}
|N_{G_{t'}}(u) \cap (C'' \cap C)| &\geq |C'' \cap C| - |(C'' \cap C) \setminus N_{G_{t'}}(u)| && (1)\\
&\geq |C'' \cap C| - |C'' \setminus N_{G_{t'}}(u)| && (2)\\
&\geq |C''|/2 - 9\epsilon |C''| && (3)\\
&> |C''|/3 && (4)\\
&\geq \frac{(1-20\epsilon)}{3}|C| && (5)\\
&\geq \frac{|C|}{6} && (6)
\end{aligned}$$

Where:

1. from line (3) to line (4) $(1/2 - 9\epsilon) > 1/3$ is true for ϵ small enough;
2. from line (4) to line (5) we used Lemma 16.6; and
3. from line (5) to line (6) $\frac{(1-20\epsilon)}{3} \geq \frac{1}{6}$ is true for ϵ small enough.

Consequently at time t, since $C' \cap C = \emptyset$, $|N_{G_t}(u) \setminus C'| \geq |N_{G_t}(u) \cap C| \geq |N_{G_t}(u) \cap (C'' \cap C)| > \frac{|C|}{6}$. By combining the latter with $|N_{G_t}(u) \setminus C'| < \frac{3\epsilon}{1-3\epsilon}|C'|$ from Property 10 we get

$$|C'| > \frac{1-3\epsilon}{18\epsilon}|C| > \frac{3}{2}|C|$$

for ϵ small enough.

However, because of Proposition 16.6, we get that $|C'| < \frac{3}{2}|C|$, which is a contradiction. This concludes the proof. □

Note that by combining the two latter Lemmas 16.6 and 16.7 we can deduce that an origin cluster which gets assigned by an assignment function $f_{t'}$ cluster id s has a large intersection with a cluster of $\widetilde{\mathcal{C}}_t$ whose nodes get assigned cluster id s by an assignment function f_t for $t > t'$. Formally:

Corollary 16.7. *Le ϵ be a small enough constant and C' a non-trivial cluster of $\widetilde{\mathcal{C}}_t$. Then $|Origin(f_t(C')) \cap C'| > (1 - 20\epsilon)|Origin(f_t(C'))|$.*

Using Corollary 16.7 we can deduce that there cannot be two clusters C_1, C_2 of $\widetilde{\mathcal{C}}_t$ whose nodes get assigned the same cluster id by f_t.

Lemma 16.8. *Le ϵ be a small enough constant and C', C'' be two non-trivial clusters of $\widetilde{\mathcal{C}}_t$. Then $f_t(C') \neq f_t(C'')$.*

Proof. Towards a contradiction assume that nodes in C', C'' get assigned the same cluster id s by f_t and let $C = Origin(s)$. Then from Corollary 16.7 we have that $|C \cap C'| > (1 - 20\epsilon)|C|$, $|C \cap C''| > (1 - 20\epsilon)|C|$. In addition, C', C'' are different clusters of the same clustering $\widetilde{\mathcal{C}}_t$, hence they should not intersect, that is $C' \cap C'' = \emptyset$. Thus, we have $|C| \geq |C \cap C'| + |C \cap C''| > (2 - 40\epsilon)|C|$ which is false for ϵ small enough. □

An important quantity we need to bound is the number of nodes which belong to trivial clusters of $\widetilde{\mathcal{C}}_t$ and get assigned by f_t the same cluster id as nodes which belong to non-trivial clusters of $\widetilde{\mathcal{C}}_t$. To that end, Corollary 16.7 is crucial. The following lemmas bound the latter quantity.

Lemma 16.9. *Let ϵ be a small enough constant, C' a non-trivial cluster of $\widetilde{\mathcal{C}}_t$ and C the origin cluster which corresponds to cluster id $f_t(C')$, that is $C = Origin(f_t(C'))$. Then every node $u \in S_t^{C'}$ has at least $2|C|/9$ edges to nodes of C and the total number of edges between nodes in $S_t^{C'}$ and $C \cap C'$ is at least $\frac{|S_t^{C'}||C|}{10} > \frac{|S_t^{C'}||C'|}{15}$.*

Proof. Let $t_0 \leq t$ be the time when the origin cluster C was formed. By Corollary 16.7 we know that $|C \cap C'| > (1 - 20\epsilon)|C|$ and since by Proposition 16.6 it holds that $|C'| < 3/2|C|$ we get that $|C \cap C'| > |C'|/2$ for ϵ small enough.

Let $x \in S_t^{C'}$. Since $f_t(x) = f_{t_0}(C)$ there exists $t' \in [t_0, t]$ and a cluster C'' of $\widetilde{\mathcal{C}}_{t'}$ such that: (1) $f_{t'}(C'') = f_{t_0}(C)$; and (2) $x \in C''$. Again by the Corollary 16.7 we get that $|C \cap C''| \geq |C''|/2$ for ϵ small enough. Since $x \in C''$ we also have $|N_{G_{t'}}(x) \cap C''| \geq (1 - 9\epsilon)|C''|$. By combining the two latter inequalities we get that:

$$\begin{aligned}
|N_{G_{t'}}(x) \cap (C'' \cap C)| &\geq |C'' \cap C| - |(C'' \cap C) \setminus N_{G_{t'}}(x)| && (1)\\
&\geq |C'' \cap C| - |C'' \setminus N_{G_{t'}}(x)| && (2)\\
&\geq |C''|/2 - 9\epsilon|C''| && (3)\\
&> |C''|/3 && (4)\\
&> 2|C|/9 && (5)
\end{aligned}$$

Where:

1. from line (3) to line (4) we used the fact that $(1/2 - 9\epsilon) > 1/3$ holds for ϵ small enough; and
2. from line (4) to line (5) we used the fact that $|C''| < 3/2|C|$ holds by Proposition 16.6.

Consequently, every node of $S_t^{C'}$ has at least $2|C|/9$ edges to nodes of C. Using the latter we can deduce that the total number of edges between nodes of $S_t^{C'}$ and $C \cap C'$ is at least:

$$|S_t^{C'}|\frac{2}{9}|C| - |C \setminus C'||S_t^{C'}| > |S_t^{C'}|\frac{2}{9}|C| - 20\epsilon|C||S_t^{C'}| = |S_t^{C'}||C|\left(\frac{2}{9} - 20\epsilon\right) \geq \frac{|S_t^{C'}||C|}{10}$$

Where:

1. the first inequality comes from Corollary 16.7; and
2. the last inequality holds since $(\frac{2}{9} - 20\epsilon) \geq \frac{1}{10}$ for ϵ small enough.

To conclude the lemma just note that by Proposition 16.6 $|C'| < 3/2|C|$ and consequently $\frac{|S_t^{C'}||C|}{10} > \frac{|S_t^{C'}||C'|}{15}$.

□

Lemma 16.10. *Let ϵ be a small enough constant and C' a non-trivial cluster of $\widetilde{\mathcal{C}}_t$. Then $|S_t^{C'}| < 100\epsilon|C'|$.*

Proof. Let C be the origin cluster corresponding to cluster id $f_t(C')$, that is $C = Origin(f_t(C'))$. By Lemma 16.9 we know that the total number of edges between nodes of $S_t^{C'}$ and $C \cap C'$ is at least $\frac{|S_t^{C'}||C|}{10}$. Thus, by averaging there is a node $r \in C \cap C'$ with at least $\frac{|S_t^{C'}||C|}{10|C \cap C'|} > \frac{|S_t^{C'}||C|}{10|C|} > \frac{|S_t^{C'}|}{10}$ edges to nodes of $S_t^{C'}$. Thus, for that node r it holds that:

1. $|N_{G_t}(r) \setminus C'| > \frac{|S_t^{C'}|}{10}$; and

2. $|N_{G_t}(r) \setminus C'| < \frac{3\epsilon}{1-3\epsilon}|C'|$ from Property 10

By combining these two bounds we conclude that $|S_t^{C'}| < \frac{30\epsilon}{1-3\epsilon}|C'| < 100\epsilon|C'|$ for ϵ small enough. □

Before stating the main theorem of this section, we underline that by f_t^{OPT} we denote the assignment function which induces the optimal correlation clustering solution for graph G_t.

Theorem 16.8. *The Online Agreement algorithm is a constant competitive ratio algorithm, that is, for any time t*

$$\mathrm{cost}(f_t) \leq \Theta(1) \cdot \mathrm{cost}(f_t^{\mathrm{OPT}})$$

Proof. Fix a time t, from Lemma 16.1 we have that the cost of clustering $\widetilde{\mathcal{C}}_t$ is a constant factor approximation to the cost of the optimal correlation clustering solution for graph G_t. Thus to prove the theorem it suffices to prove that the cost of our solution is a constant factor approximation to the cost of $\widetilde{\mathcal{C}}_t$. W.l.o.g. we assume that $\mathrm{cost}(\tilde{f}_t) < \mathrm{cost}(f_t)$, otherwise the theorem becomes trivial.

First, note that for any non-trivial cluster C of $\widetilde{\mathcal{C}}_t$ both f_t and $\tilde{f}_t$ cluster all nodes of C together. Now, fix a node u which forms a trivial cluster in $\widetilde{\mathcal{C}}_t$. While $\tilde{f}_t$ clusters u as a singleton cluster f_t may cluster u in a larger cluster. We concentrate on the case where f_t clusters u in a larger cluster C^*, as this is the case where the cost of the two assignment functions may differ. Both assignment functions maintain, at all time, the following invariant: if two nodes v, v' belong to different non-trivial clusters of $\widetilde{\mathcal{C}}_t$ then they are assigned to different clusters in $\mathcal{C}_t$. Thus, C^* may contain the nodes of at most one non-trivial cluster of $\widetilde{\mathcal{C}}_t$.

We start by arguing that if $C^* \subseteq S(\widetilde{\mathcal{C}}_t)$ we can safely charge the cost that f_t pays for clustering all nodes of C^* together to what $\tilde{f}_t$ pays for clustering them apart. To argue the latter, it suffices to prove that $|(u,v) \notin E : u \in C^*, v \in C^*| \leq \Theta(1)|(u,v) \in E : u \in C^*, v \in V|$. To that end, note that $C^* \subseteq C' \cup S_{t'}^{C'}$ for some $t' < t$ and non-trivial cluster $C' \in \widetilde{\mathcal{C}}_{t'}$. W.l.o.g. assume t' is the last time the latter holds and let C be the origin cluster corresponding to $f_{t'}(C')$. Then, by Lemma 16.9 any node in C^* has at least $2|C|/9$ edges to nodes in C, hence $|(u,v) \in E : u \in C^*, v \in C^*| \geq |C^*| \cdot 2|C|/9 = \Theta(1)|C||C^*|$. At the same time we have that $|(u,v) \notin E : u \in C^*, v \in C^*| \leq |C^*|^2 \leq |C^*|(|C'| + |S_{t'}^{C'}|) \leq |C^*|(|C'| + 100\epsilon|C'|) = \Theta(1)|C^*||C|$, where we used Lemma 16.10 and the fact that C is the origin cluster of C'. Thus, for the rest of the proof we can assume that $C^* \not\subseteq S(\widetilde{\mathcal{C}}_t)$ by loosing only a constant factor in the approximation guarantees.

Since $C^* = C \cup S_t^C$ for some non-trivial cluster $C \in \widetilde{\mathcal{C}}_t$, we have that under the assignment function f_t all nodes in $C \cup S_t^C$ are clustered together while under the assignment function $\tilde{f}_t$ nodes in C are clustered together and each node in S_t^C is clustered as a singleton. To relate the two costs $\mathrm{cost}(\tilde{f}_t), \mathrm{cost}(f_t)$ we define the following sets of pairs of nodes and assume that pair of nodes are not ordered so that they are not double counted:

- $PCS(C) = \{(u,v) \in E : u \in C, v \in S_t^C\}$.
- $NCS(C) = \{(u,v) \notin E : u \in C, v \in S_t^C\}$

- $PSS(C) = \{(u,v) \in E : u \in S_t^C, v \in S_t^C\}$
- $NSS(C) = \{(u,v) \notin E : u \in S_t^C, v \in S_t^C\}$
- $PVS(C) = \{(u,v) \in E : u \in V_t \setminus (C \cup S_t^C), v \in S_t^C\}$
- $NVS(C) = \{(u,v) \notin E : u \in V_t \setminus (C \cup S_t^C), v \in S_t^C\}$
- $PCC(C) = \{(u,v) \in E : u \in C, v \in C\}$
- $NCC(C) = \{(u,v) \notin E : u \in C, v \in C\}$

Note that: (1) pairs of nodes in sets $PCS(C), PSS(C)$ contribute to $\text{cost}(\tilde{f}_t)$ but not to $\text{cost}(f_t)$; (2) pairs of nodes in sets $NCS(C), NSS(C)$ contribute to $\text{cost}(f_t)$ but not to $\text{cost}(\tilde{f}_t)$; (3) pairs of nodes in $PVS(C)$ and $NCC(C)$ contribute to both costs; and (4) pairs of nodes in $NVS(C)$ and $PCC(C)$ do not contribute to none of the two costs. In addition, we have that for every two different clusters $C, C' \in \widetilde{\mathcal{C}}_t$ by Lemma 16.8 the sets $S_t^C, S_t^{C'}$ do not intersect, hence the difference of the two costs can be rewritten as:

$$\begin{aligned}\text{cost}(f_t) - \text{cost}(\tilde{f}_t) &= \sum_{C \in \widetilde{\mathcal{C}}_t : |C|>1} (|NCS(C)| + |NSS(C)| - |PCS(C)| - |PSS(C)|) \\ &\leq \sum_{C \in \widetilde{\mathcal{C}}_t : |C|>1} (|NCS(C)| + |NSS(C)|)\end{aligned}$$

We are left to prove that $|NCS(C)| \leq \Theta(1)|PCS(C)|$ and $|NSS(C)| \leq \Theta(1)|PCS(C)|$ for every $C \in \widetilde{\mathcal{C}}_t$. This way we deduce that $\text{cost}(f_t) - \text{cost}(\tilde{f}_t) \leq \Theta(1) \sum_{C \in \widetilde{\mathcal{C}}_t : |C|>1} |PCS(C)|$. Note that the latter suffices to prove the Lemma since $\text{cost}(\tilde{f}_t) \geq \sum_{C \in \widetilde{\mathcal{C}}_t : |C|>1} |PCS(C)|$.

We lower bound the size of $PCS(C)$ by $\frac{|S_t^C||C|}{15}$ using[3] Lemma 16.9. In addition from the definition of set $NCS(C)$ we can upper bound its size by $|S_t^C||C|$, hence, deduce that $|NCS(C)| \leq 15|PCS(C)|$. Finally, note that for set $NSS(C)$ we have:

$$|NSS(C)| \leq |S_t^C|^2 < 100\epsilon |S_t^C||C| \leq 100\epsilon |PSC(C)|$$

where the first inequality comes from the definition of set $NSS(C)$, the second inequality from Lemma 16.10 and the third inequality from the lower bound on the size of $PSC(C)$. This concludes the proof. □

16.4 Bounding the worst-case recourse

In this section we bound the worst case recourse of any node by $O(\log(n))$ where n is the number of nodes in the final graph. To do the latter we prove that whenever a node changes from a non-trivial cluster $C_1' \in \widetilde{\mathcal{C}}_{t_1}$ whose nodes get assigned cluster id s_1 by f_{t_1} to another non-trivial cluster $C_2' \in \widetilde{\mathcal{C}}_{t_2}$ whose nodes get assigned cluster id s_2 by f_{t_2} then the origin cluster corresponding to the new cluster id is bigger than the one corresponding to the old cluster id by a multiplicative term. Since the maximum size of an origin cluster is n, changing non–trivial

[3]Note that cluster C corresponds to cluster C' in Lemma 16.9

clusters and consequently changing the origin cluster corresponding to the assigned cluster id can happen at most $\log(n)$ times.

We will first prove that if two origin clusters are "close" in size then they cannot be intersecting.

Lemma 16.11. *Let ϵ be a small enough constant and let C_1, C_2 be two origin clusters with cluster ids s_1, s_2 formed at different times t_1, t_2 respectively with $t_1 < t_2$. If $|C_2| < 5/4|C_1|$ then $C_1 \cap C_2 = \emptyset$.*

Proof. Towards a contradiction assume that $C_1 \cap C_2 \neq \emptyset$ and let $u \in C_1 \cap C_2$. As a first step of the proof we will argue that since $C_1 \cap C_2$ is non-empty then $|C_1 \cap C_2|$ must be large. Since $u \in C_1 \cap C_2$ from Property 4 we have that $|N_{Gt_1}(u) \cap C_1| > (1-9\epsilon)|C_1|$. Thus, $|N_{G_{t_2}}(u) \setminus C_2| \geq (1-9\epsilon)|C_1| - |C_1 \cap C_2|$. Combining the latter with the upper bound on $|N_{G_{t_2}}(u) \setminus C_2|$ from Property 10 we get:

$$(1-9\epsilon)|C_1| - |C_1 \cap C_2| < \frac{3\epsilon}{1-3\epsilon}|C_2| \Rightarrow \quad (1)$$

$$(1-9\epsilon)|C_1| - \frac{3\epsilon}{1-3\epsilon}|C_2| < |C_1 \cap C_2| \Rightarrow \quad (2)$$

$$(1-9\epsilon)|C_1| - \frac{15\epsilon}{4(1-3\epsilon)}|C_1| < |C_1 \cap C_2| \Rightarrow \quad (3)$$

$$(1-20\epsilon)|C_1| < |C_1 \cap C_2| \quad (4)$$

Where:

1. from line (2) to line (3) we used that $|C_2| < 5/4|C_1|$; and
2. from line (3) to line (4) we used that $(1-9\epsilon) - \frac{15\epsilon}{4(1-3\epsilon)} > (1-20\epsilon)$ is true for ϵ small enough.

Now, we will argue that at time $t_2 - 1$ there should be a node $v \in C_1 \cap C_2$ such that $f_{t_2-1}(v) \neq s_1$.

Let v_{t_2} be the newly arrived node at time t_2 and note that because of lemma 16.3, C_2 is formed either:

> **Case 1:** exclusively from nodes in $S(\widetilde{\mathcal{C}}_{t_2-1}) \cup \{v_{t_2}\}$ (by using **Rule 2**); or
>
> **Case 2:** by a subset of nodes $C' \neq \emptyset$ in a non-trivial cluster $C'' \in \widetilde{\mathcal{C}}_{t_2-1}$ and nodes in $S(\widetilde{\mathcal{C}}_{t_2-1}) \cup \{v_{t_2}\}$.

We will continue by proving that in neither of these two cases all nodes in $C_1 \cap C_2$ can have the same cluster id s_1 at time $t_2 - 1$.

Before doing so, note that by combining $(1-20\epsilon)|C_1| < |C_1 \cap C_2|$ and $|C_2| < 5/4|C_1|$ we can deduce that $(4/5)(1-20\epsilon)|C_2| < |C_1 \cap C_2|$ and consequently $|C_1 \cap C_2| > |C_2|/2$ for ϵ small enough.

In **Case 1** where C_2 is formed exclusively by nodes in $S(\widetilde{\mathcal{C}}_{t_2-1}) \cup \{v_{t_2}\}$, if all nodes $v \in C_1 \cap C_2$ have a cluster id s_1 at time $t_2 - 1$ then, since $|C_1 \cap C_2| > |C_2|/2$ by **Rule 2** of Algorithm 10 the new cluster would have had s_1, and not s_2, as a cluster id before the ***Assignment refinement***

phase. In addition, since $|C_2| < 5/4|C_1| < 3/2|C_1|$ the assignment function f_{t_2} will still assign s_1 after the ***Assignment refinement phase***, which is a contradiction. Thus, in this case all nodes in $C_1 \cap C_2$ cannot have the same cluster id s_1 at time $t_2 - 1$.

