Fakultät für Informatik

Online Algorithms for Scheduling and Data Management

Maximilian M. Janke

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzende/-r: Prof. Dr.-Ing. Pramod Bhatotia

Prüfende der Dissertation:

1. Prof. Dr. Susanne Albers
2. Assistant Prof. Dr. Antonios Antoniadis

Die Dissertation wurde am 26.10.2021 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 06.04.2022 angenommen.

To my parents.

Abstract

Online Makespan Minimization and List Update are two of the most basic online problems. Both problems have been studied since the sixties and are employed in scheduling and data management.

Online problems conceptualize interactive settings where *online algorithms* make decisions without knowing the whole input in advance. This prevents optimal performance on general instances. Instead, online algorithms are judged by their *competitive ratio*, the maximum ratio by which their performance strays from being optimal due to not knowing the future.

The first online problem, *List Update*, studies the linear linked list. Lists are often-used data structures for managing requests to a small number of items. Frequently requested items are served faster at the front of the list. Moving them there requires *paid exchanges*. The costs of paid exchanges, when compared to practical list implementations, are underestimated in the *standard model*. This led to the P^d *model*, which considers a general exchange cost of d.

In the *Makespan Minimization* problem, one is tasked to assign jobs to identical and parallel machines. Preemption is not allowed. The objective is to minimize the time it takes to process all jobs, called the *makespan*. The dual problem, *Machine Covering*, asks to maximize the time all machines are busy.

Classical analyses, on the one hand, optimistically omit and neglect errors in the input. They are, on the other hand, too pessimistic in that they present online algorithms with worst-case inputs arranged in worst-case orders. Considering uniformly random orders instead, gives the recently popular *random-order model*. A stronger model of ours considers nearly worst-case orders. To model errors in the input, *budgeted uncertainty* assumptions are used.

This thesis presents the currently best upper and lower bounds for five online problems. For the List Update problem in the P^d model, the main result is a 2.2442-competitive online algorithm and a lower bound of 2. For Makespan Minimization in the random-order model, a 1.8478-competitive algorithm is provided, which beats prior adversarial lower bounds already on 24 machines [145]. In another random-order model, the input length n is known in advance. Here, a 1.535-competitive online algorithm beats even the pessimistic adversarial lower bounds for randomized algorithms. For Makespan Minimization under budgeted uncertainty, an upper bound of 2.9052 is complemented by a lower bound of 2. For Machine Covering in the random-order model, an improved algorithm is $\tilde{O}(\sqrt[4]{m})$-competitive while a lower bound of $\tilde{\Omega}(\log(m))$ is provided.

Zusammenfassung

List-Update und Online-Makespan-Minimierung sind zwei grundlegende Online-Probleme mit wichtigem Anwendungsbezug zu Scheduling und Datenmanagement.

In Anwendungen mit interaktiven Komponenten treffen Algorithmen Entscheidungen, ohne bereits die gesamte Eingabe zu kennen. Solche Algorithmen, *Online-Algorithmen* genannt, können im Allgemeinen keine optimalen Resultate für sämtliche Eingaben garantieren. Stattdessen bewertet man sie nach dem maximalen Faktor, um den sie schlechter als das Optimum abschneiden, dem sogenannten *kompetitiven Faktor*.

Im *List-Update-Problem* werden linear verkettete Listen betrachtet. Listen werden oft verwendet um Zugriffe auf eine kleine Anzahl von Einträgen zu verwalten. Um Kosten zu sparen, lohnt es sich Einträge, auf die oft zugegriffen wird, vorne in der Liste anzuordnen. Hierfür verwendet man *bezahlte Täusche*. Die Kosten eines solchen Tausches werden im *Standardmodell* unterschätzt. Deswegen betrachtet das P^d*-Modell* allgemeine Tauschkosten d.

Beim *Makespan-Minimierungs-Problem* sind Aufgaben identischen und parallelen Maschinen zuzuweisen. Präemption ist nicht erlaubt. Ziel ist es, die Zeit bis alle Aufgaben fertiggestellt sind, auch *Makespan* genannt, zu minimieren. Im dualen *Machine-Covering-Problem* ist stattdessen die Zeit, die alle Maschinen gleichzeitig beschäftigt sind, zu maximieren.

Traditionell werden Online-Algorithmen auf Worst-Case-Eingaben betrachtet. Diese Eingaben bearbeiten sie in einer Worst-Case-Reihenfolge. Ist die Reihenfolge stattdessen uniform zufällig gewählt, erhält man das optimistischere *Random-Order-Modell*. Für ein pessimistischeres Modell, auf der anderen Seite, betrachten wir *Budgeted-Uncertainty-Annahmen*, die Fehler und Fehlfunktionen in der Eingabe berücksichtigen. Solche Fehler können die Ausführungszeiten einzelner Aufträge verlängern.

Diese Arbeit enthält die derzeit besten oberen und unteren Schranken für fünf Probleme. Für das List-Update-Problem im P^d-Modell trifft ein 2,2442-kompetitiver Online-Algorithmus auf eine untere Schranke von 2. Für Makespan-Minimierung schlägt ein 1,8478-kompetitiver Algorithmus bereits die untere Worst-Case-Schranke für 24 Maschinen [145]. Im dazu verwandten Random-Order-Modell, in dem die Auftragszahl n von Anfang an bekannt ist, unterbietet ein 1,535-kompetitiver Algorithmus die als pessimistisch geltende allgemeine untere Schranke für randomisierte Algorithmen. Für Makespan-Minimierung mit Budgeted-Uncertainty-Annahmen findet sich eine obere Schranke von 2,9052 und eine untere Schranke von 2. Für Machine-Covering im Random-Order-Modell präsentieren wir einen $\tilde{O}(\sqrt[4]{m})$-kompetitiven Algorithmus und eine untere Schranke von $\tilde{\Omega}(\log(m))$.

Acknowledgments

This thesis would not have existed without the generous support, help and assistance of many people.

I foremost thank my supervisor Prof. Dr. Susanne Albers for her advice during my time as a PhD student at the Technical University of Munich. Without her, I would not be the independent researcher I am today.

Special thanks go to my colleagues, Prof. Dr. Harald Räcke, the post-docs, Dr. Arindam Khan, Dr. Kevin Schewior, Dr. Waldo Gàlvez and Dr. Kelin Luo, fellow and former doctoral students Dr. Richard Stotz, Alexander Eckl, Jens Quedenfeld, Luisa Peter, Leon Ladewig, Gunther Bidlingmaier, Sebastian Schraink and all others. The time spent and discussions had, during and after work, contributed to this thesis as well as my general well-being. My only regret is that none of them picked up the game of Go.

I thank 吕扬帆 and my friends. They keep me sane, help me in trying times and fill my life with cheer.

Deep thanks go to my family, my three brothers, Jonathan, Philipp and Florian and, above all, my parents Stephanie and Martin on whose trust, encouragement and patience I could always count. Year by year, I grow more grateful of their support and the family environment they create.

Contents

Chapter 1

Introduction

Makespan Minimization and List Update are long-standing problems studied since the sixties [94, 133]. Both have applications in scheduling and data management. In the *List Update* problem, the task is to dynamically maintain an unordered list of items at minimum cost. Unordered item lists are sensible dictionary structures for managing up to a few dozens of entries. The *Makespan Minimization* problem, on the other hand, studies one of the most basic scheduling problems where jobs have to be assigned to many parallel and identical machines. The goal is to minimize the time it takes to process all jobs, called the *makespan*. Preemption is not allowed. Makespan Minimization is used in Manufacturing Planning, Multiprocessor Scheduling and Load Balancing. The latter is a key process in big data centers distributing requests to numerous servers. The dual problem of Makespan Minimization is *Machine Covering*. For Machine Covering, jobs represent resources that machines consume in order to run. The goal is to keep the whole system running as long as possible. Machine Covering has applications in the sequencing of maintenance actions for aircraft engines and in storage area networks.

In theoretical research, problems that stem from such applications are modeled using precise mathematical language and definitions. This, while required for the thorough and comprehensive treatment expected from theoretical results, by necessity diminishes applicability to real-world settings. In the real world, we encounter open problems—rarely completely clear-cut ones as studied in theory. Theoretical research adapts to the ever rising needs in practical settings by refining and developing ever more realistic models that improve applicability.

An example central to this thesis are *online algorithms*. Traditionally, algorithms are analyzed in the *offline paradigm* where the whole problem data arrives and is processed at once. Contrary to that, real-world programs often contain interactive components and serve requests one by one. Neither does a website know who its next visitor will be, nor does a search engine receive all its questions at once in one gigantic chunk. To study requests that arrive one by one the *online paradigm* was introduced. An online algorithm needs to treat each request before the next one is revealed. Such treatment involves permanent and irreversible decisions. It makes mistakes in the beginning unavoidable and irrevocable and it rules out optimal results on general inputs. Thus, online algorithms are measured in terms of *competitive analysis*. They are judged by how far their solutions deviate from the optimum.

Nowadays, the area of online algorithms is well established. Online Makespan Minimization, Online Machine Covering and (Online) List Update have been studied in depth. Many relevant and exciting variants were discovered. Despite this, some assumptions and simplifications common in theory still do not completely translate to practical situations. This diminishes their transferability to real-world applications. Such diminishing transferability may result from too simple a cost-charging scheme; too pessimistic worst-case analyses; or too much optimism signified by the fact that practical input data is rarely free of errors and uncertainty. For each of these issues, we explore new models that mitigate them.

A more realistic cost-charging scheme for *List Update* leads to the P^d *model*. In any realistic list implementation, swapping items, an action which reassigns several memory pointers, takes vastly more time than simply following a single pointer. This difference in cost has been neglected in the *standard model*. The standard model indifferently charges a cost of 1 for each of both operations. Considering a more general exchange cost of $d \geq 1$ gave rise to the P^d *model*, or *paid exchange model*.

To go beyond worst-case analyses, we consider the recently popular *random-order model*. Inputs are randomly permuted before being presented to the online algorithm. In this setting, we study Machine Covering and Online Makespan Minimization. For Online Makespan Minimization, we consider a stricter version of the random-order model. This stricter model considers "nearly worst-case" sequences. Only a tiny fraction of extremely badly ordered sequences is excluded. This caters to typical real-world applications that need to be robust against bad scenarios but do not confront a single malicious actor that conjures up absolute worst-case inputs.

It is unclear in the random-order model whether online algorithms should know the input length n in advance. This information is not profitable to algorithms facing adversarial inputs but has been, so far, crucial knowledge of every prior algorithm designed with random-order arrival in mind. We call the model where input size n is known in advance the *secretary model*. On the other hand, in our *random-order model* input size n is not revealed. The secretary model allows for vastly superior performance guarantees.

With regards to errors and uncertainty, we study *Online Makespan Minimization with budgeted uncertainty*. Budgeted uncertainty is popular in the analysis of offline algorithms but has, to the best of my knowledge, never been considered in an online problem before. A recently popular line of online algorithm research, which studies explorable uncertainty, is thus complemented by a model of unexplorable budgeted uncertainty.

In total we explore three models. The P^d model provides a more general and realistic cost-charging scheme. The random-order model provides more optimistic performance guarantees that go beyond worst-case analysis. Budgeted uncertainty assumptions prohibit overmuch optimism by including uncertainty and the possibility of errors in the input.

1.1. Brief Problem Overview

This section gives a brief overview over the problems studied in this thesis. A more comprehensive introduction then follow in Chapter 2.

For the **List Update** problem in the P^d *model*, or *paid exchange model*, one is given a linear item list L. Then, a sequence $\sigma = \sigma(1), \ldots, \sigma(m)$ of requests to items in L arrives. Request $\sigma(t)$, $1 \leq t \leq m$, is served by linearly searching through the list until the requested item is found. If $\sigma(t)$ asks for the ith item in the current list, serving the request incurs cost i. In order to reduce this cost, items can be moved to the front of the list using paid exchanges. A *paid exchange* swaps two neighboring items in the current list. Each such swap incurs cost d, for some real number $d \geq 1$. The *standard model* sets $d = 1$, which underestimates exchange costs compared to realistic list implementations. Traditionally, the standard model also allows certain *free exchanges*, discussed in Chapter 3.

To an online algorithm A, requests $\sigma(1), \ldots, \sigma(m)$ are revealed one by one and each has to be served permanently and irrevocably before the next one is shown. Let us denote the cost incurred by algorithm A on request sequence σ by $A(\sigma)$, while $\mathrm{OPT}(\sigma)$ is the cost of an optimal offline algorithm that knows the whole request sequence σ in advance. Online algorithm A is said to be *c-competitive* if $A(\sigma) \leq c\,\mathrm{OPT}(\sigma) + \alpha$. The constant α has to be independent of request sequence σ but may depend on the list size n. A randomized online algorithm A is *c-competitive* if $\mathbf{E}[A(\sigma)] \leq c\,\mathrm{OPT}(\sigma) + \alpha$ for all inputs σ. Here, the expectation is taken over all of A's random choices.

For **Makespan Minimization** and **Machine Covering**, a *job set* $\mathcal{J} = \{J_1, \ldots, J_n\}$ of *size* n has to be assigned to m parallel and identical *machines*. Job J_i has *processing time* p_i and runs on precisely one machine. In particular, preemption is not allowed. The *load* of a machine is the total processing time of jobs assigned to it. For *Makespan Minimization*, the goal is to minimize the maximum load of a machine, also called the *makespan*. For the dual problem, called *Machine Covering*, the goal is to maximize the minimum load of a machine.

An online algorithm A receives job set $\mathcal{J}$ in some order σ. Formally, order σ is a permutation of the integers 1 to n. Algorithm A then treats job sequence $\mathcal{J}^\sigma = J_{\sigma(1)}, J_{\sigma(2)}, \ldots, J_{\sigma(n)}$ in order. Only after it assigns job $J_{\sigma(i)}$, $1 \leq i < n$, permanently and irrevocably will the next job $J_{\sigma(i+1)}$ be revealed. Let $A(\mathcal{J}^\sigma)$ denote the resulting makespan and let $\mathrm{OPT}(\mathcal{J})$ be the optimum makespan. $\mathrm{OPT}(\mathcal{J})$ does not depend on the input order σ. Classically for Makespan Minimization, the *adversarial competitive ratio* $c = \sup_{\mathcal{J},\sigma} \frac{A(\mathcal{J}^\sigma)}{\mathrm{OPT}(\mathcal{J})}$ is considered. If A is randomized, $c = \sup_{\mathcal{J},\sigma} \frac{\mathbf{E}[A(\mathcal{J}^\sigma)]}{\mathrm{OPT}(\mathcal{J})}$. For Machine Covering, which is a maximization problem, the adversarial competitive ratio is defined as the inverse value $c = \sup_{\mathcal{J},\sigma} \frac{\mathrm{OPT}(\mathcal{J})}{A(\mathcal{J}^\sigma)}$, or $c = \sup_{\mathcal{J},\sigma} \frac{\mathrm{OPT}(\mathcal{J})}{\mathbf{E}[A(\mathcal{J}^\sigma)]}$ if randomization is involved.

One focus of this thesis is on the *random-order model*. Herein, the input order σ is picked uniformly at random among all $n!$ possible choices. One important variant, which we call the *secretary model*, reveals the input length n to the online algorithm in advance. In adversarial settings, such knowledge is not useful, but for random-order arrival it makes a big difference. For Makespan Minimization, the *competitive ratio in the random-order (or secretary) model* is $c = \sup_{\mathcal{J}} \mathbf{E}_\sigma[\frac{A(\mathcal{J}^\sigma)}{\mathrm{OPT}(\mathcal{J})}] = \sup_{\mathcal{J}} \frac{1}{n!} \sum_\sigma \frac{A(\mathcal{J}^\sigma)}{\mathrm{OPT}(\mathcal{J})}$. Inversely, for Machine Covering the *competitive ratio in the random-order (or secretary) model* is $c = \sup_{\mathcal{J}} \frac{\mathrm{OPT}(\mathcal{J})}{\mathbf{E}_\sigma[A(\mathcal{J}^\sigma)]}$.

We also study *budgeted uncertainty* assumptions, where the processing time of each job J_t consists of a *regular processing time* $\tilde{p}_t$ and an *additional processing time* Δp_t. Up to Γ jobs may fail and require an extended processing time $\tilde{p}_t + \Delta p_t$ while all other jobs are processed in time $\tilde{p}_t$. The number $\Gamma \in \mathbb{N} \cup \{0, \infty\}$ of failing jobs is part of the input. The maximum possible makespan if the up to Γ jobs fail is called the *uncertain makespan*. This uncertain makespan should be minimized. For $\Gamma \in \{0, \infty\}$, the problem is equivalent to the old problem of Makespan Minimization; for general Γ, this problem is, as we show, strictly harder.

1.2. Brief Literature Overview

A very brief overview of results most relevant to our work is provided. A more comprehensive literature review then follows in Chapter 2.

For **List Update** in the standard model, the best competitive ratio of deterministic algorithms is 2 cf. [1, 71, 153]. This cannot be improved further [115]. Using randomization, competitive ratios as small as 1.6 are possible [20]. Better ratios are only possible for so called non-projective algorithms [25]. Analyzing non-projective algorithms is considered extremely difficult and even for these algorithms, a lower bound of 1.5 holds [156]. The best deterministic algorithm in the P^d model achieves a competitive ratio of $\frac{5+\sqrt{17}}{2} \approx 4.5616$ [21]. Moreover, a general lower bound of 3 holds [143]. Using randomization, a competitive ratio of $\frac{5+\sqrt{17}}{4} \approx 2.2808$ is possible [143]. Prior to our work, no lower bound for randomized algorithms was known.

For **Makespan Minimization**, the deterministic Greedy strategy is $(2 - \frac{1}{m})$-competitive [94]. This ratio is tight. Even in the random-order model, Greedy still remains 2-competitive for $m \to \infty$ [139]. The best known adversarial competitive ratio of any deterministic online algorithm is 1.9201 [84]. No ratio below 1.88 is possible [145]. The role of randomization is poorly understood. The best randomized online algorithm barely outperforms deterministic guarantees with a competitive ratio of 1.916 [3]. Opposed to that, the best randomized lower bound is only $\frac{1}{e-1} \approx 1.581$ [51, 151]. All algorithms currently developed for random-order arrival [8, 15, 91, 97] fall into what we call the secretary model. They know the input size n in advance.

For **Machine Covering**, no deterministic online algorithm can be better than m-competitive, where m is the number of machines [157]. This competitive ratio is already obtained by the Greedy strategy. Using randomization, the best competitive ratio is $\tilde{O}(\sqrt{m})$ [33]. Recall that the notation $\tilde{O}$ hides logarithmic factors in the parameter function, here $\sqrt{m}$. The upper bound of $\tilde{O}(\sqrt{m})$ is optimal up to this hidden logarithmic factor [157].

1.3. Summary of Contributions

For the **List Update** problem, this thesis provides an improved randomized algorithm, which beats prior performance guarantees with its competitive ratio of 2.2442. To establish this competitive ratio, we develop new analysis methods that allow to use the advanced randomized TIMESTAMP algorithm from [1] together with COUNTER-strategies developed for the P^d model from [143]. We moreover establish first lower bounds. No online algorithm can be better than 1.8654-competitive in the full cost model. For the partial cost model, we can even prove a lower bound of $2 - \frac{1}{2d}$. To the best of my knowledge, there is no algorithm for general size lists that is known to perform better in the partial cost model than the full cost model.

Three additional new results on the List Update problem not contained in my prior publications follow. We first discuss formal reasons why it is difficult to improve deterministic results using COUNTER-strategies, something which we successfully do in randomized settings. Then, a simple and efficient description of the optimum offline algorithm on two-item-lists provides intuition for our lower bounds. We finally recover and verify the bounds of RANDOM RESET from [143]. These bounds and their proofs were known to the authors but are only partially published. Our analysis uses and exemplifies the new techniques required to derive our main upper bound, the competitive ratio of 2.2442.

For **Makespan Minimization in the random-order model**, we provide a deterministic 1.8478-competitive algorithm. For deterministic algorithms, this result separates the random-order model from the adversarial model. In fact, we use a stronger performance measure called *nearly-competitiveness* that considers nearly worst-case orders. Only a vanishing fraction of input permutations is excluded. For this, our analysis classifies *stable sequences*. These sequences can be interpreted as the kernel of the problem: other sequences are either trivially treated well by any sensible online algorithm or are extremely rare. The main challenge becomes to obtain good results on stable sequences. Here, an intricate adversarial analysis leads to the desired competitive ratio of 1.8478. Next to this upper bound, we establish first lower bounds. No algorithm can be better than 4/3-competitive in the random-order model and no algorithm can be better than nearly 1.5-competitive for our stronger model of nearly-competitiveness.

For **Makespan Minimization in the secretary model**, a simple adaption of the algorithm LightLoad from the literature [10] is nearly 1.75-competitive. We show that its competitive ratio in the random-order model is less than $1.75 + \frac{4.4}{\sqrt{m}} + O(\frac{1}{m})$. Next, a more sophisticated algorithm is nearly 1.535-competitive, which even beats the pessimistic randomized lower bound of $\frac{1}{e-1} \approx 1.581$. These results are complemented by first lower bounds. No online algorithm, deterministic or randomized, can be better than 1.043-competitive in the secretary model, neither can it be better than nearly 1.257-competitive.

For **Makespan Minimization with budgeted uncertainty**, we first analyze the Greedy strategy and prove that it is $(3 - \frac{2}{m})$-competitive. This result is tight and nicely complements Graham's classical competitive ratio of $(2 - \frac{1}{m})$. We then consider an improved strategy, which has competitive ratio approaching 2.9052. On the other side, a lower bound of 2 separates this problem from classical Makespan Minimization.

For **Machine Covering in the secretary model**, we again analyze the Greedy strategy and prove that it is $O(\frac{m}{\log(m)})$-competitive. This ratio is tight up to a factor of $2 + o(1)$. We then develop a randomized algorithm whose competitive ratio of $\tilde{O}(\sqrt[4]{m})$ beats adversarial upper and lower bounds. We complement this result by a general lower bound of $\tilde{\Omega}(\log(m))$ in the secretary model. The lower bound follows from results on a new variant of the famous Secretary Problem, called the *Talent Contest Problem*. This problem might be of independent research interest. Intuitively, the Talent Contest Problem formalizes the main algorithmic challenge in Machine Covering—the challenge of distinguishing small and large jobs.

1.4. Publication Summary

This publication-based dissertation is based on the following five papers [8, 12–15]. Each paper is published or accepted for publication in peer-reviewed computer science conferences and journals.

Makespan Minimization in the Secretary Model. S. Albers, M. Janke. To appear in: 41st Annual Conference on Foundations of Software Technology and Theoretical Computer Science (**FSTTCS**) 2021

Machine Covering in the Random-Order Model. S. Albers, W. Gálvez, M. Janke. To appear in: The 32nd International Symposium on Algorithms and Computation (**ISAAC**) 2021

Online Makespan Minimization with Budgeted Uncertainty. S. Albers, M. Janke. 17th Algorithms and Data Structures Symposium (**WADS**) 2021

Scheduling in the Random-Order Model. S. Albers, M. Janke. *Algorithmica* (2021): 1-30.
A preliminary version appeared in: The 47th International Colloquium on Automata, Languages and Programming (**ICALP**) 2020

The List-Update Problem in the Paid Exchange Model. S. Albers, M. Janke. 37th Symposium on Theoretical Aspects of Computer Science (**STACS**) 2020

All papers can be found in the appendix.

1.5. Outline of the Thesis

Chapter 2 introduces the fundamental notions of *online algorithms* and *competitive analysis*, as well as the basic problems studied in this thesis: *Online Makespan Minimization*, its dual problem *Online Machine Covering* and the *List Update* problem. Each problem description is followed up with a detailed literature review.

Five chapters follow, which provide a technical overview over each contribution. Every chapter describes one of the five papers. Chapter 3 studies the List Update problem in the paid exchange model. Chapter 4 investigates Makespan Minimization in the random-order model. Chapter 5 discusses Makespan Minimization in the secretary model. Chapter 6 explores Makespan Minimization under budgeted uncertainty. And Chapter 7 considers Machine Covering in the secretary model. Each chapter recalls the problem definition in a brief introduction. The main contributions are named, and followed by proof sketches and, possibly, additional results that did not make it into publication. We end each chapter with a brief summary of open problems and possible future research directions.

A final appendix contains the five papers underlying this publication based thesis.

Chapter 2

Literature

Online algorithms are central to this thesis. We begin with a brief recollection of their definition and introduce *competitive analysis*. Then the three problems of interest, the *List Update* problem, *Online Makespan Minimization* and its dual problem, *Online Machine Covering*, are discussed. These problems are studied in three models: the P^d *model*, or *paid exchange model* for the List Update problem, the *secretary model* for Makespan Minimization and Machine Covering, as well as additionally the *random-order model* and *budgeted uncertainty* assumptions for Makespan Minimization.

2.1. Online Algorithms

The area of online algorithms studies the impact of "not knowing the future". Traditional analyses consider what we call *offline algorithms*. These follow the Input-Process-Output-Paradigm. Offline algorithms first receive the whole input, then perform some computations and finally return an output. Opposed to this, practical applications often do not allow full knowledge of the whole input in advance. Instead, a sequence $\sigma = \sigma(1), \sigma(2), \ldots, \sigma(m)$ of requests arrives, and each request has to be served by an *online algorithm* A permanently and irrevocably before the next one is revealed. This typically prevents optimal performance on general input sequences. Instead, one judges online algorithms in terms of *competitive analysis*, which measures how well the online algorithm adapts to the changing requirements of new requests or, equivalently, by how far the online solution may stray from the optimum offline one.

Let us first consider a *minimization problem*. We denote the cost of a deterministic online algorithm A on input sequence σ by $A(\sigma)$ and the optimum, i.e. minimal, cost on σ by $\mathrm{OPT}(\sigma)$. The *competitive ratio* c of A is defined as $c = \limsup_{\mathrm{OPT}(\sigma)\to\infty} \frac{A(\sigma)}{\mathrm{OPT}(\sigma)}$. Equivalently, c is chosen minimal such that for every input σ there holds $A(\sigma) \leq c\,\mathrm{OPT}(\sigma) + \alpha$ for some constant α independent of σ. If the online algorithm A is randomized, one considers $\mathbf{E}[A(\sigma)]$ instead of $A(\sigma)$ where the expectation is taken over all random choices of online algorithm A. Again, the *competitive ratio (against oblivious adversaries)* is $c = \limsup_{\mathrm{OPT}(\sigma)\to\infty} \frac{\mathbf{E}[A(\sigma)]}{\mathrm{OPT}(\sigma)}$. Historically, other types of adversaries were considered, cf. [41], but they do not receive much attention in the recent literature. For maximization problems, the

competitive ratio is defined inversely. It is $c = \limsup_{\mathrm{OPT}(\sigma)\to\infty} \frac{\mathrm{OPT}(\sigma)}{A(\sigma)}$ for deterministic and $c = \limsup_{\mathrm{OPT}(\sigma)\to\infty} \frac{\mathrm{OPT}(\sigma)}{\mathbf{E}[A(\sigma)]}$ for randomized algorithms.

The area of online algorithms is too vast to be reviewed fully in this thesis. Famous online problems include Ski Rental [113,114], List Update [108,143], Paging [82,153], Bin Packing [37,105], Scheduling [5,141], Matching [92,116] and the k-Server Problem [38,125]. The survey [4] and the books [46,83] contain more information on this topic.

2.2. The List Update Problem

Figure 2.1: Example of an unordered list. Serving a request to item z requires linearly searching through the list starting from the beginning at y. Searching item z incurs cost 3 given by its list position.

In the List Update problem the goal is to dynamically manage an linear list of items to minimize access cost. The List Update problem is one of the most basic online problems [46,153] and has been studied since the 1960s [133] leading to a vast body of literature, see [1,19,25,27,46,49,65,103,108,143,153] and references therein.

Formally, a list L of n items is given. A sequence $\sigma = \sigma(1), \sigma(2), \ldots, \sigma(m)$ of requests to the items in L then has to be served in order. To serve the item requested by $\sigma(t)$, $1 \leq t \leq m$, one has to start at the beginning of the list L and search linearly until it is found. Therefore, requests to the ith item incur cost i. In the *standard model*, the item, immediately after being requested, may be moved to any position closer to the front of the list free of charge. Such exchanges are called *free exchanges*. Moreover, at any time, two neighboring items in L can be swapped at cost 1. These exchanges are called *paid exchanges*. List Update in the standard model is well researched and further progress is considered extremely difficult. [25]

Additional list operations, in particular insertion and deletions of items, can be implemented without incurring meaningful extra cost using accesses and are thus typically not included in the problem formulation [21,23,153].

To an *online algorithm* A, requests $\sigma(1), \ldots, \sigma(m)$ arrive one by one. Upon arrival, each request has to be served before the next one is revealed. Online algorithm A is called c-competitive if it satisfies the previous definition for every list size n. For deterministic online algorithms, this is equivalent to $A(\sigma) \leq c\,\mathrm{OPT}(\sigma) + \alpha$ for every input sequence σ. The constant α may be used to initialize the list and can depend on the list size n but not on the input σ. A randomized online algorithm A is c-competitive if there exists a constant α such that $\mathbf{E}[A(\sigma)] \leq c\mathrm{OPT}(\sigma) + \alpha$ for all inputs σ.

Related Work: The offline List Update problem is NP-complete [22] but polynomial algorithms exist for any fixed list size n [142]. Almost all of the literature focuses on the online problem. Since results are numerous, we only review the ones most relevant to this thesis. Sleater and Tarjan [153] first have shown that their MOVE-TO-FRONT strategy is 2-competitive. MOVE-TO-FRONT uses free exchanges to move every request to the front of its list after serving. This is best possible. No deterministic online algorithm can have competitive ratio below 2 [115]. Many other 2-competitive deterministic algorithms were discovered subsequently [1, 71].

Using randomization, the first improvement has been made by Irani [103]. Her algorithm SPLIT is 1.9375-competitive. Next, Reingold et al. [143] introduced their 1.75-competitive algorithm BIT. In the same paper, they generalize their approach to a family of COUNTER algorithms. The best COUNTER algorithm has a competitive ratio of 1.7346. They also included the even more general family of RANDOM RESET algorithms for ratios up to $\sqrt{3} \approx 1.731$. Albers [1] presented a family of TIMESTAMP algorithms whose optimal competitiveness reaches the golden ratio $\frac{1+\sqrt{5}}{2} \approx 1.6180$. Albers, Stengel and Werchner [20] later devised the online algorithm COMB, which combines BIT and the TIMESTAMP algorithm, to achieve the currently best competitive ratio of 1.6.

In terms of lower bounds, Teia [156] has established that no randomized online algorithm can be better than 1.5-competitive. Ambühl et al. [24] and Ambühl [23] improved this ratio to 1.50115 in the *partial cost model*. In this model the cost of serving each request is reduced by 1. Ambühl et al. [25] also show that no *projective* algorithm in the partial cost model is better than 1.6-competitive. Intuitively, an algorithm is *projective* if it suffices to consider two-item-lists. All aforementioned upper bounds hold in the partial cost model, too, and all but Irani's algorithm SPLIT-algorithm are projective. Improving the best competitive ratio of 1.6 is thus considered extremely difficult and intractable by techniques developed so far.

More recent research exonerates paid exchanges. Early work conjectured them unnecessary [153], which has later been disproven [129]. The optimal offline algorithm does not need free exchanges [142].

Recent research has proposed models for locality of reference [19, 27, 65], advice [49], untrusted advice [26], access model tailored to specific forms of memory management [138] or reduced cost models relevant to data compression [107, 109]. In the strongest such model [107], the whole list can be rearranged free of charge and the serving the ith item only costs $a\lfloor \log(i) \rfloor + b$ for some $a, b > 0$. This cost corresponds to the number of bits required to encode the access costs of a deterministic List Update algorithm in binary. Fotakis et al. [85] study a more general version of the List Update problem, where sets of items are requested. The cost of serving a request to set S is the smallest position any element of S has in list L.

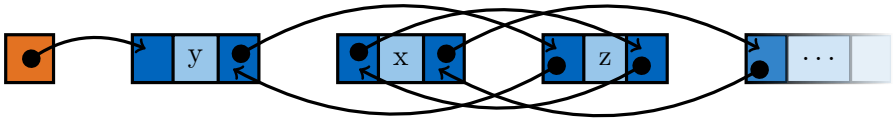


Figure 2.2: Swapping items x and z in Figure 2.1. Rearranging six pointers should be more expensive than simply traversing a single one.

We refer to the excellent surveys [21, 108] for further results.

The P^d Model

The P^d *model*, or *paid exchange model*, was introduced by Manasse, McGeoch and Sleator [131] and Reingold, Westbrook and Sleator [143]. In the P^d model, only free exchanges are prohibited and each paid exchange incurs general cost d, for some real number $d \geq 1$.

Reingold, Westbrook and Sleator [143] introduced COUNTER and RANDOM RESET algorithms. Deterministic COUNTER algorithms achieve competitive ratios up to $\frac{5+\sqrt{17}}{2} \approx 4.5616$ for large values of d [21] while, again for large d, no deterministic algorithm can be better than 3-competitive [143]. Reingold et al. [143] show that randomized COUNTER algorithms have half the competitive ratio of their deterministic counterparts, namely $\frac{5+\sqrt{17}}{4} \approx 2.2808$. Their RANDOM RESET algorithms allow further improvements for small d by mitigating the integer rounding required for counter values.

2.3. Makespan Minimization

For Makespan Minimization a set of n *jobs* $\mathcal{J}$ is given. Each job is defined by its non-negative *processing time* (or *size*). One is tasked to assign these jobs $\mathcal{J}$ to m parallel and identical machines. Preemption is not allowed. The goal is to minimize the *makespan*, that is the time it takes to complete them all. Formally, the sum of processing times of jobs assigned to a machine is called its *load*; the maximum load is then the makespan.

To an *online algorithm* A, jobs arrive one by one and each has to be scheduled immediately and irrevocably before the next one is revealed. A deterministic online algorithm A is called c-competitive if $A(\mathcal{J}^\sigma) \leq c\,\mathrm{OPT}(\mathcal{J}^\sigma)$ holds for every job set $\mathcal{J}$ and every order σ jobs can be presented to the online algorithm. The constant term α in the original definition is superfluous since scaling processing times by a factor λ allows for arbitrarily large values of $\mathrm{OPT}(\mathcal{J})$ without changing the behavior of sensible online algorithms.

Related Work: Makespan Minimization is one of the most basic scheduling problems and has been studied extensively both in its online and offline

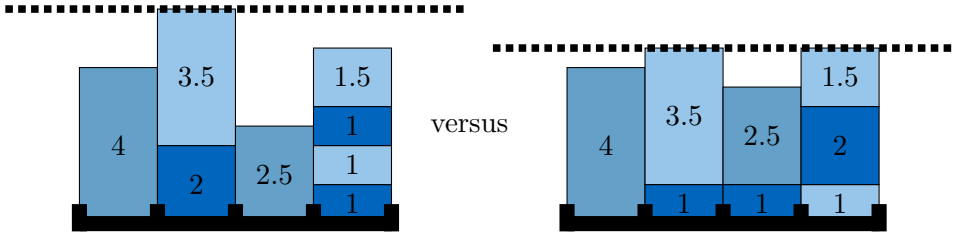


Figure 2.3: Two possible schedules of a job set with makespan 5.5 on the left, and optimum makespan 4.5 on the right. If an online algorithm produces the left schedule, it cannot be better than 11/9-competitive.

variant. The offline variant is NP-complete and admits an EPTAS [52, 100, 104]. It has received considerable research interest, see [52, 100, 104] and references therein. For the remainder of this section we focus on the online variant. Even here, we restrict ourselves to the work most relevant to this thesis.

Already in the 1960s, Graham [94] established that his famed Greedy strategy is $(2 - \frac{1}{m})$-competitive. It took nearly thirty years for Galambos and Woeginger [89] to develop techniques that improve upon this result by some $\epsilon_m > 0$ where $\epsilon_m \to 0$ for $m \to \infty$. Their result first uses the new bound $2P_{m+1}$, twice the $(m+1)$th largest processing time of a job, and the idea of avoiding flat schedules fundamental to every work further on. Building on that, Bartal et al. [40] established their competitive ratio of 1.986. This was improved by Karger et. al [112] to 1.945 and by Albers [2] to 1.923. Finally, Fleischer and Wahl [84] provided a competitive ratio of $1 + \sqrt{\frac{1+\ln(2)}{2}} < 1.9201$, which remains the state of the art since 2000.

Günther et al. [96] and later Chen et al. [53] devise algorithms whose competitive ratio is $1 + \varepsilon$ times the best possible one by searching through a suitably pruned game tree of the corresponding request answer game. No further explicit bounds on the optimum competitive ratio are provided.

The first general lower bound is $1 + \frac{\sqrt{2}}{2} \approx 1.707$ [78]. It has later been improved to 1.837 [40], 1.852 [2] and 1.853 [93]. Rudin III [145] established the currently best bound of 1.88.

For classical Online Makespan Minimization randomization is poorly understood. The best randomized algorithm by Albers [3] requires involved combinatorial arguments to barely outperform the best deterministic result. Its competitive ratio is 1.916. Opposed to that, the best general lower bound is only $\frac{1}{e-1} \approx 1.581$ [51, 151].