In **Case 2** note that all nodes in C' at time $t_2 - 1$ have a cluster id different from s_1, otherwise f_{t_2} would have assigned cluster id s_1 to all nodes in C_2 before the ***Assignment refinement phase*** and also after the ***Assignment refinement phase*** since $|C_2| < 5/4|C_1| < 3/2|C_1|$. Thus, since in **Case 2** C' is non-empty,

- either $C_1 \cap C_2 \cap C'$ is non-empty, and consequently there is a node in $C_1 \cap C_2 \supseteq C_1 \cap C_2 \cap C'$, whose cluster id at time $t_2 - 1$ is different from s_1; or
- $C_1 \cap C_2 \cap C'$ is empty and all nodes in $C_1 \cap C_2$ form trivial clusters in $\widetilde{\mathcal{C}}_{t_2-1}$. In that case, as in **Case 1** there should be a node in $C_1 \cap C_2$ whose id at time $t_2 - 1$ is different than s_1, otherwise Algorithm 10 would have assigned the same cluster id s_1 to C_2.

Consequently, there exists a node $v \in C_1 \cap C_2$ such that $f_{t_2-1}(v) \neq s_1$.

Since there exists a node $v \in C_1 \cap C_2$ such that $f_{t_2-1}(v) \neq s_1$, w.l.o.g. we can assume that v was the last node to change cluster id before time t_2. Let $C''' \in \widetilde{\mathcal{C}}_{t'}$ be last non-trivial cluster to which v belonged for $t' \in (t_1, t_2 - 1)$ before changing cluster id. We proceed by bounding the size of C'''. From Property 3 $|N_{G_{t_2}}(v)| \leq \frac{|C_2|}{1-3\epsilon} < \frac{5}{4}\frac{1}{1-3\epsilon}|C_1|$, hence $|N_{G_{t_2}}(v)| < \frac{5}{4}\frac{1}{1-3\epsilon}|C_1|$. At the same time from Property 6 $|N_{G_{t_2}}(v)| \geq |N_{G_{t'}}(v)| > (1 - 9\epsilon)|C'''|$, thus

$$|C'''| < \frac{1}{1-9\epsilon}\frac{5}{4}\frac{1}{1-3\epsilon}|C_1| < \frac{4}{3}|C_1|$$

for ϵ small enough. Now using the same arguments as in the beginning of the proof where we lower bounded the size of $C_1 \cap C_2$ we can argue that $|C''' \cap C_1| > (1 - 20\epsilon)|C_1|$ for ϵ small enough. Consequently, because w.l.o.g. we assumed that v is the last node of $C_1 \cap C_2$ to change cluster id, we have that $f_{t_2-1}(w) = f_{t'}(C'''), \forall w \in C_1 \cap C_2 \cap C'''$. In addition note that from $|C_1 \cap C_2| > (1 - 20\epsilon)|C_1|$ and $|C''' \cap C_1| > (1 - 20\epsilon)|C_1|$ we can deduce:

$$|C_1 \cap C_2 \cap C'''| > (1 - 40\epsilon)|C_1| > \frac{4}{5}(1 - 40\epsilon)|C_2| > \frac{|C_2|}{2}$$

where the last inequality holds for ϵ small enough.

Let C'''' be the origin cluster corresponding to $f_{t'}(C''')$. From corollary 16.7 we know that $|C''''| < \frac{|C'''|}{(1-20\epsilon)}$. Combining that with $|C_1 \cap C_2 \cap C'''| > \frac{4}{5}(1 - 40\epsilon)|C_2| \Rightarrow \frac{5}{4(1-40\epsilon)}|C''''| > |C_2|$. We conclude that $|C_2| < (3/2)|C''''|$.

Again, we will continue the proof by distinguishing between two cases:

> **Case 1:** where C_2 is formed exclusively by nodes in $S(\widetilde{\mathcal{C}}_{t_2-1}) \cup \{v_{t_2}\}$; and
>
> **Case 2:** where C_2 is formed by a subset of nodes $C' \neq \emptyset$ in a non-trivial cluster $C'' \in \widetilde{\mathcal{C}}_{t_2-1}$ and nodes in $S(\widetilde{\mathcal{C}}_{t_2-1}) \cup \{v_{t_2}\}$.

In **Case 1** we have that C_2 is formed by nodes in $S(\widetilde{\mathcal{C}}_{t_2-1}) \cup \{v_t\}$ where at least half of the nodes have cluster id $f_{t'}(C''')$ at time $t_2 - 1$. By **Rule 2** all nodes in C_2 get as a cluster id

$f_{t'}(C''')$ before the ***Assignment refinement phase*** and remain with that cluster id also after the ***Assignment refinement phase*** since $|C_2| < (3/2)|C''''|$. Thus all nodes in C_2 get cluster id $f_{t'}(C''')$ and C_2 is not an origin cluster in that case, which is a contradiction.

In **Case 2**, we can argue again that nodes in C' cannot have as a cluster id $f_{t'}(C''')$ since $|C_2| < (3/2)|C''''|$ and C_2 would not be an origin cluster. Thus, all nodes of C_2 whose cluster id is $f_{t'}(C''')$ at time $t_2 - 1$ form trivial clusters in $\widetilde{\mathcal{C}}_{t_2-1}$. From proposition 16.5 such nodes are at most $|C_2|/2$ which contradicts the fact that $|C_1 \cap C_2 \cap C'''| > \frac{|C_2|}{2}$.

Which leads to a contradiction. The lemma follows. □

We continue by proving that if a node gets a new cluster id by changing non-trivial cluster, then its new cluster must be larger than the old one by a at least a constant multiplicative factor.

Lemma 16.12. *Let node v be a node which belongs to a non-trivial cluster $C'_1 \in \widetilde{\mathcal{C}}_{t_1}$ and to a non-trivial cluster $C'_2 \in \widetilde{\mathcal{C}}_{t_2}$ for $t_1 < t_2$. If $f_{t_1}(C'_1) \neq f_{t_2}(C'_2)$ then denote by C_1 and C_2 the origin clusters corresponding to cluster ids $f_{t_1}(C'_1)$ and $f_{t_2}(C'_2)$ respectively. Then $|C_2| > (5/4)|C_1|$.*

Proof. Towards a contradiction assume that $|C_2| < 5/4|C_1|$.

From lemma 16.6 we have that $|C_1 \cap C'_1| \geq (1-20\epsilon)|C_1|$ and from Proposition 16.6 we also have that $|C'_1| < (3/2)|C_1|$. Combining the latter two inequalities we get $|C_1 \cap C'_1| \geq \frac{|C'_1|}{2}$ for ϵ small enough. Note that $|N_{G_{t_1}}(v) \cap (C'_1 \cap C_1)| \geq |N_{G_{t_1}}(v) \cap C'_1| - |C'_1 \setminus C_1| \geq (1-9\epsilon)|C'_1| - |C'_1|/2 > |C'_1|/3 > |C_1|/4$ where: (1) in the second inequality we used Property 6 and the fact that $|C_1 \cap C'_1| > |C'_1|/2$; (2) in the third inequality we used that $(1 - 9\epsilon - 1/2) > 1/3$ for ϵ small enough; and (3) in the last inequality we used that $|C_1| < \frac{|C'_1|}{1-20\epsilon}$ from lemma 16.6 and $(1 - 20\epsilon)/3 > 1/4$ for ϵ small enough.

We will continue by upper bounding $|N_{G_{t_2}}(v) \setminus C_2|$. To that end from lemma 16.6 and following the same reasoning as we did to bound $|C_1 \cap C'_1|$ we get that $|C_2 \cap C'_2| \geq |C'_2|/2$. In addition:

$$|N_{G_{t_2}}(v) \setminus C_2| \leq |N_{G_{t_2}}(v) \setminus C_2 \cap C'_2| \leq |N_{G_{t_2}}(v) \setminus C'_2| + |C'_2 \setminus C_2| \leq \frac{3\epsilon}{1-3\epsilon}|C'_2| + |C'_2|/2 < \\ < \left(\frac{3\epsilon}{1-3\epsilon} + \frac{1}{2}\right)\frac{|C_2|}{1-20\epsilon}$$

Where in the third inequality we used Property 10 and in the last lemma 16.6.

From lemma 16.11 we have that $C_1 \cap C_2 = \emptyset$ and $N_{G_{t_1}}(v) \subseteq N_{G_{t_2}}(v)$ so $|N_{G_{t_1}}(v) \cap C_1| \leq |N_{G_{t_2}}(v) \setminus C_2|$. Combining the lower and upper bound of the latter two quantities, for ϵ small enough we end up in a contradiction.

$$|C_1|/4 < \left(\frac{3\epsilon}{1-3\epsilon} + \frac{1}{2}\right)\frac{|C_2|}{1-20\epsilon} \Rightarrow |C_2| > 5/4|C_1|$$

Which leads to a contradiction. □

We are now ready to bound the recourse of our algorithm.

Theorem 16.9. *The Online Agreement algorithm has a worst case recourse of $O(\log n)$, that is:*

$$r(u) = O(\log n), \forall v \in V$$

Proof. Fix a node u and assume that at time t node u belongs to a non-trivial cluster $C \in \widetilde{\mathcal{C}}_t$ with cluster id s assigned by f_t. Note that node u will get assigned a new cluster id by an assignment function $f_{t''}$ whenever one of the following scenarios happen:

1. node u forms a trivial cluster in $\widetilde{\mathcal{C}}_{t'}$ for $t' > t$ and at a later time $t'' > t'$ enters a non-trivial cluster in $\widetilde{\mathcal{C}}_{t''}$ with a different cluster id s'; and
2. node u forms a trivial cluster in $\widetilde{\mathcal{C}}_{t'}$ for $t' > t$ and at a later time $t'' > t'$ its cluster id changes to s' by the ***Assignment refinement phase***.

Note that (1) can happen at most $O(\log n)$ times due to lemma 16.12 and each time it happens the recourse of u increases by 1. In addition, note that after (2) happens there are two possibilities; either u remains in a trivial cluster and it never changes its cluster id again or it enters a new non-trivial cluster with cluster id s''. The former can happen at most once and increases the recourse by 1, while the latter increases the recourse by 2 and by lemma 16.12 can happen at most $O(\log n)$ times. Thus the overall recourse of u is $O(\log n)$. □

17 Experiments

17.1 Datasets

Our study includes four graphs that are formed by user-to-user interactions. Specifically, we consider a Social network (musae-facebook), an email network (email-Enron), a collaboration network (ca-AstroPh), and a paper citation network (cit-HepTh). All but the cit-HepTh datasets are static undirected graphs. cit-HepTh has timestamps on the nodes indicating a natural arrival order and since it is directed, we transform it to an undirected graph by ignoring edge directions. In all of our datasets we removed all parallel-edges. Our datasets are obtained from SNAP [70] and their basic characteristics are summarized in Table 17.1.

17.2 Baselines

Pivot. As one of the baselines, we consider the 3-approximate pivoting algorithm introduced by Ailon et al. [2]. We refer to this algorithm as Pivot. In the offline model, Pivot works as follows. First, it creates a random order of the nodes, marks all nodes as unclustered, and then iterates over the nodes in the aforementioned random order. If the current node v is still unclustered, then v forms a cluster together with all of its unclustered neighbors and they are marked as clustered.

The naive way to use Pivot in the online setting is to re-run it from scratch following every node arrival, each time with a fresh random order. However, we re-use the same random order of the previously arrived nodes, which leads to an improved practical performance both in terms of recourse and running time. That is, whenever a new node arrives it is inserted in the preexisting random order at a random position (the relative order of the previously arrived nodes remains unchanged). We call the latter order the *new random order*. It is not hard to verify that all clustering choices of the algorithm regarding nodes which precede the newly arrived node in the new random order remain the same compared to the execution of the algorithm in the previous

	#nodes	#edges	online?
musae-facebook	22,470	171,002	no
email-Enron	36,692	183,831	no
ca-AstroPh	18,772	198,110	no
cit-HepTh	27,770	352,807	yes

Table 17.1: Basic characteristics of our datasets.

iterations (before the arrival of the last node). Thus, we only simulate the algorithm starting from visiting the index of the newly arrived node in the new random order onward.

Agree-Off. This baseline reruns the Agreement algorithm following each node arrival. In our experiments we set both parameters β and λ of [28] equal to 0.2, as this setting exhibited the best behavior in [28].

Greedy recourse minimization. For the baselines we assume that there is no consistent cluster id assignment to the clusters produced after each node arrival, and hence, measuring recourse as per our definition would give an unfair advantage to our (more designated) algorithm. To have a fair comparison, we measure recourse independently of the actual cluster ids that each algorithm assigns. That is, given two clusterings $\mathcal{C}_{t-1}, \mathcal{C}_t$ from consecutive runs of some algorithm we reassign the cluster ids of $\mathcal{C}_t$ such that we minimize the number of nodes that have distinct cluster ids in $\mathcal{C}_{t-1}$ and $\mathcal{C}_t$. Specifically, we construct a bipartite graph $B = (U \cup V, E)$ where each node of U (resp., V) represents a cluster $C_u \in \mathcal{C}_{t-1}$ (resp., $C_v \in \mathcal{C}_t$), and there is an edge $(u, v) \in E \cap \{U \times V\}$ with weight w if C_u overlaps with C_v on w nodes (if C_u and C_v do not overlap then there is no edge (u, v)). To re-assign the ids of the clusters in $\mathcal{C}_t$, we run the greedy $1/2$-approximate maximum bipartite matching on B, and each cluster $C \in \mathcal{C}_t$ that is matched with a cluster $C' \in \mathcal{C}_t$ gets the same cluster id as C; if C' remains unmatched then it receives a new unique cluster id [1]. Finally, the recourse is computed by counting the number of nodes with distinct cluster ids in $\mathcal{C}_{t-1}$ and $\mathcal{C}_t$. We apply this post-processing to all algorithms (including ours).

17.3 Setup

Our code is written in C++ and is available online[2]. We run our experiments on a e2-standard-16 Google Cloud instance, with 16 cores, 2.20GHz Intel(R) Xeon(R) processor, and 64 GiB main memory.

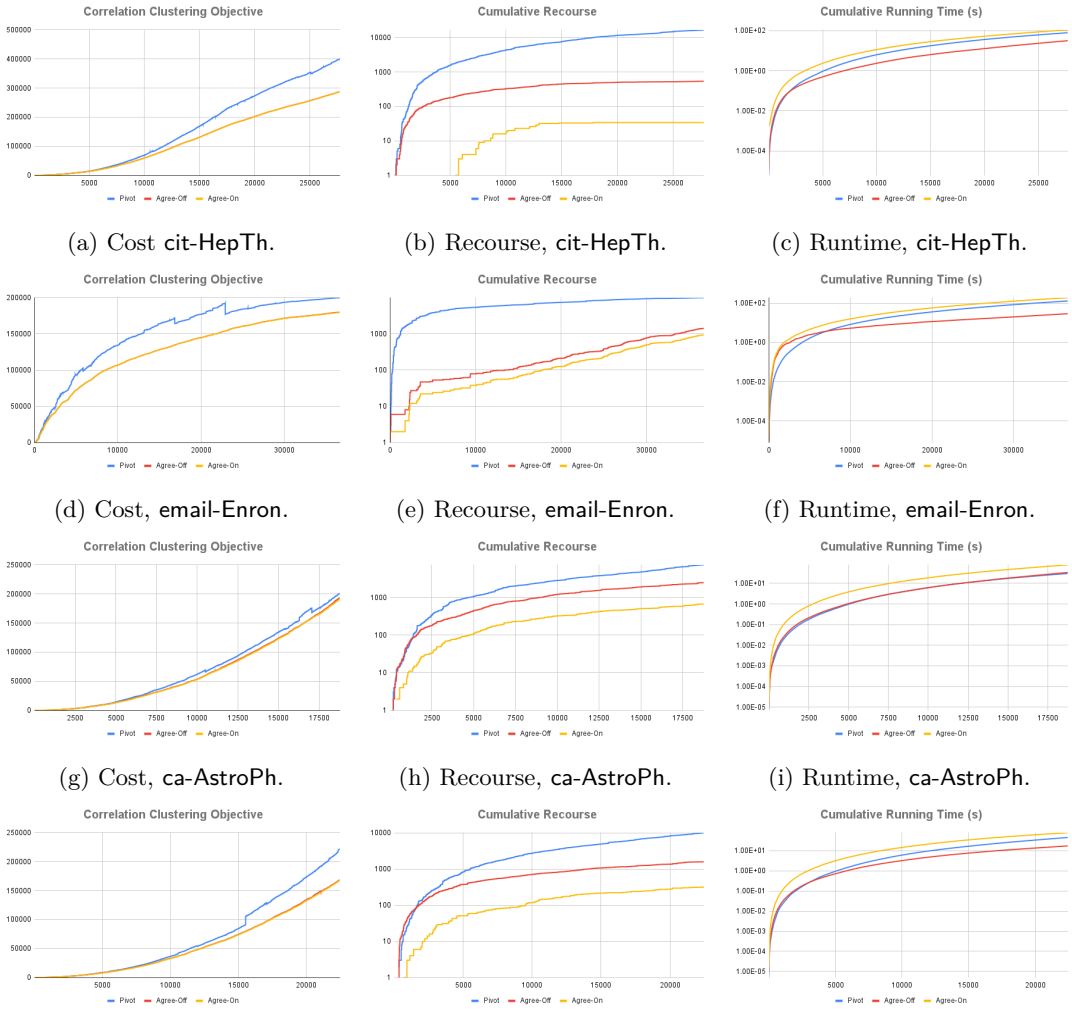
For the datasets email-Enron, ca-AstroPh, musae-facebook, we use a random arrival order, as the nodes are not timestamped. For the dataset cit-HepTh, we consider the arrival order implied by the timestamps on the nodes. Once a node v arrives, it reveals only its edges to the previously arrived nodes (the remaining edges are revealed once the other endpoint of each such edge arrives).

17.4 Results

Solution quality. Figure 17.1 shows the cost of the cluster produced by each of the algorithms that we consider throughout the sequence of node arrivals, on all datasets. In all datasets but ca-AstroPh, after a few node arrivals, the Agree-Off and Agree-On perform significantly better than Pivot. The latter implies that the solution calculated by Agree-On is at most a 3-approximation of the offline optimum.

[1] Although one can solve the maximum bipartite matching exactly, this task is expensive and orthogonal to our study.

[2] https://github.com/google-research/google-research/tree/master/online_correlation_clustering



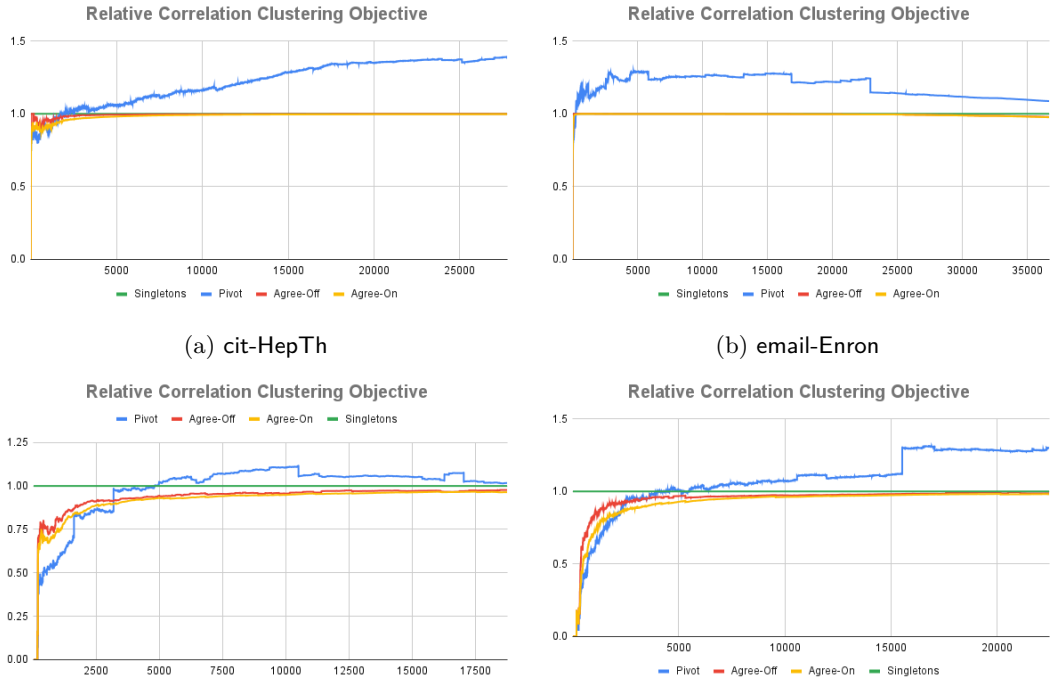
(a) Cost cit-HepTh. (b) Recourse, cit-HepTh. (c) Runtime, cit-HepTh.

(d) Cost, email-Enron. (e) Recourse, email-Enron. (f) Runtime, email-Enron.

(g) Cost, ca-AstroPh. (h) Recourse, ca-AstroPh. (i) Runtime, ca-AstroPh.

(j) Cost, musae-facebook (k) Recourse, musae-facebook. (l) Runtime, musae-facebook.

Figure 17.1: Comparison of our algorithm, Agree-On, with the two baselines Pivot and Agree-Off for the datasets cit-HepTh, email-Enron, ca-AstroPh, and musae-facebook. The first column compares the cost of the solution produced by each algorithm, the second and third column contain respectively the cumulative recourse and total run time of each algorithm in log-scale.



(a) cit-HepTh (b) email-Enron

(c) ca-AstroPh (d) musae-facebook

Figure 17.2: The quality of the clustering produced by the different algorithms relatively to SINGLETONS for the datasets cit-HepTh, email-Enron, ca-AstroPh, and musae-facebook.

.

	Pivot	Agree-Off	Agree-On
email-Enron	14	3	1
ca-AstroPh	10	2	1
musae-facebook	10	3	1
cit-HepTh	13	5	1

Table 17.2: The maximum recourse per node over the whole sequence of node arrivals, for the four datasets that we considered.

While not immediately visible from the plots, Agree-On performs slightly better than Agree-Off in all datasets. This is due to the fact that once a set of nodes C is clustered together at time t_1, Agree-On keeps the nodes in C clustered together at times $t > t_1$ even if the nodes in C obtain many outgoing edges from C, whereas Agree-Off would split C under such a scenario.

Our datasets are sparse throughout the arrival sequence. Sparse graphs tend to not have a good correlation clustering structure, and often the clustering that consists of only singleton clusters is a competitive solution; we denote such a solution as Singletons. We illustrate this in Figure 17.2, where we present the cost function relatively to Singletons. After a small number of node arrivals, Pivot performs clearly worse than Singletons, while Agree-Off and Agree-On always perform better or equally well compare to Singletons. These findings are not surprising. On the one hand, Pivot is more likely to create clusters with very small density which results to a lot of negative intra-cluster edges. On the other hand, Agree-Off (and hence Agree-On) creates singleton clusters if the graph contains no clear clustering structure; which implies that Agree-Off produces a clustering that is never worse than the one produced by Singletons. Unlike the case of our particular datasets, Agree-Off (and consequently Agree-On) often produces many medium-sized clusters on sparse graphs[3] [28] implying performance significantly better that Singletons.

We would like to underline that the best algorithms in terms of approximation guarantees for the correlation clustering problem are a 2.06-approximation LP-based variation of Pivot in [21] and a $1.94 + \epsilon$-approximation algorithm based on $O(1/\epsilon^2)$ rounds of the Sherali-Adams hierarchy in [26]. However, these algorithms due to their high computational cost become quickly infeasible to run even for graphs where the number of nodes is one or two orders of magnitude smaller than the ones we use in our experiments. Thus, the state-of-the-art algorithm for the correlation clustering problem both in terms of practical performance and provable guarantees is Pivot, and we successfully compare against it.

Recourse. Table 17.2 shows the maximum recourse of a node, for each dataset. Both Agree-Off and Agree-On perform significantly better than Pivot, while Agree-On never changes the cluster id of a node twice. Recall that the greedy reassignment post-processing benefits all algorithms.

Next, we investigate the total recourse that each algorithm incurs over the whole node-arrival sequence, i.e., the sum of the recourse of each node. The cummulative recourse plots for all datasets are in the second column of Figure 17.1. Agree-On consistently incurs 1-2 order of magnitude less recourse compared to Pivot, and up to an order of magnitude less recourse compared to Agree-Off.

[3]Unfortunately, it was not reasonable to run the whole arrival sequence on the same datasets, due to their size.

Running time. For simplicity the version of AGREE-ON described in Algorithm 10 computes the clustering $\tilde{C}_t$ by naively rerunning the Agreement algorithm on the graph G_t. However, we can easily use most of the prior information to compute the clustering $\tilde{C}_t$ without rerunning the Agreement algorithm from scratch. To this end, note that for any node $u \in G_t \setminus \{v_t\}$ its degree can be updated in $O(1)$ by checking if u and the newly arrived node v_t share an edge. Moreover let $u, v \in G_{t-1}$ be two nodes which share an edge, then the size of the symmetric difference of their neighborhoods can be updated in $O(1)$ by checking if the newly arrived node is the neighbor of exactly one of them. Thus, all $\epsilon-$agreements between nodes $u, v \in G_{t-1}$ can be updated in linear time. In addition, checking the $\epsilon-$agreement between two nodes u, v without any prior information takes time $O(\min\{d_u, d_v\})$. Consequently we can check the $\epsilon-$agreement between the newly arrived node and all its neighbors in time $O(d_{v_t}^2)$. Finally, Lines $7-43$ of AGREE-ON for each new node arrival can be easily implemented in linear time. Overall, a careful implementation of AGREE-ON would yield to an algorithm whose complexity for the whole arrival sequence is $O(\sum_u d_u^2) + n \cdot O(n+m) = O(nm)$ (matching Pivot's complexity) instead of $O(n^2 m)$ from a naive implementation of the pseudocode.

As expected, in terms of running time, AGREE-OFF performs comparably to PIVOT as shown in the third column of Figure 17.1. On the other hand, due to the required post-processing (Lines $7-43$ of the pseudocode) following the computation made by AGREE-OFF, AGREE-ON performs less than $2\times$ slower compared to AGREE-OFF and PIVOT. Since the latter post-processing requires linear time, this seems to imply that in practice all these three algorithms require $O(n)$ time per node arrival. This is in contrast to their trivial worst-case performance of $O(\sum_{v \in V} d^2(v))$ time for AGREE-OFF and $O(m)$ time for PIVOT, per update.

18 Conclusion

This thesis concentrated on beyond worst-case analysis. After Part I which served as a motivation for why we do need a beyond worst-case analysis framework, Part II and Part III were devoted to learning-augmented online algorithms and Part IV to online clustering with recourse.

One promising direction would be to generalize the techniques of Part II and design general methods for power management using machine learning predictions. Indeed, in the classical online speed scaling problem Gupta et al. [50] propose an optimal online algorithm which they analyze using a primal-dual technique tailored to the non-linear optimization objective. Thus, an interesting open direction would be to translate the primal-dual analysis and techniques of [50] into the learning-augmented setting so as to handle more general power management problems.

Another interesting direction in the field of learning-augmented algorithms is to use machine learning predictions to improve the runtime of classical algorithms. In [33] the authors show how to speed up the Hungarian algorithm which is a classical primal-dual algorithm for computing a maximum weight matching in a bipartite graph. Although [33] considers a specific problem and speeds up a specific algorithm for that problem, it shows at a conceptual level how machine learning predictions can be used to guide an algorithm to find an optimal solution faster in a robust way. There are general algorithmic primitives which can be used to solve a large class of problems. Can we incorporate machine learning predictions in those primitives and speed up the algorithms' runtime?

Finally, in learning-augmented algorithms, predictions have been used in two ways: (1) estimating problem parameters which permit us to take better decisions [33]; or (2) the prediction itself gives us a hint about the optimal solution [72]. To the best of our knowledge the interplay between different types of prediction is unexplored and the closest work to that line of thought [47] studies how to combine multiple machine learning predictors of the same type. An interesting direction would be to understand the interplay between different forms of advice. What are the limits of a specific type of advice? Can different forms of advice provide complementary information? The high-level goal would be to find a problem and two natural types of prediction where each type, individually, is not enough to get guarantees that are better than the worst-case of classical online algorithms. However, when both predictions are combined, the improvement is significant.

A Extensions and deferred proofs of Part II

A.1 Pure online algorithms for uniform deadlines

Since most related results concern the general speed scaling problem, we give some insights about the uniform speed scaling problem in the online setting without predictions. We first give a lower bound on the competitive ratio for any online algorithm for the simplest case where $D = 2$ and then provide an almost tight analysis of the competitive ratio of AVR.

Theorem A.1. *There is no (randomized) online algorithm with an (expected) competitive ratio better than* $\Omega\left((6/5)^{\alpha}\right)$.

Proof. Consider $D = 2$ and two instances J_1 and J_2. Instance J_1 consists of only one job that is released at time 0 with workload 1 and J_2 consists of the same first job with a second job which starts at time 1 with workload 2.

In both instances, the optimal schedule runs with uniform speed at all time. In the first instance, it runs the single job for 2 units of time at speed $1/2$. The energy-cost is therefore $1/2^{\alpha-1}$. In the second instance, it first runs the first job at speed 1 for one unit of time and then the second job at speed 1 for 2 units of time. Hence, it has an energy-cost of 3.

Now consider an online algorithm. Before time 1 both instances are identical and the algorithm therefore behaves the same. In particular, it has to decide how much work of job 1 to process between time 0 and 1. Let us fix some $\gamma \geq 0$ as a threshold for the amount of work dedicated to job 1 by the algorithm before time 1. We have the following two cases depending on the instance.

1. If the algorithm processes more that γ units of work on job 1 before time 1 then for instance J_1 the energy cost is at least γ^{α}. Hence the competitive ratio is at least $\gamma^{\alpha} \cdot 2^{\alpha-1}$.

2. On the contrary, if the algorithm works less than γ units of work before the release of the second job then in instance J_2 the algorithm has to complete at least $3 - \gamma$ units of work between time 1 and 3. Hence, its competitive ratio is at least $2/3 \cdot ((3-\gamma)/2)^{\alpha}$.

Choosing γ such that these two competitive ratios are equal gives $\gamma = \frac{3}{3^{1/\alpha}4^{1-1/\alpha}+1}$ and yields a lower bound on the competitive ratio of at least:

$$2^{\alpha-1}\left(\frac{3}{3^{1/\alpha}4^{1-1/\alpha}+1}\right)^{\alpha}.$$

This term asymptotically approaches $1/2 \cdot (6/5)^\alpha$ and this already proves the theorem for deterministic algorithms. More precisely, it proves that any deterministic algorithm has a competitive ratio of at least $\Omega\left((6/5)^\alpha\right)$ on at least one of the two instances J_1 or J_2. Hence, by defining a probability distribution over inputs such that $p(J_1) = p(J_2) = \frac{1}{2}$ and applying Yao's minimax principle we get that the expected competitive ratio of any randomized online algorithm is at least

$$(1/2) \cdot 2^{\alpha-1} \left(\frac{3}{3^{1/\alpha} 4^{1-1/\alpha} + 1} \right)^\alpha .$$

which again gives $\Omega\left((6/5)^\alpha\right)$ as lower bound, this time against randomized algorithms. □

We now turn ourselves to the more specific case of the AVR algorithm with the following two results. We recall that the AVR algorithm was shown to be $2^{\alpha-1} \cdot \alpha^\alpha$-competitive by Yao et al. [88] in the general deadlines case. In the case of uniform deadlines, the competitive ratio of AVR is actually much better and proofs are much less technical than the original analysis of Yao et al. Recall that for each job i with workload w_i, release i, and deadline $i + D$; AVR defines a speed $s_i(t) = w_i/D$ if $t \in [i, i + D]$ and 0 otherwise.

Theorem A.2. *AVR is 2^α-competitive for the uniform speed scaling problem.*

Proof. Let (w, D, T) be a job instance and s_{OPT} be the speed function of the optimal schedule for this instance. Let s_{AVR} be the speed function produced by the Average Rate heuristic on the same instance. It suffices to show that for any time t we have

$$s_{\mathrm{AVR}}(t) \leq 2 \cdot s_{\mathrm{OPT}}(t).$$

Fix some t. We assume w.l.o.g. that the optimal schedule runs each job j isolated for a total time of p_j^*. By optimality of the schedule, the speed during this time is uniform, i.e., exactly w_j/p_j^*. Denote by j_t the job that is processed in the optimal schedule at time t.

Let j be some job with $j \leq t \leq j + D$. It must be that

$$\frac{w_j}{p_j^*} \leq \frac{w_{j_t}}{p_{j_t}^*} = s_{\mathrm{OPT}}(t). \tag{A.1}$$

Note that all jobs j with $j \leq t \leq j + D$ are processed completely between $t - D$ and $t + D$. Therefore,

$$\sum_{j:j\leq t\leq j+D} p_j^* \leq 2D.$$

With (A.1) it follows that

$$\sum_{j:j\leq t\leq j+D} w_j \leq s_{\mathrm{OPT}}(t) \sum_{j:j\leq t\leq j+D} p_j^* \leq 2D \cdot s_{\mathrm{OPT}}(t).$$

We conclude that

$$s_{\mathrm{AVR}}(t) = \sum_{j:j\le t\le j+D} \frac{w_j}{D} \le 2 \cdot s_{\mathrm{OPT}}(t). \qquad \square$$

Next, we show that our upper bound on the exponential dependency in α of the competitive ratio for AVR (in Theorem A.2) is tight for the uniform deadlines case.

Theorem A.3. *Asymptotically (α approaches ∞), the competitive ratio of the AVR algorithm for the uniform deadlines case is at least*

$$\frac{2^\alpha}{e\alpha}$$

Proof. Assume $\alpha > 2$ and consider a two-job instance with one job arriving at time 0 of workload 1 and one job arriving at time $(1 - 2/\alpha)D$ with workload 1. One can check that the optimal schedule runs at constant speed throughout the whole instance for a total energy of

$$\left(\frac{2}{(2-2/\alpha)D}\right)^\alpha \cdot (2 - 2/\alpha)D.$$

On the other hand, on interval $[(1 - 2/\alpha)D, D]$, AVR runs at speed $2/D$. This implies the following lower bound on the competitive ratio:

$$\frac{(2/D)^\alpha \cdot (2/\alpha)D}{\left(\frac{2}{(2-2/\alpha)D}\right)^\alpha \cdot (2 - 2/\alpha)D} = \frac{2^\alpha}{\alpha}\left(1 - \frac{1}{\alpha}\right)^{\alpha-1}$$

which approaches to $2^\alpha/(e\alpha)$ as α tends to infinity. $\square$

A.2 Impossibility results for learning-augmented speed scaling

This section is devoted to prove some impossibility results about learning augmented algorithms in the context of speed scaling. We first prove that our trade-offs between consistency and robustness are essentially optimal. Again, we describe an instance as a triple (w, D, T).

Theorem A.4. *Assume a deterministic learning-augmented algorithm is $(1 + \epsilon/3)^{\alpha-1}$-consistent for any $\alpha \ge 1$ and any small enough constant $\epsilon > 0$ (independently of D). Then the worst case competitive ratio of this algorithm cannot be better than $\Omega\left(\frac{1}{\epsilon}\right)^{\alpha-1}$.*

Proof. Fix D big enough so that $\lceil \epsilon D \rceil \le 2 \cdot (\epsilon D)$. Consider two different job instances J_1 and J_2: J_1 contains only one job of workload 1 released at time 0 and J_2 contains an additional job of workload $1/\epsilon$ released at time $\lceil \epsilon D \rceil$. On the first instance, the optimal cost is $1/D^{\alpha-1}$ while the optimum energy cost for J_2 is $(1/\lceil \epsilon D \rceil)^{\alpha-1} + D/(\epsilon D)^\alpha \le (1/\epsilon)^\alpha \cdot ((1 + \epsilon)/D^{\alpha-1})$.

Assume the algorithm is given the job of workload 1 released at time 0 and additionally the prediction consists of one job of workload $1/\epsilon$ released at time $\lceil \epsilon D \rceil$. Note that until time $\lceil \epsilon D \rceil$ the algorithm cannot tell the difference between instances J_1 and J_2.

Depending on how much the algorithm works before time $\lceil \epsilon D \rceil$, we distinguish the following cases.

1. If the algorithm works more that $1/2$ then the energy spent by the algorithm until time $\lceil \epsilon D \rceil$ is at least

$$(1/2)^\alpha / (\lceil \epsilon D \rceil)^{\alpha-1} = \Omega\left(\frac{1}{\epsilon D}\right)^{\alpha-1}.$$

2. However, if it works less than $1/2$ then on instance J_2, a total work of at least $(1/\epsilon + 1 - 1/2) = (1/2 + 1/\epsilon)$ remains to be done in D time units. Hence the energy consumption on instance J_2 is at least

$$\frac{(1/2 + 1/\epsilon)^\alpha}{D^{\alpha-1}}.$$

If the algorithm is $(1+\epsilon/3)^{\alpha-1}$-consistent, then it must be that the algorithm works more that $1/2$ before time $\lceil \epsilon D \rceil$ otherwise, by the second case of the analysis, the competitive ratio is at least

$$\frac{(1/2 + 1/\epsilon)^\alpha}{(1/\epsilon)^\alpha (1+\epsilon)} = \frac{(1+\epsilon/2)^\alpha}{1+\epsilon} > (1+\epsilon/3)^{\alpha-1},$$

where the last inequality holds for $\alpha > 4$ and ϵ small enough.

However it means that if the algorithm was running on instance J_1 (i.e. the prediction is incorrect) then by the first case the approximation ratio is at least $\Omega\left(\frac{1}{\epsilon}\right)^{\alpha-1}$. □

We then argue that one cannot hope to rely on some l_p norm for $p < \alpha$ to measure error.

Theorem A.5. *Fix some α and D and let p such that $p < \alpha$. Suppose there is an algorithm which on some prediction w^{pred} computes a solution of value at most*

$$C \cdot \text{OPT} + C' \cdot \|w - w^{\text{pred}}\|_p^p.$$

Here C and C' are constants that can be chosen as an arbitrary function of α and D.

Then it also exists an algorithm for the online problem (without predictions) which is $(C+\epsilon)$-competitive for every $\epsilon > 0$.

In other words, predictions do not help, if we choose $p < \alpha$.