Better results are possible for small numbers of machines. The optimum deterministic competitive ratio is 1.5 for $m = 2$ and 5/3 for $m = 3$ machines, see [78, 94]. It lies between $\sqrt{3} \approx 1.732$ [146] and 1.7333 [89] on $m = 4$ machines. Already on $m = 24$ machines, no competitive ratio below 1.8566

is possible [145]. The best randomized competitive ratio is 4/3 on $m = 2$ machines [40].

Other variants of the Online Makespan Minimization problem cover more general machines. *Related machines* have different speeds. A job of processing time p assigned to a machine with speed s requires processing time p/s. Upper bounds are presented in [30, 42], the best being 5.828 for deterministic algorithms [30] and $2e \approx 5.436$ using randomization [42]. Lower bounds are provided in [42] and [70]. The currently best is 2.564. On *unrelated machines*, jobs have machine dependent processing times (p_M). A special case is the *Restricted Assignment* problem, where all processing times p_M are either 0 or 1. For both variants, the optimum competitive ratio is logarithmic in the number of machines m [29, 34]. Other variants generalize the makespan objective, the l_∞-norm of the machine loads, to l_p-norms with general p, see [31, 75, 102]. The area of *Vector Scheduling* considers jobs with multidimensional sizes representing diverse resource requirements. For any fixed vector dimension d, constant competitive ratios are possible [56, 101]. In particular in dimension $d = 2$, an 8/3-competitive algorithm was recently provided by Cohen et al. [56].

We now return to the main problem of Online Makespan Minimization on many parallel and identical machines. Since there has been no improvement in the pure online setting for the last twenty years, semi-online models have been considered where the online algorithm is given extra resources. A highly popular model is *Bin Stretching* where the optimum makespan is known in advance. A prolific line of research [35, 45, 88, 117] lead to an upper bound of 1.5. Interestingly, only an almost trivial lower bound of 4/3 is known for general m [35, 119]. If instead of the optimum makespan the total processing volume is known in advance, the problem is solved. A line of papers [10, 54, 118, 119] conclude that the optimum competitive ratio is 1.585 [10, 118]. In *advice complexity* settings, algorithms are allowed to receive a certain number of *advice bits* in advance. Albers and Hellwig [11] as well as Dohrau [64] show, seemingly independently, that a constant number of advice bits allows competitive ratios arbitrarily close to 4/3, while $O(m)$ advice bits allow competitive ratios arbitrarily close to 1. Moreover, it is shown in [11] that $\Omega(m)$ advice bits are, in fact, required for competitive ratios below 4/3.

In the *bounded migration* model, whenever a job of processing time p arrives, jobs of total processing time up to $\beta \cdot p$ can be reassigned to different machines. The factor β is called the *migration factor*. This allows to retroactively righten mistakes made in the past. Sanders et al. [148] provide a competitive ratio of 1.5 for $\beta = 4/3$ and a ratio of $1 + \varepsilon$ for $\beta = 2^{\tilde{O}(1/\varepsilon)}$. If preemption is allowed, Epstein and Levin [75] design a best-possible online algorithm, which has $\beta = 1 - \frac{1}{m}$. As long as the processing times involved are small, bounded migration allows to perform any number of

migrations. Albers and Helwig [9] study *limited migration*. For limited migration, only a bounded number of jobs can be reassigned after all jobs are treated. Albers and Helwig [9] provide a 1.4659-competitive algorithm migrating $O(m)$ jobs and show that no better algorithm is possible using $o(n)$ migrations. Englert et al. [72] extend this model to uniform machines. They provide a 1.7992-competitive algorithm migrating $O(m)$ jobs. They also show that no ratio below 2 is possible using $o(m)$ migrations.

There are two semi-online models that pertain the input order. Quite early, Graham [95] established that his Greedy strategy is $(\frac{4}{3} - \frac{1}{3m})$-competitive if jobs are presented by decreasing processing times. A better algorithm for such orders is 1.25-competitive [55]. Lower bounds are provided in [150], namely 7/6 for $m = 2$ and $\frac{1+\sqrt{37}}{6}$ for $m = 3$. For these numbers of machines matching upper bounds are presented in [95] respectively [55]. "The power of reordering" is further studied by Englert, Özmen and Westermann [73], who equip the online algorithm with a buffer, which can be used for the dual purpose of lookahead and reordering the sequence "on the fly". They show that a buffer of size $O(m)$ suffices for a competitive ratio approaching 1.4659 for $m \to \infty$. No sensible buffer size can improve upon their result. The ratio of 1.4659 is the same one obtained in the limited migration setting [9].

For more results on semi-online scheduling consult the surveys [6, 74].

Machine Covering

The dual problem to Makespan Minimization is known as *Machine Covering*. Here, the goal is to maximize the minimum load of a machine. Machine Covering is relevant to applications where jobs represent resources. These resources are consumed by the machines in order to work. The goal is to keep all machines running for as long as possible. Machine Covering has applications in the sequencing of maintenance actions for aircraft engines [87] and in storage area networks [149].

Related Work: The offline problem, also known as *Santa-Claus* and *Max-Min Allocation* Problem, is strongly NP-hard but admits a PTAS [157]. For further research in this setting consult [28, 39, 157] and references therein.

General results for Online Machine Covering are extremely restrictive. Recall that an online algorithm A receives jobs one by one. Each job has to be assigned permanently and irrevocably before the next one is revealed. Since Machine Covering is a maximization problem, the competitive ratio is $c = \sup_{\mathcal{J}^\sigma} \frac{\mathrm{OPT}(\mathcal{J})}{A(\mathcal{J}^\sigma)}$. This uses the convention $0/0 = 0$ and $r/0 = \infty$ for $r > 0$. The goal is to find online algorithms obtaining small competitive ratios c. For deterministic algorithms, the best competitive ratio is m, which is achieved by the Greedy strategy [157]. The lower bound is depicted in Figure 2.4. First, m jobs of processing time 1 have to be assigned by a competitive deterministic online algorithm A to different machines. Then, subsequent

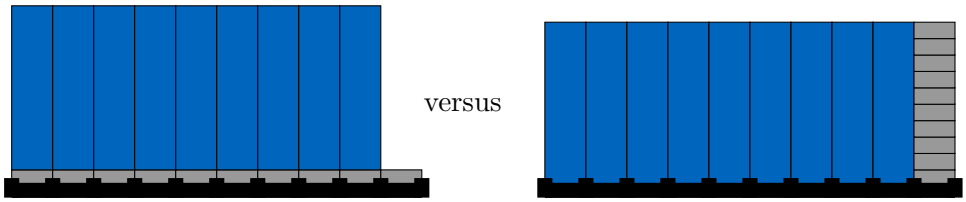


Figure 2.4: The lower bound for deterministic Machine Covering shows that classical adversarial analyses are extremely pessimistic. A competitive online algorithm, depicted on the left, may only achieve minimum load 1, while load m is possible on the right.

$m-1$ jobs of processing time m make online algorithm A perform poorly. Using randomization, a better bound of $\tilde{O}(\sqrt{m})$ is possible. This bound is optimal up the logarithmic term hidden in the $\tilde{O}$-notation [33]. The lower bound again uses that no online algorithm is able to schedule a prefix of m jobs correctly; at least not with probability exceeding $1/\sqrt{m}$.

These pessimistic guarantees have motivated the study of various semi-online models that enhance the online algorithm with extra resources or provide extra information. Azar et al. [33] provide a $(2-\frac{1}{m})$-competitive algorithm, if OPT is known in advance. They also show that no competitive ratio better than 1.75 is possible. Ebenlendr et al. [69] improve the upper bound to $11/6 \approx 1.833$ and the lower bound to 1.791. Recall that in the *bounded migration model*, for some $\beta > 0$, jobs of total processing volume up to $\beta \cdot p$ can be reassigned after a job of processing time p arrives. The algorithm of Sanders et al. [149] is 2-competitive for $\beta = 1$. Gálvez et al. [90] provide an algorithm with competitive ratio $4/3+\varepsilon$ for $\beta = O(\varepsilon^{-3})$ and show that no competitive ratio below $17/16$ is possible with constant migration. Epstein et al. [76] study Machine Covering with a *reordering buffer*, which can store and rearrange a fixed amount jobs. They provide a competitive ratio of $H_{m-1}+1$, where H_{m-1} is the $(m-1)$th harmonic number. They require buffer size $m-1$ and show that this result cannot be improved for any sensible buffer size.

Random-Order Models

The goal of the classical *Secretary Problem* is to pick the best secretary out of a linearly ordered set of n candidates. To the online algorithm, candidates are presented one by one and have to be hired immediately and irrevocably. Once a candidate is hired, all remaining ones are automatically rejected. The online algorithm fails unless it manages to hire the best candidate. It is impossible for deterministic algorithms to pick the best candidate on worst-case arrival orders. Instead, the arrival order is picked uniformly randomly. The goal is to maximize the probability of finding the correct candidate.

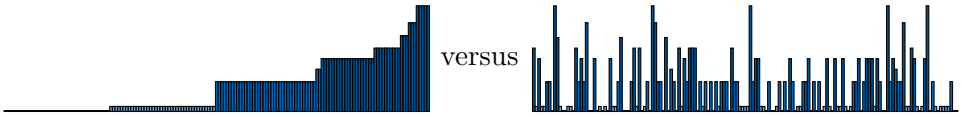


Figure 2.5: Lower bounds in the literature order jobs by increasing processing time. Here, the lower bound from [2] is depicted in the original and a random order.

This classical problem gave rise to the *random-order model*, a popular tool to combat the pessimism inherent in worst-case orders. Given an online algorithm A, its random-order cost on input set $\mathcal{J}$ is $A^{\text{rom}}(\mathcal{J}) = \mathbf{E}_\sigma[A(\mathcal{J}^\sigma)] = \frac{1}{n!}\sum_\sigma A(\mathcal{J}^\sigma)$ where σ ranges over all $n!$ possible orders of input set $\mathcal{J}$. If online algorithm A is randomized, the expected value also includes its random choices. The *competitive ratio of online algorithm A in the random-order model* is $c = \limsup_{\text{OPT}(\mathcal{J})\to\infty} \frac{A^{\text{rom}}(\mathcal{J})}{\text{OPT}(\mathcal{J})}$, or respectively the inverse if a maximization problem is considered. Recall our convention that $0/0 = 1$ and $r/0 = \infty$ for $r > 0$. Then we consider for Makespan Minimization $c = \sup \frac{A^{\text{rom}}(\mathcal{J})}{\text{OPT}(\mathcal{J})}$, and for Machine Covering $c = \sup \frac{\text{OPT}(\mathcal{J})}{A^{\text{rom}}(\mathcal{J})}$.

Related Work: The Secretary Problem has already been studied by Lindley [128] and Dynkin [68] in the 1960s. They show independently that the optimum strategy finds the best secretary with probability approaching $\frac{1}{e}$ as the number n of candidates goes to infinity. Later research focused on the surprisingly non-trivial difference between secretaries being ordered via an abstract linear order or via valuations in the real numbers. The latter is also known as the *game of Googol*. A prolific line of papers [140, 147, 154] has shown that both versions are equivalent unless $n = 2$ candidates are considered. Modern work generalizes the traditional problem in many ways, including picking several secretaries [18, 123] possibly including matroid constraints [36,63,80,126], prior sampling [58,110] or even prophet inequalities and the game of Googol [32, 57, 59]. Consult the surveys [63, 81, 86] and references therein for a more thorough literature review.

Coming from the Secretary Problem, the random-order model became recently popular for online problems such as Matching [92, 111, 116, 130], Knapsack [16, 36], Bin Packing [17, 120, 127], Facility Location [134], Packing LPs [121], Convex Optimization [97], Welfare Maximization [124], Budgeted Allocation [135] and Scheduling [8, 14, 15, 91, 136, 139]. See also [98] for a survey chapter on random-order models.

For Scheduling in the random-order model little is known so far. Osborn and Torng [139] prove that Graham's Greedy strategy is still not better than 2-competitive for $m \to \infty$. Molinaro [136] studies a very general scheduling problem where jobs have different machine dependent processing times bounded by 1. His algorithm has expected makespan $(1 + \varepsilon)\text{OPT} +$

$O(\log(m)/\varepsilon)$ but no further guarantees on the random-order competitive ratio are given. Göbel et al. [91] study Scheduling on a single machine where the goal is to minimize weighted completion times. They provide a constant competitive ratio for unit-sized jobs and a competitive ratio of $O(\log(n))$ for general job sizes. They also show that no sublinear competitive ratio is possible if worst-case orders are considered.

Results in the literature differ depending on whether the input length n is known in advance. For the Secretary Problem, such information is essential. For Online Matching, such information is implicit and hence not required. Any algorithm developed for Scheduling in the random-order model [8, 15, 91, 97] excluding ours in [14] needs to know the input size n in advance. As far as traditional worst-case orders are involved, knowing n is of no help. For random-order arrival, knowing n most likely poses a strong advantage albeit no formal proof is known yet. To distinguish both models, we use the term *secretary model* when n is known and the term *random-order model* else.

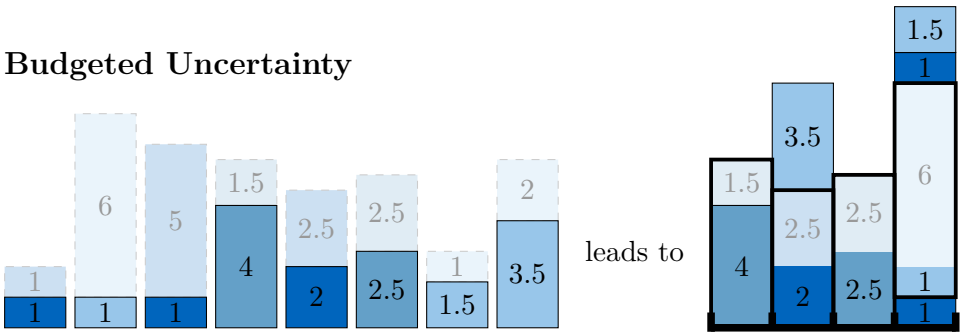


Figure 2.6: The first schedule from Figure 2.3 (left) if jobs can fail. Failing jobs requires additional job-specific processing time depicted on the right. Here, $\Gamma = 1$ jobs fail per machine in the worst possible way. If only the job on the right-most machine failed, the makespan would not change.

For Makespan Minimization with *budgeted uncertainty*, jobs have both a *regular processing time* and an *additional processing time*. Normally, a job requires the former regular processing time to be executed. Unfortunately, up to Γ jobs may fail and require the sum of both processing times. The *uncertain makespan* of a schedule is then the makespan obtained if these Γ jobs are chosen worst possible. Equivalently, up to Γ jobs may fail per machine. The resulting uncertain makespan does not change in this more general setting. For Makespan Minimization with budgeted uncertainty, the goal is again to minimize the uncertain makespan.

In general, budgeted uncertainty equips each input parameter, such as job processing times, with an interval of uncertainty. Up to Γ parameters may *fail* and may be reset to any value in the uncertainty interval. All other parameters retain their original value. The goal is to give guarantees if these failing parameters and their values are chosen worst possible. The prior

formulation of Makespan Minimization with budgeted uncertainty arises since it is always worst to choose processing times of failing jobs maximally.

Related Work: Offline models incorporating uncertainty date back to the 1950s [61]. Earliest models consider stochastic settings where scenarios eventualize according to a probability distribution. The more recent budgeted uncertainty setting from [43] considers worst-case scenarios. It has been applied to many offline problems such as Scheduling [47, 48, 155], Bin Packing [144] and Linear Optimization [43, 44]. In particular, for the offline variant of Makespan Minimization under budgeted uncertainty, which we study as an online problem, an EPTAS has been provided by Bougeret et. al [47].

Budgeted uncertainty has, to the best of my knowledge, never been studied in online settings. This is surprising since uncertainty forms the core area of study for online algorithms. To date, a complementary line of research already introduced explorable uncertainty. The key difference to unexplorable budgeted uncertainty is that explorable uncertainty only impacts the online algorithm, who can resolve it at a certain extra cost. In 1991, Kahan [106] investigated the number of queries necessary to determine median and maximum element of a set of "uncertain numbers". Later work generalized this to finding general rank-k-elements [79, 99, 106, 122], Caching [137] and recently Makespan Minimization [7, 66, 67]. Confer to the survey by Erlebach et. al [77] and references therein for more results on this topic.

Chapter 3

The List Update Problem in the P^d Model

We revisit the List Update problem in the paid exchange model P^d. An unordered item list L has to be maintained dynamically so that a sequence σ of requests to these items can be served with minimum cost. Serving each item request requires linearly searching through L and thus incurs cost given by the item's list position. In order to reduce cost, frequently requested items should be moved closer to the front. For such movement, paid exchanges are used. Paid exchanges may, at any time, swap neighboring items in the list L at a given cost $d \geq 1$ per exchange. The goal is to serve input σ at minimum total cost.

The List Update problem is one of the most basic online problems and has applications to data compression schemes. For example, the compression program BZIP2 applies the Burrows-Wheeler transform together with linear list encoding [50, 60, 132, 152].

The most studied model of the List Update problem is the standard model where d, the cost for exchanging neighboring items, is 1. Traditionally, *free exchanges* are allowed. Their usefulness depends on minutiae in the model description discussed in Paragraph 3.1. The paid exchange model, or P^d *model*, introduced by Manasse, McGeoch and Sleator [131] as well as Reingold, Westbrook and Sleator [143] never included free exchanges and considers general exchange cost d, for $d \geq 1$ a real-valued constant. This is motivated by the fact that in practice one iteration of the traversal loop for servicing an item will be much cheaper than swapping two neighboring items, compare Figure 2.1 and 2.2.

3.1. Preliminaries

Problem Definition

Formally, one is given a list L of n items and a sequence $\sigma(1), \ldots, \sigma(m)$ of requests to items in L that must be served in order. To serve a request to item $\sigma(t)$, $1 \leq t \leq m$, one needs to traverse list L from the beginning until item $\sigma(t)$ is found. If item $\sigma(t)$ is the ith item in the list, a cost of i is incurred. This is called the *full cost model*. In the literature, there is a

second prominent model where the cost for serving a request to the ith item in L is only $i-1$. This is called the *partial cost model*. While the partial cost model is not sensible in practical applications, there is, to the best of my knowledge, no upper bound for arbitrary-sized item lists that does not hold in both models. At any time, the algorithm is allowed to perform paid exchanges. These swap two neighboring items at cost d where $d \geq 1$ is a given real-valued constant. The goal is to minimize the total cost incurred.

Online Algorithms

To an *online algorithm*, item requests arrive one by one and each has to be served without knowledge of future requests. When presented with a request $\sigma(t)$, online algorithm A can make arbitrary paid exchanges before serving the requested item. Given a request sequence σ and an online algorithm A, we denote the cost of A on σ by $C_A(\sigma)$. Let OPT denote the optimum offline algorithm and let $C_{\mathrm{OPT}}(\sigma)$ denote its cost on σ. Then a deterministic online algorithm A is called *c-competitive*, for some real number $c \geq 1$, if $C_A(\sigma) \leq c \cdot C_{\mathrm{OPT}}(\sigma) + \alpha$ for some constant α that is independent of input σ but may depend on the list size n. Similarly, a randomized online algorithm A is *c-competitive (against an oblivious adversary)* if $\mathbf{E}[C_A(\sigma)] \leq c \cdot C_{\mathrm{OPT}}(\sigma) + \alpha$. The expectation is taken over the random choices of A.

Almost all analyses in the literature are carried out in the partial cost model. They carry over to the full cost model due to the following observation.

Lemma 3.1
If an online algorithm, deterministic or randomized, is c-competitive in the partial cost model, then it is also c-competitive in the full cost model.

Proof. By passing over to the full cost model, the cost of serving each request increases by precisely 1. In particular, $C_A(\sigma)$ and $C_{\mathrm{OPT}}(\sigma)$, the cost of A and OPT, increase by precisely the length m of σ. The ratio $\frac{C_A(\sigma)}{C_{\mathrm{OPT}}(\sigma)}$ becomes $\frac{C_A(\sigma)+m}{C_{\mathrm{OPT}}(\sigma)+m}$ when passing over to the full cost model. It can only decrease. □

Exchanges in Between

Assume a request $\sigma(t)$ is revealed and has to be served. Is request $\sigma(t)$ served immediately leaving no time for paid exchanges? Or, can an online algorithm quickly make some paid exchanges moving the requested item closer to the front before serving it? The latter is called an *exchanges in between*.

In the standard model, exchanges in between can be replaced by exchanges, free or paid, after request $\sigma(t)$ is served. The total cost does not increase. Ambühl et al. [23–25] argue conversely that free exchanges are unnecessary in the standard model. They can be replaced by paid exchanges right before serving the request. Such paid exchanges would be in between. This is

evidence that earlier authors did not consider exchanges in between possible but also gives a reason to include such exchanges for a simpler problem definition.

Reingold, Westbrook and Sleator on the one hand state, quoting from [143], "We can think of any algorithm as servicing each request as follows: first some number of paid exchanges are performed, then the request is satisfied, and then any number of free exchanges are done. An on-line list update algorithm must service each request without knowledge of future requests." This indicates that exchanges in between are allowed. On the other hand, their COUNTER-algorithm for the P^d model only moves items to the front of the list after serving them. This would be insensible if moving them before serving, i.e. in between, was also an option.

In our work, we explicitly allow exchanges in between. For the standard model, this has no major impact besides simplifying the model by making free exchanges unnecessary [23–25]. For the P^d model, this leads to slightly better competitive ratios. Moreover, the standard model is equivalent to the P^1 model with exchanges in between.

Projective Algorithms

An online algorithm A is *projective* if the relative order of any two items x, y in its list is fully determined by prior requests to x and y. The popularity of projective algorithms for List Update results from the following proposition.

Proposition 3.1
Consider the partial cost model. A projective online algorithm A is c-competitive if and only if it is c-competitive on request sequences referencing only two items, which are served on a two-item-list.

The result has been long known and holds in both the standard and the P^d model. Reducing to two-item-lists seems central to current List Update analyses. Every algorithm besides Irani's SPLIT algorithm [103] is projective. Early results, such as [143], do not use Proposition 3.1 explicitly. Irani [103] uses different techniques to reduce to two-item-lists. The result of Ambühl et al. [25] establishes that competitive ratios smaller than 1.6 cannot rely on Proposition 3.1 anymore, explaining the lack of progress since the projective algorithm COMB [20] obtained that ratio.

COUNTER-Algorithms

A blueprint for algorithms in the P^d model are COUNTER-algorithms. Consider a well performing online algorithm A from the standard model, respectively the P^1 model. In the P^d model, exchanges become expensive and should be performed less often. To achieve this, COUNTER-algorithms thin out the input sequence via a mod l counter. Ignoring requests with

positive counter value slows down A.

Formally, the COUNTER analogue of online algorithm A, denoted by $A(l)$, maintains a mod l counter for each item x in the list. Whenever a request $\sigma(t)$ asks for item x with counter value $c(x) > 0$, the corresponding counter value $c(x)$ is decremented by 1. Request $\sigma(t)$ is a *minor request*. Once the counter $c(x)$ reaches 0 on request $\sigma(t')$ it gets reset to $l-1$. Request $\sigma(t')$ is a *master request*. The modified algorithm $A(l)$ simulates A on the sequence of master requests. It serves minor requests without rearranging its list. When faced with a master request $\sigma(t)$, algorithm $A(l)$ feeds $\sigma(t)$ to A. Algorithm A thus only "sees" the master requests. Algorithm $A(l)$ makes exchanges according to A's behavior. If algorithm A uses a free exchange on request $\sigma(t)$, there are two possibilities for $A(l)$. Preferably, the free exchange is replaced by a paid exchange right before serving request $\sigma(t)$. This would be an exchange in between. If exchanges in between are not allowed, $A(l)$ has to perform a paid exchange right after serving request $\sigma(t)$. This option is slightly more expensive.

Deterministic COUNTER-algorithms will initialize all counters to $l-1$ at the beginning of the sequence. Any other initialization value leads to the same performance. The adversary can always reinitialize counters arbitrarily by prepending a suitable prefix to input σ. This can be helped using randomization. Randomized COUNTER-algorithms initialize all counters independently and uniformly at random. Now, counters cannot be controlled and predicted by the adversary anymore.

Reingold, Westbrook and Sleator [143] first used the COUNTER-strategy to adapt the MOVE-TO-FRONT algorithm from [153] to the P^d model. MOVE-TO-FRONT simply moves every item to the front of the list before serving it. The resulting archetype of COUNTER-algorithms is simply called COUNTER(l). Important special cases of COUNTER(l)[1] are the randomized COUNTER(2), known as the BIT-algorithm [143], and deterministic COUNTER(1), which, if exchanges in between are used, recovers the MOVE-TO-FRONT strategy.

For large d and the correct choice of l, the competitive ratio of the deterministic COUNTER(l) is $\frac{5+\sqrt{17}}{2} \approx 4.5616$. Its randomized counterpart is $\frac{5+\sqrt{17}}{4} \approx 2.2808$-competitive. In the randomized case, setting the counter value to $l \approx 2.561\,d$ is optimal. Unfortunately, since l has to be an integer we need to round $2.561\,d$. Such rounding is circumvented by a more general algorithm family called RANDOM RESET [143]. RANDOM RESET is parameterized with a *reset distribution* ℓ. Whenever a counter reaches 0, RANDOM RESET resets it to some value $l-1$ where l is drawn from ℓ. Distribution ℓ's expected value $\mathbf{E}_{l\sim\ell}[l]$ can obtain general real values. This

[1] Another family of COUNTER algorithms, considered in [143], is more general than COUNTER(l) but less general than RANDOM RESET. In the standard model, it allows competitive ratios as small as 1.7346. We do not consider this family further in this thesis.

leads to improvements for small values of d where $2.561\,d$ is not close to an integer, see Figure 3.2. For $d = 1$, it improves the ratio of BIT, which is 1.75, to $\sqrt{3} \approx 1.7321$. For $d \to \infty$ the improvement vanishes.

Thus, our main result, which focuses on large d, uses COUNTER-strategies instead of "RANDOM RESET-strategies" for a simpler analysis.

3.2. Our contributions

Recall that the classical COUNTER(l) algorithm simply uses the COUNTER-strategy together with the MOVE-TO-FRONT algorithm. In the next paragraph, we prove that known sensible deterministic algorithms cannot lead to better results. Using randomization, algorithms superior to MOVE-TO-FRONT are known. The second best, TIMESTAMP(p), obtains competitive ratios up to $\frac{1+\sqrt{5}}{2} \approx 1.6180$. In our work, we consider the COUNTER-version of TIMESTAMP(p), called TIMESTAMP(l,p).

The algorithm TIMESTAMP(l,p) maintains, for every list item x, a mod l counter $c(x) \in \{0, \ldots, l-1\}$. In the beginning, each counter $c(x)$ is initialized independently uniformly at random.

Given a request $\sigma(t)$ to item x, there are two possibilities. If $c(x) > 0$, request $\sigma(t)$ is a *minor request*. Minor requests are served without list rearrangement. Counter $c(x)$ is then decremented by 1. Once $c(x) = 0$, request $\sigma(t)$ is a *master request*. For a master request, we proceed according to the TIMESTAMP(p) strategy from [1]. With probability p, where p is the second parameter of TIMESTAMP(l,p), Policy 1 moves item x greedily to the front of the list. With probability $1-p$, item x moves according to the more cautious Policy 2. Consider $\lambda(t)$, the longest suffix of the current request sequence ending with $\sigma(t)$ that contains no master request to x

Algorithm 1 How TIMESTAMP(l,p) serves request $\sigma(t) = x$.

if $c(x) > 0$ **then** // $\sigma(t)$ is a minor request
    $c(x) \leftarrow c(x) - 1$;
else // $\sigma(t)$ is a master request
    $c(x) \leftarrow l - 1$;
    With probability p, serve $\sigma(t)$ using **Policy 1** and with probability $1-p$ serve it using **Policy 2**;
    Policy 1: Move x to the front of the list.
    Policy 2: Let $\lambda(t)$ be the longest suffix of the sequence ending with request $\sigma(t)$ in which exactly one master request to x occurs. Let z be the first item in the current list for which at most one master request occurs in $\lambda(t)$ and the possible master request was served using Policy 2. If $z \neq x$ move x in front of z in the list.

besides $\sigma(t)$. Suffix $\lambda(t)$ begins right after the previous master request to x or at the beginning of σ if none exists. Policy 2 determines the first item z in the current list such that either (1) no master request to z occurs in $\lambda(t)$ or (2) precisely one master request to z occurs in $\lambda(t)$ and this master request has been treated using Policy 2. Then, Policy 2 moves item x in front of z in the list. Note that x satisfies the conditions for item z. If $z = x$, item x is not moved. In particular, item x never moves backwards. Intuitively, Policy 2 only surpasses items that are requested less. In particular, both policies together reduce needless exchanges on alternating sequences $xyxy$. Such useless exchanges are precisely what prevents improvement in deterministic settings, cf. Definition 3.1.

Our main result is that TIMESTAMP(l,p) is 2.2442-competitive with the correct choice of parameters for $d \to \infty$. This ratio c improves upon the randomized COUNTER(l)-algorithm. More precise evaluations of the terms involved reveal that the competitive ratio c improves for small values of d so that the limit is only a formality. Such more precise ratios are included in Figure 3.2. If exchanges in between are not allowed, the opposite is true. TIMESTAMP(l,p) approaches a competitive ratio of 2.2442 from above. One disadvantage is that in this case the upper bounds become fairly complicated due to involved additive terms in $O(1/d)$. Of course, for $d \to \infty$ the bounds do not change. If exchanges in between are allowed, these additive terms are negative. We can thus omit them for a slightly worse but simpler result.

Theorem 3.1
Let $\varphi = \frac{l}{d}$. TIMESTAMP(l,p) is c-competitive, where c is the maximum of the following expressions:

$$c_1 = 1 + \left(\tfrac{1}{2} + \max\{1, 2p\}(1-p)\right)\varphi, \qquad c_2 = \tfrac{7-3p}{4} + \tfrac{1}{\varphi},$$
$$c_3 = 1 + \tfrac{3p}{2} - p^2 + \tfrac{2p}{\varphi}, \qquad c_4 = \tfrac{3-p+p^2}{2} + \tfrac{2(1-2p+2p^2)}{\varphi},$$
$$c_5 = \tfrac{3+p-p^2}{2} + \tfrac{2p^2}{\varphi}, \qquad c_6 = 2 - p + \tfrac{1-p}{\varphi}.$$

In particular, the TIMESTAMP(l,p) algorithm is 2.2442-competitive for $p = 0.45797$ and $l \approx 1.19390d$.

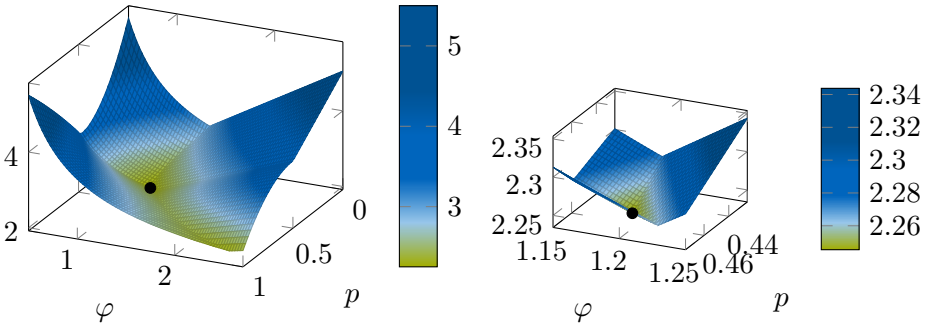


Figure 3.1: The function of Theorem 3.1. (The plot is colored.)

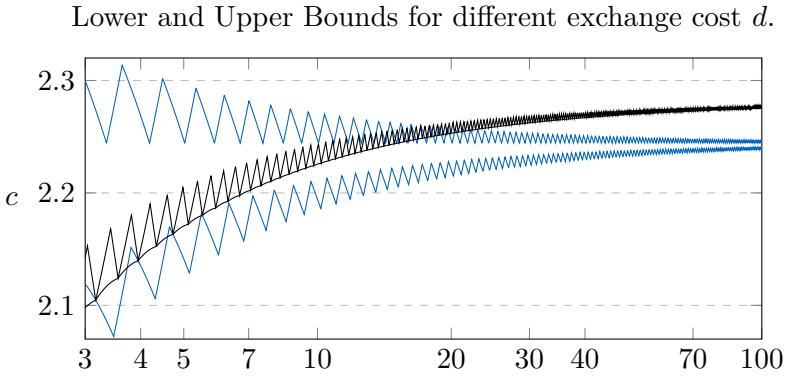


Figure 3.2: A comparison of TIMESTAMP, COUNTER and RANDOM RESET for different values of $d \geq 3$. Exchanges in between are allowed. The plot is colored and the x-axis is log-scaled. The black lines indicate the competitive ratios of COUNTER and the slightly better ratio of RANDOM RESET from [143]. Blue lines indicate the competitive ratio of TIMESTAMP from Theorem 3.1 and a better bound incorporating small improvements in $O(1/d)$ omitted in the main result. For all algorithms, optimal choice of parameters is assumed. The lines of COUNTER strategies are jagged due to the integer rounding involved.

First lower bounds are provided if d is an integer. Our main lower bound only holds in the partial cost model where the cost of serving the ith item is $i-1$ instead of i. So far, no algorithm is known to perform better in the partial cost model for arbitrary-size lists. I moreover conjecture that the lower bound of $2-\frac{1}{2d}$ holds, in fact, for the full cost model, too.

Theorem 3.2
No randomized algorithm for List Update in the partial cost model is better than $(2-\frac{1}{2d})$-competitive.

The main challenge in the proof is to analyze the optimum algorithm on a certain randomized lower bound sequence. We solve this challenge for two-item-lists, where it is already quite involved. If the challenge was solved for lists of general size n, Theorem 3.2 would also hold in the full cost model. Instead of analyzing the optimum algorithm on general size lists, we consider the simplified deterministic algorithm B_h. Algorithm B_h moves the current item to the front of its list if and only if it is requested h times in a row. From the analysis of B_h, we conclude the following general lower bound.

Theorem 3.3
If A is an online algorithm for List Update in the P^d model, then its competitive ratio is at least $1/(1+2W(\frac{-1}{2e}))-O(1/d) \approx 1.8654$.

For $d=1$, we can recover Teia's lower bound of 1.5 [156].

3.3. Complementary results

In the following, we mention a few results, which did not make it in the main paper, but should be of interest.

Deterministic COUNTER Strategies

If we look at potential candidates for deterministic COUNTER-strategies, such as the deterministic TIMESTAMP algorithm [1] or one of the algorithms mentioned in [71], we observe that these algorithms continuously rearrange their lists on alternating two-item-sequences $xyxy$. We call such a behavior *fickle*. We show that fickle algorithms cannot improve the state of the art when plugged into deterministic COUNTER-algorithms. While it is easy to design non-fickle online algorithms, these are either non-projective or perform worse in the standard model. When lifted to the P^d model, they are unlikely to make up for this.

Definition 3.1. Consider the infinite sequence $(xy)^*$ of alternating requests to items x and y. An online algorithm A for List Update in the standard model is called *fickle* if, after possibly excluding a finite prefix of sequence $(xy)^*$, algorithm A moves every request to the front of the list before serving it.

Theorem 3.4
If A is a fickle algorithm, then the COUNTER-algorithm $A(l)$ cannot outperform the classical algorithm $COUNTER(l)$.

Proof. COUNTER(l) has competitive ratio $\max(1+\frac{l}{2\,d}\pm\frac{1}{2\,d}, \frac{3}{2}+\frac{2\,d-1/2}{l}\pm\frac{1}{l})$. For the $\pm$-term, a $-$ is to be chosen if exchanges in between are used to move each requested element to the front before serving. A $+$ is chosen if these exchanges have to occur after serving. The latter bound is published in [143]. Both bounds are presented in Corollary 3.1 stated later.

Consider any algorithm A that behaves like MOVE-TO-FRONT on the sequence $(xy)^*$. One can verify that the COUNTER-algorithm $A(l)$ has (strict) competitive ratio at least $1+\frac{l}{2\,d}\pm\frac{1}{2\,d}$ on input $x^l y^l$. Also, $A(l)$ has (strict) competitive ratio at least $\frac{3}{2}+\frac{2\,d-1/2}{l}\pm\frac{1}{l}$ on input $x^{l-1}y^l x y^{2l-1} x^l y x^l$. Therefore, algorithm $A(l)$ does not outperform COUNTER(l).

These two bounds also hold for the non-strict competitive ratio since we can repeat these sequences arbitrarily often. For the same reason, the bounds still hold if algorithm A deviates on a finite prefix of $(xy)^*$. Such deviation can only improve the cost by a constant. □

Although the randomized TIMESTAMP(l,p) family outperforms the COUNTER(l) family, the opposite holds true with regards to the deterministic TIMESTAMP(l,1) algorithm. It performs worse than the deterministic COUNTER(l) family. Recall that the deterministic COUNTER(l) algorithm achieves competitive ratios approaching 4.5616.

Proposition 3.2
Deterministic TIMESTAMP(l,1) is at most 5-competitive for $d \to \infty$.