Proof. In the online algorithm we use the prediction-based algorithm A_P as a black box. We set the prediction $\tilde{w}$ to all 0. We forward each job to A_P, but scale its work by a large factor M. It is obvious that by scaling the optimum of the instance increases exactly by a factor M^α. The error in the prediction, however, increases less:

$$\|M \cdot w - M \cdot w^{\text{pred}}\|_p^p = M^p \cdot \|w - w^{\text{pred}}\|_p^p.$$

We run the jobs as A_P does, but scale them down by M again. Thus, we get a schedule of value

$$M^{-\alpha}(M^\alpha \cdot C \cdot \mathrm{OPT} + M^p \cdot C' \cdot \|w - w^{\mathrm{pred}}\|_p^p) = C \cdot \mathrm{OPT} + M^{p-\alpha} \cdot C' \cdot \|w - w^{\mathrm{pred}}\|_p^p. \quad \text{(A.2)}$$

Now if we choose M large enough, the second term in (A.2) becomes insignificant. First, we relate the prediction error to the optimum. First note that

$$\mathrm{OPT} \geq (1/D^\alpha) \cdot ||w||_\alpha^\alpha$$

since the optimum solution cannot be less expensive than running all jobs i disjointly at speed w_i/D for time D. Second note that $\|w\|_p^p \leq ||w||_\alpha^\alpha$ since $|x|^p \leq |x|^\alpha$ for any $x \geq 1$ (recall that we assumed our workloads to be integral). Hence we get that,

$$\|w - w^{\mathrm{pred}}\|_p^p = \|w\|_p^p \leq D^\alpha \cdot \mathrm{OPT}\,.$$

Choosing M sufficiently large gives $M^{p-\alpha}C'D^\alpha < \epsilon$, which implies that (A.2) is at most $(C + \epsilon)\,\mathrm{OPT}$. □

We terminate this section by proving an impossibility result regarding the evolving prediction model of Chapter 8. Indeed, we prove that the dependence on the λ parameter in Theorem 8.3 is essentially tight. For completeness we repeat the error definition in that model:

$$\mathrm{err}^{(\lambda)} = \sum_t \sum_{i=t+1}^{\infty} |w_i^{\mathrm{real}} - w_i^{\mathrm{pred}}(t)|^\alpha \cdot \lambda^{i-t}.$$

In the following we allow the parameter λ to be a function of D and we write $\lambda(D)$.

Theorem A.6. *Let* $\mathrm{err}\,(\lambda)$ *the error in the evolving prediction model be defined with some* $0 < \lambda(D) < 1$ *that can depend on* D*. Suppose there is an algorithm which computes a solution of value at most*

$$C \cdot \mathrm{OPT} + C'(D) \cdot \mathrm{err}^{(\lambda)},$$

where C *is independent of* D *and* $C'(D) = o\left(\frac{1-\lambda(D)^D}{\lambda(D)^D} \cdot \frac{1}{D^\alpha}\right)$*. Then there also exists an algorithm for the online problem (without predictions) which is* $(C + \epsilon)$*-competitive for every* $\epsilon > 0$*.*

In particular, note that for λ independent of D, it shows that an exponential dependency in D is needed in $C'(D)$ as we get in Theorem 8.3.

Proof. The structure of the proof is similar to that of Theorem A.5. We pass an instance to the assumed algorithm, but set the prediction to all 0. Unlike the previous proof, we keep the same workloads when passing the jobs, but subdivide D in to $D \cdot k$ time steps where k will be specified later. This will decrease the cost of every solution by k^α.

Take an instance with interval length D. Like in the proof of Theorem A.5 we have that

$$\|w^{\mathrm{real}}\|_\alpha^\alpha \leq D^\alpha \cdot \mathrm{OPT}\,.$$

Consider the error parameter $\mathrm{err}^{(\lambda)\prime}$ for the instance with $D' = D \cdot k$. We observe that

$$\begin{aligned}
\mathrm{err}^{(\lambda)\prime} &= \sum_t \sum_{i=t+1}^{\infty} |w^{\mathrm{real}}_{k\cdot i}|^\alpha \cdot \lambda(D')^{k(i-t)} \\
&\leq ||w^{\mathrm{real}}||_\alpha^\alpha \cdot \sum_{i=1}^{\infty} \lambda(D')^{k\cdot i} \\
&\leq ||w^{\mathrm{real}}||_\alpha^\alpha \frac{\lambda(D')^k}{1-\lambda(D')^k} \\
&\leq D^\alpha \frac{\lambda(D')^k}{1-\lambda(D')^k} \cdot \mathrm{OPT}
\end{aligned}$$

Hence, by definition the algorithm produces a solution of cost

$$C \cdot \mathrm{OPT}\,/k^\alpha + C'(D')\,\mathrm{err}^{(\lambda)\prime} \leq (C/k^\alpha + D^\alpha \frac{\lambda(D')^k}{1-\lambda(D')^k} C'(D')) \cdot \mathrm{OPT}$$

for the subdivided instance. Transferring it to the original instance, we get a cost of

$$(C + k^\alpha D^\alpha \frac{\lambda(D')^k}{1-\lambda(D')^k} C'(D')) \cdot \mathrm{OPT}$$

Therefore, if $k^\alpha \frac{\lambda(D\cdot k)^k}{1-\lambda(D\cdot k)^k} C'(D\cdot k)$ tends to 0 as k grows, for any $\epsilon > 0$, we can fix k big enough so that the cost of the algorithm is at most $(C+\epsilon)\,\mathrm{OPT}$. □

A.3 A shrinking lemma

Recall that by applying the earliest-deadline-first policy, we can normalize every schedule to run at most one job at each time. We say, it is run *isolated*. Moreover, if a job is run isolated, it is always better to run it at a uniform speed (by convexity of $x \mapsto x^\alpha$ on $x \geq 0$). Hence, an optimal schedule can be characterized solely by the total time p_j each job is run. Given such p_j we will give a necessary and sufficient condition of when a schedule that runs each job isolated for p_j time exists. Note that we assume we are in the general deadline case, each job j comes with a release r_j and deadline d_j and the EDF policy might cause some jobs to be preempted.

Lemma A.7. *Let there be a set of n jobs with release times r_j and deadlines d_j for each job j. Let p_j denote the total duration that j should be processed. Scheduling the jobs isolated earliest-deadline-first, with the constraint to never run a job before its release time, will complete every job j before time d_j if and only if for every interval $[t, t']$ it holds that*

$$\sum_{j : t \leq r_j, d_j \leq t'} p_j \leq t' - t \tag{A.3}$$

Proof. For the one direction, let t, t' such that Equation (A.3) is not fulfilled. Since the jobs with $t \leq r_j$ cannot be processed before t, the last such job j' to be completed must finish after

$$t + \sum_{j : t \leq r_j, d_j \leq t'} p_j > t + t' - t = t' \geq d_{j'}$$

For the other direction, we will schedule the jobs earliest-deadline-first and argue that if the schedule completes some job after its deadline, then Equation (A.3) is not satisfied for some interval $[t, t']$.

To this end, let j' be the first job that finishes strictly after $d_{j'}$ and consider the interval $I_0 = [r_{j'}, d_{j'}]$. We now define the following operator that transforms our interval I_0 into an interval I_1. Consider $t_{\inf}$ to be the smallest release time among all jobs that are processed in interval I_0 and define $I_1 = [t_{\inf}, d_{j'}]$. We apply iteratively this operation to obtain interval I_{k+1} from interval I_k. We claim the following properties that we prove by induction.

1. For any $k \geq 0$, the machine is never idle in interval I_k.
2. For any $k \geq 0$, all jobs that are processed in I_k have a deadline $\leq d_{j'}$.

For $I_0 = [r_{j'}, d_{j'}]$, since job j' is not finished by time $d_{j'}$ it must be that the machine is never idle in that interval. Additionally, if a job is processed in this interval, it must be that its deadline is earlier that $d_{j'}$ since we process in EDF order. Assume both items hold for I_k and then consider I_{k+1} that we denote by $[a_{k+1}, d_{j'}]$. By construction, there is a job denoted j_{k+1} released at time a_{k+1} that is not finished by time a_k. Therefore the machine cannot be idle at any time in $[a_{k+1}, a_k]$ hence at any time in I_{k+1} by the induction hypothesis. Furthermore, consider a job processed in $I_{k+1} \setminus I_k$. It must be that its deadline is earlier that the deadline of job j_{k+1}. But job j_{k+1} is processed in interval I_k which implies that its deadline is earlier than $d_{j'}$ and ends the induction.

Denote by k' the first index such that $I_{k'} = I_{k'+1}$. We define $I_\infty = I_{k'}$. By construction, it must be that all jobs processed in I_∞ have release time in I_∞ and by induction the machine is never idle in this interval and all jobs processed in I_∞ have deadline in I_∞.

Since job j' is not finished by time $d_{j'}$ and by the previous remarks we have that

$$\sum_{j : r_j, d_j \in I_\infty} p_j > |I_\infty|$$

which yields a counter example to (A.3). □

We can now prove two shrinking lemmas that are needed in the procedure Robustify and its generalization to general deadlines.

Lemma A.8. *Let $0 \leq \mu < 1$. For any instance $\mathcal{I}$ consider the instance $\mathcal{I}'$ where the deadline of job j is set to $d'_j = r_j + (1-\mu)(d_j - r_j)$ (i.e. we shrink each job by a $(1-\mu)$ factor). Then*

$$\mathrm{OPT}(\mathcal{I}') \leq \frac{\mathrm{OPT}(\mathcal{I})}{(1-\mu)^{\alpha-1}}$$

Additionally, assuming $0 \leq \mu < 1/2$, consider the instance $\mathcal{I}''$ where the deadline of job j is set to $d''_j = r_j + (1-\mu)(d_j - r_j)$ and the release time is set to $r''_j = r_j + \mu(d_j - r_j)$. Then

$$\mathrm{OPT}(\mathcal{I}'') \leq \frac{\mathrm{OPT}(\mathcal{I})}{(1-2\mu)^{\alpha-1}}$$

Proof. W.l.o.g. we can assume that the optimal schedule s for $\mathcal{I}$ runs each job isolated and at a uniform speed. By optimality of the schedule and convexity, each job j must be run at a constant speed s_j for a total duration of p_j. Consider the first case and define a speed $s'_j = \frac{s_j}{1-\mu}$ for all j (hence the total processing time becomes $p'_j = (1-\mu) \cdot p_j$).

Assume now in the new instance $\mathcal{I}'$ we run jobs earliest-deadline-first with the constraint that no job is run before its release time (with the processing times p'_j). We will prove using Lemma A.7 that all deadlines are satisfied. Consider now an interval $[t, t']$ we then have that

$$\sum_{j:t\leq r_j, d'_j \leq t'} p'_j = (1-\mu) \cdot \sum_{j:t\leq r_j, d'_j \leq t'} p_j \leq (1-\mu) \cdot \sum_{j:t\leq r_j, d_j \leq \frac{t'-\mu t}{1-\mu}} p_j$$

where the last inequality comes from the fact that $t' \geq d'_j = d_j - \mu(d_j - r_j)$ which implies that $d_j \leq \frac{t'-\mu r_j}{1-\mu} \leq \frac{t'-\mu t}{1-\mu}$ by using $r_j \geq t$. By Lemma A.7 and the fact that s is a feasible schedule for $\mathcal{I}$ we have that

$$\sum_{j:t\leq r_j, d'_j \leq t'} p'_j \leq (1-\mu) \cdot \left(\frac{t'-\mu t}{1-\mu} - t\right) = (1-\mu) \cdot \frac{t'-t}{1-\mu} = t' - t$$

which implies by Lemma A.7 that running all jobs EDF with processing time p'_j satisfies all deadlines d'_j. Now notice the cost of this schedule is at most $\frac{1}{(1-\mu)^{\alpha-1}}$ times the original schedule s which ends the proof (each job is ran $\frac{1}{1-\mu}$ times faster but for a time $(1-\mu)$ times shorter).

The proof of the second case is similar. Note that for any $[t, t']$, if

$$\begin{aligned} d''_j &= r_j + (1-\mu)(d_j - r_j) = (1-\mu)d_j + \mu r_j \leq t' \\ r''_j &= r_j + \mu(d_j - r_j) = (1-\mu)r_j + \mu d_j \geq t \end{aligned}$$

then we have

$$\begin{aligned} (1-\mu)d_j \leq t' - \mu r_j &\leq t' - \frac{\mu}{1-\mu}(t - \mu d_j) \\ \iff (1-\mu)d_j - \frac{\mu^2}{1-\mu}d_j &\leq t' - \frac{\mu}{1-\mu} \cdot t \\ \iff d_j((1-\mu)^2 - \mu^2) &\leq (1-\mu)t' - \mu t \\ \iff d_j &\leq \frac{(1-\mu)t' - \mu t}{1-2\mu} \end{aligned}$$

Similarly, we have

$$\begin{aligned} (1-\mu)r_j \geq t - \mu d_j &\geq t - \frac{\mu}{1-\mu}(t' - \mu r_j) \\ \iff (1-\mu)r_j - \frac{\mu^2}{1-\mu}r_j &\geq t - \frac{\mu}{1-\mu} \cdot t' \\ \iff r_j &\geq \frac{(1-\mu)t - \mu t'}{1-2\mu} \end{aligned}$$

Notice that $\frac{(1-\mu)t'-\mu t}{1-2\mu} - \frac{(1-\mu)t-\mu t'}{1-2\mu} = \frac{t'-t}{1-2\mu}$

Therefore, if we set the speed that each job s''_j is processed to $s''_j = \frac{s_j}{1-2\mu}$ then we have a processing time $p''_j = (1-2\mu) \cdot p_j$ and we can write

$$\begin{aligned}\sum_{j:t\le r''_j, d''_j \le t'} p''_j &= (1-2\mu) \cdot \sum_{j:t\le r''_j, d''_j \le t'} p_j \\ &\le (1-2\mu) \cdot \sum_{j:\frac{(1-\mu)t-\mu t'}{1-2\mu} \le r_j, d_j \le \frac{(1-\mu)t'-\mu t}{1-2\mu}} p_j \\ &\le (1-2\mu) \cdot \frac{t'-t}{1-2\mu} = t'-t\end{aligned}$$

by Lemma A.7. Hence we can conclude similarly as in the previous case. □

A.4 Making an algorithm noise tolerant

The idea for achieving noise tolerance is that by Lemma A.8 we know that if we delay each job's arrival slightly (e.g., by ηD) we can still obtain a near optimal solution. This gives us time to reassign arriving jobs within a small interval in order to make the input more similar to the prediction. We first, in Appendix A.4.1, generalize the error function err to a more noise tolerant error function err_η. We then, in Appendix A.4.2, give a general procedure for making an algorithm noise tolerant (see Theorem A.9).

A.4.1 Noise tolerant measure of error

For motivation, recall the example given in the main body. Specifically, consider a predicted workload w^{pred} defined by $w_i^{\text{pred}} = 1$ for those time steps i that are divisible by a large constant, say 1000, and let $w_i^{\text{pred}} = 0$ for all other time steps. If the real instance w^{real} is a small shift of w^{pred} say $w_{i+1}^{\text{real}} = w_i^{\text{pred}}$ then the prediction error $\text{err}(w^{\text{real}}, w^{\text{pred}})$ is large although w^{pred} intuitively forms a good prediction of w^{real}. To overcome this sensitivity to noise, we generalize the definition of err.

For two workload vectors w, w', and a parameter $\eta \ge 0$, we say that w is in the η-neighborhood of w', denoted by $w \in N_\eta(w')$, if w can be obtained from w' by moving the workload at most ηD time steps forward or backward in time. Formally $w \in N(w')$ if there exists a solution $\{x_{ij}\}$ to the following system of linear equations[1]:

$$\begin{aligned} w_i &= \sum_{j=i-\eta D}^{i+\eta D} x_{ij} \qquad \forall i \\ w'_j &= \sum_{i=j-\eta D}^{j+\eta D} x_{ij} \qquad \forall j \end{aligned}$$

The concept of η-neighborhood is inspired by the notion of earth mover's distance but is adapted to our setting. Intuitively, the variable x_{ij} denotes how much of the load w_i has been moved to time unit j in order to obtain w'. Also note that it is a symmetric and reflexive relation, i.e., if

[1]To simplify notation, we assume that ηD evaluates to an integer and we have extended the vectors w and w' to take value 0 outside the range $[0, T-D]$.

$w \in N_\eta(w')$ then $w' \in N_\eta(w)$ and $w \in N_\eta(w)$.

We now generalize the measure of prediction error as follows. For a parameter $\eta \geq 0$, an instance w^{real}, and a prediction w^{pred}, we define the η-prediction error, denoted by err_η, as

$$\text{err}_\eta(w^{\text{real}}, w^{\text{pred}}) = \min_{w \in N_\eta(w^{\text{pred}})} \text{err}(w^{\text{real}}, w)\,.$$

Note that by symmetry we have that $\text{err}_\eta(w^{\text{real}}, w^{\text{pred}}) = \text{err}_\eta(w^{\text{pred}}, w^{\text{real}})$. Furthermore, we have that $\text{err}_\eta = \text{err}$ if $\eta = 0$ but it may be much smaller for $\eta > 0$. To see this, consider the vectors w^{pred} and $w_i^{\text{real}} = w_{i+1}^{\text{pred}}$ given in the motivational example above. While $\text{err}(w^{\text{pred}}, w^{\text{real}})$ is large, we have $\text{err}_\eta(w^{\text{pred}}, w^{\text{real}}) = 0$ for any η with $\eta D \geq 1$. Indeed the definition of err_η is exactly so as to allow for a certain amount of noise (calibrated by the parameter η) in the prediction.

A.4.2 Noise tolerant procedure

We give a general procedure for making an algorithm $\mathcal{A}$ noise tolerant under the mild condition that $\mathcal{A}$ is monotone: we say that an algorithm is monotone if given a predictor w^{pred} and duration D, the cost of scheduling a workload w is at least as large as that of scheduling a workload w' if $w \geq w'$ (coordinate-wise). That increasing the workload should only increase the cost of a schedule is a natural condition that in particular all our algorithms satisfy.

Theorem A.9. *Suppose there is a monotone learning-augmented online algorithm $\mathcal{A}$ for the uniform speed scaling problem, that given prediction w^{pred}, computes a schedule of an instance w^{real} of value at most*

$$\min\{C \cdot \text{OPT} + C' \,\text{err}(w^{\text{real}}, w^{\text{pred}}), C'' \,\text{OPT}\}\,.$$

Then, for every $\eta \geq 0$, $\zeta > 0$ there is a learning-augmented online algorithm Noise-Robust($\mathcal{A}$)*, that given prediction w^{pred}, computes a schedule of w^{real} of value at most $((1+\eta)(1+\zeta))^{O(\alpha)}$ times*

$$\min\{C \cdot \text{OPT} + (1/\zeta)^{O(\alpha)}(C + C') \,\text{err}_\eta(w^{\text{real}}, w^{\text{pred}}), C'' \,\text{OPT}\}\,.$$

The pseudo-code of the online algorithm Noise-Robust($\mathcal{A}$), obtained from $\mathcal{A}$, is given in Algorithm 12.

The algorithm constructs a vector $w^{\text{online}} \in N_\eta(w^{\text{real}})$ while trying to minimize $\mathsf{err}(w^{\text{online}}, w^{\text{pred}})$. Each component w_i^{online} will be finalized at time $i + \eta D$. Hence, we forward the jobs to $\mathcal{A}$ with a delay of ηD.

The vector is constructed as follows. Suppose a job w_i^{real} arrives. The algorithm first (see Steps 4-15) greedily assigns the workload to the time steps $j = i - \eta D, i - \eta D + 1, \ldots, i + \eta D$ from left-to-right subject to the constraint that no time step receives a workload higher than $(1+\zeta)w_j^{\text{pred}}$. If not all workload of w_i^{real} was assigned in this way, then the overflow is assigned uniformly to the time steps from $i - \eta D$ to $i + \eta D$ (Steps 17-20). Since each w_j^{online} can only receive workloads during time steps $j - \eta D, \ldots, j + \eta D$, it will be finalized at time $j + \eta D$. Thus,

Algorithm 12 NOISE-ROBUST($\mathcal{A}$)

Require: Algorithm $\mathcal{A}$, prediction w^{pred}, and $\eta \geq 0, \zeta > 0$
1: Initialize $\mathcal{A}$ with prediction $\overline{w}_i^{\text{pred}} = (1+\zeta)w_{i-\eta D}^{\text{pred}}$ and duration $(1-2\eta)D$
2: Let w^{online} and $\overline{w}^{\text{real}}$ be workload vectors, initialized to 0
3: **on time step** i **do**
4: $W \leftarrow w_i^{\text{real}}$
5: **for** $j \in \{i-\eta D, \ldots, i+\eta D\}$ **do**
6: **if** $w_j^{\text{online}} + W \leq (1+\zeta)w_j^{\text{pred}}$ **then**
7: $x_{ij} \leftarrow W$
8: $W \leftarrow 0$
9: $w_j^{\text{online}} \leftarrow w_j^{\text{online}} + W$
10: **else if** $w_j^{\text{online}} < (1+\zeta)w_j^{\text{pred}}$ **then**
11: $x_{ij} \leftarrow (1+\zeta)w_j^{\text{pred}} - w_j^{\text{online}}$
12: $W \leftarrow W - x_{ij}$
13: $w_j^{\text{online}} \leftarrow (1+\zeta)w_j^{\text{pred}}$
14: **end if**
15: **end for**
16: *// Distribute remaining workload W evenly*
17: **for** $j \in \{i-\eta D, \ldots, i+\eta D\}$ **do**
18: $x_{ij} \leftarrow x_{ij} + W/(2\eta D+1)$
19: $w_j^{\text{online}} \leftarrow w_j^{\text{online}} + W/(2\eta D+1)$
20: **end for**
21: $\overline{w}_i^{\text{real}} \leftarrow w_{i-\eta D}^{\text{online}}$
22: Feed the job with workload $\overline{w}_i^{\text{real}}$ to $\mathcal{A}$
23: **end on**

at time i we can safely forward $w_{i-\eta D}^{\text{online}}$ to the algorithm $\mathcal{A}$. Hence, we set the workload of the algorithm's instance to $\overline{w}_i^{\text{real}} = w_{i-\eta D}^{\text{online}}$ (Steps 21-22). This shift together with the fact that a job w_i^{real} may be assigned to $w_{i+\eta D}^{\text{online}}$, i.e., ηD time steps forward in time, is the reason why we run each job with an interval of length $(1-2\eta)D$. Shrinking the interval of each job allows to make this shift and reassignment while still guaranteeing that each job is finished by its original deadline.

For an example, consider Figure A.1. Here we assume that $\eta D = 1$ and for illustrative purposes that $\zeta = 0$. At time 0, a workload $w_0^{\text{real}} = 1$ is released. The algorithm NOISE-ROBUST($\mathcal{A}$) then greedily constructs w^{online} by filling the available slots in $w_{-1}^{\text{pred}}, w_0^{\text{pred}}$, and w_1^{pred}. Since $w_0^{\text{pred}} = 3$, it fits all of the workload of w_0^{real} at time 0. Similarly the workloads w_2^{real} and w_3^{real} both fit under the capacity given by w^{pred}. Now consider the workload $w_4^{\text{real}} = 2$ released at time 4. At this point, the available workload at time 2 is fully occupied and one there is one unit of workload left at time 3. Hence, NOISE-ROBUST($\mathcal{A}$) will first assign the one unit of w_4^{real} to the third time slot and then split the remaining unit of workload unit uniformly across the time steps $3, 4, 5$. The obtained vector w^{online} is depicted on the right of Figure A.1. The workload w^{online} is then fed online to the algorithm $\mathcal{A}$ (giving a schedule of w^{online} and thus of w^{real}) so that at time i, $\mathcal{A}$ receives the job $\overline{w}_i^{\text{real}} = w_{i+\eta D}^{\text{online}} = w_{i+1}^{\text{online}}$ with a deadline of $i + (1-2\eta)D = i + D - 2$. This deadline is chosen so as to guarantee that a job is finished by $\mathcal{A}$ within its original deadline. Indeed, by this selection, the last part of the job w_4^{real} that was assigned to w_5^{online} is guaranteed to finish by time $6 + D - 2 = 4 + D$ which is its original deadline.

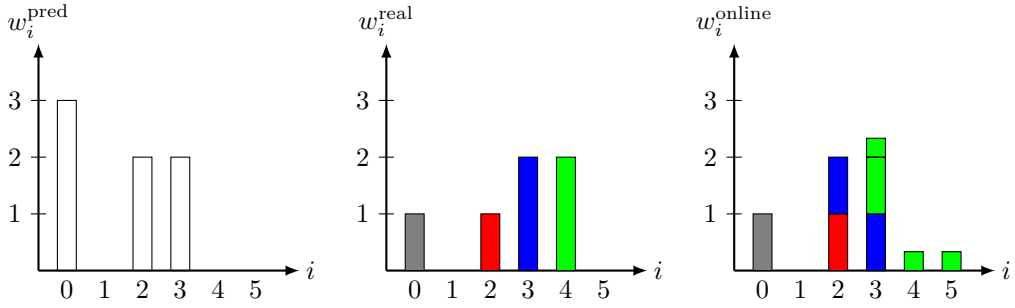


Figure A.1: An example of the construction of the vector w^{online} from w^{real} and w^{pred}.

Having described the algorithm, we proceed to analyze its guarantees which will prove Theorem A.9.

Analysis. We start by analyzing the noise tolerance of Noise-Robust($\mathcal{A}$).

Lemma A.10. *The schedule computed by Noise-Robust($\mathcal{A}$) has cost at most* $(1+O(\eta))^{\alpha}C''$ OPT.

Proof. Let OPT and OPT$'$ denote the cost of an optimum schedule of the original instance w^{real} with duration D and the instance $\overline{w}^{\text{real}}$ with duration $(1-2\eta)D$ fed to $\mathcal{A}$, respectively. The lemma then follows by showing that

$$\text{OPT}' \leq (1+O(\eta))^{\alpha}\,\text{OPT}\ .$$

To show this inequality, consider an optimal schedule s of w^{real} subject to the constraint that every job w_i^{real} is scheduled within the time interval $[i+2\eta D, i+(1-2\eta)D]$. By Lemma A.8, we have that the cost of this schedule is at most $(1+O(\eta))^{\alpha}$ OPT. The statement therefore follows by arguing that s also gives a feasible schedule of $\overline{w}^{\text{real}}$ with duration $(1-2\eta)D$. To see this note that Noise-Robust($\mathcal{A}$) moves the workload w_i^{real} to a subset of $\overline{w}_i^{\text{real}}, \overline{w}_{i+1}^{\text{real}}, \ldots, \overline{w}_{i+2\eta D}^{\text{real}}$. All of these jobs are allowed to be processed during $[i+2\eta D, i+(1-2\eta)D]$. It follows that the part of these jobs that corresponds to w_i^{real} can be processed in the computed schedule s (whenever it processes w_i^{real}) since s process that job in the time interval $[i+2\eta D, i+(1-2\eta)D]$. By doing this "reverse-mapping" for every job, we can thus use s as a schedule for the instance $\overline{w}^{\text{real}}$ with duration $(1-2\eta)D$. □

We now proceed to analyze the consistency and smoothness. The following lemma is the main technical part of the analysis. We use the common notation $(a)^+$ for $\max\{a,0\}$.

Lemma A.11. *The workload vector w^{online} produced by Noise-Robust($\mathcal{A}$) satisfies*

$$\sum_i \left[\left(w_i^{online} - (1+\zeta)w_i^{\text{pred}}\right)^+\right]^{\alpha} \leq O(1/\zeta)^{3\alpha} \cdot \min_{w\in N_\eta(w^{\text{real}})} \sum_i \left[\left(w_i - w_i^{\text{pred}}\right)^+\right]^{\alpha}\ .$$

The more technical proof of this lemma is given in Appendix A.4.2. Here, we explain how it implies the consistency and smoothness bounds of Theorem A.9. For a workload vector w, we

use the notation $\mathrm{OPT}(w)$ and $\mathrm{OPT}'(w)$ to denote the cost of an optimal schedule of workload w with duration D and $(1-2\eta)D$, respectively. Now let $\hat{w}^{\text{online}}$ be the workload vector defined by

$$\hat{w}_i^{\text{online}} = \max\{w_i^{\text{online}}, (1+\zeta)w_i^{\text{pred}}\}\,.$$

We analyze the cost of the schedule produced by $\mathcal{A}$ for $\hat{w}^{\text{online}}$ (shifted by ηD). This also bounds the cost of running $\mathcal{A}$ with $\overline{w}^{\text{real}}$: Since $\mathcal{A}$ is monotone, the cost of the schedule computed for the workload $\hat{w}^{\text{online}}$ (shifted by ηD) can only be greater than that computed for $\overline{w}^{\text{real}}$ which equals w^{online} (shifted by ηD). Furthermore, we have by Lemma A.11 that

$$\begin{aligned}\mathsf{err}(\hat{w}^{\text{online}}, (1+\zeta)w^{\text{pred}}) &= \sum_i \left[\left(w_i^{\text{online}} - (1+\zeta)w_i^{\text{pred}}\right)^+\right]^\alpha \\ &\leq O(1/\zeta)^{3\alpha}\mathsf{err}_\eta(w^{\text{real}}, w^{\text{pred}})\,.\end{aligned} \tag{A.4}$$

It follows by the assumptions on $\mathcal{A}$ that the schedule computed by Noise-Robust($\mathcal{A}$) has cost at most

$$\begin{aligned}&C\cdot\mathrm{OPT}'(\hat{w}^{\text{online}}) + C'\cdot\mathsf{err}(\hat{w}^{\text{online}}, (1+\zeta)w^{\text{pred}}) \\ &\qquad\leq C\cdot\mathrm{OPT}'(\hat{w}^{\text{online}}) + O(1/\zeta)^{3\alpha}\cdot C'\cdot\mathsf{err}_\eta(w^{\text{real}}, w^{\text{pred}})\,.\end{aligned}$$

The following lemma implies the consistency and smoothness, as stated in Theorem A.9, by relating $\mathrm{OPT}'(\hat{w}^{\text{online}})$ with the cost $\mathrm{OPT} = \mathrm{OPT}(w^{\text{real}})$.

Lemma A.12. *We have*

$$\mathrm{OPT}'(\hat{w}^{\text{online}}) \leq ((1+\eta)(1+\zeta))^{O(\alpha)}\left(\mathrm{OPT}(w^{\text{real}}) + O(1/\zeta)^{4\alpha}\,\mathrm{err}_\eta(w^{\text{real}}, w^{\text{pred}})\right)\,.$$

Proof. By the exact same arguments as in the proof of Theorem 6.1, we have that for any $\eta' > 0$

$$\begin{aligned}\mathrm{OPT}'(\hat{w}^{\text{online}}) &\leq (1+\eta')^\alpha\,\mathrm{OPT}'((1+\zeta)w^{\text{pred}}) + O(1/\eta')^\alpha\,\mathrm{err}(\hat{w}^{\text{online}}, (1+\zeta)w^{\text{pred}}) \\ &\leq (1+\eta')^\alpha\,\mathrm{OPT}'((1+\zeta)w^{\text{pred}}) + O(1/\eta')^\alpha O(1/\zeta)^{3\alpha}\,\mathrm{err}_\eta(w^{\text{real}}, w^{\text{pred}})\,,\end{aligned}$$

where we used (A.4) for the second inequality.

By Lemma A.8, we have that decreasing the duration by a factor $(1-2\eta)$ only increases the cost by factor $(1+O(\eta))^\alpha$ and so $\mathrm{OPT}'((1+\zeta)w^{\text{pred}}) \leq (1+O(\eta))^\alpha\,\mathrm{OPT}((1+\zeta)w^{\text{pred}})$. Furthermore, as a schedule for a workload w^{pred} gives a schedule for $(1+\zeta)w^{\text{pred}}$ by increasing the speed by a factor $(1+\zeta)$, we get

$$\mathrm{OPT}'((1+\zeta)w^{\text{pred}}) \leq (1+O(\eta))^\alpha(1+\zeta)^\alpha\,\mathrm{OPT}(w^{\text{pred}})\,.$$

Hence, by choosing $\eta' = \zeta$,

$$\mathrm{OPT}'(\hat{w}^{\text{online}}) \leq (1+O(\eta))^\alpha(1+\zeta)^{2\alpha}\,\mathrm{OPT}(w^{\text{pred}}) + O(1/\zeta)^{4\alpha}\,\mathrm{err}_\eta(w^{\text{real}}, w^{\text{pred}}).$$

It remains to upper bound $\mathrm{OPT}(w^{\text{pred}})$ by $\mathrm{OPT}(w^{\text{real}})$. Let $w = \mathrm{argmin}_{w\in N_\eta(w^{\text{pred}})}\mathrm{err}(w, w^{\text{real}})$ and so $\mathrm{err}_\eta(w^{\text{real}}, w^{\text{pred}}) = \mathrm{err}(w^{\text{real}}, w)$. By again applying the arguments of Theorem 6.1, we

have for any $\eta' > 0$

$$\mathrm{OPT}(w) \leq (1+\eta')^{\alpha}\,\mathrm{OPT}(w^{\mathrm{real}}) + O(1/\eta')^{\alpha}\,\mathrm{err}(w^{\mathrm{real}}, w)\,.$$

Now consider an optimal schedule of w subject to that for every time t the job w_t is scheduled within the interval $[t+\eta D, t+(1-\eta)D]$. By Lemma A.8, we have that this schedule has cost at most $(1+O(\eta))^{\alpha}\,\mathrm{OPT}(w)$. Observe that this schedule for w also defines a feasible schedule for w^{pred} since the time of any job is shifted by at most ηD in w. Hence, by again selecting $\eta' = \zeta$,

$$\begin{aligned}\mathrm{OPT}(w^{\mathrm{pred}}) &\leq (1+O(\eta))^{\alpha}\,\mathrm{OPT}(w)\\ &\leq (1+O(\eta))^{\alpha}\left((1+\zeta)^{\alpha}\,\mathrm{OPT}(w^{\mathrm{real}}) + O(1/\zeta)^{\alpha}\,\mathrm{err}_{\eta}(w^{\mathrm{real}}, w^{\mathrm{pred}})\right)\end{aligned}$$

Finally, by combining all inequalities, we get

$$\mathrm{OPT}'(\hat{w}^{\mathrm{online}}) \leq (1+O(\eta))^{2\alpha}\left((1+\zeta)^{3\alpha}\,\mathrm{OPT}(w^{\mathrm{real}}) + O(1/\zeta)^{4\alpha}\,\mathrm{err}_{\eta}(w^{\mathrm{real}}, w^{\mathrm{pred}})\right)$$

□

Proof of Lemma A.11

The lemma is trivially true if there were no jobs that had remaining workloads to be assigned uniformly, i.e., if we always have $W = 0$ at Step 16 of Noise-Robust($\mathcal{A}$). So suppose that there was at least one such job and consider the directed bipartite graph G with bipartitions A and B defined as follows:

- A contains a vertex for each component of w^{real} and B contains one for each component of w^{online}. In other words, A and B contain one vertex for each time unit.
- There is an arc from $i \in A$ to $j \in B$ if $|i-j| \leq \eta D$, that is, if w_i^{real} could potentially be assigned to w_j^{online}.
- There is an arc from $j \in B$ to $i \in A$ if part of the workload of w_i^{real} was assigned to w_j^{online} by Noise-Robust($\mathcal{A}$), i.e., if $x_{ij} > 0$.

Now let t be the *last* time step such that the online algorithm had to assign the remaining workload of w_t^{real} uniformly. So, by selection, $t+\eta D$ is the last time step so that $w_{t+\eta D}^{\mathrm{online}} > (1+\zeta) w_{t+\eta D}^{\mathrm{pred}}$. For $k \geq 0$, define the sets

$$\begin{aligned}A_k &= \{i \in A : \text{the shortest path from } t \text{ to } i \text{ has length } 2k \text{ in } G\},\\ B_k &= \{j \in B : \text{the shortest path from } t \text{ to } j \text{ has length } 2k+1 \text{ in } G\}.\end{aligned}$$

Here t stands for the corresponding vertex in A. The set A_k consists of those time steps, for which the corresponding jobs in w^{real} have been moved in w^{online} to the time slots in B_{k-1} but not to any time slot in $B_{k-2}, B_{k-3}, \ldots, B_0$; and B_k are all the time slots where the jobs corresponding to A_k could have been assigned (but no job in $A_{k-1}, A_{k-2}, \ldots, A_0$ could have been assigned). By the selection of t, and the construction of w^{online}, these sets satisfy the following two properties:

Claim A.1. *The sets $(A_k, B_k)_{k\geq 0}$ satisfy*

- *For any time step $j \in \bigcup_k B_k$ we have $w_j^{online} \geq (1+\zeta)w_j^{\text{pred}}$.*
- *For any two time steps $i_k \in A_k$ and $i_\ell \in A_\ell$ with $k > \ell$, we have $i_k - i_\ell \leq 2\eta D(k-\ell+2)$.*

Proof of claim. In the proof of the claim we use the notation $\ell(A_k)$ and $\ell(B_k)$ to denote the left-most (earliest) time step in A_k and B_k, respectively. The proof is by induction on $k \geq 0$ with the following induction hypothesis (IH):

1. For any time step $j \in B_k$ we have $w_j^{\text{online}} \geq (1+\zeta)w_j^{\text{pred}}$.
2. $B_0 = \{t - \eta D, \ldots, t + \eta D\}$ and for any (non-empty) B_k with $k > 1$ we have $B_k = \{\ell(B_k), \ldots, \ell(B_{k-1}) - 1\}$ and $\ell(B_k) - \ell(B_{k-1}) \leq 2\eta D$.

The first part of IH immediately implies the first part of the claim. The second part implies the second part of the claim as follows: Any time step in A_ℓ has a time step in B_ℓ that differs by at most ηD. Similarly, for any time step in A_k there is a time step in B_{k-1} at distance at most ηD. Now by the second part of the induction hypothesis, the distance between these time steps in B_{k-1} and B_ℓ is at most $(k-\ell+1)2\eta D$.

We complete the proof by verifying the inductive hypothesis. For the base case when $k = 0$, we have $B_0 = \{t - \eta D, \ldots, t + \eta D\}$ by definition since $A_0 = \{t\}$. We also have that the first part of IH holds by the definition of Noise-Robust($\mathcal{A}$) and the fact that the overflow of job $w^{\text{real}}(t)$ was uniformly assigned to these time steps.

For the inductive step, consider a time step $i \in A_k$. By definition w_i^{real} was assigned to a time step in B_{k-1} but to no time step in $B_{k-2} \cup \ldots \cup B_0$. Now suppose toward contradiction that there is a time step $j \in A_{k-1}$ such that $j < i$. But then by the greedy strategy of Noise-Robust($\mathcal{A}$) (jobs are assigned left-to-right), we reach the contradiction that w_i^{real} must have been assigned to a time step in $B_{k-2} \cup \ldots \cup B_0$ if $k \geq 2$ since then w_j^{real} is assigned to a time step in B_{k-2}. For $k = 1$, we have $j = t$ and so all time steps in B_0 were full (with respect to capacity $(1+\zeta)w^{\text{pred}}$) after t was processed. Hence, in this case, w_i^{real} could only be assigned to a time step in B_0 if it it had overflow that was uniformly assigned by Noise-Robust($\mathcal{A}$), which contradicts the selection of t.