Proof Sketch. Evaluate the cost of OPT and TIMESTAMP(l,1) on input sequence $\left(x^{l-1}y^l xy^{3l}x^l y^{2l-1}x^l yx^{3l}y^l x^l\right)^N$ for general N. □

Remark 3.1
Based on my analyses, I know that deterministic TIMESTAMP(l,p) is, in fact, 5-competitive for $d \to \infty$.

OPT on Two-Item-Lists

Assume that in the two-item-list of OPT item y is currently at the front and x at the back. If the next request goes to y, it is optimal to serve this request at minimum cost. Hence, consider the case that the next request goes to x. Should OPT move item x to the front to reduce serving cost? In the standard model, the following rule applies [142]: move item x to the front if and only if it has to be served twice in a row. In other words, OPT is close to being an online algorithm. It does only need to know the next two requests. For $d > 1$ the optimum algorithm is fully offline and might need to know the whole input sequence in advance.

Given an input sequence σ, let the first element in the two-item-list of OPT be y and the last one x. The optimum algorithm will serve any leading requests to y without rearrangements till it encounters a request to x. The first sequence λ starts with this request. For $z \in \{x, y\}$, let $|\lambda|_z$ denote the number of requests to z in λ. We run through the following requests in σ and add them to λ. We stop once one of the following events occurs: Either (1) $|\lambda|_x = |\lambda|_y$, or (2) $|\lambda|_x \geq |\lambda|_y + 2\,d$, or (3) the end of input σ is reached. In case (1) we call λ a *zero-sequence*; in case (2) it is called an *up-sequence*; else, in case (3), it is called the *final sequence*. If λ is a zero-sequence, OPT serves it without rearranging its list. Following λ, more requests to item y may follow, which are served free of charge. These requests form what we call a *post-sequence*. After the post-sequence—that is at the next request to x—we repeat this process to define a new sequence λ'. Sequence λ' will be again an up-, zero- or the final sequence. If λ is an up-sequence, OPT moves item x to the front of its list before serving λ. Afterwards, a *post-sequence* of requests to x may be served free of charge. After these are served, the following sequence λ' begins with a request to y. It is defined analogously with the roles of x and y reversed. If λ is the final sequence, we check whether $|\lambda|_x \geq |\lambda|_y + d$ holds. If so, we move item x to the front before serving λ.

Theorem 3.5
The previously described algorithm is optimal.

Proof. Assume that the proposition was untrue. Let OPT′ be an optimal offline algorithm and consider the first sequence $\lambda = \lambda_i$ in which OPT′ does not behave like the previously described algorithm. An exchange belongs to λ_i if it is performed right before serving a request in this sequence. We choose OPT′ among all optimal algorithms such that i is maximal. In other words, the diverging sequence $\lambda = \lambda_i$ occurs latest possible in σ. Let x again denote the first element requested in λ and let y be the other element in the list L. Element y will be at the front and x at the back of the lists of OPT and OPT′ before treating λ.

First, observe that OPT′ can make at most one exchange in λ. For this, one needs to verify that any proper contiguous subsequence λ' of λ satisfies $||\lambda'|_x - |\lambda'|_y| < 2\,d$. If OPT′ were two make two exchanges in λ, let λ' denote the sequence of requests served in between. Removing these two exchanges would reduce the cost of OPT′ by $2\,d - (|\lambda'|_x - |\lambda'|_y) > 0$. This contradicts optimality.

Next, observe that if OPT′ makes one exchange in λ, this exchange occurs right at the beginning of λ. For this, verify that for any non-empty prefix λ' of λ there holds $|\lambda'|_x > |\lambda'|_y$. If OPT′ did make an exchange after such a non-empty prefix, moving this exchange to the front would reduce the total cost by $|\lambda'|_x - |\lambda'|_y > 0$. This would again contradict optimality.

We now derive a contradiction to the fact that OPT′ is chosen such that the critical sequence λ_i occurs latest possible. There are three cases. Case 1) Sequence λ is a zero-sequence and OPT′ makes an exchange right at the beginning of λ. Moving this exchange to the back of λ would not change the cost of OPT′ since $|\lambda|_x = |\lambda|_y$. Further moving said exchange past the post-sequence would, in fact, decrease the cost unless said post-sequence is empty. We thus found a way to modify OPT′ without increasing its cost and the critical sequence λ_i on which OPT′ does not behave similar to OPT now occurs later. This is the desired contradiction to λ_i occurring latest possible. Case 2) Sequence λ is an up-sequence and OPT′ does not make an exchange right at the beginning. We now add one exchange right at the beginning of λ and another one right after all elements in λ and the post-sequence are served. These exchanges incur cost $2\,d$ on the one hand but, on the other hand, the serving cost for the elements in λ decreases. This decrease is $|\lambda|_x - |\lambda|_y = 2\,d$ plus the length of the post-phase. It makes up for the cost $2\,d$ of adding these two exchanges. We again reached the desired contradiction to λ_i being latest possible. Case 3) If sequence λ is the final sequence, verify that it is optimal to perform an exchange at the beginning if and only if $|\lambda|_x \geq |\lambda|_y + d$. The theorem follows. □

RANDOM RESET

Recall that RANDOM RESET is initialized with a random variable ℓ taking values in the natural numbers. Whenever a counter value reaches 0, we pick

an element $l \sim \ell$ and reset the counter to $l-1$. Each counter needs to be initialized the following way. First pick l with probability $\frac{\mathbf{P}[\ell=l]l}{\mathbf{E}_{l\sim\ell}[l]}$. Then pick the counter value uniformly among the integers $0, 1, \ldots, l-1$. One can verify that this describes a stationary distribution. This distribution of counter values does not change, as counters are decremented and randomly reset. Let $r = \mathbf{E}_{l\sim\ell}[l]$ be the *expected reset value*. Then the *expected counter value* at any time is $R = \mathbf{E}_{l\sim\ell}[\frac{1}{\mathbf{E}_{l\sim\ell}[l]}\sum_{i=1}^{l-1} i - 1] = \mathbf{E}_{l\sim\ell}[\frac{l(l-1)}{r}]$. The competitive ratio of RANDOM RESET depends on these two values r and R.

Proposition 3.3
The competitive ratio of RANDOM RESET is $\max\left(1+\frac{R}{d}, 1+\frac{R+(2\,d-1)}{r}\right)$. If requests in between are not allowed, R has to be replaced by $R+1$ and $2\,d-1$ by $2\,d$ in the previous term.

For simplicity, our proof will focus on the setting with exchanges in between. Proposition 3.3 was likely known to the authors of [143] but to the best of my knowledge never published. We will prove it in the next section. This proof exemplifies the techniques required for our more involved analysis of the TIMESTAMP(l,p)-algorithm.

Corollary 3.1
The COUNTER(l) algorithm in the P^d model has competitive ratio $\max(1+\frac{l}{2\,d}\pm\frac{1}{2\,d}, \frac{3}{2}+\frac{2\,d-1/2}{l}\pm\frac{1}{l})$. For the $\pm$-term, a $-$ is to be chosen if exchanges in between are used to move each requested element to the front of the list before serving. A $+$ needs to be chosen if these exchanges have to occur after serving.

Proof. For the special case of COUNTER(l), the (expected) reset value is $r = l$ and the expected counter value is $R = \frac{l-1}{2}$. Now, apply Proposition 3.3. □

Remark 3.2
Using Proposition 3.3, we verified the results for RANDOM RESET in the P^d model and the standard model from [143].

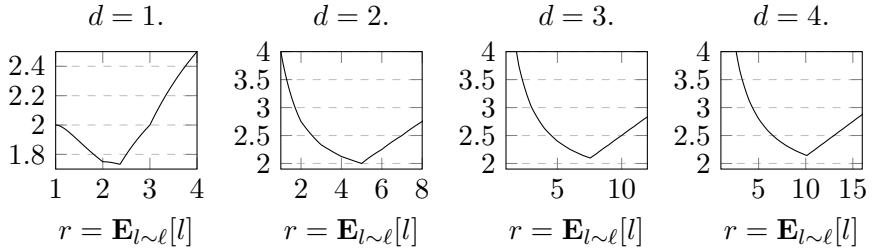


Figure 3.3: The competitive ratios of RANDOM RESET for $d = 1, 2, 3, 4$. Reset distribution ℓ is chosen optimal with given expected value $r = \mathbf{E}_{l\sim\ell}[l]$.

3.4. Summary of Techniques and Proof Sketches

3.4.1. Upper Bound

We are going to prove the upper bound on RANDOM RESET, Proposition 3.3. The proof allows us to showcase a simplified version of our analysis of TIMESTAMP(l,p). The result may also be of independent interest. Proposition 3.3 has, to the best of my knowledge, never been published, although it was known to the authors of [143].

Proof of Proposition 3.3. Observe that RANDOM RESET, or RR for short, is projective. Its behavior on any request only depends on the current counter value, but not on any prior request. In particular, by Proposition 3.1, we only need to analyze RANDOM RESET in the partial cost model on two-item-lists. The first item in the list can be served free of charge. The last item incurs service cost 1. Let σ be an input sequence and let $c = \max\left(1+\frac{R}{d}, 1+\frac{R+2\,d-1}{r}\right)$ be the desired competitive ratio. We have to show that $C_{\mathrm{RR}}(\sigma)$, the cost of RANDOM RESET, satisfies $\mathbf{E}[C_{\mathrm{RR}}(\sigma)] \leq cC_{\mathrm{OPT}}(\sigma)+\alpha$ with α independent of σ.

Phase partitioning: We partition input sequence σ into phases. The first phase starts with the first requests in σ. Whenever OPT exchanges the two items in its list, the current phase ends and a new one begins. The request before OPT's exchange is the last one of the current phase. The next request starts the new phase. Let $\lambda_1, \ldots, \lambda_k$ be the resulting phases. Then it suffices to show that $\frac{\mathbf{E}[C_{\mathrm{RR}}(\lambda_i)]}{C_{\mathrm{OPT}}(\lambda_i)} \leq c$ for every phase λ_i.

Consider now an arbitrary phase $\lambda = \lambda_i$ during which OPT does not rearrange its list. In OPT's list, let y denote the first item and let x be the last one. Then OPT incurs cost 1 for serving requests to x while serving requests to element y free of charge, at cost 0. Next to this, OPT pays exchange cost d for the exchange at the beginning of the phase. We also charge this cost in the first phase, which technically does not begin with an exchange. This overestimates OPT's cost on σ by d in total but, due to the constant α in the definition, does not affect the competitive ratio. The cost of OPT is thus d plus the number of requests to element x in phase λ.

Counter fixing for $\boldsymbol{x}$: We wish to evaluate the algorithms' cost C_{RR} and C_{OPT} on λ by only focusing on master requests. First, consider a request $\sigma(t)$ to x. The probability of $\sigma(t)$ being a master request is precisely $1/r$ where $r = \mathbf{E}_{l\sim\ell}[l]$. Let us switch to a *pre-refined cost scheme*. We charge cost r at any master requests to x and cost 0 else. Then the expected costs of OPT at any request to x remain $\frac{1}{r}\cdot r = 1$. Since RANDOM RESET moves item x to the front at a master request to x, it only incurs cost at minor requests. We thus only need to charge RANDOM RESET cost $r-1$ at master requests to x and cost 0 at minor requests to x. The expected costs of RANDOM RESET can only increase.

For item x, these new costs only depend on master requests. These master requests are fully determined by the random choices of RANDOM RESET, namely by the initial counter value of x at the beginning of σ and the reset values at each master request to x. Consider a possible way b these choices could have been made by RANDOM RESET and let $C^b_{\mathrm{OPT}}(\lambda)$ respectively $C^b_{\mathrm{RR}}(\lambda)$ denote the resulting pre-refined cost of OPT respectively RANDOM RESET. We previously argued that $\mathbf{E}_b[\mathbf{E}[C^b_{\mathrm{RR}}(\lambda)]] = \mathbf{E}[C_{\mathrm{RR}}(\lambda)]$ and $\mathbf{E}_b[C^b_{\mathrm{OPT}}(\lambda)] = C_{\mathrm{OPT}}(\lambda)$. Now, using Jensen's inequality

$$\frac{\mathbf{E}[C_{\mathrm{RR}}(\lambda)]}{C_{\mathrm{OPT}}(\lambda)} = \frac{\mathbf{E}_b[\mathbf{E}[C^b_{\mathrm{RR}}(\lambda)]]}{\mathbf{E}_b[C^b_{\mathrm{OPT}}(\lambda)]} \leq \mathbf{E}_b\left[\frac{\mathbf{E}[C^b_{\mathrm{RR}}(\lambda)]}{C^b_{\mathrm{OPT}}(\lambda)}\right] \leq \max_b \left[\frac{\mathbf{E}[C^b_{\mathrm{RR}}(\lambda)]}{C^b_{\mathrm{OPT}}(\lambda)}\right].$$

In the following, we thus consider a set of choices b where the maximum on the right is obtained. Once these choices b are fixed, all master requests to x become fixed, too. It suffices to show that $\frac{\mathbf{E}[C^b_{\mathrm{RR}}(\lambda)]}{C^b_{\mathrm{OPT}}(\lambda)} \leq c$ for these choices b.

Counter fixing for y: Unlike OPT, algorithm RANDOM RESET may incur cost at requests to y. These costs are only incurred precisely if the master request preceding y in λ was to x or if there was no preceding master request in λ. We will pay the cost incurred at y in advance.

Consider a master request X to x. Let Y be the next master request to y. We now charge the cost for all minor requests to y in between X and Y ahead of time. RANDOM RESET will already pay them at the master request X to x. This number of minor requests is precisely c_y, where c_y is the counter value of y at master request X. Instead of cost c_y, we may as well charge RANDOM RESET the expected cost $R = \mathbf{E}[c_y]$. In expectation, the charged cost stays the same. We similarly charge cost R at the beginning of the phase for minor requests to y that are not preceded by a master request X to x in phase λ. At these minor requests no further cost is incurred. Our *refined cost scheme* will thus charge no cost at requests to y but makes up for this by charging additional cost R at any master request to x and at the beginning of the phase.

Finally, we have to consider exchange costs of RANDOM RESET. We simply charge cost $2\,d$ at any master request to x for a possible exchange made here and another one made at the next master request to y. Moreover, we charge cost d at the beginning of the phase for a possible exchange made at the first master request to y, which might not be preceded by a master request to x. These refined costs are now fully independent of the counters of y, which is convenient. Most importantly, passing over to refined cost cannot reduces RANDOM RESET's cost in expectation.

Refined costs: Summarizing the preceding changes in the cost scheme, we obtain the following *refined cost scheme*:

- At the beginning of the phase, OPT incurs cost d while RANDOM RESET incurs cost $R + d$.

- At any master request to x, the optimum algorithm OPT pays cost r while RANDOM RESET incurs cost $r - 1 + R + 2\,d$.
- No costs are incurred at other requests.
- The counter values of x at the beginning of the phase and the reset value of the counter at any master request are chosen worst-possible. In particular, the sequence of master requests to x is fixed.

Recall that we argued that passing over to this cost scheme will not improve the ratio $\frac{\mathbf{E}[C^b_{\mathrm{RR}}(\lambda_i)]}{C^b_{\mathrm{OPT}}(\lambda_i)}$. Now, let K be the number of master requests to x in λ. Then we have shown that $\frac{\mathbf{E}[C_{\mathrm{RR}}(\lambda)]}{C_{\mathrm{OPT}}(\lambda)} \leq \frac{\mathbf{E}[C^b_{\mathrm{RR}}(\lambda)]}{C^b_{\mathrm{OPT}}(\lambda)}$ is at most

$$\frac{R + d + K \cdot (r - 1 + R + 2\,d)}{d + K \cdot r} \leq \max\Big(1 + \frac{R}{d}, 1 + \frac{R + 2\,d - 1}{r}\Big). \qquad \square$$

Analyzing TIMESTAMP(l,p) is much more difficult than analyzing RANDOM RESET. The main challenge is that TIMESTAMP's decisions on a given master request depend on preceding ones. This introduces quite a few special cases that need to be considered. Refined costs need to be charged at master requests to both x and y. Moreover, the behavior of TIMESTAMP(l,p) at a master request X to x depends on the counter value of y at this request. Thus, introducing *refined costs* becomes more challenging. We briefly sketch how to modify the proof of Proposition 3.3 to analyze TIMESTAMP(l,p).

Projectivity and phase partitioning: These steps work similar to the previous proof. Again, projectivity reduces us to two-item-lists while phases are subdivided according to the behavior of OPT. From now on, a single phase λ is considered.

Pre-refined cost: The goal is to shift all the costs of TIMESTAMP(l,p) to master requests. We move the costs of each request to the next master request. We need to be more careful with requests that are not followed by a master requests in λ. For this, *pre-refined costs* are introduced. Pre-refined costs charge for these requests in a roundabout manner. We rigorously guarantee that no cost is lost. It is crucial to ensure that we do not undercharge TIMESTAMP in expectation when passing over to pre-refined cost.

Counter fixing for x: Let y denote the item at the front of OPT's list in phase λ, while x is at the back. For item x, we shift the cost of OPT to master requests similar to the proof of Proposition 3.3.

Refined costs: These cost form the crux of the analysis. The costs of the online algorithm are changed such that they are fully determined by the sequence of master requests. Again, it is important that passing over to refined cost does not, in expectation, increase the cost of the online algorithm.

The definition of refined costs is quite subtle and involved. To illustrate this, we will discuss several options regarding the following previously encountered scenario. We are right before a master request X to x. Let

Y denote the next master request to y. We want to pay the cost for all minor requests to y before Y in advance. These costs are c_y where c_y is the counter of y at X. Previously, for RANDOM RESET, we would have simply charged cost $R = \mathbf{E}[c_y] = \frac{l-1}{2}$ at the master request X. A crucial feature of TIMESTAMP(l,p) is that it may keep item x at the back of its list if master request X is preceded by many master requests to y. It is important to capitalize on this and not charge costs if item x is kept at the back. In other words, always charging cost $\frac{l-1}{2}$ at master requests to x will not work.

Instead, we could try charging cost $\frac{l-1}{2}$ only if x is moved to the front of the list at X. This undercharges the TIMESTAMP. Consider the infinite sequence $(x^l y)^*$. Whenever item x is moved to the front at a given master request X to x the preceding request to y has been a master request. Thus, the counter c_y at X is $l-1$. Note that RANDOM RESET has the same problem. In the proof of Proposition 3.3, it is important to charge cost $\mathbf{E}[c_y]$ independent of whether item x has been moved to the front at this request.

The correct charging scheme is the following. We charge cost $\frac{l-1}{2}$ if item x is at the front of the list after serving master request X. This cost is also charged, if the item was already moved at an earlier master request. The counter c_y is still not independent of this event. It requires a careful analysis to see that the expected value of c_y conditioned on x being at the front after master request X is at most $\frac{l-1}{2}$. Formally, we establish a negative correlation between this event and the counter value c_y.

Counter fixing for y: After introducing the refined costs, we can fix the counter of y in a way maximizing the refined cost of TIMESTAMP(l,p). The refined costs only depend on the sequence of master requests. On this sequence TIMESTAMP(l,p) behaves like the original TIMESTAMP(p) from [1].

Patterns and Subphases: We now need to analyze TIMESTAMP(p) on the sequence of master requests using our refined costs. This requires a subdivision of the phase. First, a *pre-phase* is cut off. In this phase, OPT only pays exchange cost d. If the pre-phase is not *degenerated*, this results in a competitive ratio of c_1 from Theorem 3.1. The requests following the pre-phase form the *post-phase*.

The post-phase is further divided into two types of *subphases*. These types broadly follow the patterns $XX\cdots YY$ respectively $X\cdots YY$. We use capital letters to highlight the fact that these are master requests. Minor requests, do not matter at this point in the analysis. The "$\cdots$" may be filled with an arbitrary numbers of repetitions of the pattern YX. Zero repetitions are also allowed. Intuitively, the bound c_2 in Theorem 3.1 corresponds to $XX\cdots YY$, the bound c_3 in Theorem 3.1 corresponds to $X\cdots YY$ and the bound c_4 to the pattern YX inserted into "$\cdots$". We previously mentioned degenerated pre-phases. These are covered by the additional bounds c_5 and c_6 in Theorem 3.1. We only include them to make sure that our analysis covers

degenerate pre-phases, too. Technically, the maximum is always obtained at c_1 to c_4. Similarly, the $\max\{1, 2p\}$-term in the definition of c_1 could be replaced by 1. This requires more thorough case distinctions and a different definition of pre-refined cost. It also does not lead to better competitive ratios for $d \geq 7$. Getting rid of the $\max(1, 2p)$-term is interesting for small values of d and if one were to analyze a COUNTER-version COMB(l) of COMB from [20]. Algorithm COMB(l) is a combination of a COUNTER($2l$) and a TIMESTAMP(l,1) algorithm.

3.4.2. Lower Bound

The crux of the lower bound is the randomized sequence $\mathcal{S}_N$ on which all sensible online algorithms perform equally good. To sample an element from $\mathcal{S}_N$, one cyclically moves through the initial list L starting with the last element and proceeding to the front. Once the first element is reached, one again continues with the last one and repeats the process until N items have been considered in total. While cycling through L, an input sequence $\sigma \sim \mathcal{S}_N$ is sampled the following way: Initially σ is empty. When an item z from L is considered, i request to z are appended. The value i is drawn from a one-based geometric distribution with parameter $p = \frac{1}{2\,d}$, i.e. $\mathbf{P}[i = j] = p(1-p)^{j-1}$ for $j \geq 1$ and zero else.

We establish the following result, which can be easily generalized to lists containing more than two items.

Lemma 3.2
Consider two-item-lists in the partial cost model. If σ is sampled according to $\mathcal{S}_N$, the expected cost of any online algorithm on σ are at least dN.

Proof Sketch. Consider the following situation. An online algorithm encounters two segments $x^i y^j$ where x is at the back of its list and i, j are independently sampled from a one-based geometric distribution with parameter $\frac{1}{2\,d}$. We consider different strategies that could be applied by the online algorithm. On the one extreme, proactive online algorithm MOVE-TO-FRONT would move x to the front immediately paying cost d. On the other extreme, apathetic online algorithm STAY-BACK would wait for the segment x^i to pass by without additional rearrangements. Even if segment length i turns out to be large, STAY-BACK does not budge. STAY-BACK's expected cost on segment x^i are $\mathbf{E}[i] = 2\,d$. To make up for this, segment y^j can now be served free of charge. Similar to MOVE-TO-FRONT, algorithm STAY-BACK still pays cost d when averaged over two segments x^i and y^i.

A moderate online algorithm A may try to strike a balance between STAY-BACK and MOVE-TO-FRONT. After serving the first request to x without list rearrangement, algorithm A decides how to proceed on the remaining sequence. On the first request to x, an initial cost 1 is payed. Algorithm A's choice to wait would definitely be correct if this first request

was the only request to x. In other words, if $i = 1$. If $i = 1$, algorithm A saves cost $2\,d$ ignoring the initial cost 1. Since $i = 1$ with probability $\frac{1}{2\,d}$, algorithm A saves cost $\frac{1}{2\,d} \cdot 2\,d$ in expectation. This saving cancels out with the initial cost of 1 payed on the first request to x. On the other hand, if $i > 1$, we use that geometric distributions are forgetful: $i - 1$ conditioned on $i > 1$ is again geometrically distributed with parameter $\frac{1}{2\,d}$. Online algorithm A has not learned anything new and cannot achieve better cost on the remaining suffix than it could have achieved on the initial sequence $x^i y^j$. Thus, the moderate algorithm A cannot outperform the extreme strategy MOVE-TO-FRONT in expectation. Every strategy incurs, in expectation, cost dN.

This informal reasoning is easily made formal in the full paper, see Appendix A. □

In order to make the following results more intuitive, we consider the randomized STAY-BACK algorithm. At the beginning, STAY-BACK picks a uniform permutation of its starting list. We waive the constant costs required here, which can only improve STAY-BACK's performance. After the initial exchanges, STAY-BACK simply serves each request without further list rearrangements. The previous lemma implies that STAY-BACK is optimal on sequences sampled according to $\mathcal{S}_N$. We again restrict ourselves to two-item-lists for simplicity.

Corollary 3.2
Consider two-item-lists. After waiving the initial rearrangement cost, STAY-BACK pays (expected) cost 1/2 on each request and is an optimal online algorithm for sequences $\sigma \sim \mathcal{S}_N$.

Proof. Since STAY-BACK has every item at the front of its list with probability 1/2, it serves each request, no matter which of the two items is requested, with expected cost 1/2. Its total cost on any input σ accumulate to $\frac{|\sigma|}{2}$. The lemma follows by observing that $\mathbf{E}_{\sigma \sim \mathcal{S}_N}[|\sigma|] = 2\,d \cdot N$. The expected cost of STAY-BACK are thus dN. By Lemma 3.2, no online algorithm surpasses this. □

Now, proving Theorem 3.2 and Theorem 3.3 comes down to analyzing OPT on the randomized sequence $\mathcal{S}_N$ for $N \to \infty$.

Proof sketch of Theorem 3.2. We focus on $d \to \infty$ so that we can ignore terms that vanish in the limit. The main paper in Appendix A considers general integer values d. It suffices to evaluate OPT on sequences σ sampled from $\mathcal{S}_N$ with regards to two-item-lists in the partial cost model. For this, we compare OPT against the STAY-BACK.

The optimal algorithm OPT, as defined in Section 3.3, subdivides input sequence σ into up-phases, zero-phases, post-phases and possibly one final

phase. We can show that STAY-BACK and OPT incur the same (expected) cost on zero-phases and up-phases. Since there is at most one final phase, the advantage that OPT gains here over STAY-BACK is irrelevant. We are left to consider post-phases. The length of each post-phase is geometrically distributed with parameter $p = \frac{1}{2\,d}$. This geometric distribution is zero-based and thus the expected length of each post-phase is $2\,d - 1$. Each post-phase is served by OPT free of charge and while STAY-BACK pays expected cost $\frac{2\,d-1}{2} \approx d$. The number of post-phases can be determined by a Markov analysis. In the main paper, we implicitly show that σ contains in expectation $\frac{N}{2} - o(N)$ post-phases. Thus, for $d \to \infty$ and $N \to \infty$ the expected competitive ratio of STAY-BACK on $\sigma \sim \mathcal{S}_N$ is at most $\frac{dN}{dN - dN/2} = 2$.

The main paper requires more notation to make these arguments formal. It also introduces a simplified variant of OPT that omits the final phase. All computations in the paper work for general values of d, not only for $d \to \infty$. For general d, the lower bound of $2 - \frac{1}{2\,d}$ is obtained. □

Proof sketch of Theorem 3.3. Lower bounds in the full-cost model are challenging since one has to consider general sized lists. In the full cost model, algorithm STAY-BACK for example is 1.5-competitive on two-item-lists. We need to consider lists L of general size n. On arbitrary lists L, it is extremely challenging to understand the behavior of OPT on $\sigma \sim \mathcal{S}_N$. Instead, we consider the simplified offline algorithm B_h, which only moves the current element to the front if it is requested for the next h times. The number h is a variable, which we optimize later. For offline algorithm B_h, the relative order of two items only depends on prior requests to these items. This resembles the projective property of online algorithms. The same techniques as in Proposition 3.1 can be used to reduce the analysis of B_h to two-item-lists in the partial cost model. This reduction holds without loss of generality. On two-item-lists, a Markov analysis shows that the cost of algorithm B_h are $k(h)N - o(N)$ for $k(h) = \frac{\frac{1-(1-p)^h}{p} - h(1-p)^{h-1} + (1-p)^{h-1} d}{2-(1-p)^{h-1}}$ where $p = \frac{1}{2\,d}$. By Lemma 3.2, this establishes a lower bound of $\frac{2\,d}{k(h)}$ on the competitive ratio of any online algorithm. Theorem 3.3 follows for $d \to \infty$ by setting $h = W(-1/2e)\,2\,d$. □

3.5. Open Problems

The following are prime open problems for List Update, which are highly interesting even in the standard model.

Conjecture 3.1
If an online algorithm is c-competitive in the full-cost model, then there exists a, possibly different, online algorithm that is c-competitive in the

partial cost model.

Note that this conjecture is obviously untrue when we consider small list sizes. The previously introduced STAY-BACK algorithm is 1.5-competitive when restricted to two-item-lists in the full cost model. But STAY-BACK will not have any constant competitive ratio in the partial cost model. The advantage of the full cost model seems to vanish as the list size n increases. This is why I conjecture it not to matter in the grand scope of arbitrary sized lists.

Proving or disproving the following hypothesis is the major open problem concerning List Update algorithms. Since non-projective algorithms are poorly understood, I cannot fathom whether it is false or true.

Hypothesis 3.1 (Projectivity-Hypothesis)
If an online algorithm is c-competitive in the full-cost model, then there exists a projective online algorithm that is c-competitive in the partial cost model.

We now proceed with special conjectures for the P^d model. Consider our randomized lower bound sequence $\mathcal{S}_N$ for general size n lists.

Conjecture 3.2
There holds $\mathbf{E}_{\sigma\sim\mathcal{S}_N}[\mathrm{OPT}(\sigma)] = \frac{n(n-1)d^2}{4d-1}N + o(N)$, where n denotes the size of list L.

Using a generalization of Lemma 3.2 to general list sizes n, this conjecture implies the following.

Conjecture 3.3
No algorithm for List Update in the full cost P^d model is better than $(2 - \frac{1}{2d})$-competitive.

Naturally, the previous conjecture would also be a consequence of our lower bound and the more general Conjecture 3.1.

The work of Ambühl et al. [25] develops advanced techniques to prove lower bounds for projective algorithms in the standard model. These techniques can be generalized to the P^d model. It is easy to recreate a lower bound of $2-1/d$, a weaker statement than Theorem 3.2, using the techniques of [25]. Their techniques though, allow further improvement. It would be particularly interesting to see if these techniques can lead to lower bounds exceeding 2.

Problem 3.1
Do the techniques of Ambühl et al. [25] allow for lower bounds exceeding 2 for d large enough?

For $d \to \infty$ our results establish a lower bound of 2 in the full cost model, a bound of 1.8654 in the partial cost model and an upper bound of 2.2442 in

both models. A final obvious open problem would be to close these gaps.

Problem 3.2
Tighten the bounds on the competitive ratio, in particular for $d \to \infty$.

I conclude with a simple observation that has been puzzling me. If we compare deterministic and the randomized COUNTER(l) algorithms for $d \to \infty$, the latter has half the competitive ratio of the former cf. [21, 143]. From my own analyses, I know that the same holds for the TIMESTAMP(l,1) family. Namely for $d \to \infty$ and optimum choice of l, deterministic TIMESTAMP(l,1) is 5-competitive where as Theorem 3.1 implies that randomized TIMESTAMP(l,1) is 2.5-competitive. To date, I could not find a simple and convincing explanation why these competitive ratios halve when counter values are randomly initialized.

Chapter 4

Makespan Minimization in the Random-Order Model

4.1. Preliminaries

Problem Definition

We study Online Makespan Minimization in the random-order model. A set of n *jobs* $\mathcal{J} = \{J_1, \ldots, J_n\}$, where each job J_i is fully defined by its *processing time* p_i, has to be assigned to m parallel and identical machines. Each job runs on precisely one machine. In particular, preemption is not allowed. Given such an assignment, the *load* l_M of machine M is the sum of all processing times of jobs assigned to it. The goal is to minimize the *makespan*, that is the time it takes to process all jobs or, formally, $\max(l_M)$, the maximum load of a machine.

To an online algorithm A, jobs arrive one by one and each has to be assigned permanently and irrevocably before the next one is revealed. Input size n is not known in advance. One typically measures online algorithm A's performance in terms of competitive analysis. Each element σ of S_n, the permutation group of the integers 1 to n, describes a way to order the elements in $\mathcal{J}$ resulting in job sequence $\mathcal{J}^\sigma = J_{\sigma(1)}, \ldots, J_{\sigma(n)}$. The makespan of online algorithm A, denoted by $A(\mathcal{J}^\sigma)$, depends on both job set $\mathcal{J}$ and job order σ. The optimum makespan $\mathrm{OPT}(\mathcal{J})$ only depends on the former. Traditionally, the goal is to minimize the *adversarial competitive ratio* $c = \sup_{\mathcal{J},\sigma} \frac{A(\mathcal{J}^\sigma)}{\mathrm{OPT}(\mathcal{J})}$. This classic performance measure considers worst-case sequences. In the literature, these are highly ordered. They arrange jobs by increasing processing times. Our results imply that general worst-case sequences are extremely unlikely after random input permutation.

To accommodate less order-sensitive applications, the *random-order model* was introduced. One still considers worst-case input sets $\mathcal{J}$ but job order σ is chosen uniformly at random. Each possible job order $\sigma \in S_n$ is picked with probability $\frac{1}{n!}$. This turns the set of job orders S_n into a Laplace space. The *competitive ratio (in the random-order model)* of an online algorithm A is then $c = \sup_{\mathcal{J}} \mathbf{E}_{\sigma \sim S_n}\left[\frac{A(\mathcal{J}^\sigma)}{\mathrm{OPT}(\mathcal{J})}\right] = \sup_{\mathcal{J}} \frac{1}{n!} \sum_{\sigma \in S_n} \frac{A(\mathcal{J}^\sigma)}{\mathrm{OPT}(\mathcal{J})}$. While worst-case orders are often criticized for being highly pessimistic, random orders may be considered rather optimistic. We thus propose the *nearly competitive ratio*, which considers nearly worst-case orders on large numbers of machines.

Only a vanishing fraction of particularly bad orders is omitted.

An online algorithm A is *nearly c-competitive* if it has a constant adversarial competitive ratio and $\lim_{m\to\infty} \mathbf{P}_{\sigma\sim S_n}[\frac{A(\mathcal{J}^\sigma)}{\mathrm{OPT}(\mathcal{J})} > c+\varepsilon] = 0$ for every $\varepsilon > 0$. In other words, for every $\varepsilon > 0$, there exists a number of machines m_0 such that on $m \geq m_0$ machines $\mathbf{P}_{\sigma\sim S_n}[\frac{A(\mathcal{J}^\sigma)}{\mathrm{OPT}(\mathcal{J})} > c+\varepsilon] < \varepsilon$. This is a much stronger condition than the random-order competitive ratio. In particular, the following holds.

Proposition 4.1
If an online algorithm A is nearly c-competitive, then it is c-competitive in the random-order model for $m \to \infty$, i.e. if its competitive ratio in the random-order model on m machines is c_m, then $\lim_{m\to\infty} c_m \leq c$.

In this chapter, we assume that online algorithms do not know the number of jobs n in advance. This information cannot help improve the adversarial competitive ratio but allows vastly more powerful results in the random-order model. The random-order model where n is known in advance is called the *secretary model*. Makespan Minimization in the secretary model is discussed in Chapter 5.

4.2. Our contributions

We provide a new online algorithm ALG, which is nearly c-competitive. Here, c is the unique real root of the polynomial $Q[x] = 4x^3 - 14x^2 + 16x - 7$,

$$c = \frac{7 + \sqrt[3]{28-3\sqrt{87}} + \sqrt[3]{28+3\sqrt{87}}}{6} < 1.8478.$$

This result is significant. Even for $m = 24$ machines, no adversarial competitive ratio below 1.856 is possible. For $m \to \infty$, the best possible competitive ratio cannot lie below 1.88 [145].

Consider an ordered input sequence $\mathcal{J}^\sigma = J_{\sigma(1)}, J_{\sigma(2)}, \ldots, J_{\sigma(n)}$. After online algorithm ALG has assigned the first t jobs $J_{\sigma(1)} \ldots, J_{\sigma(t)}$ for some $0 \leq t \leq n$, we denote the m machines by $M_1^t, \ldots, M_m^t$. The ordering is chosen such that machine loads are non-increasing. In other words, if l_j^t denotes the load of machine M_j^t at time t, then $l_1^t \geq \ldots \geq l_m^t$. In particular, l_1^t is the makespan of the current schedule.

In order to assign an incoming job $J_{\sigma(t)}$, algorithm ALG considers up three machines: the least loaded machine M_m^{t-1}, an almost least loaded machine M_{m-h}^{t-1} and a medium loaded machine M_i^{t-1}. There is some freedom in the choice of parameter $h = h(m)$. We only require that $h(m) \in \omega(1)$ and $h(m) \in o(\sqrt{m})$; reasonable choices are $h(m) = \lfloor \sqrt[3]{m} \rfloor$, $h(m) = \lfloor \sqrt[4]{m} \rfloor$ or $h(m) = \lfloor \log m \rfloor$. We set $i = \lceil (2c-3)m \rceil + h \approx 0.6956m$ for the index of the medium machine M_i^{t-1}.

Assigning job J_t to one of the machines M_i^{t-1} or M_{m-h}^{t-1} risks creating a particularly high makespan. This would be hazardous. Whenever online

algorithm ALG uses any machine besides the least loaded one, it has to ensure that its makespan is at most $c\,\mathrm{OPT}(\mathcal{J})$. This requires an estimate $O^t \leq \mathrm{OPT}$, which only depends on jobs $J_{\sigma(1)} \ldots, J_{\sigma(t)}$. Previous algorithms estimate the optimum makespan $\mathrm{OPT}(\mathcal{J})$ by the average load L^t of the current schedule, a well-known lower bound. Online algorithm ALG is the first approach that requires a more sophisticated lower bound. Its bound O^t incorporates, in addition to L^t, the currently largest processing time $p^t_{\max}$ and $2P^t_{m+1}$, twice the processing time of the $(m+1)$th largest job seen so far. These are well known lower bounds but do not allow improved performance guarantees when combined with prior approaches [2, 40, 84, 89, 112].