We thus have that each time step in A_k is smaller than the earliest time step in A_{k-1}. It follows that $B_k = \{\ell(B_k), \ldots, \ell(B_{k-1}) - 1\}$ where $\ell(B_k) = \ell(A_k) - \eta D$. The bound $\ell(B_k) - \ell(B_{k-1}) \leq 2\eta D$ then follows since, by definition, $\{\ell(A_k) - \eta D, \ldots, \ell(A_k) + \eta D\}$ must intersect B_{k-1}. This completes the inductive step for the second part of IH. For the first part, note that the job $w_{\ell(A_k)}^{\text{real}}$ was also assigned to B_{k-1} by Noise-Robust($\mathcal{A}$). By the greedy left-to-right strategy, this only happens if the capacity of all time steps B_k is saturated. □

Now let p be the smallest index such that $w^{\text{real}}(A_{p+1}) + w^{\text{real}}(A_{p+2}) \leq \zeta' \sum_{i=0}^{p} w^{\text{real}}(A_i)$ where we select $\zeta' = \zeta/10$. We have

$$\sum_{i=0}^{p+1} w^{\text{real}}(A_i) \geq \sum_{i=0}^{p} w^{\text{online}}(B_i) \geq (1+\zeta) \sum_{i=0}^{p} w^{\text{pred}}(B_i) \tag{A.5}$$

where the first inequality holds by the definition of the sets and the second is by the first part of the above claim. In addition, by the selection of p,

$$\sum_{i=0}^{p} w^{\text{real}}(A_i) \geq (1-\zeta') \sum_{i=0}^{p+2} w^{\text{real}}(A_i)\,. \tag{A.6}$$

Now let $q = \max\{p - 4/(\zeta')^2, 0\}$. We claim the following inequality

$$\sum_{i=q}^{p} w^{\text{real}}(A_i) \geq (1-\zeta') \sum_{i=0}^{p} w^{\text{real}}(A_i)\,. \tag{A.7}$$

The inequality is trivially true if $q = 0$. Otherwise, we have by the selection of p,

$$\begin{aligned}
\sum_{i=q}^{p} w^{\text{real}}(A_i) &= (1-\zeta') \sum_{i=q}^{p} w^{\text{real}}(A_i) + \zeta' \sum_{i=q}^{p} w^{\text{real}}(A_i) \\
&\geq (1-\zeta') \sum_{i=q}^{p} w^{\text{real}}(A_i) + \frac{(p-q)}{2}(\zeta')^2 \sum_{i=0}^{q-1} w^{\text{real}}(A_i) \\
&\geq (1-\zeta') \sum_{i=q}^{p} w^{\text{real}}(A_i) + 2 \sum_{i=0}^{q-1} w^{\text{real}}(A_i)
\end{aligned}$$

and so (A.7) holds.

We are now ready to complete the proof of the lemma. Let w^* be a minimizer of the right-hand-side, i.e.,

$$w^* = \operatorname*{argmin}_{w \in N_\eta(w^{\text{real}})} \sum_i \left[\left(w_i - w_i^{\text{pred}} \right)^+ \right]^\alpha$$

Divide the time steps of the instance into T_1, B_{p+1}, T_2 and T_3 where T_1 contains all time steps earlier than $\ell(B_{p+1})$, T_2 contains the time steps in $\cup_{i=0}^{p} B_i$, and T_3 contains the remaining time steps, i.e., those after $t + \eta D$. By the selection of t, we have $w_i^{\text{online}} \leq (1+\zeta) w_i^{\text{pred}}$ for all $i \in T_3$. We thus have that $\sum_i \left[\left(w_i^{\text{online}} - (1+\zeta) w_i^{\text{pred}} \right)^+ \right]^\alpha$ equals

$$\sum_{i \in T_1} \left[\left(w_i^{\text{online}} - (1+\zeta) w_i^{\text{pred}} \right)^+ \right]^\alpha + \sum_{i \in B_{p+1} \cup T_2} \left[\left(w_i^{\text{online}} - (1+\zeta) w_i^{\text{pred}} \right)^+ \right]^\alpha .$$

We start by analyzing the second sum. The only jobs in w^{real} that contribute to the workload of w^{online} at the time steps in $B_{p+1} \cup T_2$ are by definition those corresponding to time steps in $A_0 \cup \ldots \cup A_{p+2}$. In the worst case, we have that w^{pred} is 0 during these time steps and that the jobs in w^{real} are uniformly assigned to the same $2\eta D + 1$ time steps. This gives us the upper

bound:

$$\sum_{i\in B_{p+1}\cup T_2}\left[\left(w_i^{\text{online}}-(1+\zeta)w_i^{\text{pred}}\right)^+\right]^\alpha \le \left(\frac{\sum_{i=0}^{p+2} w^{\text{real}}(A_i)}{2\eta D+1}\right)^\alpha \cdot (2\eta D+1)$$
$$\le (1+\zeta')^\alpha \left(\frac{\sum_{i=0}^{p} w^{\text{real}}(A_i)}{2\eta D}\right)^\alpha 2\eta D\,.$$

At the same time, combining (A.5) (A.6), and (A.7) give us

$$\sum_{i=q}^{p} w^{\text{real}}(A_i) \ge (1-\zeta')^2(1+\zeta)\sum_{i=0}^{p} w^{\text{pred}}(B_i) \ge (1+\zeta/2)\sum_{i=0}^{p} w^{\text{pred}}(B_i)\,.$$

By definition, the jobs in w^{real} corresponding to time steps $\cup_{k=q}^{p} A_k$ can only be assigned to w^{online} during time steps $T_2=\cup_{k=0}^{p} B_k$. Therefore, as the difference between the largest time and smallest time in $\cup_{k=q}^{p} A_k$ is at most $2\eta D(p-q+2)$ (second statement of the above claim) and thus the workload of those time steps can be assigned to at most $2\eta D(p-q+4)$ time steps, we have

$$\sum_{i\in T_2}\left[\left(w_i^*-w_i^{\text{pred}}\right)^+\right]^\alpha \ge \left(\frac{\sum_{i=q}^{p} w^{\text{real}}(A_i)-\sum_{i=0}^{p} w^{\text{pred}}(B_i)}{(p-q+4)\cdot 2\eta D}\right)^\alpha \cdot (p-q+4)\cdot 2\eta D$$
$$\ge \left(c\cdot\zeta^3\right)^\alpha \left(\frac{\sum_{i=0}^{p} w^{\text{real}}(A_i)}{2\eta D}\right)^\alpha \cdot 2\eta D$$

for an absolute constant c. It follows that

$$\sum_{i\in B_{p+1}\cup T_2}\left[\left(w_i^{\text{online}}-(1+\zeta)w_i^{\text{pred}}\right)^+\right]^\alpha \le \left(\frac{1+\zeta'}{c\zeta^3}\right)^\alpha \sum_{i\in T_2}\left[\left(w_i^*-w_i^{\text{pred}}\right)^+\right]^\alpha .$$

We have thus upper bounded the sum on the left over time steps in $B_{p+1}\cup T_2$ by the sum on the right over only time steps in T_2. Since NOISE-ROBUST($\mathcal{A}$) does not assign the workload w_i^{real} for $i\in T_1$ to w^{online} on any of the time steps in T_2, we can repeatedly apply the arguments on the time steps in T_1 to show

$$\sum_{i\in T_1}\left[\left(w_i^{\text{online}}-(1+\zeta)w_i^{\text{pred}}\right)^+\right]^\alpha \le \left(\frac{1+\zeta'}{c\zeta^3}\right)^\alpha \sum_{i\in T_1\cup B_{p+1}}\left[\left(w_i^*-w_i^{\text{pred}}\right)^+\right]^\alpha ,$$

yielding the statement of the lemma.

A.5 Robustify for general deadlines

In this Appendix, we discuss generalizations of our techniques to general deadlines. Recall that an instance with general deadlines is defined by a set $\mathcal{J}$ of jobs $J_j=(r_j,d_j,w_j)$, where r_j is the time the job becomes available, d_j is the deadline by which it must be completed, and w_j is the work to be completed. For $\delta>0$, we use the notation $\mathcal{J}^\delta$ to denote the instance obtained from $\mathcal{J}$ by shrinking the duration of each job by a factor $(1-\delta)$. That is, for each job $(r_j,d_j,w_j)\in\mathcal{J}$, $\mathcal{J}^\delta$ contains the job $(r_j, r_j+(1-\delta)(d_j-r_j), w_j)$.

Our main result in this appendix generalizes ROBUSTIFY to general deadlines.

Theorem A.13. *For any $\delta > 0$, given an online algorithm for general deadlines that produces a schedule for $\mathcal{J}^\delta$ of cost C, we can compute online a schedule for $\mathcal{J}$ of cost at most*

$$\min\left\{\left(\frac{1}{1-\delta}\right)^{\alpha-1} C, (2\alpha/\delta^2)^\alpha/2 \cdot \mathrm{OPT}\right\},$$

where OPT *denotes the cost of an optimal schedule of $\mathcal{J}$.*

Since it is easy to design a consistent algorithm by just blindly following the prediction, we have the following corollary.

Corollary A.2. *There exists a learning augmented online algorithm for the General Speed Scaling problem, parameterized by $\varepsilon > 0$, with the following guarantees:*

- Consistency: *If the prediction is accurate, then the cost of the returned schedule is at most* $(1+\varepsilon)\,\mathrm{OPT}$.
- Robustness: *Irrespective of the prediction, the cost of the returned schedule is at most* $O(\alpha^3/\varepsilon^2)^\alpha \cdot \mathrm{OPT}$.

Proof of Corollary. Consider the algorithm that blindly follows the prediction to do an optimal schedule of $\mathcal{J}^\delta$ when in the consistent case. That is, given the prediction of $\mathcal{J}$, it schedules all jobs that agrees with the prediction according to the optimal schedule of the predicted $\mathcal{J}^\delta$; the workload of the remaining jobs j that were wrongly predicted is scheduled uniformly during their duration from release time r_j to deadline d_j. In the consistent case, when the prediction is accurate, the cost of the computed schedule equals thus the cost $\mathrm{OPT}(J^\delta)$ of an optimal schedule of J^δ. Furthermore, we have by Lemma A.8

$$\mathrm{OPT}(\mathcal{J}^\delta) \leq \left(\frac{1}{1-\delta}\right)^{\alpha-1} \mathrm{OPT},$$

where OPT denotes the cost of an optimal schedule to $\mathcal{J}$. Applying Theorem A.13 on this algorithm we thus obtain an algorithm that is also robust. Specifically, we obtain an algorithm with the following guarantees:

- If prediction is accurate, then the computed schedule has cost at most $\left(\frac{1}{1-\delta}\right)^{2(\alpha-1)} \cdot \mathrm{OPT}$.
- The cost of the computed schedule is always at most $(2\alpha/\delta^2)^\alpha/2 \cdot \mathrm{OPT}$.

The corollary thus follows by selecting $\delta = \Theta(\varepsilon/\alpha)$ so that $1/(1-\delta)^{2(\alpha-1)} = 1+\varepsilon$.

□

We proceed by proving the main theorem of this appendix, Theorem A.13.

The procedure General-Robustify. We describe the procedure General-Robustify that generalizes Robustify to general deadlines. Its analysis then implies Theorem A.13. Let $\mathcal{A}$ denote the online algorithm of Theorem A.13 that produces a schedule of $\mathcal{J}^\delta$ of cost C. To simplify the description of General-Robustify, we fix $\Delta > 0$ and assume that the schedule s output by $\mathcal{A}$ only changes at times that are multiples of Δ. This is without loss of generality as we can let Δ tend to 0. To simplify our calculations, we further assume that $\delta(d_j - r_j)/\Delta$ evaluates to an integer for all jobs $(r_j, d_j, w_j) \in \mathcal{J}$.

The time line is thus partitioned into time intervals of length Δ so that in each time interval either no job is processed by s or exactly one job is processed at constant speed by s. We denote by $s(t)$ the speed at which s processes the job $j(t)$ during the t:th time interval, where we let $s(t) = 0$ and $j(t) = \perp$ if no job was processed by s (during this time interval).

To describe the schedule computed by General-Robustify, we further divide each time interval into a *base* part of length $(1-\delta)\Delta$ and an *auxiliary* part of length $\delta\Delta$. In the t:th time interval, General-Robustify schedules job $j(t)$ at a certain speed $s^{\text{base}}(t)$ during the base part, and a subset $\mathcal{J}(t) \subseteq \mathcal{J}$ of the jobs is scheduled during the auxiliary part, each $i \in J(t)$ at a speed $s_i^{\text{aux}}(t)$. These quantities are computed by General-Robustify online at the start of the t:th time interval as follows:

- Let $s^{\text{aux}}(t) = \sum_{i \in \mathcal{J}(t)} s_i^{\text{aux}}(t)$ be the current speed of the auxiliary part and let $D_{j(t)} = d_{j(t)} - r_{j(t)}$ be the duration of job $j(t)$.
- If $s(t)/(1-\delta) \leq s^{\text{aux}}(t)$, then set $s^{\text{base}}(t) = s(t)/(1-\delta)$.
- Otherwise, set $s^{\text{base}}(t)$ so that

$$(1-\delta)\Delta s^{\text{base}}(t) + \left(s^{\text{base}}(t) - s^{\text{aux}}(t)\right)\delta^2 D_{j(t)} = s(t)\Delta \tag{A.8}$$

  and add $j(t)$ to $J(t), J(t+1), \ldots, J(t+\delta D_{j(t)}/\Delta - 1)$ with all auxiliary speeds $s_{j(t)}^{\text{aux}}(t), s_{j(t)}^{\text{aux}}(t+1), \ldots, s_{j(t)}^{\text{aux}}(t+\delta D_{j(t)}/\Delta - 1)$ set to $s^{\text{base}}(t) - s^{\text{aux}}(t)$.

This completes the formal description of General-Robustify. Before proceeding to its analysis, which implies Theorem A.13, we explain the example depicted in Figure A.2. Schedule s, illustrated on the left, schedules a blue, red, and green job during the first, second, and third time interval, respectively. We have that δ/Δ times the duration of the blue job and the red job are 3 and 4, respectively. General-Robustify now produces the schedule on the right where the auxiliary parts are indicated by the horizontal stripes. When the the blue job is scheduled it is partitioned among the base part of the first interval and evenly among the auxiliary parts of the first, second and third intervals so that the speed at the first interval is the same in the base part and auxiliary part. Similarly, when the red job is scheduled, General-Robustify splits it among the base part of the second interval and evenly among the auxiliary part of the second, third, fourth and fifth intervals so that the speed during the base part equals the speed at the auxiliary part during the second interval. Finally, the green job is processed at a small speed and is thus only scheduled in the base part of the third interval (with a speed increased by a factor $1/(1-\delta)$).

Analysis. We show that General-Robustify satisfies the guarantees stipulated by Theorem A.13. We first argue that General-Robustify produces a feasible schedule to $\mathcal{J}$. During

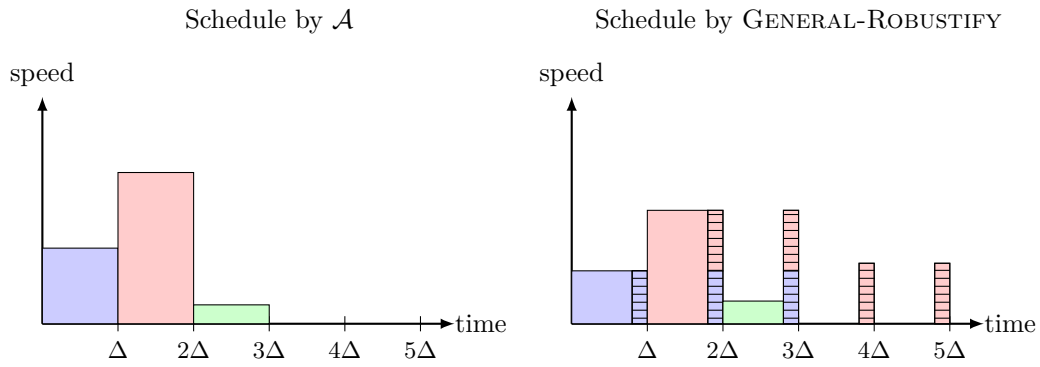


Figure A.2: Given the schedule on the left, General-Robustify produces the schedule on the right.

the t:th interval, the schedule s computed by $\mathcal{A}$ processes $\Delta \cdot s(t)$ work of job $j(t)$. We argue that General-Robustify processes the same amount of work from this time interval. At the time when this interval is considered by General-Robustify, there are two cases:

- If $s(t)/(1-\delta) \leq s^{\text{aux}}(t)$ then $s^{\text{base}}(t) = s(t)/(1-\delta)$ so General-Robustify processes $(1-\delta)\Delta s(t)/(1-\delta) = s(t)\Delta$ work of $j(t)$ during the base part of the t:th time interval.
- Otherwise, we have that General-Robustify processes $(1-\delta)\Delta s^{\text{base}}(t)$ of $j(t)$ during the base part of the t:th time interval and $\delta\Delta\left(s^{\text{base}}(t) - s^{\text{aux}}(t)\right)$ during the auxiliary part of each of the $\delta D_{j(t)}/\Delta$ time intervals $t, t+1, \ldots, t+\delta D_{j(t)}/\Delta - 1$. By the selection (A.8), it thus follows that General-Robustify processes all work $s(t)\Delta$ from this time interval. in this case as well.

The schedule of General-Robustify thus completely processes every job. Furthermore, since each job is delayed at most $\delta D_{j(t)}$ time steps we have that it is a feasible schedule to $\mathcal{J}$ since we started with a schedule for $\mathcal{J}^\delta$, which completes each job j by time $r_j + (1-\delta)D_j$. It remains to prove the robustness and consistency guarantees of Theorem A.13

Lemma A.14 (Robustness)**.** *General-Robustify computes a schedule of cost at most* $(2\alpha/\delta^2)^\alpha/2 \cdot$ OPT.

Proof. By the definition of the algorithm we have, for each time interval, that the speed of the base part is at most the speed of the auxiliary part. Letting $s^{\text{base}}(t)$ and $s^{\text{aux}}(t)$ denote the speed of the base and auxiliary part of the t:th time interval, we thus have

$$\sum_t \left((1-\delta)s^{\text{base}}(t)^\alpha + \delta s^{\text{aux}}(t)^\alpha\right) \leq \sum_t s^{\text{aux}}(t)^\alpha \,.$$

Now we have that the part of a job j that is processed during the auxiliary part of a time interval has been uniformly assigned to at least $\delta^2 D_j$ time steps. It follows that the speed at any auxiliary time interval is at most $1/\delta^2$ times the speed at that time of the Average Rate heuristic (AVR). The lemma now follows since that heuristic is known [88] to have competitive ratio at most $(2\alpha)^\alpha/2$. □

Lemma A.15 (Consistency)**.** *General-Robustify computes a schedule of cost at most* $\left(\frac{1}{1-\delta}\right)^{\alpha-1} \cdot C$ *where* C *denotes the cost of the schedule* s *computed by* $\mathcal{A}$*.*

Proof. For $t \geq 0$, let $h^{(t)}$ be the schedule that processes the workload during the first t time intervals as in the schedule computed by General-Robustify, and the workload of the remaining time intervals is processed during the base part of that time interval by increasing the speed by a factor $1/(1-\delta)$. Hence, $h^{(0)}$ is the schedule that processes the workload of all time intervals during the base part at a speed up of $1/(1-\delta)$, and $h^{(\infty)}$ equals the schedule produced by General-Robustify. By definition, the cost of $h^{(0)}$ equals $\left(\frac{1}{1-\delta}\right)^{\alpha}(1-\delta)\cdot C$ and so the lemma follows by observing that for every $t \geq 1$ the cost of $h^{(t)}$ is at most the cost of $h^{(t-1)}$. To see this consider the two cases of General-Robustify when considering the t:th time interval:

- If $s(t)/(1-\delta) \leq s^{\text{aux}}(t)$ then General-Robustify processes all the workload during the base part at a speed of $s^{\text{base}}(t) = s(t)/(1-\delta)$. Hence, in this case, the schedules $h^{(t)}$ and $h^{(t-1)}$ processes the workload of the t:th time interval identically and so they have equal costs.
- Otherwise, General-Robustify partitions the workload of the t:th time interval among the base part of the t:th interval and $\delta D_{j(t)}/\Delta$ many auxiliary parts so that the speed at each of these parts is strictly less than $s(t)/(1-\delta)$. Hence, since $h^{(t)}$ processes the workload of the t:th time interval at a lower speed than $h^{(t-1)}$ we have that its cost is strictly lower if $\alpha > 1$ (and the cost is equal if $\alpha = 1$).