Formally, for $j = 1, \ldots, m$, let L^t_j be the *average load* of the $m - j + 1$ least loaded machines $M^t_j, \ldots, M^t_m$, i.e. $L^t_j = \frac{1}{m-j+1} \sum_{r=j}^{m} l^t_r$. In particular, we let $L^t = L^t_1 = \frac{1}{m} \sum_{r=1}^{t} p_r$ be the average load of all the machines. Let P^t_j, for $1 \leq j \leq n$, be the processing time of the jth largest job among the first t jobs $J_{\sigma(1)}, \ldots, J_{\sigma(t)}$ in $\mathcal{J}^\sigma$. If $t < j$, we set $P^t_j = 0$. We let $p^t_{\max} = P^t_1$ be the processing time of the largest job among the first t jobs in $\mathcal{J}^\sigma$. Finally, let $L = L^n$, $P_j = P^n_j$ and $p_{\max} = p^n_{\max}$.

Proposition 4.2
$OPT(\mathcal{J}) \geq \max\{L, p_{\max}, 2P_{m+1}\}$ for any input $\mathcal{J}$.

Proof. Even the optimum algorithm cannot have all machine loads below average. Thus, $L \leq \mathrm{OPT}(\mathcal{J})$. Neither can OPT avoid scheduling job $J_{\max}$ of processing time $p_{\max}$ on some machine. Its load will be at least $p_{\max}$. Thus, $p_{\max} \leq \mathrm{OPT}(\mathcal{J})$. By the pigeonhole principle, one machine holds two of the $m + 1$ largest jobs. Its load is at least $2P_{m+1}$. Thus, $2P_{m+1} \leq \mathrm{OPT}$. □

Hence, $O^t = \max\{L^t, p^t_{\max}, 2P^t_{m+1}\} \leq \mathrm{OPT}(J_{\sigma(1)}, \ldots, J_{\sigma(t)}) \leq \mathrm{OPT}(\mathcal{J})$ provides a lower bound for $\mathrm{OPT}(\mathcal{J})$ that ALG can use to schedule job $J_{\sigma(t)}$. Note that online algorithm ALG can compute $L^t = \frac{1}{m} \sum_{r=1}^{t} p_r$ and hence O^t right before job $J_{\sigma(t)}$ is scheduled.

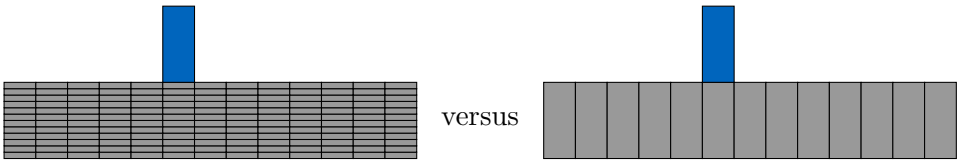


Figure 4.1: The lower bound for Graham's Greedy strategy [94] (on the left) presents many tiny jobs that create an extremely flat schedule. A follow-up large job $J_{n'}$ creates a large makespan. OPT may "sink down" job $J_{n'}$ to schedule it efficiently. It is hence paramount to avoid flat schedules. On the right, another instance shows that flat schedules are sometimes unavoidable. For m jobs of equal processing time, no sensible online algorithm can avoid a flat schedule. Fortunately, even OPT cannot handle a follow-up job $J_{n'}$ that uses the flat schedule to create a high makespan.

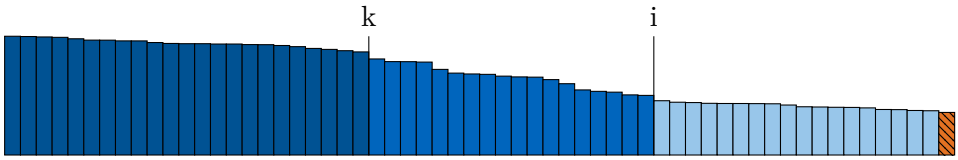


Figure 4.2: A steep schedule. ALG only considers the least loaded machine.

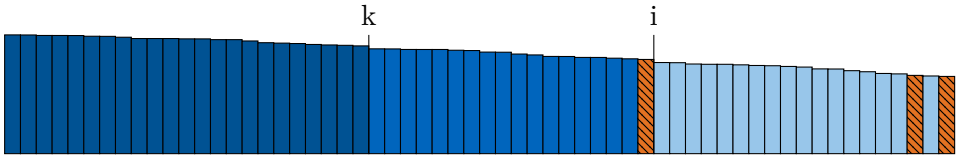


Figure 4.3: A flat schedule. The three machines considered by ALG are marked for $h = 2$.

The left of Figure 4.1 depicts the main deficiency of the Greedy-Strategy, which always uses a least loaded machine: very flat schedules. Lower bounds exploit such schedules by following up with a single large job $J_{n'}$ that causes a particularly high makespan but may be effectively handled by the offline algorithm. To avoid this, Algorithm ALG follows the tried and tested approach of creating steep schedules and avoiding flat ones. Such avoidance is impossible on particularly difficult inputs, cf. the right side of Figure 4.1. The main part of the analysis establishes that inputs exploiting very flat schedules of ALG are inherently difficult. They contain $m + 1$ big jobs that cannot be handled effectively. A high offline makespan thus ameliorates the high online makespan of ALG.

Formally, consider the *schedule at time* t, which is the schedule just before job $J_{\sigma(t)}$ has to be assigned. The schedule is *flat* if $l_k^{t-1} < \alpha L_{i+1}^{t-1}$ where $\alpha = (1 - \frac{1}{2(c-1)})^{-1} \approx 2.7376$ and $k = 2i - m$. It is *steep* otherwise. Job $J_{\sigma(t)}$ is scheduled *flatly* (*steeply*) if the schedule at time t is flat (steep).

A steep schedule, on the one hand, allows ALG to safely use the least loaded machine. It will indeed simply do so. For a flat schedule on the other hand, algorithm ALG desires to use a higher loaded machine in order to make the schedule steep again. For this purpose, algorithm ALG first samples medium machine M_i^{t-1}. If the resulting makespan is at most $c\,O^t$, formally $l_i^{t-1} + p_{\sigma(t)} \leq c\,O^t$, medium machine M_i^{t-1} is used. Else, the close to least loaded machine M_{m-h}^{t-1} is sampled. Machine M_{m-h}^{t-1} is, again, used if the resulting makespan is at most $c\,O^t$, or formally $l_{m-h}^{t-1} + p_{\sigma(t)} \leq c\,O^t$. Else, algorithm ALG shelves its ambition for a steep schedule and makes due with the least loaded machine M_m^{t-1}. It is non-obvious that even least loaded machine M_m^{t-1} is safe to use. In fact, establishing this for our choice of c comprises the core of the analysis. The job assignment rules are also illustrated in Figures 4.2 and 4.3.

Algorithm 2 How online algorithm ALG schedules current job $J_{\sigma(t)}$.

if the schedule at time t is steep **then**
    Assign job $J_{\sigma(t)}$ to the least loaded machine M_m^{t-1};
else // the schedule is flat
    if $l_i^{t-1} + p_{\sigma(t)} \leq c\,O^t$ **then** Assign job $J_{\sigma(t)}$ to M_i^{t-1};
    else if $l_{m-h}^{t-1} + p_{\sigma(t)} \leq c\,O^t$ **then** Assign job $J_{\sigma(t)}$ to M_{m-h}^{t-1};
    else Assign job $J_{\sigma(t)}$ to M_m^{t-1};

Our main result is Theorem 4.1, which establishes the nearly competitive ratio of ALG.

Theorem 4.1
ALG is nearly c-competitive with $c < 1.8478$ defined as above.

Proposition 4.1 implies Corollary 4.1, which gives competitive guarantees in the random-order model.

Corollary 4.1
ALG is c-competitive in the random-order model for $m \to \infty$.

4.2.1. Lower Bounds

We also provide first deterministic lower bounds for Online Makespan Minimization in the random-order model.

Theorem 4.2
Let A be a deterministic online algorithm that is c-competitive in the random-order model. Then $c \geq 4/3$ if $m \geq 8$.

Theorem 4.3
Let A be a deterministic online algorithm that is nearly c-competitive. Then $c \geq 3/2$.

These lower bounds make use of the fact that suitably few jobs of equal processing time have to be scheduled flatly in order to maintain the competitive ratio 4/3 respectively 3/2. A single large job then creates a high makespan. In the following, we focus on the upper bound in Theorem 4.1, which is our main contribution.

4.3. Summary of Techniques and Proof Sketches

The analysis follows three steps. First, *simple* job sets are introduced. Simple job sets are easy for the algorithm but defy our general analysis. Think of a one-job-set. While hardly problematic to any online algorithm there is only one possible input arrangement. This problem still persists should we add a second and third job; or many jobs of processing time so small that they are

Figure 4.4: Let $\varepsilon > 0$. Online algorithm ALG is $(c+\varepsilon)$-competitive on simple and stable sequences and 2-competitive on the problematic unstable remainder (hashed). Dashes lines mark orbits of input sequences under permutation. At most a vanishing fraction of each orbit is neither simple nor stable. Therefore, online algorithm ALG is nearly c-competitive.

negligible. In general, we need to exclude sets where few jobs carry almost all the load. Establishing that ALG is c-competitive on simple job sets will be easy but necessary. Focusing from now on non-simple inputs, we introduce *stable* sequences. These sequences arise with almost certainty after random permutation. Stable job sequences require problematic jobs to be evenly distributed. The $m+1$ largest jobs are of particular concern. These jobs should not be clustered towards the end as was common in prior adversarial worst-case sequences; neither should the h largest jobs hide at the end of the sequence. We prove that non-stable sequences are extremely unlikely for $m \to \infty$. For the nearly-competitive ratio, such unlikely sequences are negligible. Thus, we may focus on stable sequences. A third step poses the main part of the analysis. An intricate adversarial analysis shows that algorithm ALG is $(c+\varepsilon)$-competitive on stable sequences.

Simple Job Sets

Figure 4.5: A surprisingly problematic input set $\mathcal{J}$ for random-order arguments. A gigantic job carries almost all the processing volume; other jobs are negligible. All permutations of $\mathcal{J}$ look basically the same. Such "simple" inputs need to be excluded before undertaking a random-order analysis.

For job set $\mathcal{J}$, let $R[\mathcal{J}] = \min(1, \frac{L[\mathcal{J}]}{p_{\max}[\mathcal{J}]})$. Recall that $p_{\max}[\mathcal{J}]$ denotes the largest processing time in $\mathcal{J}$. In particular, $L[\mathcal{J}] \leq R[\mathcal{J}] \max(L[\mathcal{J}], p_{\max}[\mathcal{J}])$ and thus $L[\mathcal{J}] \leq R[\mathcal{J}]\,\mathrm{OPT}(\mathcal{J})$. We call job set $\mathcal{J}$ *simple* if it either contains at most m jobs or if $R[\mathcal{J}] \leq (c-1)$. Simple job sets are easy to schedule.

Lemma 4.1
Algorithm ALG is 2-competitive in the adversarial model. Moreover, if job set $\mathcal{J}$ is simple $\mathrm{ALG}(\mathcal{J}^\sigma) \leq c\,\mathrm{OPT}(\mathcal{J})$ for every job order σ.

Proof. The proof relies on the ancient arguments that Graham uses to analyze his Greedy strategy. Consider any input set $\mathcal{J}$. First observe that ALG only

considers a machine other than the least loaded machine if the resulting load is less than $c\,O^t \leq c\,\mathrm{OPT}$. If job J^t of processing time p_t is scheduled on the least loaded machine M_m^{t-1}, the resulting load becomes $l_m^{t-1} + p_t$. Further observe that $p_t \leq p_{\max} \leq \mathrm{OPT}$, cf. Proposition 4.2, and that the smallest machine load cannot exceed the average load, i.e. $l_m^{t-1} \leq L^{t-1} \leq L$. The first statement of the lemma, $l_m^{t-1} + p_t \leq 2\,\mathrm{OPT}$, follows from $L \leq \mathrm{OPT}$. If input set $\mathcal{J}$ is simple, either $L \leq (c-1)\mathrm{OPT}$ or there are at most m jobs, in which case $l_m^{t-1} = 0$ for $t = 1, \ldots, m$. Either way, $l_m^{t-1} + p_t \leq c\,\mathrm{OPT}$. The second statement of the lemma follows. □

4.3.1. Stable Sequences

Intuitively, random-order sequences have no concentrations of problematic jobs. Formally, they are *stable* with high probability. The properties of stable sequences satisfies two needs: They are useful in the analysis and they are extremely likely after random permutation.

The Load Lemma

One crucial result for establishing stable job sequences is the Load Lemma, which relates two time measures. Our probabilistic arguments, on the one hand, depend on the fraction $\frac{t}{n}$ of jobs currently scheduled. The arguments used to analyze our algorithm, on the other hand, only rely on the fraction $\frac{L^t}{L^n}$ of currently assigned processing volume. The left side of Figure 4.6 indicates that these measures are essentially unrelated on worst-case sequences that order jobs by increasing processing time. The Load Lemma establishes the opposite for random-order sequences.

For any time $0 \ll t \leq n$ we have $\mathbf{E}\left[\frac{L^t}{L}\right] = \frac{t}{n}$. The Load Lemma states that, in fact, $\frac{L^t}{L} \approx \frac{t}{n}$ with probability $1 - o(1)$. Here, $o(1)$ denotes a function vanishing for $m \to \infty$ while $\approx$ hides a small margin of error ε where $\varepsilon > 0$ can be chosen arbitrarily small. In the sequence in Figure 4.5 such a result does not hold. The Load Lemma applicable after we to exclude simple sequences beforehand. In the following, we state a more advanced and more general result from our next paper found in Appendix C.

Lemma 4.2 (Load Lemma)
Let $R_{\mathrm{low}} = R_{\mathrm{low}}(m) > 0$, $1 \geq \varphi = \varphi(m) > 0$ and $\varepsilon = \varepsilon(m) > 0$ be three functions in m such that $\varepsilon^{-4}\varphi^{-1}R_{\mathrm{low}}^{-1} = o(m)$. Then there exists a variable $m(R_{\mathrm{low}}, \varphi, \varepsilon)$ depending on these three functions such that we have for all $m \geq m(R_{\mathrm{low}}, \varphi, \varepsilon)$ and all job sets $\mathcal{J}$ with $R(\mathcal{J}) \geq R_{\mathrm{low}}$ and $|\mathcal{J}| \geq m$:

$$\mathbf{P}_{\sigma \sim S_n}\left[\left|\frac{L^{\varphi t}[\mathcal{J}^\sigma]}{L[\mathcal{J}]} - \varphi\right| \geq \varepsilon\right] < \varepsilon.$$

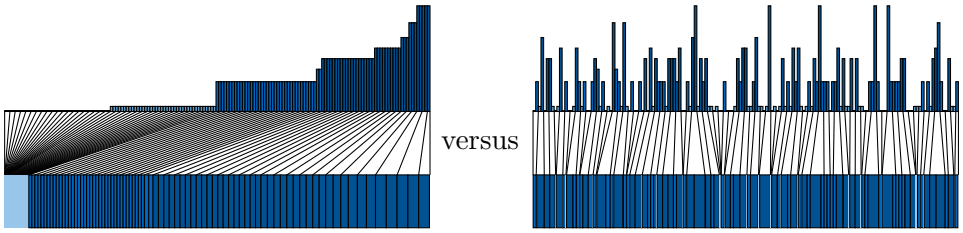


Figure 4.6: Comparing $\frac{t}{n}$ and $\frac{L^t}{L^n}$ on Albers's lower bound sequence [2] in the original worst-case order (left) and a random order (right). On the top, processing volume is indicated by the height of a job. On the bottom, it is indicated by the width. In the worst-case order, $\frac{t}{n}$ and $\frac{L^t}{L^n}$ clearly do not agree. In the random order, they agree up to some margin of error. For larger values of m this margin decreases, cf. Figure 4.7and 5.1.

In this chapter, we are only interested in the case where R_{low}, φ and ε are constants. In our case, we set $R_{\text{low}} = c - 1$, which implies that the Load Lemma holds unless job set $\mathcal{J}$ is simple. Figure 4.7 shows how the function $F_{\mathcal{J},\sigma}(\varphi) = L^{\varphi t}[\mathcal{J}^\sigma]$ approaches the line $G_{\mathcal{J}}(\varphi) = \varphi L[\mathcal{J}]$ for three random input permutations σ of the lower bound sequence from [2].

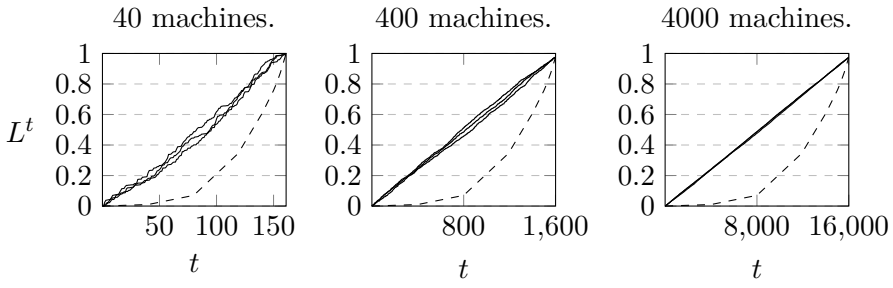


Figure 4.7: The average load L^t over time on the classical lower bound sequence from [2], depicted in Figure 4.6, for 40, 400 and 4000 machines. The dashed line shows the original adversarial order. The three solid lines correspond to random orders. They approximate a straight line.

Formal definition

We provide the formal definition of stable sequences and an informal description why these properties are extremely likely under random-order arrival. The definition relies on some value $\varepsilon > 0$, which can be picked arbitrarily small.

Definition 4.1. An (ordered) input sequence $\mathcal{J}^\sigma = J_{\sigma(1)}, \ldots, J_{\sigma(n)}$ is *stable* if set $\mathcal{J}$ is not simple and the following conditions hold:

- Once $L^t \geq (c-1)\frac{i}{m}L$, there holds $p^t_{\max} \geq P_h$.

- For every $j \geq i$, the sequence ending once we have $L^t \geq (\frac{j}{m} + \frac{\varepsilon}{2})L$ contains at least $j + h + 2$ many of the $m + 1$ largest jobs in $\mathcal{J}$.
- Consider the sequence ending right before either (a) $L^t \geq \frac{i}{m}(c-1)\varepsilon L$ holds or (b) one of the h largest job of $\mathcal{J}$ is scheduled. This sequence contains at least $h + 1$ many of the $m + 1$ largest jobs in $\mathcal{J}$.

Given $\varepsilon > 0$ and m, let $P_\varepsilon(m)$ be the minimum probability with which the permutation of a non-simple job set is stable. Formally

$$P_\varepsilon(m) = \inf_{\mathcal{J} \text{ not simple}} \mathbf{P}_{\sigma \sim S_n}[\mathcal{J}^\sigma \text{ is stable}].$$

The main result of this section shows that $P_\varepsilon(m)$ tends to 1 as $m \to \infty$.

Main Lemma 4.1
For every $\varepsilon > 0$, there holds $\lim_{m\to\infty} P_\varepsilon(m) = 1$.

Proof Sketch. Consider a non-simple job set $\mathcal{J}$. Since $R[\mathcal{J}] \geq c - 1$, we can use the Load Lemma to conclude that $\mathbf{P}_{\sigma\sim S_n}\left[\left|\frac{L^{\varphi t}[\mathcal{J}^\sigma]}{L[\mathcal{J}]} - \varphi\right| \geq \varepsilon\right] < \varepsilon$ for any constant choice ε, φ if we choose m large enough. We use the informal, sketchy term $\gtrsim$ for inequalities that hold up to an ε-margin of error excluding an $o(1)$-fraction of input permutations. Then $L^t \geq \varphi L$ implies $t \gtrsim \varphi n$ by the Load Lemma. The analyses in our main paper provide, of course, a more rigorous treatment.

The first property of stable job sequences states that one of the h largest jobs arrives before $L^t \geq (c-1)\frac{i}{m}L$. Using the Load Lemma, this implies $t \gtrsim (c-1)\frac{i}{m}n$. The probability of a single large job independently arriving before time $(c-1)\frac{i}{m}n$ is at least $1 - (c-1)\frac{i}{m} > 1 - 0.59$. For h jobs, it is at least $1 - 0.59^h$, which quickly approaches 1 since $h = h(m) \in \omega(1)$.

For the second property, note that before $L^t \geq (\frac{j}{m} + \frac{\varepsilon}{2})L$ or, equivalently, before $t \gtrsim (\frac{j}{m} + \frac{\varepsilon}{2})n$, at least $\gtrsim (\frac{j}{m} + \frac{\varepsilon}{2})(m+1)$ of the $m + 1$ largest jobs arrived. Since $\frac{\varepsilon}{2}(m+1)$ grows faster than $h(m) + 2$ and the small margin of error hidden in the $\gtrsim$-term, $\frac{\varepsilon}{2}(m+1)$ exceeds $h + 2$ for m large enough. The second property follows.

The reasoning for part a) of the third property mirrors the reasoning for the second property only that not only one but $h + 1$ of the $m + 1$ largest jobs are considered. For part b) we have to observe that $\gtrsim \frac{m}{h+1}$ of the $m + 1$ largest jobs arrive before one of the h largest jobs arrives. Since $h \in o(\sqrt{m})$, the term $\frac{m}{h+1}$ grows asymptotically faster than $h + 1$ and the result follows for m large enough. □

The main lemma implies that we only have to consider stable sequences from now on.

Corollary 4.2
If online algorithm ALG is $(c + \varepsilon)$-competitive on stable sequences for every ε, then it is already nearly c-competitive.

Proof. Consider an input set $\mathcal{J}$. If $\mathcal{J}$ is simple, $A(\mathcal{J}^\sigma) \leq c\,\mathrm{OPT}(\mathcal{J})$ for all $\sigma \in S_n$ by Lemma 4.1. Else, the condition of the lemma implies that $\mathbf{P}[A(\mathcal{J}^\sigma) \leq (c+\varepsilon)\mathrm{OPT}(\mathcal{J})] \leq \mathbf{P}_\varepsilon(m)$. By Main Lemma 4.1, there exists m_0 such that $\mathbf{P}_\varepsilon(m) \leq \varepsilon$ for all $m \geq m_0$. Since ALG has a constant adversarial competitive ratio of 2, cf. Lemma 4.1, the corollary follows. □

4.3.2. The main analysis

Consider any stable input sequence $\mathcal{J}^\sigma = J_{\sigma(1)}, \ldots, J_{\sigma(n)}$. By Corollary 4.2, it suffices to show that $\mathrm{ALG}(\mathcal{J}^\sigma) \leq (c+\varepsilon)\mathrm{OPT}(\mathcal{J})$. Let us for contradiction sake assume that this was not the case. For simplicity, we will without loss of generality assume that $\mathcal{J}^\sigma = J_1, \ldots, J_n$. We furthermore simply write OPT for $\mathrm{OPT}(\mathcal{J})$.

Let $J_{n'}$ be the first job that caused $\mathrm{ALG}(\mathcal{J}^\sigma)$ to exceed $(c+\varepsilon)$OPT. Online algorithm ALG must have scheduled job $J_{n'}$ on a least loaded machine, else the resulting load would be at most $c\,O^{n'} \leq c\,\mathrm{OPT}$. Let $b_0 = l_m^{n'-1}$ be the load of this least loaded machine $M_m^{n'-1}$ right before receiving job $J_{n'}$. Since $b_0 + p_{n'} > (c+\varepsilon)\mathrm{OPT}$ and $p_{n'} \leq \mathrm{OPT}$ and $b_0 \leq L^{n'-1} \leq \mathrm{OPT}$, there holds $b_0 \geq (c-1+\varepsilon)\mathrm{OPT}$ and $p_{n'} \geq (c-1+\varepsilon)\mathrm{OPT}$. Let $\lambda_{\mathrm{end}} = \frac{1}{2(c-1+\varepsilon)} \approx 0.5898$. Since $(c+\varepsilon)\mathrm{OPT} < b_0 + p_{n'} \leq (1+\frac{1}{c-1+\varepsilon})b_0 = (c+\varepsilon)\frac{b_0}{c-1+\varepsilon} = (c+\varepsilon)2\,\lambda_{\mathrm{end}} b_0$, we get $\mathrm{OPT} < 2\,\lambda_{\mathrm{end}} b_0$ and hence

$$P_{m+1} \leq \mathrm{OPT}/2 < \lambda_{\mathrm{end}} b_0. \tag{4.1}$$

Because $p'_n > (c-1)\mathrm{OPT} \geq (c-1)b_0 > 0.847\,b_0 > \lambda_{\mathrm{end}} b_0 \approx 0.5898\,b_0$ for ε not too large, we must have in particular

$$P_m^{n'-1} < \lambda_{\mathrm{end}} b_0. \tag{4.2}$$

The main goal of the analysis is to derive a contradiction to either inequality, $P_{m+1} < \lambda_{\mathrm{end}} b_0$ or $P_m^{n'-1} < \lambda_{\mathrm{end}} b_0$.

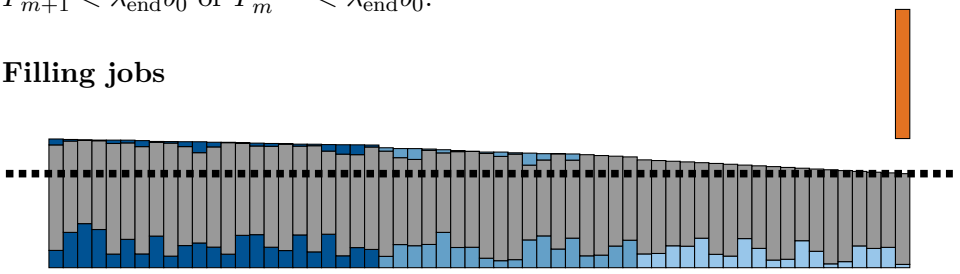


Figure 4.8: Filling jobs if ALG has to schedule large job $J_{n'}$ on a very flat schedule. If we remove all other jobs and only consider the filling jobs and job $J_{n'}$, the situation becomes similar to the right side of Figure 4.1. In particular, the optimum algorithm OPT also has a high makespan.

The standard way to derive such a contradiction considers filling jobs. Let $b \geq b_0$ such that at least $m - h$ machines reach load b before job $J_{n'}$ arrives.

We call a job J_t, $1 \leq t < n'$, a *filling job (with regards to b)* if it causes the load of a machine to reach or exceed b. Consider the most important special case $b = b_0$. Since every machine reaches load b_0 before job J'_n is scheduled, there are m such filling jobs. If we could show that all these filling jobs had processing time at least $\lambda_{\text{end}} b_0$, then we would have achieved the desired contradiction to $P_m^{n'-1} < \lambda_{\text{end}} b_0$, Inequality 4.2. This is the essence of prior analyses and also our main line of reasoning. Our analysis includes two novel approaches with regards to filling jobs. In some cases, random-order arrival allows us to take shortcuts that do not give guarantees on the processing times of filling jobs. In some other cases, arguments require us to consider filling jobs for values $b > b_0$.

We number filling jobs according to their arrival order. For $1 \leq j \leq m$, let $J^{(j)}$ denote the jth filling job. Choose s maximal such that the sth filling job is scheduled steeply. If $s \leq i$, set $s = i + 1$. Filling job $J^{(j)}$ with $j > i$ is called a *late filling jobs* if it is scheduled flatly. All other filling jobs are called *early filling jobs*. We first establish that late filling jobs do exist.

Proposition 4.3
We have $s < m - h$. In other words, filling jobs $J^{(m-h)}$, as well as $J^{(m-h+1)}, \ldots, J^{(m)}$ if they exist, are late filling jobs.

Proof Sketch. The proposition follows by computing the average load of a steep schedule where all but h machines have load $b \geq (c - 1 + \varepsilon)\text{OPT}$. If the schedule was steep after $m - h$ machines where full, i.e. $s \geq m - h$, this leads to a high lower bound of $\alpha \frac{m-i-h}{m-i} b$ on the k highest machine loads. Using these observations, we can compute the total average load of such a steep schedule. It is at least 1.32 OPT and, in particular, exceeds OPT. This is impossible by Proposition 4.2. □

4.3.3. Early filling jobs

The most difficult argument in prior analyses provides a lower bound on the processing times of early filling jobs. The goal is to establish that early filling jobs have processing time at least $\lambda_{\text{end}} b_0$, contradicting Inequality 4.2. In our case the argument becomes significantly simpler. Whenever the key statement in prior arguments does not hold, we can conclude $P_{m+1}^{n'-1} \geq \min(\lambda b, \lambda_{\text{end}} b)$ using the properties of stable sequences.

To comply with arguments in the next section, we need a more general statement. This statement includes general values $\lambda < \lambda_{\text{end}}$. Note that this inequality for $\lambda = \lambda_{\text{end}}$ gives us a desired contradiction to Inequality 4.1.

Proposition 4.4
Assume that all late filling jobs have processing time at least λb. Then $P_{m+1}^{n'-1} \geq \min(\lambda b, \lambda_{\text{end}} b)$.

The main technical ingredient for proving this proposition is the following Lemma 4.3. After it is established, the previous proposition follows by techniques known from the literature.

Lemma 4.3
Consider the time $t(s)$ when the sth filling job $J^{(s)}$ arrived, i.e. $J^{(s)} = J_{t(s)}$. One of the following holds: a) the average load of the machines $M_s^{t(s)-1}$ to $M_{m-h}^{t(s)-1}$ at time $t(s)-1$ is at most $\alpha^{-1}b$ or b) $P_{m+1}^{n'-1} \geq \lambda b$.

Let us first sketch why Lemma 4.3 is crucial to proving Proposition 4.4.

Proof Sketch of Proposition 4.4. Let us assume that part a) of Lemma 4.3 holds, else the conclusion is immediate. One can now observe that filling jobs $J^{(k+1)}, J^{(k+2)}, \ldots, J^{(i)}$ are scheduled steeply. Indeed, the smallest $m-i$ machines will have average load less than $\alpha^{-1}b = (1-\lambda_{\text{end}})b$ while the k most loaded machines have load at least b. Jobs $J^{(k+1)}, \ldots, J^{(i)}$ and later filling jobs are thus scheduled on a least loaded machine. Its load was at most $(1-\lambda_{\text{end}})b$. By definition, this machine reached load b afterwards. Filling jobs $J^{(k+1)}, \ldots, J^{(i)}$ thus had processing time at least $\lambda_{\text{end}}b$. Consider a time t before jobs $J^{(k+1)}, \ldots, J^{(i)}$ arrived. At this time, $i-k = m-i$ machines had load at most $(1-\lambda_{\text{end}})b$. In particular, the medium machine M_i^{t-1} did not have load exceeding $(1-\lambda_{\text{end}})b$. Hence, jobs $J^{(1)}, \ldots, J^{(k)}$ also caused machines of load at most $(1-\lambda_{\text{end}})b$ to reach load b. Again, we conclude that these filling jobs have processing time at least $\lambda_{\text{end}}b$, too. □

Previously, proving Lemma 4.3 required a clever and sophisticated analysis using exponential functions. This line of reasoning was not only involved, it also cannot be improved beyond the currently best competitive ratio of about 1.9201 from [84]. Our approach not only simplifies the proof using the properties of stable sequences. It also vastly lowers the competitive ratio c. This proof is also the reason why we chose the index of the medium machine to be $i \approx (2c-3)m$ different to prior work.

Proof Sketch Lemma 4.3. Assume that at time $t(s)-1$ machines $M_s^{t(s)-1}$ to $M_{m-h}^{t(s)-1}$ had average load exceeding $\alpha^{-1}b$. It then suffices to show that $P_{m+1}^{n'-1} \geq \lambda b$.

By definition, the full machines $M_{k+1}^{t(s)-1}$ to $M_{s-1}^{t(s)-1}$ had load at least b. Moreover, the schedule was steep, which leads to better bounds on the loads of $M_1^{t(s)-1}$ to $M_k^{t(s)-1}$. From these bounds we can compute that $L^{t(s)-1} \geq (\frac{s}{m} + \frac{\varepsilon}{2})$. The computation is long but straight-forward. Since $L^{t(s)-1} \geq (\frac{s}{m} + \frac{\varepsilon}{2})$, the second property of stable sequences establishes that at most $m+1-(s+h+2) = m-s-h-1$ of the largest $m+1$ jobs came after time $t(s)-1$. In particular, one of the $m-h-s$ late filling jobs $J^{(j)}$ arrived after time $t(s)-1$ was not among the $m+1$ largest jobs. Since this filling job $J^{(j)}$ had processing time at least λb we have that $P_{m+1} \geq \lambda b$.

To conclude $P_{m+1}^{n'-1} \geq \lambda b$, observe that $s+h+2$ of the largest $m+1$ jobs arrived before time $t(s)$ while the $m-h-s$ late filling jobs $J^{(s+1)}, \ldots, J^{(m-h-1)}$ came after time $t(s)$. These are more than $m+1$ jobs, each of which had processing time at least λb. Thus, $P_{m+1}^{n'-1} \geq \lambda b$. □

4.3.4. Late filling jobs

In the following we attempt an intuitive explanation to help with the subtle and interwoven proofs in the main paper cf. Appendix B. We first describe the easiest way one could profit from the random-order model before coming to our main argument, which forms a reciprocity between P_h the processing time of the hth largest job and P_{m+1} the processing time of the $(m+1)$th largest job.

The analysis of late filling jobs hinges on analyzing O^{mid}, the best lower bound O^t available for OPT once i machines are full. Formally, $O^{\text{mid}} = O^t$, where t is the time after the ith machine became full. At this time t, there holds $L^t \geq (c-1)\frac{i}{m}L \approx 0.721L$. Intuitively, if we can give better guarantees on O^{mid} our online algorithm will perform better. In fact, this has already been leveraged in [10]. We first describe a simple way to do so in the classical random-order model before explaining how this idea can be improved and lifted to the new model of *nearly-competitiveness*.

A simple approach

Take the algorithm from [84] and plug in $\max(L^t, p_{\max}^t)$ for L^t. Then, redo its analysis assuming that the following condition holds: $p_{\max}^t \geq (c-1)\text{OPT}$ whenever $L^t \geq 0.721L$. If this condition holds, one can show that the algorithm achieves competitive ratio $c^* \approx 1.890$. We now adapt the arguments at the beginning of the section to observe that a makespan above $c^*\,\text{OPT}$ is only possible if the following holds: No job $J_{n'}$ of processing time $p_{n'} \geq (c^*-1)\text{OPT}$ arrives once all machines reached load at least $(c^*-1)\text{OPT}$. This condition is impossible if a) no such job $J_{n'}$ arrived after $L^t \geq (c^*-1)L$, if b) $L \leq (c^*-1)\text{OPT}$ or c) if the input set $\mathcal{J}$ consisted of less than m jobs. We again ignore "simple" sets that fulfill the latter two conditions b) or c). This allows us to apply the Load Lemma. The condition $L^t \geq 0.721\,L$ becomes $t \geq 0.721\,n$. Also the condition a) $L^t \geq (c^*-1)L$ becomes $t \geq (c^*-1)n$.

We now call input permutation $\mathcal{J}^\sigma$ *semi-stable* if either a *large* job of processing time at least $(c^*-1)\text{OPT}$ arrived before time $0.721n$ or no such job arrived after time $(c^*-1)n$. We argued before that the algorithm then is c^*-competitive on semi-stable sequences. On general sequences, the makespan is at most $2\,\text{OPT}$. For this, we need to adapt the proof Lemma 4.1.

Let $P = P[\mathcal{J}]$ be the probability that an input permutation of input set $\mathcal{J}$ is not semi-stable. Then, on input set $\mathcal{J}$ our algorithms has makespan at most $((1-P)c^* + P \cdot 2)\text{OPT}$. Probability P only depends on the number $\tilde{h}$

of *large* jobs of processing time at least $(c^* - 1)\mathrm{OPT}$ in $\mathcal{J}$. Using some basic probability theory, one can show that P is maximal if $\tilde{h} = 1$. Here, $P = 1-(c^*-1) = 2-c^*$. In expectation, this approach leads to a competitive ratio of $c^* + (2 - c^*)^2 \approx 1.9021$.

Figure 4.9: An adversary looking to thwart our approach needs to place a large job at time $t \geq (c^* - 1)n = 0.89n$ but no such job must appear at time $t \leq 0.721n$. This, of course, is depends on the random job order. For the adversary it is best to only include one large job J in input $\mathcal{J}$. The adversary is lucky if job J arrives after $t \geq 0.89n$.

The ratio of $c^* + (2 - c^*)^2$ can be improved by further observations, which lead to several case distinctions. For example, the worst-possible competitive ratio of $2 - 1/m$ is only achieved if a job of processing time OPT arrives right at the end, after all machines have reached load $(1 - 1/m)\mathrm{OPT}$. Further improvements require case distinctions. They allow competitive ratios down to 1.896. My results still fell short of the adversarial lower bound of 1.88 from [145].

In the next section, we will see a way of forcing the input sequence to contain at least $\tilde{h} \geq h = h(m)$ large jobs of processing times at least $(c - 1)\mathrm{OPT}$. Thus, a job of processing time at least $(c - 1)\mathrm{OPT}$ is observed with almost certainty once i machines are full. This not only improves the competitive ratio in the random-order model, it also fulfills to the more strict requirements of nearly-competitiveness.

Oracle-Like properties

Recall that our long-term goal is to obtain a good lower bound on O^{mid}, the estimate on OPT once i machines are full. More precisely, we require $O^{\mathrm{mid}} \geq (1 - \varepsilon)b_0$. In our previous example, $\tilde{h} > 0$ *large* jobs of processing times at least $(c - 1)\mathrm{OPT}$ work as an unreliable oracle that reveals a value v, where $(c - 1)\mathrm{OPT} \leq v \leq \mathrm{OPT}$. The probability P of v being revealed depends on $\tilde{h}$, the number of jobs of size $(c - 1)\mathrm{OPT}$. In the worst-case, $\tilde{h} = 1$. If we could force $\tilde{h}$ to be high, say $\tilde{h} \geq h$ for the value $h = h(m)$ we introduced in the definition of the algorithm for this sake, the probability P approaches 1 for $m \to \infty$. In fact, the first property of stable sequences conveniently guarantees that once $L^t \geq (c - 1)\frac{i}{m}L$ there holds $P_h \leq p^t_{\max}$ and, in particular, $P_h \leq O^{\mathrm{mid}}$. Our long-term goal becomes to show that $P_h \geq (1 - \varepsilon)b_0$.

In order to force the adversary to include at least h large jobs of processing times at least $(1 - \varepsilon)b_0$ in input $\mathcal{J}$ or, equivalently, in order to perform well

on sequences that contain less than h large jobs, ALG samples the $(m-h)$th least loaded machine. Consider the time t_{end} when precisely $m-h$ machines were full. Filling jobs after time t_{end} were necessarily scheduled flatly on the least loaded machine, cf. Proposition 4.3. In particular, they were too large to fit on the $(m-h)$th least loaded machine. This gives a lower bound on their processing times, which is determined by the value of $O^{t_{\text{end}}}$. We write $O^{t_{\text{end}}} = O^{\text{end}}$ for short. Thus, a good lower bound on O^{end} leads to a good lower bound on the processing time of the h last filling jobs, thus on P_h and thus on O^{mid}. In order to prove our long-term goal $P_h \geq (1-\varepsilon)b_0$, we now need to establish a good lower bound on O^{end}.

Recall that $O^t = \max(L^t, p^t_{\max}, 2P^t_{m+1})$. The bound $p^t_{\max}$ was used to get a lower bound O^{mid}. In order to get a lower bound on O^{end}, we use P^t_{m+1} at time $t \geq t_{\text{end}}$. Assume for contradiction sake that the long-term goal $P_h \geq (1-\varepsilon)b_0$ did not hold. Using our current techniques and our current bound on O^{mid}, we have, for some $\lambda > 0$, shown a lower bound of $\lambda \cdot b_0$ on the $m-h$ filling jobs that arrived so far. We can then use the third property of stable sequences to show that $P^{t_{\text{end}}}_{m+1} \geq \lambda b_0$.

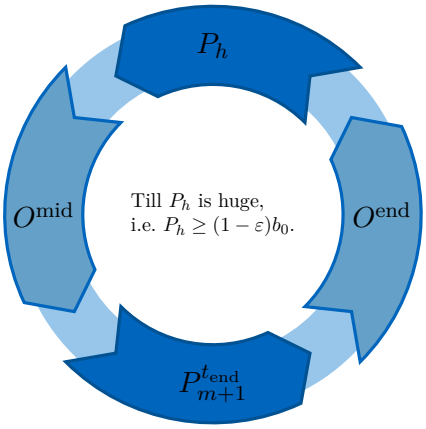


Figure 4.10: The reciprocity if O^{mid} and O^{end}. The bound O^{mid} leads to a good bound on the processing time of the $m+1$ largest job P_{m+1}. If $P_h \geq (1-\varepsilon)b_0$, we can derive a contradiction to Inequality 4.1, $P_{m+1} \geq \lambda_{\text{end}} b_0$, and thus to the assumption that our algorithm was not c-competitive. If $P_h < (1-\varepsilon)b_0$, this leads to a good bound on $P^{t_{\text{end}}}_{m+1}$ and thus on O^{end}. Bound O^{end} in turn, can be used to derive a lower bound on the processing times of the h last filling jobs and thus on P_h. Since on stable sequence $P_h \leq O^{\text{mid}}$, we obtain an improved bound on O^{mid}, which is higher than the bound we started with. The cycle repeats. We can either conclude $P_{m+1} \geq \lambda_{\text{end}} b_0$ or take another round of improvements.

Proposition 4.5
If $P_h < (1-\varepsilon)b_0$, then $h+1$ of the $m+1$ largest jobs arrived all filling jobs.

Proof. Consider the first filling job $J^{(1)}$ and let $t(1)$ be the time of its arrival, that is $J^{(1)} = J_{t(1)}$. If the first filling job $J^{(1)}$ was among the h largest jobs, the proposition follows from part b) of the third property of stable sequences. Else, the first filling job $J^{(1)}$ had load at most $P_h < (1-\varepsilon)b_0$. Therefore, the machine $M_j^{t(1)-1}$ it was scheduled on must have had load at least $l_j^{t(1)-1} \geq \varepsilon b_0 > \varepsilon(c-1)$ for it to reach load b_0 after receiving job $J^{(1)}$. By the definition of algorithm ALG, we have $j \in \{i, m-h, m\}$. In particular, at least i machines had load at least $l_j^{t(1)-1} \geq \varepsilon(c-1)$ at time $t(1)-1$. We conclude that the average load reached $\frac{i}{m}(c-1)\varepsilon$ before the first filling job $J^{(1)}$ arrived. The proposition follows from part a) of the third condition of stable sequences. □

In total, we argued that a good lower bound on O^{mid} leads to a good lower bound on the b_0-filling jobs and thus on P_{m+1}. If $P_h < (1-\varepsilon)b_0$, we get a good bound on $P_{m+1}^{t_{\text{end}}} \leq O^{\text{end}}$. Earlier, we showed the reciprocal, how a good bound O^{end} helps us improve the bound O^{mid}. We can now subsequently improve O^{end} and O^{mid} until $O^{\text{mid}} \geq P_h \geq (1-\varepsilon)b_0$. We chose our competitive ratio c such this in turn implies $P_{m+1} \geq \lambda_{\text{end}} b_0$, the desired contradiction to our algorithm being c-competitive, cf. Inequality 4.1. This idea of proof is also depicted in Figure 4.10.

The previous informal arguments swept some technicalities under the rug, which are required in our main paper. We draw attention to the two most important aspects.

First, we need an initial lower bound on O^{mid}. For this, we use the remaining term in $O^t = \max(L^t, p_{\max}^t, 2P_{m+1}^t)$, the average load L^t. This resembles prior analyses in the literature. They choose their competitive ratio c minimal such that bound L^t suffices to establish $P_{m+1} \geq \lambda_{\text{end}} b_0$. For our choice of c we only establish $P_{m+1} \geq \lambda_{\text{start}} b_0$ where $\lambda_{\text{start}} \approx 0.5426$ falls short of $\lambda_{\text{end}} \approx 0.5898$. The previous sections are used to bridge this gap.

Second, we previously sketched how a lower bound on the processing times of filling jobs $J^{(1)}, \ldots, J^{(m-h)}$ can lead to a lower bound on O^{end}. This is useful, since it gives a lower bound on the processing time of filling job $J^{(j)}$, for $m-h < j \leq m$. The argument is the following: Since job $J^{(j)}$ was not scheduled on machine M_{m-h}^{t-1}, it had processing time exceeding $c\,O^{\text{end}} - b_1$. Here, b_1 was the load of machine M_{m-h}^{t-1}. Our previous bound O^{end} is only sufficient for $b_1 = b_0$. For $b_1 > b_0$, we need to use our bound O^{mid} to obtain an even better bound on the filling jobs with respect to b_1. Only then do we obtain a sufficiently good lower bound on O^{end}.

4.4. Open Problems

The main problem is, of course, to improve the upper bound here.

Problem 4.1
Find an algorithm that is nearly c-competitive for some $c \leq 1.8477$.

After having thought about the problem for quite a while, I know that such an improvement is possible if one considers the classical random-order model. So far, the goal was to use $P_h \geq (1-\varepsilon)b_0$ as an oracle. A close look at the analysis reveals that worst-case guarantees only occur if $b_0 = (c-1+\varepsilon)\mathrm{OPT}$. In this case though, job $J_{n'}$ must have had processing time OPT. One could now combine the prior approach of betting on a large job to arrive before i machines are full with the current approach of considering the h largest jobs. Given the complexity of our current analysis, I do not recommend pursuing this approach. It requires quite involved calculations and case distinctions without adding new ideas. Still, it leads to a better competitive ratio in expectation. In particular, it solves the following problem.

Problem 4.2
Find an algorithm that is better than c-competitive in the random-order model for $c \leq 1.8477$ and $m \to \infty$.

In our paper Appendix C, discussed in Chapter 5, we introduce the *secretary model*. This is the random-order model but additionally online algorithm knows n, the number of jobs in the input, in advance. I am convinced that knowing n is highly advantageous but proving so is difficult. The main difficulty arises since it is generally hard to obtain lower bounds in random-order models. In particular, our lower bounds of 1.5 respectively 4/3 should be considered quite pessimistic. Formally separating the random-order model from the secretary model would be interesting.

Conjecture 4.1
Knowing n, the number of jobs, in advance is an advantage and allows better competitive ratios. Proving this requires improving either the lower bound for the random-order model where online algorithms do not know n or the upper bound for the secretary model where they do.

A common benchmark algorithm for Makespan Minimization is Graham's Greedy strategy. For any job, it simply uses a least loaded machine. Let c_m be its competitive ratio on m machines in the random-order model. Osborn and Torng [139] have shown that $\lim_{m\to\infty} c_m = 2$. On the other hand, it is easy to see that $c_m < 2 - 1/m$. The Greedy strategy improves in the random order model. Estimating competitive ratios for small numbers of machines is challenging in the random-order model. As one of the simplest online algorithms, Graham's Greedy strategy should be a good start for such analyses.

Problem 4.3
Provide tight ratios c_m for the Greedy strategy for explicit values of m.

Through my analyses, I know that the competitive ratio c_2 of Graham's Greedy strategy in the random-order model on $m = 2$ machines is strictly smaller than 1.4 but at least 1.25. For the lower bound, consider a sequence that has one job of processing time 1 and ε^{-1} jobs of processing time ε. I conjecture that this lower bound is tight.

Conjecture 4.2
Graham's Greedy strategy has competitive ratio $c_2 = 1.25$ in the random-order model on $m = 2$ machines.

Furthermore, the lower bound in [139] can be used to observe that $c_m = 2 - O(\sqrt{m})$. I conjecture that this is asymptotically tight.

Conjecture 4.3
Graham's Greedy strategy has competitive ratio $c_m = 2 - \Theta(\sqrt{m})$ in the random-order model on m machines.

Chapter 5

Makespan Minimization in the Secretary Model

This section studies Online Makespan Minimization in the secretary model. The problem is the same as in Chapter 4, only that online algorithms know the input size n in advance. In the adversarial model, such knowledge is useless but together with random-order arrival it vastly improves algorithmic performance, as we will show. Formally, the *secretary model* is the random-order model where n is known in advance.

For Makespan Minimization, the task is to assign n *jobs* $\mathcal{J} = \{J_1, \ldots, J_n\}$ to m parallel and identical machines. Preemption is not allowed. The goal is to minimize the *makespan*, which is the time it takes to process them all.

An online algorithm A receives jobs one by one. Only after one job is scheduled permanently and irrevocably will the next one be revealed. This prevents optimal performance on arbitrary input sequences. Instead, one judges online algorithm A by how far its makespan may stray from being optimal. The makespan of A depends on both input set $\mathcal{J}$ and input order σ. We denote this makespan by $A(\mathcal{J}^\sigma)$. Let $\mathrm{OPT}(\mathcal{J})$ be the optimum makespan, which does not depend on an input order. Then the *adversarial competitive ratio* of online algorithm A is $c = \sup_{\mathcal{J},\sigma} \frac{A(\mathcal{J}^\sigma)}{\mathrm{OPT}(\mathcal{J})}$. Classically, it has been the standard performance measure and was studied in depth.

The adversarial competitive ratio, which relies on worst-case orders, is oftentimes considered pessimistic. Thus, the *secretary model* has been introduced. In the secretary model, the input order σ is instead chosen uniformly at random. The *competitive ratio of online algorithm A in the secretary model* is thus $c = \sup_{\mathcal{J}} \mathbf{E}_{\sigma \sim S_n}\left[\frac{A(\mathcal{J}^\sigma)}{\mathrm{OPT}(\mathcal{J})}\right] = \sup_{\mathcal{J}} \frac{1}{n!} \sum_{\sigma \in S_n} \frac{A(\mathcal{J}^\sigma)}{\mathrm{OPT}(\mathcal{J})}$. All algorithms in the literature on random-order scheduling excluding ours in Chapter 4, know the input length n in advance, which certainly allows for better performance guarantees.

We use the performance measure of *nearly competitiveness*, which considers nearly worst-case input orders. Online algorithm A is *nearly c-competitive*, for $c \geq 1$, if for every $\varepsilon > 0$ there exists a number of machines m_ε such that if $m \geq m_\varepsilon$ machines are considered, $\mathbf{P}_{\sigma \sim S_n}[\frac{A(\mathcal{J}^\sigma)}{\mathrm{OPT}(\mathcal{J})} > c + \varepsilon] < \varepsilon$. Additionally, online algorithm A needs to achieve a constant competitive ratio C in the adversarial model. Nearly competitiveness is a much stronger condition than the classical notion of random-order competitiveness.

Proposition 5.1
If an online algorithm A is nearly c-competitive, then it is c-competitive in the secretary model for $m \to \infty$. Formally, if c_m is A's competitive ratio in the secretary model on m machines, then $\lim_{m\to\infty} c_m \leq c$.

5.1. Our contributions

LightLoad

The general Load Lemma, Lemma 4.2, allows us to adapt algorithm LightLoad from the literature [10]. The original algorithm LightLoad was developed for the semi-online setting where the average load L is known in advance. In the secretary model, where n is known to the online algorithm, the average load L can be guessed after any fixed fraction of the input sequence has been treated. Unfortunately, compared to the pure semi-online setting, there are three more challenges to overcome.

The three challenges are the following. First, one needs to exclude certain simple sequences and provide adversarial guarantees. In the worst case, the sampling estimate of L is completely wrong. Second, one needs to account for a margin of error. This is no problem in the limit case but becomes quite tricky when small numbers of machines m are of interest. Third, the average load $L = L[\mathcal{J}]$ cannot be known right from the start. A few jobs have to be scheduled without such knowledge. This third challenge prohibits adapting almost all semi-online algorithms [54, 118, 119] that know L in advance to the secretary model. In fact, the only remaining semi-online algorithm that does not need to know average load L right from the start is the algorithm LightLoad [10]. We now state the adaptation LightLoadROM of this algorithm to the secretary model. It estimates L by the guess $\hat{L}[\mathcal{J}^\sigma] = 4L^{n/4}[\mathcal{J}^\sigma]$, four times the average load after a fourth of the sequences has been scheduled. Note that the guess $\hat{L}$ is a good approximation of L by Lemma 4.2.

Algorithm 3 The algorithm LightLoadROM

1: *Let J_t be the current job and let p_t be its processing time.*
2: Determine the least loaded machine M_{low}^{t-1} and the $\lfloor \frac{m}{2} \rfloor$ least loaded machine M_{mid}^{t-1}. Let l_{low}^{t-1} be l_{mid}^{t-1} their respective loads.
3: **if** $t < n/4$ **or** $l_{\text{low}}^{t-1} \leq 0.25\,\hat{L}$ **or** $l_{\text{mid}}^{t-1} + p_t > 1.75\,\hat{L}$ **then**
4: Schedule job J_t on least loaded machine M_{low}^{t-1};
5: **else** schedule job J_t on $\lfloor \frac{m}{2} \rfloor$ least loaded M_{mid}^{t-1};

We obtain the original algorithm LightLoad from [10] if we replace guess $\hat{L}$ with L and remove the condition $t < n/4$ in the **if**-statement. Condition $t < n/4$ is technically superfluous. Indeed, for $t < n/4$ we have

$l_{\text{low}}^{t-1} \leq_{\text{low}}^{n/4} \leq L^{n/4}[\mathcal{J}^\sigma] = 0.25\,\hat{L}[\mathcal{J}^\sigma]$. We include the condition $t < n/4$ so that it is clear that algorithm LightLoadROM is an online algorithm that only knows $\hat{L}$ after $n/4$ jobs have arrived.

Theorem 5.1
Consider any (ordered) input sequence $\mathcal{J}^\sigma$. The makespan of algorithm LightLoadROM on $\mathcal{J}^\sigma$ is at most $1.75(1 + \frac{|\hat{L}[\mathcal{J}^\sigma]-L|}{L})\text{OPT}(\mathcal{J})$.

About the Proof. For $L = \hat{L}$, this is the main result from [10]. For $L \geq \hat{L}$, we can easily reduce ourselves to the case $L = \hat{L}$ by adding a few extra jobs at the end of the sequence. We can add these jobs in a way that the optimum makespan does not increase beyond $\hat{L}$. Then, the main result from [10] for $L = \hat{L}$ is applicable. For $\hat{L} < L$, the statement of the theorem still holds but cannot be easily deducted using the results in [10] as a black box. Instead, the proof requires redoing and adapting their analysis. □

Recall that the *mean absolute deviation* of a random variable X is $\text{MD}[X] = \mathbf{E}[|X - \mathbf{E}[X]|]$. If random variable X has positive expected value, its *normalized mean absolute deviation* is $\text{NMD}[X] = \frac{\text{MD}[X]}{\mathbf{E}[X]}$. Let us for simplicity assume that input size n is divisible by 4. Then $\mathbf{E}[\hat{L}] = L$ and $\text{NMD}[\hat{L}] = \mathbf{E}[\frac{|\hat{L}-L|}{L}]$. We obtain the following corollary to Theorem 5.1.

Corollary 5.1
Let $\mathcal{J}$ be any job set and pick $\sigma \sim S_n$ uniformly at random. The expected makespan of LightLoadROM is at most $1.75\,(1 + \text{NMD}(\hat{\text{L}}))\text{OPT}(\mathcal{J})$.

In order to establish the nearly competitive ratio of LightLoadROM, we also need to provide adversarial performance guarantees and guarantees on simple job sets. Simple sets are defined using the value $R(\mathcal{J}) = \min(\frac{L}{p_{\max}}, 1)$. Formally, a job set $\mathcal{J}$ is *simple* if $n = |\mathcal{J}| \leq m$ or if $R(\mathcal{J}) \leq \frac{3}{8}$.

Proposition 5.2
The makespan of online algorithm LightLoadROM on any (ordered) job sequence $\mathcal{J}^\sigma$ is at most $(1 + 2R(\mathcal{J}))\text{OPT}(\mathcal{J})$. In particular, it is at most $3\,\text{OPT}(\mathcal{J})$ and if $R(\mathcal{J}) \leq \frac{3}{8}$ it is at most $1.75\,\text{OPT}(\mathcal{J})$.

From now on, we only need to consider non-simple input sets $\mathcal{J}$ with $R(\mathcal{J}) \geq \frac{3}{8}$. The Load Lemma, Lemma 4.2, then ensures that for every $\varepsilon > 0$ there exists m_ε such that on $m \geq m_\varepsilon$ machines $\mathbf{P}_{\sigma\sim S_n}[\text{NMD}(\hat{L}) \geq \varepsilon] < \varepsilon$. From this we can conclude the following main result on LightLoadROM.

Theorem 5.2
LightLoadROM is nearly 1.75-competitive.

We obtain bounds for small values of m by further estimating $\text{NMD}(\hat{L})$ on sequences with $R(\mathcal{J}) \geq \frac{3}{8}$. In our main paper in Appendix C, we show that we can replace $\hat{L}$ first by a (scaled) hypergeometric variable Y and then by

a binomial variable X. Formally, we show that $\mathrm{NMD}(\hat{L}) \leq \frac{m}{R(\mathcal{J})}\mathrm{NMD}(Y) \leq \frac{m}{R(\mathcal{J})}\mathrm{NMD}(X) \leq \frac{10.02+o_m(1)}{\sqrt{m}}$. This leads to the following result.

Theorem 5.3
LightLoadROM is at most $(1.75 + \frac{18}{\sqrt{m}} + O(\frac{1}{m}))$-competitive in the secretary model .

We also evaluate $\mathrm{NMD}(\hat{L})$ on actual input sequences, indicating that the previous result can still be improved, see Figure 5.1. One possible way of improvement would be to analyze $\mathrm{NMD}(Y)$ directly using techniques from [62].

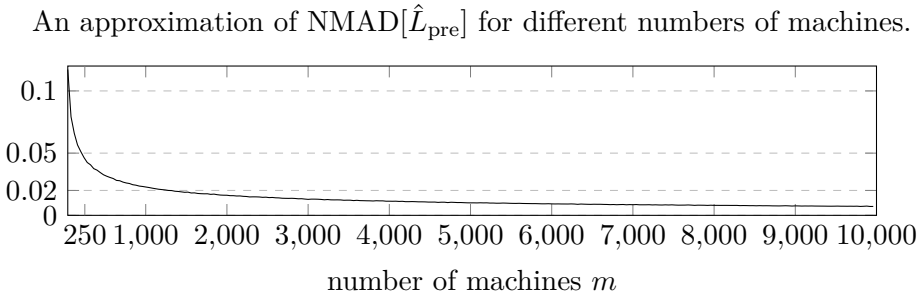


Figure 5.1: The graph shows an estimation of $\mathrm{NMD}(\hat{L})$ on the lower bound sequence from [2] based on 10,000 random samples.

Theorem 5.1 can be further improved. In general, LightLoadROM will be $1.75\,(1+\delta)$-competitive, where $\delta = d(\hat{L}, [L, \mathrm{OPT}])$, the distance of $\hat{L}$ from the non-empty, compact interval $[L, \mathrm{OPT}]$. This can be used to improve the competitive ratio LightLoadROM. Besides the estimate $\hat{L}$ also consider $p_{\max}^{n/4}$, the largest job processing times seen in the first fourth of the input. Formally, we replace the estimate $\hat{L}$ in Algorithm 3 by $\hat{B} = \max(4L_{n/4}, p_{\max}^{n/4})$. Due to the $p_{\max}^{n/4}$-term, the resulting algorithm automatically performs well on sequences that contain many jobs of processing times exceeding L. On the remaining sequences, the impact of these large jobs becomes negligible. We thus can reduce ourselves to the case $R(\mathcal{J}) = 1$. Recall that $\mathrm{NMD}(\hat{L}) \leq \frac{m}{R(\mathcal{J})}\mathrm{NMD}(X)$. Considering $R(J) = 1$ instead of $R(J) = \frac{3}{8}$ thus significantly improves performance guarantees. We could, so far, derive the following.

Theorem 5.4
The competitive ratio of LightLoadROM in the secretary model is at most $1.75 + \frac{4.4}{\sqrt{m}} + \frac{7}{m} + O(\frac{1}{m^{3/2}})$.

I believe that even this result is far from optimal and can be improved further.

Main Algorithm

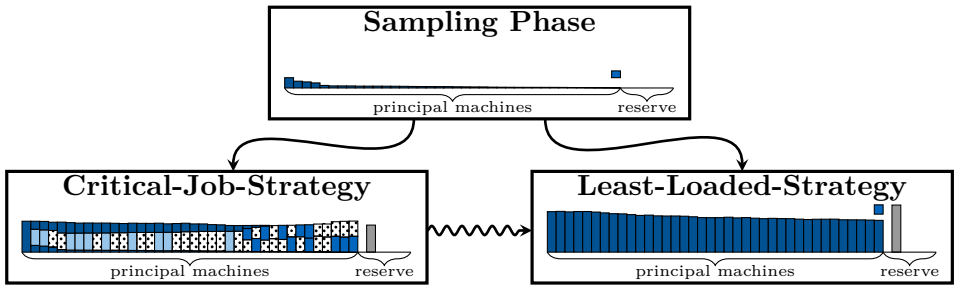


Figure 5.2: An overview over the main algorithm. First, a sampling phase learns about a certain set of critical jobs. Depending on what is learned either the Least-Loaded-Strategy is chosen or the Critical-Job-Strategy. The Critical-Job-Strategy is more sophisticated and tries to schedule critical jobs efficiently. It may still turn out to be the wrong choice. In this most troublesome case, we have to switch back to the Least-Loaded-Strategy.

In the previous section, a small sample of the input is used to estimate the average load L and to observe the close-to-largest processing time $p_{\max}^{\varphi n}$ for $\varphi = 1/4$. Our main algorithm takes this further. It first uses sampling to estimate $B = \max(L, p_{\max})$ and, depending on the estimate, defines a class $\mathcal{C}$ of critical jobs. It then further uses the sample to estimate these critical jobs. Afterwards, it has to decide between two strategies. The Critical-Job-Strategy focuses on handling critical jobs best possible. The Least-Loaded-Strategy simply enhances Graham's Greedy strategy by keeping a few machines in reserve to accommodate particularly large jobs.

The Sampling Phase.

In the sampling phase the first φn jobs, for suitable choice of $\varphi > 0$, are assigned greedily to least loaded principal machines. Principal machines exclude a few reserve machines, which need to stay empty for the following phases. After the sampling phase, we estimate L by $\hat{L} = \frac{1}{\varphi} L^{\varphi n}$. The estimate $\hat{L}$ provides a good approximation of the average load L. Technically, the general statement of the Load Lemma, Lemma 4.2, is required since the terms involved decrease as the number of machines m increases. We estimate the processing time of the largest job $p_{\max}$ by $p_{\max}^{\varphi n}$. This might not be a close estimate to $p_{\max}$ but we are guaranteed that only few, φ^{-1} in expectation, particularly large job processing times exceed this guess. We thus use the estimate $\hat{B} = \max(\hat{L}, p_{\max}^{\varphi n})$. Jobs of processing time at most $(c-1)\hat{B}$ always safely fit on a least loaded machine. We thus focus on *critical* jobs, which have processing times exceeding $(c-1)\hat{B}$.

In order to estimate the set $\mathcal{C}$ of these critical jobs, we geometrically round their processing times to powers of $(1+\varepsilon)$ for a suitable choice of $\varepsilon > 0$.

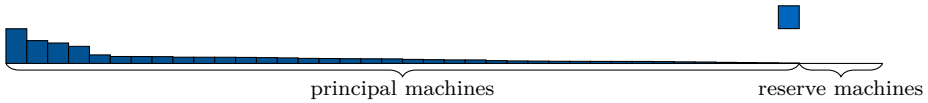


Figure 5.3: The sampling phase simply assigns each job to a least loaded principal machine. A few reserve machines are kept empty for they will be needed later. After the sampling phase, we can estimate the input set.

Consider a possible rounded job size $p = (1+\varepsilon)^i$ with $(c-1)\hat{B} < p \leq \hat{B}$. We now use the sampling phase to estimate the number n_p of jobs of processing time p. This gives us an estimate over the set of critical jobs $\mathcal{C} = \{J_t \mid (c-1)\hat{B} < p_t\}$. Using our estimations of $p_{\max}$, L and $\mathcal{C}$, we decide whether it is safe to use the Least-Loaded-Strategy. If so, it is employed. Else, the Critical-Job-Strategy focuses on assigning critical jobs correctly.

The Least-Loaded-Strategy.

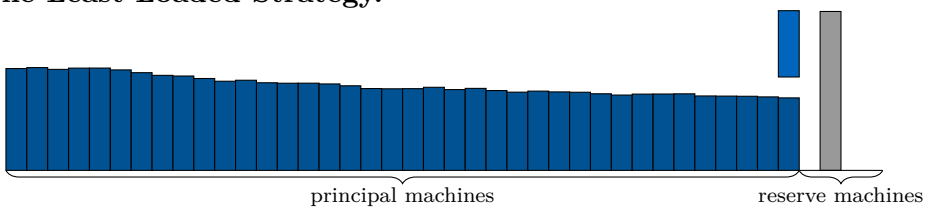


Figure 5.4: The Least-Loaded-Strategy keeps few machine in reserve to accommodate huge jobs. Such huge jobs are assigned to any least loaded machine and will, on random-order sequences, almost always find an empty reserve machine. These reserve machines are kept empty for precisely these huge jobs . Other jobs are always placed on a least loaded principal machine.

The Least-Loaded-Strategy first determines if a given job J_t is *huge*, i.e. if job J_t has processing time $p_t > \hat{B}$. As mentioned earlier, such jobs are few. There are only φ^{-1} in expectation. The number of huge jobs is, with high probability, less than the number of reserve machines. In particular, with high probability, an empty machine is available for every huge job. If a job is not huge, it will simply be scheduled onto a least loaded principal machine. Using Graham's arguments, one may observe that the Least-Loaded-Strategy causes a makespan of approximately $\max(p_{\max}, 2\hat{B})$—baring negligible terms. The Least-Loaded-Strategy is thus a good choice whenever $2\hat{B} \leq c\mathrm{OPT}$.

The Critical-Job-Strategy.

The Critical-Job-Strategy conservatively estimates the set $\mathcal{C}$ of critical jobs. It tries to predict them as well as possible but avoids overestimations at all cost. According to these predictions, fictional placeholder jobs are scheduled in advance. These jobs will be replaced once their real counterparts arrive.

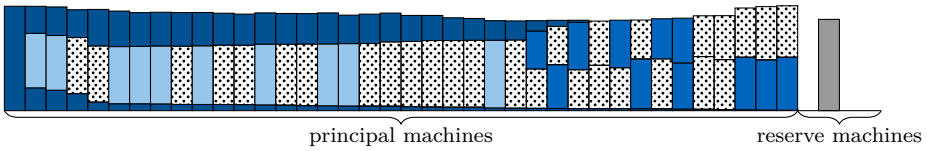


Figure 5.5: The Critical-Job-Strategy schedules critical jobs "in advance". Formally, placeholder jobs are maintained that are later exchanged with their real counterparts. Reserve machines are kept aside to accommodate huge jobs and critical jobs that could not be predicted. The Critical-Job-Strategy is fragile. If this is not possible to assign critical jobs correctly, the algorithm finds out very late and has to switch to the Least-Loaded-Strategy.

Formally, placeholders jobs are assigned right after the sampling phase but before the actual Critical-Job-Strategy begins.

Small jobs are simply assigned to a least loaded principal machine where the load takes placeholder jobs into account. A critical job J_t replaces a matching placeholder job if there still is one available. Else, we try to assign it to a reserve machine, still respecting that each machine contains either a big job or at most two medium ones. If this fails, we have to switch from the Critical-Job-Strategy to the Least-Loaded-Strategy. Showing that this switch is possible is the crux of the analysis.

Lower Bounds.

We complement our main result by first lower bounds.

Theorem 5.5
No online algorithm, deterministic or randomized, has competitive ratio better than $\frac{\sqrt{73}+29}{36} \approx 1.043$ in the secretary model.

Theorem 5.6
No online algorithm, deterministic or randomized, is better than nearly c-competitive for $c < \frac{\sqrt{73}-1}{6} \approx 1.257$.

5.2. Open Problems

The obvious problem is to improve the upper and lower bounds described in this chapter. In particular, a better upper bound is desirable.

Problem 5.1
Find an algorithm, which is better than nearly 1.535-competitive.

More open problems related to the random-order model are listed in Chapter 4. It would be particularly interesting to show Conjecture 4.1,

that the secretary model allows better performance guarantees than the random-order model.

Chapter 6

Online Makespan Minimization with Budgeted Uncertainty

6.1. Preliminaries

We study Online Makespan Minimization with budgeted uncertainty. A sequence of *jobs* $\mathcal{J}^\sigma = J_1, \ldots, J_n$ has to be assigned to m parallel and identical machines. Each job has both a *regular processing time* $\tilde{p}_i$ and an *additional processing time* Δp_i. Most jobs only require their regular processing times in order to be processed. Unfortunately, up to Γ jobs *fail* and require processing time $p_i = \tilde{p}_i + \Delta p_i$. The value of Γ, the number of failing jobs, is part of the input. The resulting makespan if these up to Γ failing jobs are chosen worst-possible, is called the *uncertain makespan*. This uncertain makespan should be minimized.

Budgeted Uncertainty models errors and malfunctions that arise in practical applications. For example, machines may represent servers that run programs. A fraction of these programs exhibits bugs that require costly restarts and clean-up operations. In other applications actual machines are considered that produce physical products and equipment. Such machines may jam or malfunction. There are even schedules made for humans. Jobs represent tasks that are to be performed. There are always a few tasks that take longer than estimated.

Parameter Γ determines the amount of bugs we have to prepare for. The naive case $\Gamma = 0$ recovers the old problem of Makespan Minimization where processing times $\tilde{p}_i$ are well-known once an assignment takes place. Bugs simply do not happen. Similarly, the paranoid case $\Gamma = \infty$ where we expect a bug in every program and a failure in every endeavor also recovers the original problem of Makespan Minimization with processing times $p_i = \tilde{p}_i + \Delta p_i$. For general Γ, the problem becomes, as we show, strictly harder.

We consider online algorithms. An *online algorithm* A schedules jobs $J_1, \ldots, J_n$ one by one, immediately and irrevocably, without knowing future ones. Its performance is measured by its *competitive ratio* c, the ratio by which its uncertain makespan approximates the optimum one. Formally $c = \sup_{\mathcal{J}^\sigma} \frac{A(\mathcal{J}^\sigma)}{\mathrm{OPT}(\mathcal{J})}$ where $A(\mathcal{J}^\sigma)$ denotes the makespan of online algorithm A on input sequence $\mathcal{J}^\sigma$ while $\mathrm{OPT}(\mathcal{J})$ denotes the optimum makespan. The goal is to find online algorithms exhibiting small competitive ratios.

Notation

In order to be compatible with other sections we use $\mathcal{J}$ to denote the job set $\{J_1, \dots, J_n\}$, while $\mathcal{J}^\sigma = J_1, \dots, J_n$ denotes the ordered sequence in which an online algorithm treats these jobs. This order has no impact on the optimum offline algorithm OPT, which is why we simply write $\text{OPT}(\mathcal{J})$.

6.1.1. Problem Definition

We are given a sequence of jobs $\mathcal{J}^\sigma = J_1, \dots, J_n$ where job J_i has *regular processing time* $\tilde{p}_i$ and an *additional processing time* Δp_i. These jobs have to be assigned to a set $\mathcal{M}$ of m machines. Given such an assignment $\sigma\colon \mathcal{J} \to \mathcal{M}$, the *regular load* $\tilde{l}_M$ of machine M is the total processing time of jobs assigned to it if none fails. Formally, $\tilde{l}_M = \sum_{J_t \in \sigma^{-1}(M)} \tilde{p}_t$, where $\sigma^{-1}(M)$ is the set of jobs scheduled on M. The *additional load* Δl_M of M is the maximum total additional processing time on M if up to Γ jobs fail in the worst-possible way. Formally, $\Delta l_M = \max_{S \subseteq \sigma^{-1}(M), |S| \leq \Gamma} \sum_{J^t \in S} \Delta p_t$. Let us fix a set S_M for machine M where the previous maximum is obtained. We then say job J_t *fails* machine M if $J_t \in S$. The *robust load* l_M of machine M, is the maximum time M may take to process all its jobs if up to Γ fail. Formally, $l_M = \tilde{l}_M + \Delta l_M$. The *uncertain makespan*, which we wish to minimize, is then $\max_{M \in \mathcal{M}} l_M$.

The previous definitions do not change, if we do not only allow Γ jobs to fail in total but instead allow Γ to fail per machine. Hence, both problems are equivalent. The latter problem may, additionally to covering more failing scenarios, also yield a better intuition in the analysis.

6.2. Our contributions

We study Online Makespan Minimization with budgeted uncertainty in depth. We first analyze the Greedy strategy, which places each job such that the resulting uncertain makespan is minimized. The Greedy strategy is a common benchmark strategy, which performs quite well. For general Γ, we show that the it is precisely $(3 - \frac{2}{m})$-competitive, which nicely complements Graham's [94] well known upper bound of $2 - \frac{1}{m}$ for $\Gamma \in \{0, \infty\}$. We also provide a tight lower bound of $3 - \frac{2}{m}$ that even holds in the special case where $\tilde{p}_i = 0$ for all i. We call this the *debugging model*, which may be applicable to settings where normal running times of programs, measured in milliseconds, are negligible. Bugs on the other hand require time-intensive debugging, which takes hours, days and weeks.

After establishing Greedy as a benchmark, we provide an improved deterministic algorithm, which is particularly suited for large values of m and Γ. It adapts prior approaches to obtain a competitive ratio of 2.9052 for $m, \Gamma \to \infty$. Precise competitive ratios for general m and $\Gamma \geq 10$ are

given in Figure 6.2. We complement this result by a lower bound of 2 for the competitive ratio achieved by deterministic algorithms for $\Gamma = 2$. This shows that the problem with budgeted uncertainty is strictly more difficult than the classical setting. For $\Gamma \in \{0, \infty\}$, even the Greedy strategy beats this bound [94].