□

B Deferred proofs of Part III

A few inequalities that are used in Part III.

Lemma B.1. *For any $d > 0$, any $0 < \epsilon \leq 1$, and any $\beta \in [0, 1]$, we have:*

$$\frac{\epsilon}{1 - e^{-\epsilon}} \geq \frac{1}{1 - e^{-1/\epsilon}} \tag{B.1}$$

$$\frac{\epsilon}{1 - (1 + 1/d)^{-\epsilon d}} \geq \frac{1}{1 - (1 + 1/d)^{-d/\epsilon}} \tag{B.2}$$

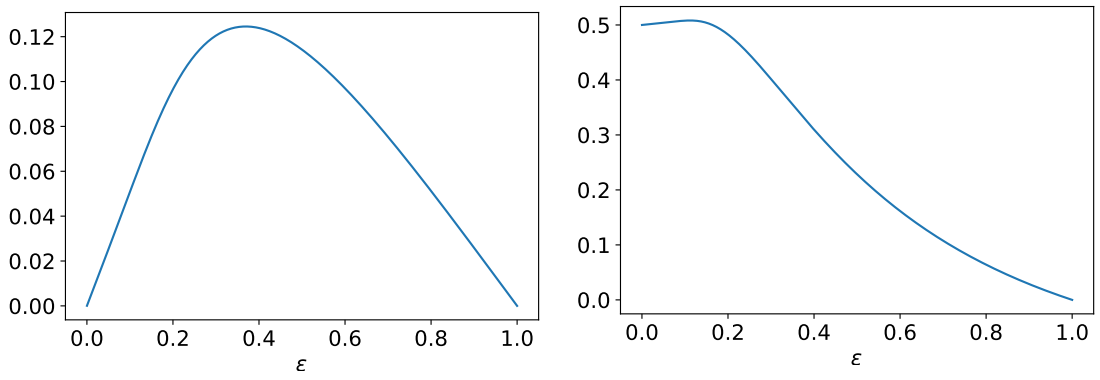
$$\frac{1}{e^{\epsilon} - 1} \geq \frac{\frac{1-\epsilon}{\epsilon} \cdot e^{1/\epsilon} + 1}{e^{1/\epsilon} - 1} \tag{B.3}$$

$$\frac{1}{(1 + 1/d)^{\epsilon d} - 1} \geq \frac{\frac{1-\epsilon}{\epsilon} \cdot (1 + 1/d)^{d/\epsilon} + 1}{(1 + 1/d)^{d/\epsilon} - 1} \tag{B.4}$$

$$\frac{\epsilon}{1 - \beta + \beta\epsilon} \cdot \frac{e^{\epsilon} - \beta}{e^{\epsilon} - 1} \geq \frac{e^{1/\epsilon} - \beta}{e^{1/\epsilon} - 1} \tag{B.5}$$

$$(\epsilon + \beta - \beta\epsilon) \cdot \frac{e^{\epsilon} - \beta}{e^{\epsilon} - 1} \geq \frac{e^{1/\epsilon} - \beta}{e^{1/\epsilon} - 1} \tag{B.6}$$

Proof. Since the formal proof of (B.1) and (B.3) seems to require heavy calculations and they are easy to check on computer we will only give a proof by a plot (see Figures B.1a and B.1b). For B.1b, note that (B.3) $\iff \frac{1}{e^{\epsilon}-1} - \frac{1-\epsilon}{\epsilon} - \frac{1}{\epsilon} \cdot \frac{e^{-1/\epsilon}}{1-e^{-1/\epsilon}} \geq 0$.



(a) Plot of $\frac{\epsilon}{1-e^{-\epsilon}} - \frac{1}{1-e^{-1/\epsilon}}$

(b) Plot of $\frac{1}{e^{\epsilon}-1} - \frac{1-\epsilon}{\epsilon} - \frac{1}{\epsilon} \cdot \frac{e^{-1/\epsilon}}{1-e^{-1/\epsilon}}$

Figure B.1: Plots for (B.1) and (B.3)

We now prove that inequality (B.1) implies inequality (B.2). For this end notice that we can write $(1+1/d)^d = e^x$ for some $x \in (0,1)$ since $(1+1/d)^d \in (1,e)$ for all $d > 0$. We prove that for any $x \in (0,1]$

$$\frac{\epsilon\left(1-e^{-x/\epsilon}\right)}{1-e^{-x\epsilon}} \geq \frac{\epsilon\left(1-e^{-1/\epsilon}\right)}{1-e^{-\epsilon}}$$

which will imply our claim since by inequality (B.1) the right hand side is bigger than 1. First note this is equivalent to prove that

$$g_\epsilon(x) = (1-e^{-\epsilon}) \cdot (1-e^{-x/\epsilon}) - (1-e^{-1/\epsilon}) \cdot (1-e^{-x\epsilon}) \geq 0$$

Taking the derivative of $g_\epsilon(x)$ we obtain

$$g'_\epsilon(x) = \frac{1-e^{-\epsilon}}{\epsilon} \cdot e^{-x/\epsilon} - \epsilon(1-e^{-1/\epsilon}) \cdot e^{-x\epsilon}$$

hence we can write

$$g'_\epsilon(x) \geq 0 \iff e^{x(\epsilon-1/\epsilon)} \geq \epsilon^2 \cdot \frac{1-e^{-1/\epsilon}}{1-e^{-\epsilon}}$$

Notice that the left hand side in this inequality is decreasing because $\epsilon \in (0,1]$. Also notice that $g_\epsilon(0) = g_\epsilon(1) = 0$. These two facts together imply that g_ϵ is first increasing for $x \in (0,c]$ then decreasing for $x \in (c,1]$ for some unknown c. In particular, we indeed have that $g_\epsilon(x) \geq 0$ which ends the proof of inequality (B.2).

Similarly, we prove that inequality (B.3) implies inequality (B.4). Again we write $(1+1/d)^d = e^x$ for some $x \in (0,1)$. We first rewrite inequality (B.4).

$$
\begin{aligned}
\text{(B.4)} &\iff \frac{1}{e^{\epsilon x}-1} \geq \frac{\frac{1-\epsilon}{\epsilon}\cdot e^{x/\epsilon}+1}{e^{x/\epsilon}-1} \\
&\iff \frac{1}{e^{\epsilon x}-1} \geq \frac{\frac{1-\epsilon}{\epsilon}\cdot (e^{x/\epsilon}-1)+\frac{1}{\epsilon}}{e^{x/\epsilon}-1} \\
&\iff \epsilon(e^{x/\epsilon}-1) \geq (1-\epsilon)(e^{x/\epsilon}-1)(e^{\epsilon x}-1)+(e^{\epsilon x}-1) \\
&\iff \epsilon(e^{x/\epsilon}-1)-(1-\epsilon)(e^{x/\epsilon}-1)(e^{\epsilon x}-1)-(e^{\epsilon x}-1) \geq 0
\end{aligned}
$$

Define the following function $h_\epsilon(x) = \epsilon(e^{x/\epsilon}-1)-(1-\epsilon)(e^{x/\epsilon}-1)(e^{\epsilon x}-1)-(e^{\epsilon x}-1)$. One can first compute:

$$
\begin{aligned}
h'_\epsilon(x) &= e^{x/\epsilon}-(1-\epsilon)\cdot\left(\epsilon e^{\epsilon x}(e^{x/\epsilon}-1)+\frac{1}{\epsilon}e^{x/\epsilon}(e^{\epsilon x}-1)\right)-\epsilon e^{\epsilon x} \\
&= e^{x/\epsilon}-\epsilon e^{\epsilon x}-(1-\epsilon)\cdot\left((\epsilon+1/\epsilon)e^{x(\epsilon+1/\epsilon)}-\epsilon e^{\epsilon x}-\frac{1}{\epsilon}e^{x/\epsilon}\right) \\
&= e^{x/\epsilon}\cdot\left(1+\frac{1-\epsilon}{\epsilon}\right)+e^{\epsilon x}\cdot(-\epsilon+\epsilon(1-\epsilon))-e^{x(\epsilon+1/\epsilon)}\cdot(1-\epsilon)\cdot\left(\epsilon+\frac{1}{\epsilon}\right) \\
&= \frac{e^{x/\epsilon}}{\epsilon}-\epsilon^2 e^{\epsilon x}-\frac{e^{x(\epsilon+1/\epsilon)}}{\epsilon}\cdot(1-\epsilon)\cdot(\epsilon^2+1)
\end{aligned}
$$

Hence we can rewrite

$$
\begin{aligned}
h'_\epsilon(x) \geq 0 &\iff \frac{e^{x/\epsilon}}{\epsilon}-\epsilon^2 e^{\epsilon x}-\frac{e^{x(\epsilon+1/\epsilon)}}{\epsilon}\cdot(1-\epsilon)\cdot(\epsilon^2+1) \geq 0 \\
&\iff e^{x/\epsilon}-\epsilon^3 e^{\epsilon x}-e^{x(\epsilon+1/\epsilon)}\cdot(1-\epsilon)\cdot(\epsilon^2+1) \geq 0 \\
&\iff 1-\epsilon^3 e^{x(\epsilon-1/\epsilon)}-e^{x\epsilon}\cdot(1-\epsilon)\cdot(\epsilon^2+1) \geq 0
\end{aligned}
$$

Let us define $i_\epsilon(x) = 1-\epsilon^3 e^{x(\epsilon-1/\epsilon)}-e^{x\epsilon}\cdot(1-\epsilon)\cdot(\epsilon^2+1)$ and we derive

$$
i'_\epsilon(x) = -\epsilon^3\cdot(\epsilon-1/\epsilon)\cdot e^{x(\epsilon-1/\epsilon)}-\epsilon e^{\epsilon x}\cdot(1-\epsilon)\cdot(\epsilon^2+1)
$$

We can now notice that

$$
\begin{aligned}
i'_\epsilon(x) \geq 0 &\iff -\epsilon^3\cdot(\epsilon-1/\epsilon)\cdot e^{x(\epsilon-1/\epsilon)}-\epsilon e^{\epsilon x}\cdot(1-\epsilon)\cdot(\epsilon^2+1) \geq 0 \\
&\iff -\epsilon^3\cdot(\epsilon-1/\epsilon)\cdot e^{-x/\epsilon}-\epsilon(1-\epsilon)\cdot(\epsilon^2+1) \geq 0 \\
&\iff \epsilon^3\cdot(1/\epsilon-\epsilon)\cdot e^{-x/\epsilon}-\epsilon(1-\epsilon)\cdot(\epsilon^2+1) \geq 0
\end{aligned}
$$

Since the left hand side is decreasing as x increases we only need to check one extreme value

which is $i'_\epsilon(0)$. We write

$$\begin{aligned} i'_\epsilon(0) \leq 0 &\iff \epsilon^3 \cdot (1/\epsilon - \epsilon) - \epsilon \cdot (1-\epsilon) \cdot (\epsilon^2+1) \leq 0 \\ &\iff \epsilon^2 - \epsilon^4 - (\epsilon^3 + \epsilon - \epsilon^4 - \epsilon^2) \leq 0 \\ &\iff -\epsilon^3 + 2\epsilon^2 - \epsilon \leq 0 \\ &\iff -\epsilon \cdot (\epsilon - 1)^2 \leq 0 \end{aligned}$$

hence we always have $i'_\epsilon(0) \leq 0$.

Therefore we get that $i'_\epsilon(x) \leq 0$ for all x and ϵ. Note that $i_\epsilon(0) = 1 - \epsilon^3 - (1-\epsilon)(\epsilon^2+1) = 1 - \epsilon^3 - \epsilon^2 - 1 + \epsilon^3 + \epsilon = \epsilon - \epsilon^2 \geq 0$. Therefore we get that h_ϵ is first positive on some interval $[0, c]$ and then negative for $x \in [c, \infty)$. Therefore h_ϵ is first increasing then decreasing. Notice that $h_\epsilon(0) = 0$ and $h_\epsilon(1) \geq 0$ by inequality (B.3). Hence inequality (B.4) is true for all $x \in [0, 1]$ which concludes the proof.

Finally, the proof of (B.5) and (B.6) are quicker and similar. Note that

$$\text{(B.5)} \iff \epsilon \cdot \frac{e^\epsilon - \beta}{e^\epsilon - 1} \geq (1 - \beta + \beta\epsilon) \cdot \frac{e^{1/\epsilon} - \beta}{e^{1/\epsilon} - 1}$$

which is equivalent to a polynomial (in β) of degree 2 being positive. The leading coefficient of this polynomial P is negative and we notice that $P(1) = 0$ and that $P(0) \geq 0$ by (B.1). All these facts together imply that $P(\beta) \geq 0$ for all $\beta \in [0, 1]$. The proof of (B.6) is similar. □

Lemma B.2. *Let $0 < \epsilon \leq 1$, $d > 0$ and define the following functions ($x \in \mathbb{R}$):*

$$f(x) = \left(1 + \frac{1}{d}\right) \cdot x + \frac{1}{d\left((1+1/d)^{\epsilon d} - 1\right)}$$

$$g(x) = \left(1 + \frac{1}{d}\right) \cdot x + \frac{1}{d\left((1+1/d)^{d/\epsilon} - 1\right)}$$

Given $S \geq 0$ and a word $w \in \{a, b\}^*$ we define a sequence S_w recursively as follows:

$$S_{w.y} = \begin{cases} S \text{ if } w.y = \epsilon \\ f(S_w) \text{ if } y = a \\ g(S_w) \text{ if } y = b \end{cases}$$

Then for any $w \in \{a, b\}^*$ such that $|w|_a + \epsilon|w|_b \geq d$ we have that $S_w \geq 1$.

Proof. Let $w' = b \dots ba \dots a = b^{|w|_b} a^{|w|_a}$ be the word made of $|w|_b$ consecutive bs followed by $|w|_a$ consecutive as. Then we claim that $S_{w'} \leq S_w$. This directly follows from the fact that for any real number x, $f(g(x)) \leq g(f(x))$. Noticing this, we can swap positions between an a followed by a b and reducing the final value. We keep doing this until all the bs in w end up in front position.

With standard computations one can check that

$$S_{b^{|w|_b}} = S \cdot (1+1/d)^{|w|_b} + \frac{(1+1/d)^{|w|_b} - 1}{(1+1/d)^{d/\epsilon} - 1}$$

For ease of notation define $S' = S_{b^{|w|_b}}$. Using the assumption that $|w|_a + \epsilon|w|_b \geq d$ and that $S \geq 0$ we get that

$$S' \geq \frac{(1+1/d)^{(d-|w|_a)/\epsilon} - 1}{(1+1/d)^{d/\epsilon} - 1}$$

Again using standard calculations we get that

$$S_{w'} \geq S' \cdot (1+1/d)^{|w|_a} + \frac{(1+1/d)^{|w|_a} - 1}{(1+1/d)^{\epsilon d} - 1}$$

which implies

$$S_{w'} \geq \frac{(1+1/d)^{(d-|w|_a)/\epsilon} - 1}{(1+1/d)^{d/\epsilon} - 1} \cdot (1+1/d)^{|w|_a} + \frac{(1+1/d)^{|w|_a} - 1}{(1+1/d)^{\epsilon d} - 1}$$

Define $h(x) = \frac{(1+1/d)^{(d-x)/\epsilon}-1}{(1+1/d)^{d/\epsilon}-1} \cdot (1+1/d)^x + \frac{(1+1/d)^x-1}{(1+1/d)^{\epsilon d}-1}$. We finish the proof by proving that for any $0 < \epsilon \leq 1$, any $d > 0$ and any $x \geq 0$, we have that $h(x) \geq 1$.

Note that $h(0) = 1$ and that

$$h'(x) = \ln(1+1/d) \cdot \left(\frac{(1+1/d)^x}{(1+1/d)^{\epsilon d} - 1} - \frac{1-\epsilon}{\epsilon} \cdot \frac{(1+1/d)^{(d-(1-\epsilon)x)/\epsilon}}{(1+1/d)^{d/\epsilon} - 1} - \frac{(1+1/d)^x}{(1+1/d)^{d/\epsilon} - 1} \right)$$

To study the sign of $h'(x)$ we can drop the $\ln(1+1/d)$ and write

$$\begin{aligned} h'(x) \geq 0 &\iff \frac{(1+1/d)^x}{(1+1/d)^{\epsilon d} - 1} - \frac{1-\epsilon}{\epsilon} \cdot \frac{(1+1/d)^{(d-(1-\epsilon)x)/\epsilon}}{(1+1/d)^{d/\epsilon} - 1} - \frac{(1+1/d)^x}{(1+1/d)^{d/\epsilon} - 1} \geq 0 \\ &\iff \frac{1}{(1+1/d)^{\epsilon d} - 1} - \frac{1-\epsilon}{\epsilon} \cdot \frac{(1+1/d)^{(d-x)/\epsilon}}{(1+1/d)^{d/\epsilon} - 1} - \frac{1}{(1+1/d)^{d/\epsilon} - 1} \geq 0 \end{aligned}$$

Clearly the last term is increasing as x increases hence we can limit ourselves to prove that $h'(0) \geq 0$ which we can rewrite

$$h'(0) \geq 0 \iff \frac{1}{(1+1/d)^{\epsilon d}-1} - \frac{1-\epsilon}{\epsilon} \cdot \frac{(1+1/d)^{d/\epsilon}}{(1+1/d)^{d/\epsilon}-1} - \frac{1}{(1+1/d)^{d/\epsilon}-1} \geq 0$$
$$\iff \frac{1}{(1+1/d)^{\epsilon d}-1} \geq \frac{\frac{1-\epsilon}{\epsilon} \cdot (1+1/d)^{d/\epsilon}+1}{(1+1/d)^{d/\epsilon}-1}$$

Which holds by equation (5) of Lemma B.1. □

Bibliography

[1] Rakesh Agrawal, Alan Halverson, Krishnaram Kenthapadi, Nina Mishra, and Panayiotis Tsaparas. Generating labels from clicks. In *Proceedings of the 2nd ACM International Conference on Web Search and Data Mining*, pages 172–181, 2009.

[2] Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: Ranking and clustering. *Journal of the ACM (JACM)*, 55(5), 2008. URL https://doi.org/10.1145/1411509.1411513.

[3] Noga Alon, Baruch Awerbuch, and Yossi Azar. The online set cover problem. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC)*, page 100–105, 2003. URL https://doi.org/10.1145/780542.780558.

[4] Lachlan LH Andrew, Minghong Lin, and Adam Wierman. Optimality, fairness, and robustness in speed scaling designs. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 37–48, 2010.

[5] Antonios Antoniadis, Christian Coester, Marek Elias, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, volume 119, pages 345–355, 2020. URL https://proceedings.mlr.press/v119/antoniadis20a.html.

[6] Arvind Arasu, Christopher Ré, and Dan Suciu. Large-scale deduplication with constraints using dedupalog. In *Proceedings of the 25th International Conference on Data Engineering*, pages 952–963, 2009.

[7] Yossi Azar, Ilan Reuven Cohen, and Alan Roytman. Online lower bounds via duality. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1038–1050, 2017.

[8] Etienne Bamas, Andreas Maggiori, Lars Rohwedder, and Ola Svensson. Learning augmented energy minimization via speed scaling. In *Proceedings of the 34th Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 15350–15359, 2020. URL https://proceedings.neurips.cc/paper/2020/file/af94ed0d6f5acc95f97170e3685f16c0-Paper.pdf.

[9] Etienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms. In *Proceedings of the 34th Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 20083–20094, 2020. URL https://proceedings.neurips.cc/paper/2020/file/e834cb114d33f729dbc9c7fb0c6bb607-Paper.pdf.

[10] N. Bansal, N. Buchbinder, and J. Naor. A primal-dual randomized algorithm for weighted paging. In *Proceedings of the 48th Symposium on Foundations of Computer Science (FOCS)*, pages 507–517, 2007.

[11] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine learning*, 56(1):89–113, 2004.

[12] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *Journal of the ACM (JACM)*, 54(1):3:1–3:39, 2007. URL https://doi.org/10.1145/1206035.1206038.

[13] Nikhil Bansal, David P. Bunde, Ho-Leung Chan, and Kirk Pruhs. Average rate speed scaling. In *Proceedings of the 8th Latin American Theoretical Informatics Symposium (LATIN)*, pages 240–251, 2008. URL https://doi.org/10.1007/978-3-540-78773-0_21.

[14] Jeff Barr. New–predictive scaling for ec2, powered by machine learning. *AWS News Blog*, 2018. URL https://aws.amazon.com/blogs/aws/new-predictive-scaling-for-ec2-powered-by-machine-learning/.

[15] Benjamin Birnbaum and Claire Mathieu. On-line bipartite matching made simple. *ACM SIGACT News*, 39(1):80–87, 2008.

[16] Francesco Bonchi, Aristides Gionis, and Antti Ukkonen. Overlapping correlation clustering. *Knowledge and information systems*, 35(1):1–32, 2013.

[17] Niv Buchbinder and Joseph (Seffi) Naor. The design of competitive online algorithms via a primal: Dual approach. *Foundations and Trends® in Theoretical Computer Science*, 3(2–3): 93–263, 2009. URL https://doi.org/10.1561/0400000024.

[18] Niv Buchbinder, Kamal Jain, and Joseph (Seffi) Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *Proceedings of the 15th Annual European Symposium on Algorithms (ESA)*, pages 253–264, 2007.

[19] Niv Buchbinder, Danny Segev, and Yevgeny Tkach. Online algorithms for maximum cardinality matching with edge arrivals. *Algorithmica*, pages 1–19, 2018.

[20] Deepayan Chakrabarti, Ravi Kumar, and Kunal Punera. A graph-theoretic approach to webpage segmentation. In *Proceedings of the 17th International World Wide Web Conference (WWW)*, pages 377–386, 2008.

[21] Shuchi Chawla, Konstantin Makarychev, Tselil Schramm, and Grigory Yaroslavtsev. Near optimal lp rounding algorithm for correlation clustering on complete and complete k-partite graphs. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC)*, pages 219–228, 2015.

[22] Yudong Chen, Sujay Sanghavi, and Huan Xu. Clustering sparse graphs. In *Proceedings of the 25th Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 2204–2212, 2012.

[23] Eunjoon Cho, Seth A. Myers, and Jure Leskovec. Friendship and mobility: User movement in location-based social networks. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 1082–1090, 2011. URL https://doi.org/10.1145/2020408.2020579.

[24] Ilan Reuven Cohen and David Wajc. Randomized online matching in regular graphs. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 960–979, 2018.

[25] Ilan Reuven Cohen, Binghui Peng, and David Wajc. Tight bounds for online edge coloring. In *Proceedings of the 60th Symposium on Foundations of Computer Science (FOCS)*, 2019.

[26] V. Cohen-Addad, E. Lee, and A. Newman. Correlation clustering with sherali-adams. In *Proceedings of the 363rd Symposium on Foundations of Computer Science (FOCS)*, pages 651–661, 2022. URL https://doi.ieeecomputersociety.org/10.1109/FOCS54457.2022.00068.

[27] Vincent Cohen-Addad, Niklas Hjuler, Nikos Parotsidis, David Saulpic, and Chris Schwiegelshohn. Fully dynamic consistent facility location. In *Proceedings of the 33rd Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[28] Vincent Cohen-Addad, Silvio Lattanzi, Slobodan Mitrovic, Ashkan Norouzi-Fard, Nikos Parotsidis, and Jakub Tarnawski. Correlation clustering in constant many parallel rounds. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, volume 139, pages 2069–2078, 2021.

[29] Vincent Cohen-Addad, Silvio Lattanzi, Andreas Maggiori, and Nikos Parotsidis. Online and consistent correlation clustering. In *Proceedings of the 39th International Conference on Machine Learning (ICML)*, pages 4157–4179, 2022. URL https://proceedings.mlr.press/v162/cohen-addad22a.html.

[30] Wikipedia contributors. Lomax distribution, 2004. URL https://en.wikipedia.org/wiki/Lomax_distribution.

[31] Erik D Demaine, Dotan Emanuel, Amos Fiat, and Nicole Immorlica. Correlation clustering in general weighted graphs. *Theoretical Computer Science*, 361(2-3):172–187, 2006.

[32] Nikhil R Devanur, Kamal Jain, and Robert D Kleinberg. Randomized primal-dual analysis of ranking for online bipartite matching. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 101–107, 2013.

[33] Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Faster matchings via learned duals. In *Proceedings of the 35th Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2021. URL https://openreview.net/forum?id=kB8eks2Edt8.

[34] Yihe Dong, Piotr Indyk, Ilya Razenshteyn, and Tal Wagner. Learning space partitions for nearest neighbor search. In *Proceedings of the 8th International Conference on Machine Learning (ICML)*, 2020. URL https://www.microsoft.com/en-us/research/publication/learning-space-partitions-for-nearest-neighbor-search/.

[35] Daniel R Dooly, Sally A Goldman, and Stephen D Scott. Tcp dynamic acknowledgment delay (extended abstract) theory and practice. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 389–398, 1998.

[36] Alon Eden, Michal Feldman, Amos Fiat, and Kineret Segal. An economic-based analysis of ranking for online bipartite matching. In *Proceedings of 2021 Symposium on Simplicity in Algorithms (SOSA)*, pages 107–110, 2021.

[37] Jack Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of research of the National Bureau of Standards B*, 69(125-130):55–56, 1965.

[38] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17(3):449–467, 1965.

[39] Leah Epstein, Asaf Levin, Danny Segev, and Oren Weimann. Improved bounds for online preemptive matching. In *Proceedings of the 30th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 389–399, 2013.

[40] Uriel Feige. Tighter bounds for online bipartite matching. *arXiv preprint arXiv:1812.11774*, 2018.

[41] Björn Feldkord, Matthias Feldotto, Anupam Gupta, Guru Guruganesh, Amit Kumar, Sören Riechers, and David Wajc. Fully-dynamic bin packing with little repacking. In *Proceedings of the 45th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 51:1–51:24, 2018.

[42] Hendrik Fichtenberger, Silvio Lattanzi, Ashkan Norouzi-Fard, and Ola Svensson. Consistent k-clustering for general metrics. In *Proceedings of the 32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2660–2678. SIAM, 2021.

[43] Rudolf Fleischer. On the bahncard problem. *Theoretical Computer Science*, 268(1):161 – 174, 2001. URL http://www.sciencedirect.com/science/article/pii/S0304397500002668.

[44] Buddhima Gamlath, Michael Kapralov, Andreas Maggiori, Ola Svensson, and David Wajc. Online matching with general arrivals. In *Proceedings of the 60th Symposium on Foundations of Computer Science (FOCS)*, pages 26–37, 2019. URL https://conferences.computer.org/focs/2019/pdfs/FOCS2019-7pBwCpNH4Mz2L4MJWVl6Xp/1pIEyZQDr2HASYIBeX3I8P/5NPI01irVYpRREDDAIQhYj.pdf.

[45] Marco E. T. Gerards, Johann L. Hurink, and Philip K. F. Hölzenspies. A survey of offline algorithms for energy minimization under deadline constraints. *Journal of Scheduling*, 19(1): 3–19, 2016. URL https://doi.org/10.1007/s10951-015-0463-8.

[46] Gagan Goel and Aranyak Mehta. Online budgeted matching in random input models with applications to adwords. In *Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 982–991, 2008.

[47] Sreenivas Gollapudi and Debmalya Panigrahi. Online algorithms for rent-or-buy with expert advice. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 2319–2327, 2019. URL http://proceedings.mlr.press/v97/gollapudi19a.html.

[48] Weibo Gong, Yong Liu, Vishal Misra, and Don Towsley. On the tails of web file size distributions. In *Proceedings of the 39th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2001.

[49] Xiangyu Guo, Janardhan Kulkarni, Shi Li, and Jiayi Xian. Consistent k-median: Simpler, better and robust. In *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics*, pages 1135–1143, 2021.

[50] Anupam Gupta, Ravishankar Krishnaswamy, and Kirk Pruhs. Online primal-dual for non-linear optimization with applications to speed scaling. In *Proceedings of the 10th Workshop on Approximation and Online Algorithms (WAOA)*, pages 173–186, 2012.

[51] Guru Prashanth Guruganesh and Sahil Singla. Online matroid intersection: Beating half for random arrival. In *Proceedings of the 19th Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 241–253, 2017.

[52] Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. Learning-based frequency estimation algorithms. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019. URL https://openreview.net/forum?id=r1lohoCqY7.

[53] Zhiyi Huang, Ning Kang, Zhihao Gavin Tang, Xiaowei Wu, Yuhao Zhang, and Xue Zhu. How to match when all vertices arrive online. In *Proceedings of the 50th Annual ACM Symposium on Theory of Computing (STOC)*, pages 17–29, 2018.

[54] Zhiyi Huang, Zhihao Gavin Tang, Xiaowei Wu, and Yuhao Zhang. Online vertex-weighted bipartite matching: Beating 1-1/e with random arrivals. In *Proceedings of the 45th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 1070–1081, 2018.

[55] Zhiyi Huang, Binghui Peng, Zhihao Gavin Tang, Runzhou Tao, Xiaowei Wu, and Yuhao Zhang. Tight competitive ratios of classic matching algorithms in the fully online model. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2875–2886, 2019.

[56] Mohammad Reza Karimi Jaghargh, Andreas Krause, Silvio Lattanzi, and Sergei Vassilvtiskii. Consistent online optimization: Convex and submodular. In *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*, pages 2241–2250, 2019.

[57] Dmitri V Kalashnikov, Zhaoqi Chen, Sharad Mehrotra, and Rabia Nuray-Turan. Web people search via connection analysis. *IEEE Transactions on Knowledge and Data Engineering*, 20 (11):1550–1565, 2008.

[58] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator. Competitive snoopy caching. In *Proceedings of the 27th Symposium on Foundations of Computer Science (FOCS)*, pages 244–254, 1986.

[59] Anna R. Karlin, Mark S. Manasse, Lyle A. McGeoch, and Susan Owicki. Competitive randomized algorithms for non-uniform problems. In *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, page 301–309, 1990.

[60] Anna R. Karlin, Claire Kenyon, and Dana Randall. Dynamic tcp acknowledgement and other stories about e/(e-1). In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC)*, page 502–509, 2001. URL https://doi.org/10.1145/380752.380845.

[61] Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 352–358, 1990.

[62] Craig Kitterman. Autoscaling windows azure applications. *Microsoft Azure Blog*, 2013. URL https://azure.microsoft.com/de-de/blog/autoscaling-windows-azure-applications/.

[63] Rohan Kodialam. Optimal algorithms for ski rental with soft machine-learned predictions. *CoRR*, 2019. URL http://arxiv.org/abs/1903.00092.

[64] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[65] Silvio Lattanzi and Sergei Vassilvitskii. Consistent k-clustering. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 1975–1984, 2017.

[66] Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1859–1877, 2020. URL https://doi.org/10.1137/1.9781611975994.114.

[67] Silvio Lattanzi, Benjamin Moseley, Sergei Vassilvitskii, Yuyan Wang, and Rudy Zhou. Robust online correlation clustering. In *Proceedings of the 34th Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[68] Euiwoong Lee and Sahil Singla. Maximum matching in the online batch-arrival model. In *Proceedings of the 19th Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 355–367, 2017.

[69] Russell Lee, Mohammad H. Hajiesmaili, and Jian Li. Learning-assisted competitive algorithms for peak-aware energy scheduling. *CoRR*, 2019. URL http://arxiv.org/abs/1911.07972.

[70] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, 2014.

[71] László Lovász and Michael D Plummer. *Matching theory*, volume 367. American Mathematical Society, 2009.

[72] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 3302–3311, 2018. URL http://proceedings.mlr.press/v80/lykouris18a.html.

[73] Mohammad Mahdian, Hamid Nazerzadeh, and Amin Saberi. Online optimization with uncertain information. *ACM Transactions on Algorithms*, 8(1), 2012. URL https://doi.org/10.1145/2071379.2071381.

[74] M. Marathe and W. Hawe. Predicted capacity of ethernet in a university environment. In *Proceedings of Southcon*, pages 1–10, 1982.

[75] Claire Mathieu, Ocan Sankur, and Warren Schudy. Online correlation clustering. In *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 573–584, 2010.

[76] Andres Muñoz Medina and Sergei Vassilvitskii. Revenue optimization with approximate bid predictions. In *Proceedings of the 30th Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 1858–1866, 2017. URL http://papers.nips.cc/paper/6782-revenue-optimization-with-approximate-bid-predictions.

[77] Aranyak Mehta. Online matching and ad allocation. *Foundations and Trends® in Theoretical Computer Science*, 8(4):265–368, 2013.

[78] Michael Mitzenmacher. Dynamic models for file sizes and double pareto distributions. *Internet Mathematics*, 1(3):305–333, 2003. URL https://projecteuclid.org:443/euclid.im/1109190964.

[79] Michael Mitzenmacher. A model for learned bloom filters and optimizing by sandwiching. In *Proceedings of the 31st Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 464–473, 2018. URL http://papers.nips.cc/paper/7328-a-model-for-learned-bloom-filters-and-optimizing-by-sandwiching.pdf.

[80] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In *Proceedings of the 31st Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 9684–9693, 2018. URL http://papers.nips.cc/paper/8174-improving-online-algorithms-via-ml-predictions.

[81] Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, page 1834–1845, 2020.

[82] Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

[83] Steven S. Seiden. A guessing game and randomized online algorithms. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC)*, page 592–601, 2000. URL https://doi.org/10.1145/335305.335385.

[84] Sumedh Tirodkar and Sundar Vishwanathan. Maximum matching on trees in the online preemptive and the incremental dynamic graph models. In *Proceedings of the 23rd International Computing and Combinatorics Conference (COCOON)*, pages 504–515, 2017.

[85] Shufan Wang, Jian Li, and Shiqiang Wang. Online Algorithms for Multi-shop Ski Rental with Machine Learned Advice. In *Proceedings of the 34th Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[86] Michael Wilson. A historical view of network traffic models. http://www.cse.wustl.edu/~jain/cse567-06/traffic_models2.htm, 2006.

[87] Yinfeng Xu and Weijun Xu. Competitive algorithms for online leasing problem in probabilistic environments. In *Proceedings of the International Symposium on Neural Networks, Part II*, pages 725–730, 2004. URL https://doi.org/10.1007/978-3-540-28648-6_116.

[88] F. Frances Yao, Alan J. Demers, and Scott Shenker. A scheduling model for reduced CPU energy. In *Proceedings of the 36th Symposium on Foundations of Computer Science (FOCS)*, pages 374–382, 1995. URL https://doi.org/10.1109/SFCS.1995.492493.

Andreas Maggiori

Contact Information

Address: Chemin de la Raye 7, Ecublens 1024, Switzerland
email: andreas.maggiori@gmail.com

Education

09/2018-present | **École Polytechnique Fédérale de Lausanne (EPFL), Switzerland**
PhD in Computer Science
Advisors: Rüdiger Urbanke and Ola Svensson

09/2011-10/2017 | **National Technical University of Athens, Greece**
Diploma (5-year joint degree; 300 ECTS),
Electrical and Computer Engineering (ECE)
Grade: 9.12 / 10 (approx. best 3%)

Thesis: Using Machine Learning Techniques to Infer Players' Valuations in Online Ad Auctions
Advisor: Dimitris Fotakis

01/2016-06/2016 | **Universidad Carlos III Madrid, Spain**
Erasmus Exchange Student Program

09/2005-06/2011 | **Lycée Léonin Nea Smirni, Greece**
High School
Grade: 19.5 / 20 - Excellent

Professional Experience

05/2022-08/2022 | **Research Intern, Google Zurich**
07/2021-10/2021 | **Research Intern, Google Zurich**

Long term Research Visits

09/2022-12/2022 | **Simons Institute for the Theory of Computing , UC Berkeley**
Visiting graduate student for the program Data-Driven Decision Processes

Research Interests

I am broadly interested in combinatorial optimization, online algorithms, machine learning and their intersection.

Currently, I am focusing on *Learning Augmented (Online) Algorithms*, where (informally) the goal is to design algorithms which provably outperform classical online algorithms when an accurate prediction about the future is available, while maintaining robustness against adversarial predictions.

Publications

Authors (as customary in theory) are in alphabetical order.

1. Online and Consistent Correlation Clustering
ICML 2022
V. Cohen-Addad, S. Lattanzi, A. Maggiori, N. Parotsidis
2. An Improved Analysis of Greedy for Online Steiner Forest
SODA 2022
É. Bamas, M. Drygala, A. Maggiori

3. The Primal-Dual method for Learning Augmented Algorithms
   NeurIPS 2020 (oral talk)
   É. Bamas, A. Maggiori, O. Svensson
4. Learning Augmented Energy Minimization via Speed Scaling
   NeurIPS 2020 (spotlight presentation)
   É. Bamas, A. Maggiori, L. Rohwedder, O. Svensson
5. Online Matching with General Arrivals **FOCS 2019**
   B. Gamlath, M. Kapralov, A. Maggiori, O. Svensson, D. Wajc

Computer Skills

Programming Languages (Excellent):	Python, C++, SQL
Programming Languages (Familiar with):	C, SML/NJ, Prolog, MATLAB, Bash
ML Frameworks (Familiar with):	PyTorch

Teaching Experience

I organized a study-group on how continuous optimization methods can be used to tackle combinatorial problems. The website of the study-group with notes and recorded lectures can be found here: https://www.epfl.ch/schools/ic/ipg/convexity-and-optimization-2020/.

I co-organized the ALPS (ALgorithms with PredictionS) workshop at EPFL in May 2022, along with Etienne Bamas and Adam Polak.

I am/was teaching assistant for the following courses:

- NTUA: Algorithms and Complexity, Discrete Mathematics
- EPFL: Theory of Computation, Machine Learning, Learning Theory, Algorithms, Advanced Probability and Applications, Foundations of Data Science

Awards

2017:	1st in the NTUA hub at `Google Hashcode` programming competition (170 in the world) with the team *Veni Vidi Vsync*
2013:	Bronze medal at SEEMOUS (South Eastern European Mathematical Olympiad for University Students) competition
2010:	Bronze medal on `Euclid` phase of high school mathematics competition organized by the Hellenic Mathematical Society
2010	Finalist in the `Physics` high school competition organized by the Union of Greek Physicists
2008, 2010:	Twice finalist in the `Archimedes` high school mathematics competition organized by the Hellenic Mathematical Society

Languages

Greek:	Mothertongue
Italian:	Mothertongue
English:	Professional working proficiency (C2, Certificate of Proficiency, Michigan)
French:	Professional working proficiency (C2, Certificate of Sorbonne)
Spanish:	Elementary proficiency (B2, Diploma Instituto Cervantes)

References

Ola Svensson:	ola.svensson@epfl.ch
Rüdiger Urbanke:	rudiger.urbanke@epfl.ch
Silvio Lattanzi:	silviol@google.com
Vincent Cohen-Addad:	cohenaddad@google.com
Nikos Parotsidis:	nikosp@google.com