6.2.1. The Greedy Strategy

The first result on Makespan Minimization is due to Graham [94] who in the 1960s showed that his famed Greedy strategy is $(2 - \frac{1}{m})$-competitive. In general, the scheduling literature differentiates between Pre-Greedy strategies and Post-Greedy strategies. Pre-Greedy strategies assign each job to a machine of minimum load. Post-Greedy strategies pick a machine such that the resulting load is minimized. For classic Online Makespan Minimization both definitions are equivalent. Under budgeted uncertainty assumptions this is not the case anymore. Post-Greedy strategies perform slightly better. We can establish the following.

Theorem 6.1
The Post-Greedy has competitive ratio $3 - \frac{2}{m}$. This bound is tight.

The main technical result concerns the average robust load. Consider the schedule of online algorithm A on input sequence $\mathcal{J}^\sigma = J_1, \ldots, J_n$. Let $L[A, \mathcal{J}^\sigma] = \frac{1}{m} \sum_M l_M$ be the *average robust load* of the schedule. The core argument of Graham is based on the fact that $L[A, \mathcal{J}^\sigma]$ is independent of the algorithm A considered. With budgeted uncertainty, this does not hold anymore. We thus need a different way to relate $L[A, \mathcal{J}^\sigma]$ to $\mathrm{OPT}(\mathcal{J})$.

Proposition 6.1
The average robust load $L[A, \mathcal{J}^\sigma]$ of algorithm A on input $\mathcal{J}^\sigma$ is at most $L[\mathrm{OPT}, \mathcal{J}] + (1 - \frac{1}{m})\mathrm{OPT}(\mathcal{J})$. In particular, $L[A, \mathcal{J}^\sigma] \leq (2 - \frac{1}{m})\mathrm{OPT}(\mathcal{J})$.

Proof Sketch. We only show $L[A, \mathcal{J}^\sigma] \leq L[\mathrm{OPT}, \mathcal{J}] + \mathrm{OPT}(\mathcal{J})$ for simplicity. Let T be the set of jobs that fail in the schedule of A but not in the schedule of OPT. Then $L[A, \mathcal{J}^\sigma] \leq L[\mathrm{OPT}, \mathcal{J}] + \frac{1}{m} \sum_{J_t \in T} \Delta p_t$. It suffices to show that $\sum_{J_t \in T} \Delta p_t \leq m\mathrm{OPT}$. Consider a job $J \in T$ with additional processing time Δp. Job J must be on a machine in OPT's schedule where Γ machines of processing time at least Δp fail. In particular, $\Gamma \cdot \Delta p \leq \mathrm{OPT}$. Thus, $\sum_{J_t \in T} \Delta p_t \leq |T| \cdot \mathrm{OPT}/\Gamma \leq m\mathrm{OPT}$. For the last inequality we use that $|T| \leq \Gamma m$ since at most Γ machines fail on each of the m machines. Thus, $L[A, \mathcal{J}^\sigma] \leq L[\mathrm{OPT}, \mathcal{J}] + \frac{1}{m} \sum_{J_t \in T} \Delta p_t \leq L[\mathrm{OPT}, \mathcal{J}] + \mathrm{OPT}(\mathcal{J})$.

For the full result, $L[\mathrm{OPT}, \mathcal{J}] + (1 - \frac{1}{m})\mathrm{OPT}(\mathcal{J})$ one also considers the set S of jobs that fail in OPT's but not in A's schedule. □

Now, Theorem 6.1 follows almost immediately from Graham's analysis.

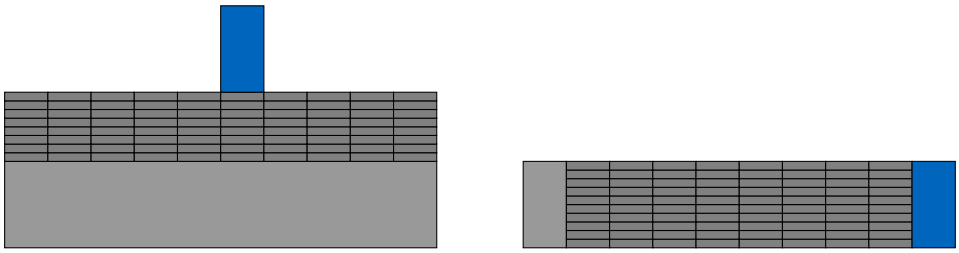


Figure 6.1: The lower bound for Post-Greedy. All regular processing times $\tilde{p}$ are 0. Tiny jobs fill the greedy schedule (left) to a height of almost 1. Small jobs increase it to $2 - \frac{2}{m}$. A final job of size 1 causes a makespan $3 - \frac{2}{m}$. The optimum schedule (right) places the tiny jobs on a single machine. Since at most Γ additional processing times count, the resulting load is 1. The large job goes onto another machine, while the remaining $m - 2$ machines are filled with small jobs.

Proof Sketch of Theorem 6.1. By Proposition 6.1, the least loaded machine in Post-Greedy's schedule has load at most $\left(2 - \frac{1}{m}\right)$ OPT. Since it receives a job J_t of size $p_t = \tilde{p}_t + \Delta p_t \leq \text{OPT}$, the load of any machine can reach at most $(3 - \frac{1}{m})\text{OPT}$. In order to establish the tight bound of $(3 - \frac{2}{m})\text{OPT}$, we apply Proposition 6.1 to the schedule obtained by assigning J_t preliminary on some machine. Here, it is important to consider a Post-Greedy strategy. □

The lower bound for Greedy even holds if all jobs have regular processing time $\tilde{p} = 0$. This models jobs, such as computer programs, that produce an output within milliseconds but, should that output be incorrect, require hours to be debugged. Since this setting may be of particular interest, we propose the name *debugging model*.

Remark 6.1
The Greedy strategy does not improve when restricted to the special case of the *debugging model*, where jobs have regular processing time $\tilde{p} = 0$.

6.2.2. General bounds

We adapt improved strategies for Online Makespan Minimization to the setting of budgeted uncertainty. The algorithm is parameterized with a competitive ratio $c = c_{\Gamma,m}$, depending on both Γ and m. This ratio will be specified later. For input sequence $\mathcal{J}^\sigma = J_1, \ldots, J_n$, let J_t, $1 \leq t \leq n$, be the current job that the online algorithm has to assign to a machine. We order machines by increasing robust loads at time $t - 1$ breaking ties arbitrarily. The $d = \left\lfloor \frac{c-2}{c} m \right\rfloor$ least loaded machines are called *small*; the next d machines are *medium* and the remaining $m - 2d$ most-loaded machines are *large*. Let M^{t-1}_{small} denote the machine of least robust load and M^{t-1}_{med} be

Algorithm 4 *How to schedule job J_t with processing time p_t.*

1: **if** $L^{t-1}_{\text{small}} > \left(1 - \frac{1}{2(c-1)}\right) L^{t-1}_{\text{large}}$ and $l^{t-1}_{\text{med}} + p_t \leq \frac{c}{2} L^{t-1}$ **then**
2: Schedule job J_t on the least loaded medium machine M^{t-1}_{med};
3: **else** Schedule job J_t on the least loaded machine M^{t-1}_{small}.

the medium machine of least robust load or, equivalently, the machine of $(d+1)$th smallest robust load. Let L^{t-1}_{small} denote the average robust load of the small machines, L^{t-1}_{large} the average robust load of large machines and L^{t-1} the average load of all machines.

We call the schedule at time $t-1 \geq 0$ *flat* if $L^{t-1}_{\text{small}} > \left(1 - \frac{1}{2(c-1)}\right) L^{t-1}_{\text{large}}$ and *steep* else. Steep schedules are desirable. We can simply assign job J_t to a least loaded machine. Else, given a flat schedule, the online algorithm aspires to make it steeper again by using a machine of higher load. The medium machine M^{t-1}_{med} is considered. Medium machine M^{t-1}_{med} is risky since it has potentially a very high load. It thus is only chosen if the online algorithm can ensure that the resulting load will not exceed $c\text{OPT}(\mathcal{J})$. Formally, the online algorithm schedules job J_t on the medium machine M^{t-1}_{med} if the schedule is flat, $L^{t-1}_{\text{small}} > \left(1 - \frac{1}{2(c-1)}\right) L^{t-1}_{\text{large}}$, and if $l^{t-1}_{\text{med}} + p_t \leq \frac{c}{2} L^{t-1}$. Here, l^{t-1}_{med} is the load of medium machine M^{t-1}_{med} at time $t-1$. Note that $\frac{1}{2} L^{t-1}$ is a lower bound for OPT by Proposition 6.1.

The values of c.

Recall that $d = \left\lfloor \frac{c-2}{c} m \right\rfloor$. Let $\Gamma \geq 2$. The competitive ratio $c = c_{\Gamma,m}$ is chosen minimally such that $c \geq \frac{7+\sqrt{17}}{4} \approx 2.7808$ and the following holds:

$$\left(1 - \frac{d}{2(c-1)m} - 2\frac{\Gamma+1}{c\Gamma}\right)\left(1 + \frac{c}{2m}\right)^d + 2\frac{\Gamma+1}{c\Gamma} \geq \left(1 - \frac{1}{m}\right).\frac{2}{c-1}. \quad (6.1)$$

Unless m is chosen extremely small, competitive ratio c is determined by Inequality (6.1) and fulfills it with equality. We can show that the competitive ratio c is below 2.9052 for $\Gamma \to \infty$.

The following is the main result of this paper.

Theorem 6.2
The algorithm is c-competitive with $c < 2.9052$ for Γ large.

We establish a lower bound of 2, which shows that the general problem involving budgeted uncertainty is strictly harder than classic Online Makespan Minimization.

Theorem 6.3
No deterministic algorithm is better than 2-competitive for general m and $\Gamma = 2$.

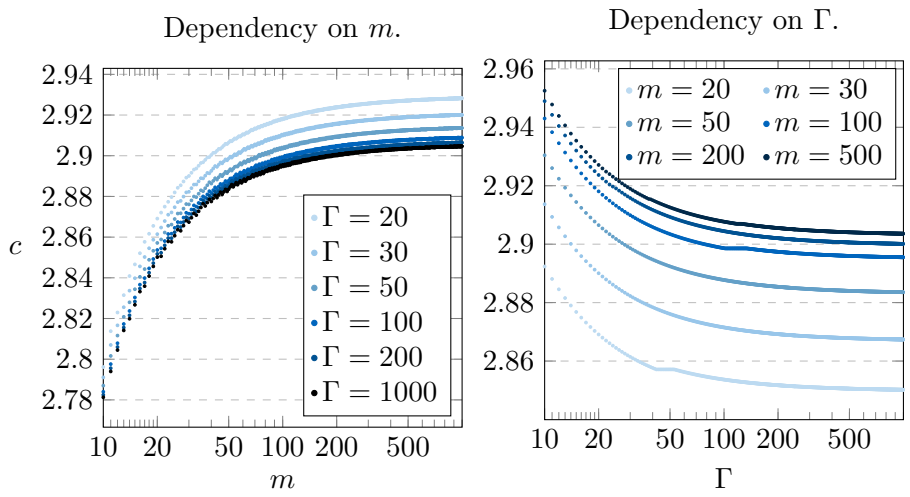


Figure 6.2: The competitive ratios for different m and Γ. The ratio c is monotonously decreasing in Γ and tends to increase in m. The latter improvement is not monotonous due to the rounding involved in d. The x-axes are log-scaled. The graphs are colored.

6.3. Open Problems

The main open problem is to improve the bounds for $m, \Gamma \to \infty$.

Problem 6.1
Improve the competitive ratios in Figure 6.2, in particular for $m, \Gamma \to \infty$.

Of course, it would also be interesting to improve the lower bounds and generalize them to larger Γ.

Problem 6.2
Provide lower bounds for general Γ or generalize the bound of 2 to $\Gamma > 2$.

Finally, we introduced the *debugging model*, which might be of independent interest. Here, jobs only have additional processing time Δp. The regular processing time $\tilde{p}$ are all set to zero. Thus, only the Γ largest on each machine count. Maybe, this model allows for improved performance guarantees. One weakness of the Greedy strategy as well as our improved strategy is that they do not try to cluster "debugging jobs". These jobs have large additional processing times but no, or small, regular processing times. Finding ways to cluster such jobs seems like a sensible avenue to further improvement.

Problem 6.3
Find improved strategies in the debugging model where $\tilde{p} = 0$ for all jobs.

Chapter 7

Machine Covering in the Secretary Model

For Online Machine Covering a set of n jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$ has to be assigned to m parallel and identical machines. Preemption is not allowed. Each job J_i has a non-negative *processing time* p_i. The *load* l_M of machine M is the sum of all processing times of jobs assigned to it. The goal is to minimize the smallest machine load or, equivalently, the time all machines are busy. Machine Covering is applicable when jobs represent resources that allow machines to run. The goal is then to keep the whole system working for as long as possible.

An *online algorithm* A has to schedule jobs one by one. Only after a job is assigned permanently and irrevocably will the next one be revealed. Not knowing the future prevents optimal performance on arbitrary input sequences. Instead, one measures online algorithm A by how well it approximates the optimum result. Each element σ of S_n, the permutation group of the integers 1 to n, describes a way to order the elements in $\mathcal{J} = \{J_1, \ldots, J_n\}$. The resulting job sequence is $\mathcal{J}^\sigma = J_{\sigma(1)}, \ldots, J_{\sigma(n)}$. After online algorithm A treats job sequence $\mathcal{J}^\sigma$ in this order, the minimum load of a machine is denoted by $A(\mathcal{J}^\sigma)$. Let $\mathrm{OPT}(\mathcal{J})$ be the minimum machine load an optimal offline algorithm achieves. Traditionally, the goal is to minimize the *adversarial competitive ratio* $c = \sup_{\mathcal{J},\sigma} \frac{\mathrm{OPT}(\mathcal{J})}{A(\mathcal{J}^\sigma)}$ respectively $c = \sup_{\mathcal{J},\sigma} \frac{\mathrm{OPT}(\mathcal{J})}{\mathbf{E}[A(\mathcal{J}^\sigma)]}$ for randomized algorithms. Here, we use the conventions $0/0 = 1$ and $r/0 = \infty$ for $r > 0$. This adversarial competitive ratio is extremely pessimistic. In a classical lower bound sequence depicted in Figure 2.4, m jobs of size 1 arrive. These jobs have to be assigned to m different machines, otherwise the deterministic online algorithm A has competitive ratio ∞. Subsequent $m-1$ jobs of size m do not allow online algorithm A to improve its minimum load of 1. The optimum offline algorithm, on the other hand, achieves minimum load m. In general, adversarial guarantees are abysmal due to an online algorithm's inability to schedule the first m jobs correctly. This also afflicts randomized algorithms [33]. On the worst-case sequence in [33], the first m jobs are treated correctly with probability at most $1/\sqrt{m}$.

This quagmire of impossibility results furthered the study of various semi-online models that enhance the online algorithm or reveal extra information to provide better performance guarantees. We instead propose a pure online setting, the *secretary model*. The worst-case input set $\mathcal{J}$ is presented to the

online algorithm in a uniformly random order. Each possible job order $\sigma \in S_n$ is picked with equal probability $\frac{1}{n!}$. An online algorithm A is now measured by its *random-order cost* $A^{\text{rom}}(\mathcal{J}) = \mathbf{E}_{\sigma \sim S_n}[A(\mathcal{J}^\sigma)] = \frac{1}{n!}\sum_{\sigma \in S_n} A(\mathcal{J}^\sigma)$. The *competitive ratio of online algorithm A in the secretary model* is $c = \sup_{\mathcal{J}} \frac{\text{OPT}(\mathcal{J})}{A^{\text{rom}}(\mathcal{J})}$, respectively $c = \sup_{\mathcal{J}} \frac{\text{OPT}(\mathcal{J})}{\mathbf{E}[A^{\text{rom}}(\mathcal{J})]}$ if A is randomized. Again, we try to find online algorithms with small competitive ratios.

Throughout this chapter, the number n of jobs is known to the online algorithm in advance. The model studied is thus a *secretary model* under the naming proposed in Section 2.3.

7.1. Our contributions

We first analyze *Greedy*, a sensible benchmark strategy for many Scheduling problems. Greedy simply assigns each job to a least loaded machine. In adversarial settings, Greedy is m-competitive [157], which is optimal as far as deterministic online algorithms are concerned. For random-order arrival the competitive ratio of Greedy improves slightly down to $\Theta(\frac{m}{\log(m)})$. This result is tight up to a factor of $2 + o(1)$.

Theorem 7.1
Greedy is $(2 + o(1))\frac{m}{H_m}$-competitive in the secretary model. Here, H_m denotes the mth harmonic number. This bound cannot be improved up to the factor $2 + o(1)$.

Following the analysis of Greedy, we provide an improved algorithm A, which beats adversarial lower bounds using the secretary model. The improved algorithm is $\tilde{O}(\sqrt[4]{m})$-competitive.

The key challenge for online algorithms is to distinguish small and large jobs. In adversarial deterministic settings, such distinction is simply impossible. For adversarial randomized settings, Azar and Epstein [33] developed a scheme that maintains a random threshold τ to separate small and large jobs. Large jobs are classified correctly with constant probability. An upper bound on the total processing volume of incorrectly labeled small jobs leads their competitive ratio of $\tilde{O}(\sqrt{m})$. Our $\tilde{O}(\sqrt[4]{m})$-competitive approach enhances their strategy by adding a sampling phase. After the sampling phase, most large jobs can be correctly determined without relying on the random threshold τ. This threshold τ can subsequently be updated more aggressively, which in turn improves our ability to identify small jobs.

A more precise description of the $\tilde{O}(\sqrt[4]{m})$-competitive algorithm A follows in the next section.

Theorem 7.2
Algorithm A is $O(\sqrt[4]{m}\log(m))$-competitive in the secretary model.

We finally conclude with a new lower bound. For the lower bound, a

novel version of the Secretary Problem, called the *Talent Contest Problem*, is introduced. The goal is to pick the Kth best among n candidates. Each candidate arrives k times. The arrival order is chosen uniformly at random. Upon each arrival of a candidate, we may make a hiring decision. In particular, information about the candidates increases at later arrivals and hiring decisions become easier. We can prove that good hiring decisions relate to the smallest machine load in a certain Machine Covering instance. Analyzing the Talent Contest Problem therefore allows us to derive a lower bound for Machine Covering in the secretary model.

Theorem 7.3
No deterministic or randomized online algorithm for Machine Covering in the secretary model is better than $O\left(\frac{\log(m)}{\log\log(m)}\right)$-competitive.

7.2. Summary of Techniques and Proof Sketches

Analysis of Greedy

For any $j = 1, \ldots, n$, let P_j denote the jth largest processing time of a job. Let $L_i = \sum_{j \geq i} P_i$ be the *total processing time of jobs smaller than* P_i. This notation uses an implicit tie breaker if there are jobs of equal sizes.

Lemma 7.1
The minimum load achieved by Greedy is at least P_m.

Proof. If the m largest jobs get assigned to different machines, the lemma is immediate. Else, consider the time at which one of the m largest jobs J_t gets assigned to a machine M that already contained another of the m largest jobs J_s, where job J_s has processing time P_j, $j \leq m$. Upon arrival of job J_t, machine M had load at least $P_j \geq P_m$. Per definition, this was the minimum load. The lemma follows as the minimum load cannot decrease later on. □

Lemma 7.2
Let $1 \leq i \leq m$. Then Greedy has minimum load at least $\frac{L_i}{m} - P_i$.

Proof Sketch. Consider the schedule formed by Greedy and let l be the minimum load. For any machine M, let $J_{\text{last}}(M)$ denote the last job it received. Consider $\mathcal{J}_{\text{last}} = \{J_{\text{last}}(M)\}_M$, the set of all last jobs, and remove these last jobs $\mathcal{J}_{\text{last}}$ from Greedy's schedule. After this removal, no machine load exceeds l. Now, add back the $m - i + 1$ smallest jobs in $\mathcal{J}_{\text{last}}$. Each of these jobs had processing time at most P_i. The total load $\tilde{L}$ of the resulting schedule is thus at most $m \cdot l + (m - i + 1)P_i$. The resulting schedule contains, on the other hand, all but $i - 1$ jobs. Thus, $\tilde{L} \geq L_i$. Altogether, we have shown that $L_i \leq \tilde{L} \leq m \cdot l + (m - i + 1)P_i \leq m \cdot l + mP_i$. Rearranging this inequality shows that the Greedy strategy achieved minimum load at least $l \geq \frac{L_i}{m} - P_i$. □

We now conclude Theorem 7.1. We show that the Greedy strategy is $(2+o(1))\frac{m}{H_m}$-competitive in the secretary model.

Proof Sketch of Theorem 7.1. We call a job *large* if its processing time is at least $\frac{H_m}{2m}\mathrm{OPT}$, else it is *small*. Let k be the number of large jobs. If $k \geq m$, we have $P_m \geq \frac{H_m}{2m}\mathrm{OPT}$ and the result follows from Lemma 7.1. Let us thus assume that $k < m$. Since in the optimum schedule at most k machines receive large jobs, $L_{k+1} \geq (m-k)\mathrm{OPT}$.

We subdivide the ordered input sequence $\mathcal{J}^\sigma$ into chunks according to the large jobs. The first chunk $\mathcal{J}_0^\sigma$ starts at the beginning of $\mathcal{J}^\sigma$ and ends right before the first large job $J^{(1)}$ arrived. The ith chunk $\mathcal{J}_i^\sigma$, $1 \leq i \leq k$, starts right after the ith large job $J^{(i)}$ arrived and ends right before the $(i+1)$th large job arrived $J^{(i+1)}$, or at the end of the input if $i = k$. Let $S_i(\mathcal{J}^\sigma)$ denote the total processing time of the jobs in $\mathcal{J}_i^\sigma$. We show next that the minimum load in Greedy's schedule is at least $\min\left(\sum_{j=0}^{k}\frac{S_j(\mathcal{J}^\sigma)}{m-j} - \frac{H_m}{2m}\mathrm{OPT}, \frac{H_m}{2m}\mathrm{OPT}\right)$.

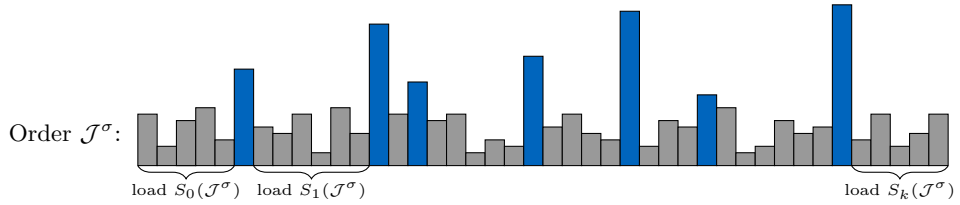


Figure 7.1: The k large jobs (blue) partition the small jobs into segments $\mathcal{J}_i^\sigma$ of different total loads $S_i(\mathcal{J}^\sigma)$. Segment $\mathcal{J}_i^\sigma$ contributes $\frac{S_i(\mathcal{J}^\sigma)}{m-i}$ to Greedy's minimum load.

A machine is *full* once it receives a large job. We may assume that no machine receives another job, once it is full. Otherwise, the minimum load would already be at least $\frac{H_m}{2m}\mathrm{OPT}$. Now, consider the situation right after sequence $\mathcal{J}_{i+1}^\sigma$, $0 \leq i \leq k$, has been scheduled and let $\tilde{L}_i$ be the average load of the non-full machines. First, consider the case $i = 0$. No machine is full, and the total processing volume is $S_0(\mathcal{J}^\sigma)$. The average load $\tilde{L}_0$ is thus $\frac{S_0(\mathcal{J}^\sigma)}{m-0}$. Job $J^{(1)}$ is scheduled on a least loaded machine. Hence, the average load of the non-full machines does not decrease. Now, the $m-1$ non-full machines receive the jobs $\mathcal{J}_1^\sigma$. Their average load increases by $\frac{S_1(\mathcal{J}^\sigma)}{m-1}$. We see that $\tilde{L}_1 \geq \frac{S_1(\mathcal{J}^\sigma)}{m-1} + \tilde{L}_0 = \frac{S_1(\mathcal{J}^\sigma)}{m-1} + \frac{S_0(\mathcal{J}^\sigma)}{m-0}$. Repeating this argument shows that $\tilde{L}_k \geq \sum_{j=0}^{k}\frac{S_j(\mathcal{J}^\sigma)}{m-j}$. Consider the most-loaded non-full machine M after all jobs are scheduled. Its load l is at least $\tilde{L}_k \geq \sum_{j=0}^{k}\frac{S_j(\mathcal{J}^\sigma)}{m-j}$. Let $J = J_{\text{last}}(M)$ be the last job scheduled onto machine M. Recall that job J had processing time less than $\frac{H_m}{2m}\mathrm{OPT}$. Hence, right before receiving it machine M had load at least $l - \frac{H_m}{2m}\mathrm{OPT} \geq \sum_{j=0}^{k}\frac{S_j(\mathcal{J}^\sigma)}{m-j} - \frac{H_m}{2m}\mathrm{OPT}$. Machine M was also the least loaded machine at this time. Thus, the minimum is load at least $\sum_{j=0}^{k}\frac{S_j(\mathcal{J}^\sigma)}{m-j} - \frac{H_m}{2m}\mathrm{OPT}$.

Pick $\sigma \sim S_n$ uniformly at random and consider any fixed small job J. Job J falls into each of the $k+1$ chunks $\mathcal{J}_0^\sigma, \mathcal{J}_1^\sigma, \dots \mathcal{J}_k^\sigma$ with equal probability $\frac{1}{k+1}$. Since the total processing time of small jobs is L_{k+1}, we have $\mathbf{E}_{\sigma\sim S_n}[S_j(\mathcal{J}^\sigma)] = \frac{L_{k+1}}{k+1}$. As mentioned in the beginning, $L_{k+1} \geq (m-k)\mathrm{OPT}$. We conclude $\mathbf{E}_{\sigma\sim S_n}[\sum_{j=0}^{\tilde{k}} \frac{S_j(\mathcal{J}^\sigma)}{m-j} - \frac{H_m}{2m}\mathrm{OPT}] \geq \frac{H_m}{2m}\mathrm{OPT}$.

Unfortunately, this does not immediately lead to a lower bound for $\min\left(\sum_{j=0}^{\tilde{k}} \frac{S_j(\mathcal{J}^\sigma)}{m-j} - \frac{H_m}{2m}\mathrm{OPT}, \frac{H_m}{2m}\mathrm{OPT}\right)$. In the full proof, we also need to consider the variance of $\sum_{j=0}^{\tilde{k}} \frac{S_j(\mathcal{J}^\sigma)}{m-j} - \frac{H_m}{2m}\mathrm{OPT}$ and apply Chebyshev's inequality.

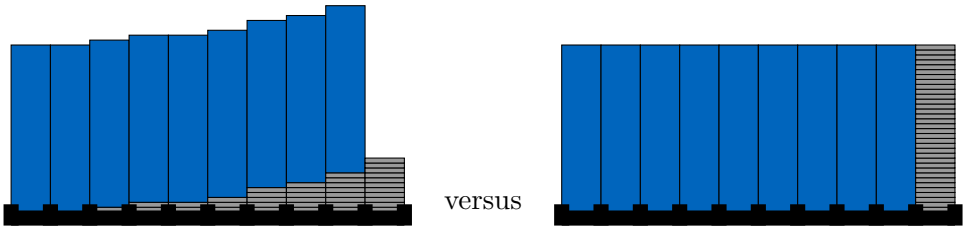


Figure 7.2: The lower bound. Comparison of the Greedy solution (left) and the optimum solution (right) for a random input permutation σ. Large jobs have size 1, small jobs have size $\varepsilon > 0$.

For the lower bound, consider $m-1$ *large* jobs of processing time 1 and $\frac{1}{\varepsilon}$ *small* jobs of tiny processing time $\varepsilon > 0$. Here, $\mathrm{OPT} = 1$. Greedy's schedule contains precisely one machine M_{small} that does not contain a large job. We can adapt the previous arguments to see that machine M_{small} has load at most $\sum_{i=0}^{m-1} \frac{S_i(\mathcal{J}^\sigma)}{m-i} + \varepsilon m$. Again, $\mathbf{E}_{\sigma\sim S_n}[S_i(\mathcal{J}^\sigma)] = \frac{L_m}{m} = \frac{1}{m}$. Thus, machine M_{small} has expected load $\sum_{i=0}^{m-1} \frac{1}{m(m-i)} + \varepsilon m = \frac{H_m}{m} + \varepsilon m$. Greedy's competitive ratio is at most $1/(\frac{H_m}{m} + \varepsilon m)$, which approaches $\frac{m}{H_m}$ for $\varepsilon \to 0$. □

Main Algorithm

Classical algorithms perform bad due to their inability to distinguish small and large jobs. The approach of Azar and Epstein [33] ameliorates this by providing a scheme, which allows to pessimistically estimate large jobs. We improve this approach via sampling. The algorithm first considers the number of large jobs k. If there are at least m large jobs or if there are less than $m - \frac{\sqrt[4]{m^3}}{50}$ large jobs, we use Lemma 7.1 and 7.2 to conclude that Greedy is $O(\sqrt[4]{m})$-competitive.

Else, if $1 \leq m-k \leq \frac{\sqrt[4]{m^3}}{50}$, the algorithm tries to guess the number $m-k$ of machines that need to be filled with small jobs. The algorithm's guess 2^t for $m-k$ is *correct* if $2(m-k) > 2^t \geq m-k$. In general, there

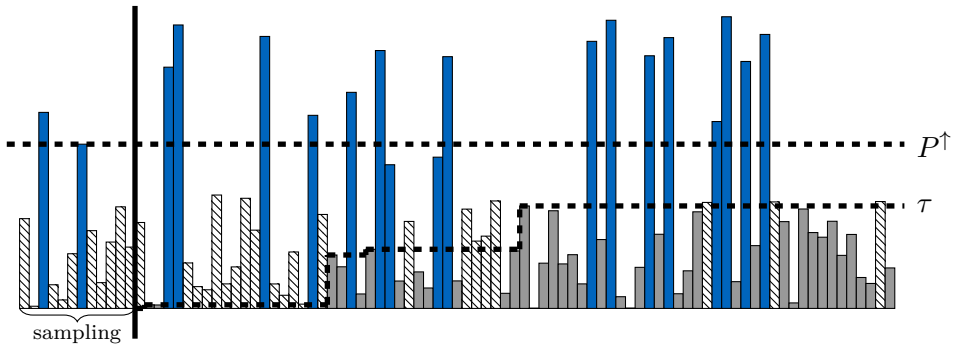


Figure 7.3: The classification of large (blue) and small (light or dashed) jobs forms the main part of the algorithm. During a sampling phase, all small jobs are misclassified (dashed). Afterwards, parameter $P^{\uparrow}$ classifies large jobs and parameter τ classifies small jobs. Normally, jobs in between are conservatively classified as large. Sometimes, with a certain small probability, threshold τ is updated on such jobs and they will be considered small. We need to ensure that this is unlikely to happen at large jobs.

are $O(\log(m))$ possible choices for the integer t, namely $0, 1, \ldots, \left\lceil \frac{3}{4}\log(m) \right\rceil$. Thus, the algorithm can guess correctly with probability $O(\frac{1}{\log(m)})$. Using such a guess 2^t, the machines are subdivided into 2^t machines $\mathcal{M}_{\text{large}}$ that only receive large jobs. The remaining machines $\mathcal{M}_{\text{small}}$ receive small jobs. The online algorithm now has to decide for every arriving job J_t whether it is *large* or *small*. If job J_t is considered large, it will be assigned greedily to the least loaded machine in $\mathcal{M}_{\text{large}}$. Else, if job J_t is small, it will be assigned to the least loaded machine in $\mathcal{M}_{\text{small}}$.

Misclassifying large jobs can be disastrous. It may cause one of the machines in $\mathcal{M}_{\text{large}}$ to remain empty. The algorithm will thus ensure that such misclassification is unlikely. It happens with probability less than $\frac{5}{9}$. The main work goes into classifying as many small jobs correctly as possible. The first $\frac{n}{8}$ jobs of the input are sampled in order to obtain a bound of $P^{\uparrow}$ that classifies all but $\sqrt{m}$ of the large jobs. During sampling, all jobs need to be cautiously classified as large. Afterwards, we run a more aggressive version of the scheme of Azar and Epstein [33] to classify small jobs correctly.

After the sampling phase, threshold $\tau < P^{\uparrow}$ is maintained. Jobs of processing time at most τ are considered small, jobs of processing time at least $P^{\uparrow}$ are considered large. When a job J has processing time p in between these values, $\tau < p < P^{\uparrow}$, threshold τ is updated to p with a certain probability Pr. If τ is updated, $p \leq \tau$ becomes satisfied and job J is considered small; else we cautiously treat it as a large job. Extra care has to be taken in choosing the correct update probability Pr. The correct value, $\frac{1}{9 \cdot 2^t \sqrt{m}}$, depends on both m and t. For this value, we establish that

the algorithm is $O(\sqrt[4]{m})$ competitive if its guess t for $m-k$ is *correct*, i.e. if $2(m-k) > 2^t \geq m-k$. From such an observation, Theorem 7.2 follows.

Algorithm 5 An Online Algorithm for Random-Order Machine Covering

Input: Job sequence $\mathcal{J}^\sigma$, input length $n = |\mathcal{J}|$ and m identical machines.

Guess $t \in \{-1, 0, 1, \ldots, \left\lceil \frac{3}{4}\log(m) \right\rceil\}$ uniformly at random;
if $t = -1$ **then** Run the Greedy and **return** the computed solution;
Partition the machines into 2^t *small* and $m - 2^t$ *large* machines;
Phase 1 – Sampling.
Schedule the first $n/8$ jobs iteratively into a least loaded large machine;
Let $P^\uparrow$ be the $\left(\frac{m-2^t}{8} - \frac{\sqrt{m}}{2}\right)$th largest processing see so far;
Phase 2 – Partition.
$\tau \leftarrow 0$;
for $j = \frac{n}{8} + 1, \ldots, n$ **do**
    if $\tau < p_{\sigma(j)} < P^\uparrow$ **then** Update τ to $p_{\sigma(j)}$ with probability $\frac{1}{9 \cdot 2^t \sqrt{m}}$;
    if $p_{\sigma(j)} \leq \tau$ **then**, Assign job $J_{\sigma(j)}$ to a least loaded small machine;
    else Schedule job $J_{\sigma(j)}$ onto a least loaded large machine;
return the computed solution;

We end in a note on rounding processing times. Our description in the main paper in Appendix E rounds job processing times to powers of two before treating the input sequence. Unlike in adversarial settings [33], this is not strictly necessary. We thus did not include such rounding in the algorithm's description here.

Lower Bound

The main challenge Online Machine Covering algorithms face is to tell apart small and large jobs. On earlier adversarial sequences, such information is hidden. Lower bound sequences simply place the critical jobs at the end of the sequence where the online algorithm only discovers them when it is too late. For random-order arrival this is impossible. Instead of withholding such information, we rather obscure it by adding noise. The algorithm sees the large jobs but cannot safely tell them apart from small ones. This is formalized in the Talent Contest Problem.

The Talent Contest Problem

To a yearly talent show contest n candidates apply. To increase entertainment value one wants to exclude perfect candidates, on the one hand. On the other hand, one needs a suitable candidate to present as a winner. Thus, the Kth best candidate, for some K, should be invited but no better one. Worse

candidates may be invited as one pleases. Each candidate will participate in precisely k trials. Each trial is considered an *arrival*. Upon every arrival, we must decide if we *mark* the candidate or not. If we mark a candidate, we consider her to be the Kth best or worse. The order of arrivals is uniformly randomly distributed. The goal is to maximize the number of trials for which the Kth best candidate has been marked but none of the better ones.

Formally, the *Talent Contest Problem* is specified by the *rank of the winner* K, *the number of arrivals* k and *the number of candidates* n, where $n \geq K$, as well as positive *candidate valuations* $v_1, v_2, \ldots, v_n$. Each candidate arrives k times. The arrival order is chosen uniformly at random. When a candidate arrives for the first time, her valuation is revealed. We may decide to mark each arrival or not, though the next candidate only arrives once such a marking decision has been made permanent and cannot be changed later on. Consider all the hth arrivals of candidates, $1 \leq h \leq k$. If we successfully marked the hth arrival of the Kth candidate but no hth arrival of a better candidate, we win a point. In total, we can win up to h points. Let $P(K, k, n)$ be the expected number of points an optimal online algorithm scores for the three values K, k and n on general valuations $v_1, \ldots, v_n$. Similar to the classical Secretary Problem, we are mostly interested in the limit value $P(K, k) = \lim_{n\to\infty} P(K, k, n)$.

In order to ensure transferability to Machine Covering, we require an extra technicality. Given $\lambda \geq 1$, we call the valuations $v_1, v_2, \ldots, v_n$ of candidates λ-*steep* if they all differ by a factor of at least λ, i.e. there exist no i, j such that $v_i < v_j < \lambda v_i$.

We can establish the following hardness result for the Talent Contest Problem.

Lemma 7.3
$P(K, k) \leq \frac{\zeta(k/2)\cdot(k+1)^{k/2}}{2\pi\sqrt{K}}$ where ζ is the Riemann Zeta Function. This bound even holds when restricted to λ-steep valuations for any $\lambda \geq 1$.

Proof Sketch. We first show that any online algorithm, which performs well in the Talent Contest Problem can be used to predict the value of a binomial distributed random variable. For the latter problem the best strategy is to guess the mode and a hardness result can be obtained. For the reduction to work, we need to first pass over to a "stable" set of input orders. These exclude certain malformed orders. The property of λ-steepness can be ensured by applying the transformation $v \mapsto \mu^v$, for μ sufficiently high, to the initial valuations. □

Reduction to Machine Covering

Our primary reason to study the talent contest Problem is that it relates to Machine Covering.

Lemma 7.4
Let $m = (K-1)\cdot k+1$ for some K and k. Then no (possibly randomized) algorithm for Machine Covering on m machines is better than $\frac{k}{P(K,k)+1}$-competitive in the secretary model.

Setting $K = (k+1)^k$ and combining Lemma 7.3 and 7.4, we can show that no algorithm for Machine Covering in the secretary model, deterministic or randomized, is better than $\frac{\lceil e^{W(\ln(m))}\rceil - 1}{1.16+o(1)}$-competitive. Here, $W(x)$ is the upper branch of the Lambert W-function, the inverse of $x \mapsto xe^x$. The main result, Theorem 7.3, follows by observing that $\frac{\lceil e^{W(\ln(m))}\rceil - 1}{1.16+o(1)} \in \Theta(\frac{\log(m)}{\log\log(m)})$.

7.3. Open Problems

The natural open problem is to improve the bounds from our result.

Problem 7.1
Find an algorithm that is better than $\tilde{O}(\sqrt[4]{m})$-competitive or a lower bound better than $O(\frac{\log(m)}{\log\log(m)})$.

A second problem could be to consider the pure random-order setting where the number n of jobs is not known in advance.

Problem 7.2
Find an algorithm that is better than $\tilde{O}(\sqrt{m})$-competitive without knowing the total number n of jobs in advance.

Finally, the Talent Contest Problem may be of independent interest.

Problem 7.3
Find better bounds on the number $P(K,k)$ defined above for the Talent Contest Problem.

Bibliography

[1] S. Albers. Improved randomized on-line algorithms for the list update problem. *SIAM Journal on Computing*, 27(3):682–693, 1998.

[2] S. Albers. Better bounds for online scheduling. *SIAM Journal on Computing*, 29(2):459–473, 1999.

[3] S. Albers. On randomized online scheduling. In *24th annual ACM symposium on Theory of computing*, pages 134–143, 2002.

[4] S. Albers. Online algorithms. In *Interactive Computation*, pages 143–164. Springer, 2006.

[5] S. Albers. Online scheduling. In *Introduction to scheduling*, pages 71–98. CRC Press, 2009.

[6] S. Albers. Recent advances for a classical scheduling problem. In *International Colloquium on Automata, Languages, and Programming*, pages 4–14, 2013.

[7] S. Albers and A. Eckl. Explorable uncertainty in scheduling with non-uniform testing times. In *Workshop on Approximation and Online Algorithms 2020*, 2020.

[8] S. Albers, W. Gálvez, and M. Janke. Machine covering in the random-order model. In *32nd International Symposium on Algorithms and Computation*, 2021.

[9] S. Albers and M. Hellwig. On the value of job migration in online makespan minimization. In *European Symposium on Algorithms*, pages 84–95, 2012.

[10] S. Albers and M. Hellwig. Semi-online scheduling revisited. *Theoretical Computer Science*, 443:1–9, 2012.

[11] S. Albers and M. Hellwig. Online makespan minimization with parallel schedules. *Algorithmica*, 78(2):492–520, 2017.

[12] S. Albers and M. Janke. New bounds for randomized list update in the paid exchange model. In *37th International Symposium on Theoretical Aspects of Computer Science*, 2020.

[13] S. Albers and M. Janke. Online makespan minimization with budgeted uncertainty. In *17th Algorithms and Data Structures Symposium*, 2021.

[14] S. Albers and M. Janke. Scheduling in the random-order model. *Algorithmica*, pages 1–30, 2021.

[15] S. Albers and M. Janke. Scheduling in the secretary model. In *41st annual Conference on Foundations of Software Technology and Theoretical Computer Science*, 2021.

[16] S. Albers, A. Khan, and L. Ladewig. Improved online algorithms for knapsack and gap in the random order model. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, 2019.

[17] S. Albers, A. Khan, and L. Ladewig. Best fit bin packing with random order revisited. *Algorithmica*, pages 1–26, 2021.

[18] S. Albers and L. Ladewig. New results for the k-secretary problem. *Theoretical Computer Science*, 863:102–119, 2021.

[19] S. Albers and S. Lauer. On list update with locality of reference. *Journal of Computer and System Sciences*, 82(5):627–653, 2016.

[20] S. Albers, B. von Stengel, and R. Werchner. A combined BIT and TIMESTAMP algorithm for the list update problem. *Information Processing Letters*, 56(3):135–139, 1995.

[21] S. Albers and J. Westbrook. Self-organizing data structures. In *Online Algorithms, The State of the Art*, Springer LNCS 1442, pages 13–51, 1998.

[22] C. Ambühl. Offline list update is np-hard. In *European Symposium on Algorithms*, pages 42–51, 2000.

[23] C. Ambühl. *On the list update problem.* PhD thesis, ETH Zurich, 2002.

[24] C. Ambühl, B. Gärtner, and B. Von Stengel. A new lower bound for the list update problem in the partial cost model. *Theoretical Computer Science*, 268(1):3–16, 2001.

[25] C. Ambühl, B. Gärtner, and B. von Stengel. Optimal lower bounds for projective list update algorithms. *ACM Transactions on Algorithms*, 9(4):31:1–31:18, 2013.

[26] S. Angelopoulos, C. Dürr, S. Jin, S. Kamali, and M. Renault. Online computation with untrusted advice. In *11th annual Innovations in Theoretical Computer Science*, 2020.

[27] S. Angelopoulos and P. Schweitzer. Paging and list update under bijective analysis. *Journal of the ACM*, 60(2):7:1–7:18, 2013.

[28] A. Asadpour and A. Saberi. An approximation algorithm for max-min fair allocation of indivisible goods. *SIAM Journal on Computing*, 39(7):2970–2989, 2010.

[29] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. In *25th annual ACM symposium on Theory of computing*, pages 623–631, 1993.

[30] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line routing of virtual circuits with applications to load balancing and machine scheduling. *Journal of the ACM*, 44(3):486–504, 1997.

[31] A. Avidor, Y. Azar, and J. Sgall. Ancient and new algorithms for load balancing in the lp norm. *Algorithmica*, 29(3):422–441, 2001.

[32] P. Azar, R. Kleinberg, and S. Weinberg. Prophet inequalities with limited information. *24th annual ACM-SIAM Symposium on Discrete Algorithms*, 07 2013.

[33] Y. Azar and L. Epstein. On-line machine covering. *Journal of Scheduling*, 1:67–77, 1998.

[34] Y. Azar, J. Naor, and R. Rom. The competitiveness of on-line assignments. *Journal of Algorithms*, 18(2):221–237, 1995.

[35] Y. Azar and O. Regev. On-line bin-stretching. *Theoretical Computer Science*, 268(1):17–41, 2001.

[36] M. Babaioff, N. Immorlica, D. Kempe, and R. Kleinberg. A knapsack secretary problem with applications. In *Approximation, randomization, and combinatorial optimization. Algorithms and techniques*, pages 16–28. Springer, 2007.

[37] J. Balogh, J. Békési, G. Dósa, L. Epstein, and A. Levin. A new and improved algorithm for online bin packing. *arXiv preprint arXiv:1707.01728*, 2017.

[38] N. Bansal, N. Buchbinder, A. Madry, and J. Naor. A polylogarithmic-competitive algorithm for the k-server problem. In *2011 IEEE 52nd annual Symposium on Foundations of Computer Science*, pages 267–276, 2011.

[39] N. Bansal and M. Sviridenko. The santa claus problem. In Jon M. Kleinberg, editor, *38th annual ACM Symposium on Theory of Computing*, pages 31–40, 2006.

[40] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. In *24th annual ACM symposium on Theory of computing*, pages 51–58, 1992.

[41] S. Ben-David and A. Borodin. A new measure for the study of on-line algorithms. *Algorithmica*, 11(1):73–91, 1994.

[42] P. Berman, M. Charikar, and M. Karpinski. On-line load balancing for related machines. *Journal of Algorithms*, 35(1):108–121, 2000.

[43] D. Bertsimas and M. Sim. Robust discrete optimization and network flows. *Mathematical programming*, 98(1-3):49–71, 2003.

[44] D. Bertsimas and M. Sim. The price of robustness. *Operations research*, 52(1):35–53, 2004.

[45] M. Böhm, J. Sgall, R. Van Stee, and P. Veselỳ. A two-phase algorithm for bin stretching with stretching factor 1.5. *Journal of Combinatorial Optimization*, 34(3):810–828, 2017.

[46] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

[47] M. Bougeret, K. Jansen, M. Poss, and L. Rohwedder. Approximation results for makespan minimization with budgeted uncertainty. In *International Workshop on Approximation and Online Algorithms*, pages 60–71, 2019.

[48] M. Bougeret, A. Pessoa, and M. Poss. Robust scheduling with budgeted uncertainty. *Discrete Applied Mathematics*, 261:93–107, 2019.

[49] J. Boyar, S. Kamali, K.S. Larsen, and A. López-Ortiz. On the list update problem with advice. *Information and Computation*, 253:411–423, 2017.

[50] M. Burrows and D. Wheeler. A block sorting lossless data compression algorithm. *Technical Report 124*, 1994.

[51] B. Chen, A. van Vliet, and G. Woeginger. A lower bound for randomized on-line scheduling algorithms. *Information Processing Letters*, 51(5):219–222, 1994.

[52] L. Chen, K. Jansen, and G. Zhang. On the optimality of approximation schemes for the classical scheduling problem. In *25th annual ACM-SIAM symposium on Discrete algorithms*, pages 657–668, 2014.

[53] L. Chen, D. Ye, and G. Zhang. Approximating the optimal algorithm for online scheduling problems via dynamic programming. *Asia-Pacific Journal of Operational Research*, 32(01):1540011, 2015.

[54] T. Cheng, H. Kellerer, and V. Kotov. Semi-on-line multiprocessor scheduling with given total processing time. *Theoretical computer science*, 337(1-3):134–146, 2005.

[55] T. Cheng, H. Kellerer, and V. Kotov. Algorithms better than lpt for semi-online scheduling with decreasing processing times. *Operations Research Letters*, 40:349–352, 2012.

[56] I. Cohen, S. Im, and D. Panigrahi. Online two-dimensional load balancing. In *47th International Colloquium on Automata, Languages, and Programming*, 2020.

[57] J. Correa, A. Cristi, B. Epstein, and J. Soto. The two-sided game of googol and sample-based prophet inequalities. In *14th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2066–2081, 2020.

[58] J. Correa, A. Cristi, L. Feuilloley, T. Oosterwijk, and A. Tsigonias-Dimitriadis. The secretary problem with independent sampling. In *32th ACM-SIAM Symposium on Discrete Algorithms*, pages 2047–2058, 2021.

[59] J. Correa, P. Dütting, F. Fischer, and K. Schewior. Prophet inequalities for iid random variables from an unknown distribution. In *20th ACM Conference on Economics and Computation*, pages 3–17, 2019.

[60] M. Crochemore, R. Grossi, J. Kärkkäinen, and G.M. Landau. Computing the Burrows-Wheeler transform in place and in small space. *Journal of Discrete Algorithms*, 32:44–52, 2015.

[61] G. Dantzig. Linear programming under uncertainty. *Management science*, 1(3-4):197–206, 1955.

[62] P. Diaconis and S. Zabell. Closed form summation for classical distributions: variations on a theme of de moivre. *Statistical Science*, pages 284–302, 1991.

[63] M. Dinitz. Recent advances on the matroid secretary problem. *ACM SIGACT News*, 44(2):126–142, 2013.

[64] J. Dohrau. Online makespan scheduling with sublinear advice. In *International Conference on Current Trends in Theory and Practice of Informatics*, pages 177–188, 2015.

[65] R. Dorrigiv, M.R. Ehmsen, and A. López-Ortiz. Parameterized analysis of paging and list update algorithms. *Algorithmica*, 71(2):330–353, 2015.

[66] C. Dürr, T. Erlebach, N. Megow, and J. Meißner. Scheduling with explorable uncertainty. In *9th Innovations in Theoretical Computer Science Conference*, 2018.

[67] C. Dürr, T. Erlebach, N. Megow, and J. Meißner. An Adversarial Model for Scheduling with Testing. *Algorithmica*, pages 1–46, 2020.

[68] E. B. Dynkin. The optimum choice of the instant for stopping a Markov process. *Soviet Mathematics*, 4:627–629, 1963.

[69] T. Ebenlendr, J. Noga, J. Sgall, and G. Woeginger. A note on semi-online machine covering. In *Approximation and Online Algorithms, Third International Workshop*, volume 3879, pages 110–118, 2005.

[70] T. Ebenlendr and J. Sgall. A lower bound on deterministic online algorithms for scheduling on related machines without preemption. *Theory of Computing Systems*, 56(1):73–81, 2015.

[71] R. El-Yaniv. *There are infinitely many competitive-optimal online list accessing algorithms.* Hebrew University of Jerusalem, 1996.

[72] M. Englert, D. Mezlaf, and M. Westermann. Online makespan scheduling with job migration on uniform machines. *Algorithmica*, pages 1–30, 2021.

[73] M. Englert, D. Özmen, and M. Westermann. The power of reordering for online minimum makespan scheduling. In *2008 49th annual IEEE Symposium on Foundations of Computer Science*, pages 603–612, 2008.

[74] L. Epstein. A survey on makespan minimization in semi-online environments. *Journal of Scheduling*, 21(3):269–284, 2018.

[75] L. Epstein and A. Levin. Robust algorithms for preemptive scheduling. *Algorithmica*, 69(1):26–57, 2014.

[76] L. Epstein, A. Levin, and R. van Stee. Max-min online allocations with a reordering buffer. *SIAM Journal on Discrete Mathematics*, 25(3):1230–1250, 2011.

[77] T. Erlebach, M. Hoffmann, and F. Kammer. Query-competitive algorithms for cheapest set problems under uncertainty. *Theoretical Computer Science*, 613:51–64, 2016.

[78] U. Faigle, W. Kern, and G. Turán. On the performance of on-line algorithms for partition problems. *Acta cybernetica*, 9(2):107–119, 1989.

[79] T. Feder, R. Motwani, R. Panigrahy, C. Olston, and J. Widom. Computing the median with uncertainty. In *32nd annual ACM symposium on Theory of computing*, pages 602–607, 2000.

[80] M. Feldman, O. Svensson, and R. Zenklusen. A simple O (log log (rank))-competitive algorithm for the matroid secretary problem. In *26th annual ACM-SIAM symposium on Discrete algorithms*, pages 1189–1201, 2014.

[81] T. S. Ferguson. Who solved the secretary problem? *Statistical science*, 4(3):282–289, 1989.

[82] A. Fiat, R. Karp, M. Luby, L. McGeoch, D. Sleator, and N. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.

[83] A. Fiat and G. Woeginger. *Online algorithms: The state of the art*. Springer, 1998.

[84] R. Fleischer and M. Wahl. On-line scheduling revisited. *Journal of Scheduling*, 3(6):343–353, 2000.

[85] D. Fotakis, L. Kavouras, G. Koumoutsos, S. Skoulakis, and M. Vardas. The online min-sum set cover problem. In *47th International Colloquium on Automata, Languages, and Programming*, 2020.

[86] PR Freeman. The secretary problem and its extensions: A review. *International Statistical Review/Revue Internationale de Statistique*, pages 189–206, 1983.

[87] D. Friesen and B. Deuermeyer. Analysis of greedy solutions for a replacement part sequencing problem. *Mathematics of Operations Research*, 6(1):74–87, 1981.

[88] M. Gabay, V. Kotov, and N. Brauner. Online bin stretching with bunch techniques. *Theoretical Computer Science*, 602:103–113, 2015.

[89] G. Galambos and G. Woeginger. An on-line scheduling heuristic with better worst-case ratio than Graham's list scheduling. *SIAM Journal on Computing*, 22(2):349–355, 1993.

[90] W. Gálvez, J. A Soto, and J. Verschae. Symmetry exploitation for online machine covering with bounded migration. *ACM Transactions on Algorithms*, 16(4):1–22, 2020.

[91] O. Göbel, T. Kesselheim, and A. Tönnis. Online appointment scheduling in the random order model. In *23rd European Symposia on Algorithms*, pages 680–692. Springer, 2015.

[92] G. Goel and A. Mehta. Online budgeted matching in random input models with applications to Adwords. In *19th annual ACM-SIAM symposium on Discrete algorithms*, volume 8, pages 982–991, 2008.

[93] T. Gormley, N. Reingold, E. Torng, and J. Westbrook. Generating adversaries for request-answer games. In *11th annual ACM-SIAM symposium on Discrete algorithms*, pages 564–565, 2000.

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

[95] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal of Applied Mathematics*, 17:416–429, 1969.

[96] E. Günther, O. Maurer, N. Megow, and A. Wiese. A new approach to online scheduling: Approximating the optimal competitive ratio. In *24th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 118–128, 2013.

[97] A. Gupta, R. Mehta, and M. Molinaro. Maximizing profit with convex costs in the random-order model. In *45th International Colloquium on Automata, Languages, and Programming*, page 71, 2018.

[98] A. Gupta and S. Singla. Random-order models. In T. Roughgarden, editor, *Beyond worst-case analysis*. Cambridge University Press, 2020.

[99] M. Gupta, Y. Sabharwal, and S. Sen. The update complexity of selection and related problems. *Theory of Computing Systems*, 59(1):112–132, 2016.

[100] D. Hochbaum and D. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM*, 34(1):144–162, 1987.

[101] S. Im, N. Kell, J. Kulkarni, and D. Panigrahi. Tight bounds for online vector scheduling. *SIAM Journal on Computing*, 48(1):93–121, 2019.

[102] S. Im, N. Kell, D. Panigrahi, and M. Shadloo. Online load balancing on related machines. In *50th annual ACM SIGACT Symposium on Theory of Computing*, pages 30–43, 2018.

[103] S. Irani. Two results on the list update problem. *Information Processing Letters*, 38(6):301–306, 1991.

[104] K. Jansen, K. Klein, and J. Verschae. Closing the gap for makespan scheduling via sparsification techniques. *Mathematics of Operations Research*, 45(4):1371–1392, 2020.

[105] D. Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8(3):272–314, 1974.

[106] S. Kahan. A model for data in motion. In *23rd annual ACM symposium on Theory of computing*, pages 265–277, 1991.

[107] S. Kamali, S. Ladra, A. López-Ortiz, and D. Seco. Context-based algorithms for the list-update problem under alternative cost models. In *2013 Data Compression Conference*, IEEE, pages 361–370, 2013.

[108] S. Kamali and A. López-Ortiz. A survey of algorithms and models for list update. In *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of his 66th Birthday*, pages 251–266, 2013.

[109] S. Kamali and A. López-Ortiz. Better compression through better list update algorithms. In *2014 Data Compression Conference*, IEEE, pages 372–381, 2014.

[110] H. Kaplan, D. Naori, and D. Raz. Competitive Analysis with a Sample and the Secretary Problem. In *14th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2082–2095, 2020.

[111] C. Karande, A. Mehta, and P. Tripathi. Online bipartite matching with unknown distributions. In *23rd annual ACM symposium on Theory of computing*, pages 587–596, 2011.

[112] D. Karger, S. Phillips, and E. Torng. A better algorithm for an ancient scheduling problem. *Journal of Algorithms*, 20(2):400–430, 1996.

[113] A. Karlin, M. Manasse, Lyle A. McGeoch, and S. Owicki. Competitive randomized algorithms for nonuniform problems. *Algorithmica*, 11(6):542–571, 1994.

[114] A. Karlin, M. Manasse, L. Rudolph, and D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):79–119, 1988.

[115] R. Karp and P. Raghavan. Personal communication cited in [143].

[116] R. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. In *22nd annual ACM symposium on Theory of computing*, pages 352–358, 1990.

[117] H. Kellerer and V. Kotov. An efficient algorithm for bin stretching. *Operations Research Letters*, 41(4):343–346, 2013.

[118] H. Kellerer, V. Kotov, and M. Gabay. An efficient algorithm for semi-online multiprocessor scheduling with given total processing time. *Journal of Scheduling*, 18(6):623–630, 2015.

[119] H. Kellerer, V. Kotov, M. Speranza, and Tuza Z. Semi on-line algorithms for the partition problem. *Operations Research Letters*, 21(5):235–242, 1997.

[120] C. Kenyon. Best-Fit Bin-Packing with Random Order. In *7th annual ACM-SIAM symposium on Discrete algorithms*, volume 96, pages 359–364, 1996.

[121] T. Kesselheim, A. Tönnis, K. Radke, and B. Vöcking. Primal beats dual on online packing LPs in the random-order model. In *26th annual ACM symposium on Theory of computing*, pages 303–312, 2014.

[122] S. Khanna and W. Tan. On computing functions with uncertainty. In *20th ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 171–182, 2001.

[123] R. Kleinberg. A multiple-choice secretary algorithm with applications to online auctions. In *16th annual ACM-SIAM symposium on Discrete algorithms*, volume 5, pages 630–631, 2005.

[124] N. Korula, V. Mirrokni, and M. Zadimoghaddam. Online submodular welfare maximization: Greedy beats 1/2 in random order. *SIAM Journal on Computing*, 47(3):1056–1086, 2018.

[125] E. Koutsoupias and C. Papadimitriou. On the k-server conjecture. *Journal of the ACM*, 42(5):971–983, 1995.

[126] O. Lachish. O (log log rank) competitive ratio for the matroid secretary problem. In *2014 IEEE 55th annual Symposium on Foundations of Computer Science*, pages 326–335, 2014.

[127] L. Ladewig. *Online Algorithms for Packing Problems in the Random-Order Model*. PhD thesis, Universität München, 2021.

[128] D. Lindley. Dynamic programming and decision theory. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 10(1):39–51, 1961.

[129] A. López-Ortiz, M.P. Renault, and A. Rosén. Paid exchanges are worth the price. In *32nd International Symposium on Theoretical Aspects of Computer Science*, LIPIcs 30, pages 636–648, 2015.

[130] M. Mahdian and Q. Yan. Online bipartite matching with random arrivals: an approach based on strongly factor-revealing lps. In *43rd annual ACM symposium on Theory of computing*, pages 597–606, 2011.

[131] M. Manasse, L. McGeoch, and D. Sleator. Competitive algorithms for on-line problems. In *Proc. 20th annual ACM Symposium on Theory of Computing*, pages 322–333, 1988.

[132] G. Manzini. An analysis of the Burrows-Wheeler transform. *Journal of the ACM*, 48(3):407–430, 2001.

[133] J. McCabe. On serial files with relocatable records. *Operations Research*, 12:609–618, 1965.

[134] A. Meyerson. Online facility location. In *42nd IEEE Symposium on Foundations of Computer Science*, pages 426–431, 2001.

[135] V. Mirrokni, S. Gharan, and M. Zadimoghaddam. Simultaneous approximations for adversarial and stochastic online budgeted allocation. In *23rd annual ACM-SIAM symposium on Discrete Algorithms*, pages 1690–1701, 2012.

[136] M. Molinaro. Online and random-order load balancing simultaneously. In *28th annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1638–1650, 2017.

[137] C. Olston and J. Widom. Offering a precision-performance tradeoff for aggregation queries over replicated data. Technical report, Stanford, 2000.

[138] Neil Olver, Kirk Pruhs, Kevin Schewior, René Sitters, and Leen Stougie. The itinerant list update problem. In *Approximation and Online Algorithms: 16th International Workshop, WAOA 2018, Helsinki, Finland, August 23-24, 2018, Revised Selected Papers*, pages 310–326. Springer, 2018.

[139] C. Osborn and E. Torng. List's worst-average-case or WAC ratio. *Journal of Scheduling*, 11(3):213–215, 2008.

[140] J. Petruccelli. Best-choice problems involving uncertainty of selection and recall of observations. *Journal of Applied Probability*, 18(2):415–425, 1981.

[141] K. Pruhs, J. Sgall, and E. Torng. *Online scheduling.* CRC Press, 2003.

[142] N. Reingold and J. Westbrook. Off-line algorithms for the list update problem. *Information Processing Letters*, 60(2):75–80, 1996.

[143] N. Reingold, J. Westbrook, and D. Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica*, 11(1):15–32, 1994.

[144] A. Roy, M. Bougeret, N. Goldberg, and M. Poss. Approximating robust bin packing with budgeted uncertainty. In *Workshop on Algorithms and Data Structures*, pages 71–84, 2019.

[145] J. Rudin III. *Improved bounds for the on-line scheduling problem.* PhD thesis, University of Phoenix, 2001.

[146] J. Rudin III and R. Chandrasekaran. Improved bounds for the online scheduling problem. *SIAM Journal on Computing*, 32(3):717–735, 2003.

[147] S. Samuels. Minimax stopping rules when the underlying distribution is uniform. *Journal of the American Statistical Association*, 76(373):188–197, 1981.

[148] P. Sanders, N. Sivadasan, and M. Skutella. Online scheduling with bounded migration. *Mathematics of Operations Research*, 34(2):481–498, 2009.

[149] P. Sanders, N. Sivadasan, and M. Skutella. Online scheduling with bounded migration. *Mathematics of Operations Research*, 34(2):481–498, 2009.

[150] S. Seiden, J. Sgall, and G. Woeginger. Semi-online scheduling with decreasing job sizes. *Operations Research Letters*, 27:215–221, 2000.

[151] J. Sgall. A lower bound for randomized on-line multiprocessor scheduling. *Information Processing Letters*, 63(1):51–55, 1997.

[152] J. Sirén. Burrows-Wheeler transform for terabases. In *2016 Data Compression Conference*, IEEE, pages 211–220, 2016.

[153] D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[154] T. Stewart. Optimal selection from a random sequence with learning of the underlying distribution. *Journal of the American Statistical Association*, 73(364):775–780, 1978.

[155] B. Tadayon and J. Smith. Algorithms and complexity analysis for robust single-machine scheduling problems. *Journal of Scheduling*, 18(6):575–592, 2015.

[156] B. Teia. A lower bound for randomized list update algorithms. *Information Processing Letters*, 47(1):5–9, 1993.

[157] G. Woeginger. A polynomial-time approximation scheme for maximizing the minimum machine completion time. *Operation Research Letters*, 20:149–154, 1997.

Appendix A

Randomized List Update in the Paid Exchange Model

Bibliographic Information *The List-Update Problem in the Paid Exchange Model.* S. Albers, M. Janke. 37th Symposium on Theoretical Aspects of Computer Science (**STACS**) 2020

Summary of Contributions The List Update problem in the paid exchange model, or P^d model, is studied in depth. A given item list has to be maintained using paid exchanges. Each exchange incurs general cost d. Free exchanges are not allowed. This resembles realistic list implementations, where rearrangement is more expensive than simple traversal.

A generalization of the TIMESTAMP algorithm from [1] is analyzed, which is 2.2442-competitive. The analysis introduces a novel approach of phase subdivision and cost shifting to master requests.

This result is complemented by new lower bounds for randomized algorithms. Such bounds were not known before. All algorithms in the literature are analyzed, in fact, in the partial cost model, where the cost for serving each request is reduced by 1. We provide a general lower bound of 2 for randomized algorithms in the partial cost model. We also show that no randomized algorithm can be better than 1.8654-competitive.

Individual Contributions

- Development of the upper bound. In particular, proposal to study TIMESTAMP(l,p) in randomized settings, introduction of refined costs and analysis leading to the upper bound.
- Development of the lower bound. In particular, the creation of the randomized sequence $\mathcal{S}_N$, as well as evaluation of OPT on two-item-lists and of algorithm family B_h for general lists.
- Composition of the first draft of the manuscript excluding introduction and abstract. Included are all other technical and non-technical parts (later significantly revised and improved by S. Albers).

Nearly Tight Bounds for Randomized List Update in the Paid Exchange Model

Susanne Albers
Department of Computer Science, Technical University of Munich
albers@in.tum.de

Maximilian Janke
Department of Computer Science, Technical University of Munich
maximilian.janke@in.tum.de

Abstract

We study the fundamental list update problem in the paid exchange model P^d. This cost model was introduced by Manasse, McGeoch and Sleator [17] and Reingold, Westbrook and Sleator [22]. Here the given list of items may only be rearranged using paid exchanges; each swap of two adjacent items in the list incurs a cost of d. Free exchanges of items are not allowed. The model is motivated by the fact that, when executing search operations on a data structure, key comparisons are less expensive than item swaps.

We develop a new randomized online algorithm that achieves an improved competitive ratio against oblivious adversaries. For large d, the competitiveness tends to 2.2442. Technically, the analysis of the algorithm relies on a new approach of partitioning request sequences and charging expected cost. Furthermore, we devise lower bounds on the competitiveness of randomized algorithms against oblivious adversaries. No such lower bounds were known before. Specifically, we prove that no randomized online algorithm can achieve a competitive ratio smaller than 2 in the partial cost model, where an access to the i-th item in the current list incurs a cost of $i - 1$ rather than i. All algorithms proposed in the literature attain their competitiveness in the partial cost model. Furthermore, we show that no randomized online algorithm can achieve a competitive ratio smaller than 1.8654 in the standard full cost model. Again the lower bounds hold for large d.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases self-organizing lists, online algorithm, competitive analysis, lower bound

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.

37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020).
Editors: Markus Bläser and Christophe Paul; Article No. ; pp. :1–:36

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In this paper we revisit the *list update problem*, one of the most basic problems in the theory of online algorithms [7, 24]. The goal is to maintain an unsorted linear linked list of items so that a sequence of accesses to these items can be served with low total cost. Unsorted linear lists are sensible when one has to store a small dictionary consisting of a few dozen items. Moreover, they have interesting applications in data compression. In fact, the standard compression program `bzip2` relies on a combination of the Burrows-Wheeler transform and linear list encoding [9, 10, 18, 23].

Early work on the list update problem dates back to the 1960s [20]. Over the past decades an extensive body of literature has been developed, see e.g. [1, 2, 5, 6, 7, 8, 11, 12, 22, 24] and references therein. List update in the *standard model* is well understood. In this setting the cost of an access is equal to the depth of the referenced item in the current list. Immediately after an access the requested item may be moved at no extra cost to any position closer to the front of the list (free exchanges). Any other swap of two adjacent items in the list incurs a cost of 1 and is called a *paid exchange*. During the last years, research on the list update problem has explored (1) alternative cost models [13, 14, 19, 21], (2) refined input models capturing locality of reference [2, 6, 11], and (3) the value of algorithmic service abilities [8, 14, 16].

We investigate the list update problem in the P^d *model*, i.e. the paid exchange model, introduced by Manasse, McGeoch and Sleator [17] as well as Reingold, Westbrook and Sleator [22]. In this model there are no free exchanges and each paid exchange, swapping a pair of adjacent items in the list, incurs a cost of d, where $d \geq 1$ is a real-valued constant. The model is motivated by the fact that the execution time of a program swapping a pair of adjacent items in the list is typically much higher than that of the program doing one iteration of the search loop. Moreover, bringing a referenced element closer to the front of the list does incur cost. As main result we develop nearly tight upper and lower bounds on the competitive ratio achieved by randomized online algorithms.

Problem formulation: In the list update problem we are given an unsorted linear linked list L of n items. An algorithm is presented with a sequence $\sigma = \sigma(1), \ldots, \sigma(m)$ of requests that must be served in the order of occurrence. Each request $\sigma(t)$, $1 \leq t \leq m$, specifies an item in the list. In order to serve a request, an algorithm has to access the requested item, i.e. it has to start at the front of the list and search linearly through the items until the referenced item is found. Hence an access to the i-th item in the list incurs a cost of i. In the *standard model*, immediately after a request, the referenced item may be moved at no extra cost to any position closer to the front of the list. These exchanges are called free exchanges. Moreover, at any time, two adjacent items in the list may be swapped at a cost of 1. Such exchanges are called paid exchanges.

In contrast, in the *paid exchange model* P^d, there are no free exchanges. The list can only be rearranged using paid exchanges. Each paid exchange incurs a cost of d, where $d \geq 1$ is an arbitrary, real-valued constant. Following Reingold, Westbrook and Sleator [22] we assume that the service of a request $\sigma(t)$, $1 \leq t \leq m$, proceeds as follows: First an algorithm performs a number of paid exchanges; then the item referenced by $\sigma(t)$ is accessed. In general, in both the standard and the P^d model, the goal is to serve a request sequence so that the total cost is as small as possible.

An online algorithm has to serve each request without knowledge of any future requests. Given a request sequence σ, let $C_A(\sigma)$ and $C_{\mathrm{OPT}}(\sigma)$ denote the costs incurred by an online algorithm A and an optimal offline algorithm OPT in serving σ. A deterministic online

algorithm A is called *c-competitive* if there exists an α such that $C_A(\sigma) \leq c \cdot C_{\mathrm{OPT}}(\sigma) + \alpha$ holds for all request sequences σ. The value α must be independent of the input σ but may depend on the list length n. A randomized online algorithm A is *c-competitive against oblivious adversaries* if there exists an α such that $\mathbf{E}[C_A(\sigma)] \leq c \cdot C_{\mathrm{OPT}}(\sigma) + \alpha$ holds for all σ. Here the expectation is taken over the random choices made by A.

Previous work: Due to the wealth of results on the list update problem, we only mention the most important contributions relevant to our work. First we focus on the standard model. In their seminal paper [24], Sleator and Tarjan showed that the deterministic MOVE-TO-FRONT algorithm is 2-competitive. This is the best competitiveness a deterministic online strategy can attain [15]. Subsequent research developed randomized online algorithms against oblivious adversaries. Irani [12] gave a SPLIT algorithm that is 1.9375-competitive. Reingold et al. [22] devised a family of COUNTER algorithms and showed that the best of these achieve a competitive ratio of $85/49 \approx 1.7346$. In the same paper a generalized RANDOM RESET algorithm attains a competitive ratio of $\sqrt{3} \approx 1.7321$. A family of TIMESTAMP algorithms was developed in [1]. They attain a competitiveness equal to the Golden Ratio $(1+\sqrt{5})/2 \approx 1.6180$. The best randomized algorithm currently known is 1.6-competitive [3]. The algorithm is a combination of specific instances of the COUNTER and TIMESTAMP families.

As for lower bounds, Teia [25] showed that no randomized online algorithm can be better than 1.5-competitive against oblivious adversaries. Ambühl et al. [5] showed that no projective randomized online algorithm can be better than 1.6-competitive against oblivious adversaries in the partial cost model. In the *partial cost model* an access to the i-th item in the list incurs a cost of $i-1$ rather than i. Moreover, an algorithm is projective if it suffices to consider pairs of items. Specifically, the relative order of any two items in the list only depends on the previous requests to those elements. All the algorithms mentioned above, except for SPLIT, are projective.

Recent research on the standard model has proposed models capturing locality of reference in request sequences [2, 6, 11]. Furthermore, Lopez-Ortiz et al. [16] analyzed the value of paid exchanges. They showed a lower bound of 12/11 on the worst-case ratio between the performance of an offline algorithm that uses only free exchanges and that of an offline algorithm that uses both paid and free exchanges. Boyar et al. [8] analyzed the list update problem with advice, where an online algorithm has partial information on future requests.

We next consider the P^d model. So far only COUNTER and RANDOM RESET algorithms have been studied. The best deterministic online algorithm currently known achieves a competitive ratio of $(5+\sqrt{17})/2 \approx 4.5616$ [4]. No deterministic online algorithm can be better than 3-competitive [22]. Reingold et al. [22] gave a randomized COUNTER algorithm. For $d=1$, the algorithm is 2.75-competitive against oblivious adversaries. For increasing d, the competitiveness decreases and tends to $(5+\sqrt{17})/4 \approx 2.2808$. For small values of d, their RANDOM RESET algorithm achieves competitive ratios that are slightly smaller than those of the COUNTER algorithm. No lower bounds on the performance of randomized online algorithms against oblivious adversaries are known.

As for other cost models, Munro [21] and Kamali et al. [13] proposed settings that are interesting in data compression applications. Specifically, after an access to the i-th item in the list, all items up to position i can be rearranged at a cost proportional to i. In an even stronger model of theirs the whole list can be rearranged free of charge.

Our contribution: We present a comprehensive study of the list update problem in the P^d model. First in Section 3 we develop a new randomized online algorithm

TIMESTAMP(l,p), which is defined for any positive integer l and any probability p, $0 \leq p \leq 1$. The strategy incorporates features of the TIMESTAMP algorithm in the standard model and the COUNTER algorithm in the P^d model. In the P^d model one cannot afford to move a referenced item closer to the front of the list on every request to that item. Therefore, for every item in the list, TIMESTAMP(l,p) maintains a mod l counter. These counters are initialized independently and uniformly at random. When an item is requested, its counter value is decremented by 1. If the counter switches from 0 to $l-1$, we say that a *master request* to that item occurs. On a master request to x, with probability p, item x is moved to the front of the list. Otherwise, with probability $1-p$, item x is inserted in front of the first item y in the list such that (a) at most one master request to y occurred since the last master request to x and (b) the possible master request to y was also handled according to this latter policy, i.e. y was not moved to the front.

TIMESTAMP(l,p) achieves an improved competitive ratio of $c < 2.2442$ against oblivious adversaries, as d grows. A main contribution of this paper is a new analysis technique. TIMESTAMP(l,p) is projective so that it can be analyzed on pairs of items. However in the P^d model, in contrast to the standard model, it is hard to keep track of the optimal offline algorithm. Specifically, it does not hold true anymore that after two consecutive requests to the same item, that item precedes the other item in the optimum list. Therefore we define a more general phase partitioning of request sequences. A phase ends whenever the optimum offline algorithm OPT changes the relative order of the two considered items in the list. Hence the analysis is guided by OPT's moves. In particular, each phase can be assigned the cost of one paid exchange performed by OPT. The challenge is to estimate the cost of TIMESTAMP(l,p) without making any assumptions on the request pattern at the end of a phase. In order to analyze any phase, we also devise a new framework to charge expected service cost.

In Section 4 we develop lower bounds. We prove that, against oblivious adversaries and as d goes to infinity, no randomized online algorithm can achieve a competitive ratio smaller than 1.8654. Furthermore, we show that no randomized online algorithm can attain a competitiveness smaller than $2-1/2d$ against oblivious adversaries in the partial cost model, for general d.No lower bound against oblivious adversaries was known before.

In order to establish our lower bounds, we devise a probability distribution on request sequences: The items of a given list are requested in cyclic order. The number of consecutive requests to the same item is distributed geometrically with mean $2d$. We then compare expected online and offline cost. The analysis of the expected cost incurred by OPT is quite involved. In the partial cost model we partition a request sequence into phases, each ending with a certain surplus of requests to the same item, so that OPT will move that item to the front of its list. As for the lower bound in the full cost model, we prove that we may restrict ourselves to the partial cost model and request sequences referencing two items, provided that we consider projective offline algorithms. Unfortunately, OPT is not projective. Therefore, we define a family of projective offline algorithms and analyze their cost using a Markov chain.

2 Preliminaries

Given any algorithm A for list update in the P^d model, let $C_A(\sigma)$ denote its costs incurred in serving a request sequence σ. We will consider both the *full cost model* and the *partial cost model*. Again, in the former model, an access to the i-th item in the current list incurs a cost of i. In the latter one the access cost is $i-1$ only. We use the notation $C_A^{\text{full}}(\sigma)$ and $C_A^{\text{part}}(\sigma)$

to refer to the respective costs when this is needed. Observe that $C_A^{\text{full}}(\sigma) = C_A^{\text{part}}(\sigma) + m$, where $m = |\sigma|$ is the length of σ. Hence, if an online algorithm is c-competitive in the partial cost model, it is also c-competitive in the full cost model. Therefore, we will analyze TIMESTAMP(l, p) in the partial cost model.

We will prove in Section 3 that TIMESTAMP(l, p) is projective so that it can be analyzed using item pairs. An algorithm is projective if, for any request sequences σ and any pair x, y of items, the relative order of x and y in the list is always the same as if only references to x and y were served on the respective two-item list. Formally consider an algorithm A, a list L and two distinct items $x, y \in L$. Let σ be an arbitrary request sequence. Starting from an initial list configuration $L(0)$, let $L_A(\sigma)$ be the list state immediately after A has served σ. Let $L_A(\sigma)_{xy}$ be the list obtained from $L_A(\sigma)$ by deleting all items other than x and y. Next consider the projected request sequence σ_{xy}, obtained from σ by deleting all requests that are neither to x nor to y. Moreover, let $L(0)_{xy}$ be the list derived from $L(0)$ by removing all items, except for x and y. Finally, starting with $L(0)_{xy}$, let $L_A(\sigma_{xy})$ be the list immediately after A has served σ_{xy}. If A is a randomized algorithm, its random choices on σ_{xy} are identical to those on the requests to x and y in σ.

▶ **Definition 1.** *An algorithm A is projective if, for any request sequence σ and any pair x, y of distinct items of a list L, there holds $L_A(\sigma)_{xy} = L_A(\sigma_{xy})$.*

The following proposition states that projective algorithms can be analyzed by focusing on item pairs. The proof is standard and given in Appendix A.

▶ **Proposition 2.** *Consider the partial cost model. A projective online algorithm A is c-competitive if and only if it is c-competitive on request sequences referencing only two items, which are served on a two-item list.*

We call an algorithm A *strictly c-competitive on σ* if we have $C_A(\sigma) \leq c \cdot C_{\text{OPT}}(\sigma)$. We will also apply this notion to subsequences $\lambda = \sigma(t) \dots \sigma(t')$ of σ. It is an obvious but very useful fact that A is strictly c-competitive on a sequence σ if we can divide σ into subsequences $\sigma = \lambda_1 \dots \lambda_h$ such that A is strictly c-competitive for each λ_i, $1 \leq i \leq h$. Here we assume that OPT serves the entire sequence σ and evaluate OPT's cost on each subsequence $\lambda_1, \dots, \lambda_h$.

3 The TIMESTAMP(l, p)-algorithm

3.1 The algorithm

Our new algorithm TIMESTAMP(l, p), we refer to it by TS(l, p) or TS for short, is the generalization of TIMESTAMP(p) for the standard cost model [1]. In the P^d model item exchanges are expensive and one can afford them only once in a while. Therefore, for every item x in the given list L, our algorithm maintains a mod l counter $c(x)$, taking values in $\{0, \dots, l-1\}$, for some positive integer l. The counter is initialized uniformly at random and independently of other items.

Consider a request $\sigma(t)$, referencing item x. There are two cases. If $c(x) > 0$ before the request, then $c(x)$ is decremented by 1 and the position of x remains unchanged in the current list. On the other hand, if $c(x) = 0$, then a *master request* occurs. TIMESTAMP(l, p) resets $c(x)$ to $l - 1$ and moves x closer to the front of the list, choosing among two policies. With probability p, item x is simply moved to the front of the list (Policy 1). With probability $1 - p$, item x is moved more reluctantly (Policy 2). Specifically, the algorithm determines the longest suffix $\lambda(t)$ of the request sequence ending with $\sigma(t)$ such that $\lambda(t)$ contains exactly

one master request to x, namely the one at $\sigma(t)$. The algorithm then identifies the first item z in the current list such that at most one master request to z occurs in $\lambda(t)$ and the possible master request was also served using Policy 2. The algorithm TIMESTAMP(l,p) moves x in front of z in the list. Observe that x satisfies the conditions formulated for item z so that x does not move backward in the list. The intuition of Policy 2 is to pass items whose request frequency, measured in terms of master requests, is not higher than that of x. A pseudo-code description of TIMESTAMP(l,p) is given in Algorithm 1.

Note that, for any item x, two master requests are separated by $l-1$ regular requests to x so that the item is not moved too often. Since $c(x)$ is initialized uniformly at random, the cycles consisting of a master request followed by $l-1$ regular requests to x are shifted in a random fashion in σ. In particular, with probability $1/l$ a request to x happens to be a master request.

Algorithm 1 TIMESTAMP(l,p)

Let $\sigma(t)=x$;
if $c(x)>0$ **then**
    $c(x) \leftarrow c(x)-1$;
else // $\sigma(t)$ is a master request
    $c(x) \leftarrow l-1$;
    With probability p, serve $\sigma(t)$ using **Policy 1** and with probability $1-p$ serve it using **Policy 2**;
    Policy 1: Move x to the front of the list.
    Policy 2: Let $\lambda(t)$ be the longest suffix of the sequence ending with $\sigma(t)$ in which exactly one master request to x occurs. Let z be the first item in the current list for which at most one master request occurs in $\lambda(t)$ and the possible master request was served using Policy 2. Move x in front of z in the list.

Theorem 3 gives the competitive ratio of TS(l,p), which is the maximum of six expressions. Nonetheless, the maximum can be determined exactly and truly optimal, algebraic values for p, l and hence the competitive ratio c can be computed. Details are given in Appendix B.1. By plugging in $p=0.45787$ and $\varphi=l/d=1.19390$, the reader can verify that indeed $c<2.2442$. For the optimal choice of p and φ, there holds $c_1=c_2=c_3$ while the other ratios are smaller.

▶ **Theorem 3.** *Let $\varphi=\frac{l}{d}$. TS(l,p) is c-competitive, where c is the maximum of the following expressions.*

$$c_1 = 1+\left(\tfrac{1}{2}+\max\{1,2p\}(1-p)\right)\varphi \qquad c_2 = \tfrac{7-3p}{4}+\tfrac{1}{\varphi} \qquad c_3 = 1+\tfrac{3p}{2}-p^2+\tfrac{2p}{\varphi}$$
$$c_4 = \tfrac{3-p+p^2}{2}+\tfrac{2(1-2p+2p^2)}{\varphi} \qquad c_5 = \tfrac{3+p-p^2}{2}+\tfrac{2p^2}{\varphi} \qquad c_6 = 2-p+\tfrac{1-p}{\varphi}$$

As d goes to infinity, there holds $c<2.2442$, when choosing $p\approx 0.45787$ and l such that $\varphi\approx 1.19390$.

3.2 The analysis

We will prove that TS(l,p) is c-competitive in the partial cost model, for the ratio c stated in Theorem 3. As explained in Section 2 this immediately implies c-competitiveness in the full cost model. It is easy to verify that TS(l,p) is projective. A proof of the following proposition in given in Appendix B.2.

▶ **Proposition 4.** *The algorithm TS(l,p) is projective.*

Therefore, we may consider an arbitrary request sequence σ, referencing two items x and y, that is served on a two-item list. We will compare the expected cost incurred by TS(l,p) to the cost of OPT on σ.

A simple observation is that at any time while TS(l,p) serves σ, the counter value of any item is uniformly distributed over $\{0,\ldots,l-1\}$, independently of the counters of other items. This holds true because the counter value of an item, at any time, is equal to its initial value minus the number of requests to that item served so far modulo l. We will use this fact repeatedly.

The analysis of TS(l,p) crucially relies on a new phase partitioning of σ, together with a sophisticated cost charging scheme. More precisely, the service cost of a request to an item is paid at the next master request to that item, provided it occurs in the current phase. Moreover, specific cost will be assigned to master requests so that expected service cost within a phase can be analyzed independently of counter values at the beginning or during the phase.

Phases and pre-refined cost

Phase partitioning: Given an arbitrary request sequence σ, we partition it into phases. The first phase starts with the first request in σ. Whenever OPT exchanges the two items x and y, the current phase ends and a new phase starts. Recall that in response to a request, an algorithm may first exchange items. Then the request is served. Hence, when OPT swaps x and y, the most recent request served is the final one of the current phase. The upcoming request is the first one in the new phase. The last phase ends with the last request in σ.

Suppose that σ has been partitioned into phases $\lambda_1,\ldots,\lambda_k$. We will prove that, for any phase λ_i, TS(l,p)'s expected cost is upper bounded by c times the cost paid by OPT, where c is the ratio given in Theorem 3. This establishes the theorem. In analyzing a phase, we charge OPT a cost of d for the item swap done at the beginning of the phase, in addition to the service cost. We will charge this cost of d also in the first phase λ_1. This way we overestimate OPT's cost on σ by d, which does not affect the competitive ratio.

Now consider an arbitrary phase $\lambda=\lambda_i$. In the following y denotes the item stored at the first position in OPT's list. Item x is the one stored at the second position, behind y. Thus OPT incurs a service cost of 1 for each reference to x. Requests to y cost 0. Item x will be the *good item* because its service cost can be used to balance TS(l,p)'s cost. Item y will be the *bad item*.

Pre-refined cost: We wish to evaluate the algorithms' cost in λ without considering neighboring phases and by focusing on master requests. Therefore, in a series of two steps, we will pass over to a *refined cost setting*. First, we define *pre-refined cost* for TS(l,p). The service cost incurred by TS(l,p) on a request to an item $z\in\{x,y\}$ is charged at the next master request to z in the phase, if it exists. This charging scheme is applied even if the reference to z itself is a master request. We have to take care of service cost incurred on requests that are not followed by a master request in the phase.

For the item y, we simply append l requests to y at the end of the phase. Then an additional master request to y occurs at the end of the phase and the service cost of previous, unpaid requests to y is covered. Indeed we will add $2l$ requests to y so that any phase ends with two master requests to y. This will be convenient in the further phase analysis. The service cost of OPT does not increase by the addition of requests to y. We remark that in analyzing a phase, we will make no assumptions regarding TS(l,p)'s list configuration at the beginning of the phase.

As for the requests to x that are not followed by a master request to x in the phase, we

have to be more careful. By adding additional requests to x at the end of a phase or at the beginning of the next phase, thereby creating new master requests to x, we can in fact lower TS(l,p)'s cost incurred for paid exchanges on subsequent requests. This would not be feasible. Therefore, regarding requests to x that are not followed by a master request to x in a given phase, we ignore their cost. Instead, at the first master request to x in the phase, if it exists, we charge TS(l,p) a cost of l no matter how many non-master requests have occured so far. We will show in Lemma 6 below that TS(l,p)'s expected cost does not decrease when changing over to the pre-refined cost setting. For further reference, the following definition summarizes this cost charging scheme.

▶ **Definition 5.** *In the pre-refined cost setting, $2l$ requests to the bad item y are appended at the end of a given phase.* TS(l,p)*'s cost incurred at a request to $z \in \{x,y\}$ is charged to the next master request to z in the phase, if such a request exists. At the first master request to the good item x in the phase, if it exists, a cost of l is charged to* TS(l,p).

▶ **Lemma 6.** TS(l,p)*'s expected cost in a phase does not decrease when passing over to the pre-refined cost setting.*

The proof is given in Appendix B.3. The main idea is to show that, for item x, the expected extra cost charged at the first master request covers expected cost ignored at the end of the phase. From now on we evaluate TS(l,p)'s cost in the pre-refined cost setting.

Counter fixing for x and refined cost

Given a phase λ, we split OPT's cost so that OPT also incurs service cost at the master requests to x, in addition to the cost for the paid exchange. In particular, this will allow us to fix the counter value of x at the beginning of λ and compare the cost of TS(l,p) to that of OPT. Let $\mathbf{E}[C_{\mathrm{TS}}(\lambda)]$ denote TS(l,p)'s expected cost in λ. Moreover, let c_x be the counter value of x at the beginning of λ. Finally $\mathbf{E}[C^c_{\mathrm{TS}}(\lambda)]$ denotes TS(l,p)'s expected cost conditioned on $c_x = c$. Since c_x takes any value in $\{0,\dots,l-1\}$ with equal probability $1/l$, there holds $\mathbf{E}[C_{\mathrm{TS}}(\lambda)] = 1/l \cdot \sum_{c=0}^{l-1} \mathbf{E}[C^c_{\mathrm{TS}}(\lambda)]$.

Assume that λ contains $kl+j$ requests to x, for some $k \geq 0$ and $0 \leq j \leq l-1$. Then OPT's cost in λ is equal to $C_{\mathrm{OPT}}(\lambda) = d + kl + j$. Let k_c be the number of master requests to x in λ conditioned on $c_x = c$. If $c < j$, then $k_c = k+1$; otherwise $k_c = k$.

Define $C^c_{\mathrm{OPT}}(\lambda) := d + k_c l$. Then $1/l \cdot \sum_{c=0}^{l-1} C^c_{\mathrm{OPT}}(\lambda) = d + \sum_{c=0}^{l-1} k_c = d + j(k+1) + (l-j)k = d + kl + j = C_{\mathrm{OPT}}(\lambda)$. This implies

$$\frac{\mathbf{E}[C_{\mathrm{TS}}(\lambda)]}{C_{\mathrm{OPT}}(\lambda)} = \frac{1/l \cdot \sum_{c=0}^{l-1} \mathbf{E}[C^c_{\mathrm{TS}}(\lambda)]}{1/l \cdot \sum_{c=0}^{l-1} C^c_{\mathrm{OPT}}(\lambda)} \leq \max_c \left\{ \frac{\mathbf{E}[C^c_{\mathrm{TS}}(\lambda)]}{C^c_{\mathrm{OPT}}(\lambda)} \right\}.$$

Hence in the following we consider a fixed $c_x = c$ and upper bound $\mathbf{E}[C^c_{\mathrm{TS}}(\lambda)]/C^c_{\mathrm{OPT}}(\lambda)$. We emphasize that $C^c_{\mathrm{OPT}}(\lambda)$ charges service cost l to every master request to x.

We next partition a given phase λ into a *prephase* and a *postphase*, i.e. $\lambda = \lambda_{\mathrm{pre}}\lambda_{\mathrm{post}}$. The prephase λ_{pre} starts at the first request in λ and ends right before the first master request to x in the phase. If no master request to x exists in λ, then λ_{pre} ends with the last request in λ and the postphase is empty. The cost of d the algorithm OPT incurs due to the paid exchange made at the beginning of the phase counts for the prephase, while the cost of l the optimum algorithm pays at any master request to x is charged in the postphase.

The remainder of this section is devoted to evaluating TS(l,p)'s expected cost on λ_{pre} and λ_{post}. For the analysis on λ_{post}, in a second step, we change over to a refined cost setting that applies in λ_{post}. In this framework TS(l,p) is charged a pessimistic cost of l at

every master request to x if item x is not at the front of the list when the request occurs. Moreover, master requests to x are charged the service cost of requests to item y: If after a master request to x algorithm $\mathrm{TS}(l,p)$ has item x at the front of the list, then the request is charged an additional cost of $l/2$ to pay the service cost of references to y until the next master request to y occurs. A master request to y is assigned a cost of l if item y has been at the back of $\mathrm{TS}(l,p)$'s list since the last master request to y. This covers the remaining cost for requests to y. Lemma 8 below shows that $\mathrm{TS}(l,p)$'s expected cost does not decrease when passing over to the refined cost.

▶ **Definition 7.** *In the refined cost setting for* $\mathrm{TS}(l,p)$ *in* λ_{post}*, a master request to* x *is charged a* base cost *of* l *if the master request is the first one to* x *in* λ_{post} *(and hence* λ*) or if* x *is not at the front of* $\mathrm{TS}(l,p)$*'s list when the request is presented. An* additional cost *of* $l/2$ *is charged at the master request to* x *if, after service of the request,* x *is at the front of* $\mathrm{TS}(l,p)$*'s list. A master request to* y *in* λ_{post} *is charged a* bad cost *of* l *if* y *has been at the back of* $\mathrm{TS}(l,p)$*'s list since the last master request to* y*.*

▶ **Lemma 8.** *The expected cost of* $\mathrm{TS}(l,p)$ *in* λ_{post} *does not decrease when passing over to the refined cost.*

The cost analysis of a phase

Consider an arbitrary phase λ. Let λ_{post} be its postphase, which starts with a master request to x and ends with at least two master requests to y, according to the phase adjustment we did when changing over to the pre-refined cost. We next modify λ_{post} so that we can partition it into subphases, each ending with at least two master requests to y. For this purpose consider two consecutive master requests to x that are preceded by a single master request to y. The next proposition shows that we can insert l new requests to y, thereby generating a new master request to y, without decreasing the strict competitiveness on λ_{post}. The proof is given in Appendix B.3.

▶ **Proposition 9.** *In a subsequence of master requests* $x_0y_1x_1x_2$ *in* λ_{post}*, add* l *new requests to* y *before* x_1*.* $\mathrm{TS}(l,p)$*'s expected refined cost on the resulting sequence of master requests* $x_0y_1y_2x_1x_2$ *is at least as high as that on the former sequence.* OPT*'s cost does not change.*

Hence in λ_{post}, whenever two master requests to x are preceded by exactly one master request to y, we insert l new requests to y. Again, this creates a second master request to y. We then partition λ_{post} into subphases, each ending with two master requests to y. More precisely, the first subphase starts with the first master request in λ_{post}. A subphase ends immediately before a master request to x that is preceded by at least two master requests to y. Observe that whenever two consecutive master requests to x occur, they start a new subphase. We obtain two types of subphases, specified by their master requests.

- Type 1: $x(yx)^k y^{2+i}$ for some $i,k \geq 0$.
- Type 2: $x^{2+j}(yx)^k y^{2+i}$ for some $i,j,k \geq 0$.

In the following four lemmas we analyze $\mathrm{TS}(l,p)$ on any subphase. If the subphase is the first one in λ, we analyze it jointly with the preceding prephase λ_{pre}. Together the four lemmas imply Theorem 3. The cost ratios c_i, $1 \leq i \leq 6$, stated in the lemmas are identical those of Theorem 3. In Lemma 10 we first consider Type 2 subphases as $\mathrm{TS}(l,p)$'s performance ratio is independent of the master requests preceding those subphases. Then Lemmas 11, 12 and 13 address Type 1 subphases with the preceding master requests. Lemma 11 assumes that a Type 1 subphase is preceded by two master requests to y. It applies, in particular,

to any Type 1 subphase that is not the first subphase in the given λ. Lemmas 12 and 13 address a Type 1 subphase that is preceded (a) by master requests to x and y, in this order, or (b) by a master request to x. In both cases the subphase must be the first one in λ. Moreover, λ_{pre} consists of less than two master requests to y. In fact in case (b), there is no master request in λ_{pre}. Full proofs of all the lemmas are presented in Appendix B.3.

▶ **Lemma 10.** TS(l,p) *is strictly* $\max\{c_1,c_2,c_4\}$*-competitive on any subphase of Type 2, including a possible preceding prephase.*

▶ **Lemma 11.** TS(l,p) *is strictly* $\max\{c_1,c_3,c_4\}$*-competitive on any subphase of Type 1, including a possible prephase, if the subphase is preceded by two master requests to* y*.*

▶ **Lemma 12.** TS(l,p) *is strictly* $\max\{c_1,c_4,c_5\}$*-competitive on any subphase of Type 1 and its leading prephase if the subphase is preceded by master requests to* x *and* y *in this order.*

▶ **Lemma 13.** TS(l,p) *is strictly* $\max\{c_1,c_4,c_6\}$*-competitive on any subphase of Type 1 and its leading prephase if the subphase is preceded by a master request to* x*.*

At the end of Appendix B.3 we conclude with some remarks on our cost and service models.

4 Lower bounds

We develop lower bounds in the partial cost and the full cost P^d models.

▶ **Theorem 14.** *Let A be a randomized online algorithm for list update in the partial cost P^d model. If A is c-competitive against oblivious adversaries, then $c \geq 2 - \frac{1}{2d}$. This holds even for request sequences referencing only two items.*

We conjecture that a lower bound of $2 - \frac{1}{2d}$ also holds for the full cost P^d model. The difficulty is to upper bound the cost of OPT on request sequences referencing a general set of n items. We can establish the following lower bound in the full cost P^d model.

▶ **Theorem 15.** *Let A be a randomized online algorithm for list update in the full cost P^d model. If A is c-competitive against oblivious adversaries, then c is at least*

$$\frac{1}{1+2W\left(\frac{-1}{2e}\right)} - O\left(\frac{1}{d}\right) \approx 1.8654.$$

Here W is the upper branch of the Lambert-W-function, i.e. $W\left(\frac{-1}{2e}\right)$ is largest value x satisfying $2xe^{x+1} = -1$.

Table 1 presents the values of the lower bound of Theorem 15, for small values of d, up to $d = 100$. For $d = 1$, i.e. the standard cost model, our bound matches the one by Teia [25].

d	$c(d)$
1	1.5
2	1.8036
3	1.8270
4	1.8337
5	1.8420

d	$c(d)$
6	1.8438
7	1.8485
10	1.8531
20	1.8594
100	1.8642

Table 1 The lower bound in the full cost P^d model for various d.

A probability distribution on request sequences: For the proof of the two theorems above we use Yao's minimax principle [26]. We define a probability distribution on request sequences and compare the expected cost incurred by any (deterministic) online algorithm A to that of OPT. For the definition of our probability distribution, we describe how to sample a request sequence according to it. Let $L(0) = [x_0 \dots x_{n-1}]$ be a starting list of n items; x_i precedes x_{i+1} for $i = 0, \dots, n-2$. Throughout the sampling process the list is static, i.e. no rearrangement of items is done. The items will be requested in a cyclic fashion, in decreasing order of index. Each item will be referenced a certain number of times.

Formally, in addition to $L(0)$, the sampling process takes as input a number $N \in \mathbb{N}$ and a starting item $x \in L(0)$. Typically, the starting item is equal to the last item x_{n-1} in the list but the process is defined for any $x \in L(0)$. The sampling process produces a request sequence consisting of N segments. Throughout this section a *segment* is a maximal subsequence of requests to the same item. Moreover, to simplify notation, we set $x_i = x_{i \bmod n}$, for all $i \in \mathbb{Z}$.

Initially, the request sequence σ to be produced is equal to the empty string. In each step of the sampling procedure, if $N > 0$, a request to the current item x is appended at the end of σ. Then with probability $p = 1/(2d)$, the item x and the value N are *decremented*, i.e. $x = x_i$ is replaced by x_{i-1} and N is reduced by 1. Hence in this event the segment of requests to x_i ends and a segment of references to x_{i-1} starts. Note that the length of a segment is geometrically distributed with $p = 1/(2d)$ and thus equal to $2d$ in expectation. The process stops when $N = 0$. A pseudo-code description of the sampling process is given in Algorithm 2. Let $\mathcal{S}_N[x]$ denote the resulting probability distribution on request sequences with starting item x. Furthermore, let $\mathcal{S}_N = \mathcal{S}_N[x_{n-1}]$.

The lower bound construction in the full cost model will work with sequences generated according to this particular process. In the partial cost model we will have to extend the process so that the request sequences admit a phase partitioning and end with a complete phase.

Algorithm 2 SampleRequestSequence

```
Input: L(0) = [x_0 ... x_{n-1}], N ∈ ℕ and x ∈ L(0).
σ := empty string;
while N > 0 do
    Append x to σ; With probability p = 1/(2d), decrement x and N;
Return σ;
```

We next introduce the notion of an algorithm being *uncompetitive*. It will be particularly useful when deriving our lower bound in the full cost model. Nonetheless, the notion applies to both the partial and the full cost models and it will always be clear from the context which model is used.

▶ **Definition 16.** *Let $c \geq 1$. An online algorithm A is* c-uncompetitive against an offline algorithm B *if, for every $\varepsilon > 0$, there exists an initial list $L(0)$ such that*

$$\lim_{N \to \infty} \frac{\mathbf{E}_{\sigma \sim \mathcal{S}_N}[C_A(\sigma)]}{\mathbf{E}_{\sigma \sim \mathcal{S}_N}[C_B(\sigma)]} \geq c - \varepsilon.$$

If $B = \mathrm{OPT}$, algorithm A is simply c-uncompetitive. *Finally, A is* c-uncompetitive on two-item sequences (against B) *if the above condition holds with $\varepsilon = 0$, for lists consisting of two items.*

The terminology is relevant due to the following fact, whose proof is given in Appendix C.1.

▶ **Lemma 17.** *If a (possibly randomized) online algorithm A is c-uncompetitive, then its competitive ratio is not smaller than c.*

In a first step we lower bound the expected cost incurred by any online algorithm A on request sequences generated according to $\mathcal{S}_N$. In Theorem 14 we consider the partial cost model and request sequences referencing two items. In the proof of Theorem 15 we show that we may restrict ourselves to the partial cost model and, again, may focus on two-item request sequences. Therefore, the following Lemma 18 will be essential in the proofs of Theorems 14 and 15. Let $L(0) = [xy]$ be a list consisting of two items x and y; where x is initially stored in front of y. Consider the distribution $\mathcal{S}_N = \mathcal{S}_N[y]$. The proof of the next lemma is given in Appendix C.2.

▶ **Lemma 18.** *Let $L(0) = [xy]$. For every online algorithm A and every N, we have*

$$\mathbf{E}_{\sigma\sim\mathcal{S}_N}\left[C_A^{\text{part}}(\sigma)\right] \geq dN.$$

The challenging part in the proofs of Theorems 14 and 15 is to upper bound the expected cost incurred by OPT on sequences drawn according to $\mathcal{S}_N$. In the following we sketch some of the main ideas. Full analyses are presented in Appendix C.3 and C.4, respectively.

Proof sketch for Theorem 14: We focus on request sequences referencing two items x and y, and hence on sequences $\sigma \sim \mathcal{S}_N$ with initial list $L(0) = [xy]$. Such sequences consist of N segments that in turn reference y and x. Given a sequence σ' and an item $z \in \{x, y\}$, let $|\sigma'|_z$ be the number of requests to z in σ'.

Instead of OPT we analyze an offline algorithm O that, loosely speaking, processes a request sequence $\sigma \sim \mathcal{S}_N$ as follows. Algorithm O moves item z, where $z \in \{x, y\}$, to the front of its list whenever there is a prefix σ' of future requests with $|\sigma'|_z = |\sigma'|_{z'} + 2d$ and no prefix σ'' of σ' satisfies $|\sigma''|_z < |\sigma''|_{z'}$. Here z' is the other item of the pair $\{x, y\}$. Additionally, we will consider the algorithm $\bar{O}$ that always keeps its list in opposite order, compared to the list of O. On each request in a given sequence, exactly one of the two algorithms has a service cost of 1.

Given any sequence σ, let $\tilde{C}_O(\sigma)$ and $\tilde{C}_{\bar{O}}(\sigma)$ be the pure service cost of O and $\bar{O}$. Furthermore, let $D_O(\sigma)$ be the cost incurred by O for paid exchanges. Define $K(\sigma) = \tilde{C}_{\bar{O}}(\sigma) - \tilde{C}_O(\sigma) - 2D_O(\sigma)$. For $\sigma \sim \mathcal{S}_N$, $K(\sigma)$ is a random variable, which we denote by E_N. We will show that $\mathbf{E}_{\sigma\sim\mathcal{S}_N}[C_O(\sigma)] = dN - \mathbf{E}[E_N]/2$. The heart of the analysis is to prove that $\lim_{N\to\infty} \mathbf{E}[E_N]/N \geq 2d(2d-1)/(4d-1)$. Together with Lemmas 17 and 18, this implies that the competitive ratio of any online algorithm is lower bounded by $c \geq dN/(dN - \frac{N}{2}\frac{2d(2d-1)}{4d-1}) = 2 - \frac{1}{2d}$.

For the analysis of $\mathbf{E}[E_N]$, we partition a request sequence $\sigma \sim \mathcal{S}_N$ into phases. Each phase λ consists of a series of subphases $\mu_1, \ldots, \mu_l$. We describe how to obtain μ_1. At the beginning of λ, let $z \in \{x, y\}$ be the current item in the sampling process, i.e. the first segment in λ consists of requests to z. Let $z' \in \{x, y\}$, $z' \neq z$, be the other item. Starting at the beginning of λ we scan the generated requests, adding them to μ_1 until one of the following events occurs. (1) $|\mu_1|_z = |\mu_1|_{z'}$ or (2) $|\mu_1|_z = |\mu_1|_{z'} + 2d$. In case (1) we call μ_1 a *zero-subphase*; in case (2) μ_1 is an *up-subphase*. When event (1) or (2) occurs, the remaining requests of the current segment are appended to μ_1. These requests form the *post-subphase*. Then μ_1 ends. Each following subphase μ_i, for $i > 1$, is obtained in the same way, starting at the end of μ_{i-1}. Phase λ ends when, for the first time, an up-subphase is obtained. Formally, algorithm O moves item z to the front of the list right before the up-subphase.

We extend the sampling process so as to obtain sequences ending with a complete phase. This induces a slightly related probability distribution of request sequences, which is not critical though. For any λ, let $C = K(\lambda)$ and let $R = R(\lambda)$ be the number of segments in λ. At the heart of the analysis, using a recurrence relation, we prove $\lim_{N\to\infty} \mathbf{E}[E_N]/N \geq \mathbf{E}[C]/\mathbf{E}[R]$. We argue that $\mathbf{E}[C]$ is the total length of all post-subphases in λ. Then we show $\mathbf{E}[C] = (2d-1)/(1-P_0)$ and $\mathbf{E}[R] = (4d-1)/(2d(1-P_0))$, where P_0 is the probability that a subphase happens to be a zero-subphase. This yields the desired bound.

Proof sketch for Theorem 15: In this setting we are given a list $L = L(0)$ with n items. Again, we focus on request sequences generated according to $\mathcal{S}_N$ (Algorithm 2). We first prove that we may restrict ourselves to the partial cost model and two-item request sequences if we focus on *projective* offline algorithms: If every deterministic online algorithm is c-uncompetitive on two-item sequences against a projective offline algorithm B in the partial cost model, then every deterministic online algorithm is c-uncompetitive against B in the full cost model. Nonetheless, the analysis of the offline algorithm O developed for the proof of Theorem 15 cannot be employed because O and OPT are not projective.

Therefore, we define a family of projective offline algorithms B_h, for $0 < h < 2d$. Consider a request sequence to be served on a list L consisting of n items. When presented with any request, B_h works as follows: If the next h requests reference the same element $z \in L$, algorithm B_h moves z to the front of its list. Otherwise it does not change the list. Algorithm B_h is projective, for sequences σ generated by $\mathcal{S}_N$, because items are requested in cyclic order in σ and a projection to item pairs does not create longer subsequences of the same item.

Given the result for projective offline algorithms stated above, it suffices to analyze B_h on two-item sequences. Let $L(0) = [xy]$ be the starting list. Technically, the main step is to show that, for any $\sigma \sim \mathcal{S}_N$, the expected cost incurred by B_h is

$$\mathbf{E}_{\sigma\sim\mathcal{S}_N}[C_{B_h}(\sigma)] = \frac{\left(\frac{1-(1-p)^h}{p} - h(1-p)^{h-1}\right) + (1-p)^{h-1}d}{2-(1-p)^{h-1}} N + o(N), \tag{1}$$

where $p = 1/(2d)$. In order to establish this equation, given $\sigma \sim \mathcal{S}_N$, we write $\sigma = z_1^{\alpha_1} z_2^{\alpha_2} \dots z_N^{\alpha_N}$, where $z_1 = y$ and $z_2 = x$. If we consider the algorithm B_h at the beginning of the subsequence $z_t^{\alpha_t}$, there can be three cases.

- If z_t is at the back of the list of B_h and $\alpha_t \geq h$ holds, we say the sequence $z_t^{\alpha_t}$ is of *type* h. The algorithm B_h will incur a cost d on it because it immediately moves z_t to the front.
- If z_t is at the back of the list of B_h and $\alpha_t = j < h$, we say the sequence $z_t^{\alpha_t}$ is of *type* $(j,1)$. The algorithm B_h will incur cost j on it.
- If z_t is at the front of the list of B_h, the algorithm will incur no cost on it. Further we have $t > 1$ and $\alpha_{t-1} = j < h$, for some j. We say the sequence $z_t^{\alpha_t}$ is of *type* $(j,2)$.

For a type $w \in W_h = \{1,\dots,h-1\} \times \{1,2\} \cup \{h\}$, let $P(t)_w$ be the probability that the sequence $z_t^{\alpha_t}$ is of this type if we sample $\sigma \sim \mathcal{S}_N$. The process forms a Markov chain: From type $(j,1)$, for $j < h$, we pass to the type $(j,2)$ with probability 1. From every other type w we pass to type $(j,1)$, for $j < h$, with probability $p(1-p)^{j-1}$ and to type h with probability $(1-p)^{h-1}$. Using basic Markov analysis we evaluate $\lim_{t\to\infty} P(t)_w$, for $w = (j,1)$, $w = (j,2)$ and $w = h$. Using the respective expressions, we obtain (1). If we set $h = \lfloor 2\tilde{h}d \rfloor$, then $\mathbf{E}_{\sigma\sim\mathcal{S}_N}[C_{B_h}(\sigma)]$ is of the form $(1 + \frac{2\tilde{h}e^{-\tilde{h}}}{e^{-\tilde{h}}-2} + O\left(\frac{1}{d}\right))dN + o(N)$. Lemma 18 implies that every online algorithm A is c-uncompetitive against the algorithm B_h in the full cost model, where c is at least $(1 + 2\tilde{h}e^{-\tilde{h}}/(e^{-\tilde{h}}-2))^{-1}$ as $d \to \infty$. Lemma 17 ensures that the competitive ratio of A is not smaller than the above expression. Theorem 15 follows if we set $\tilde{h} = W(-1/(2e)) + 1$.

References

1. S. Albers. Improved randomized on-line algorithms for the list update problem. *SIAM J. Comput.*, 27(3):682–693, 1998.
2. S. Albers and S. Lauer. On list update with locality of reference. *J. Comput. Syst. Sci.*, 82(5):627–653, 2016.
3. S. Albers, B. von Stengel, and R. Werchner. A combined BIT and TIMESTAMP algorithm for the list update problem. *Inf. Process. Lett.*, 56(3):135–139, 1995.
4. S. Albers and J. Westbrook. Self-organizing data structures. In *Online Algorithms, The State of the Art*, Springer LNCS 1442, pages 13–51, 1998.
5. C. Ambühl, B. Gärtner, and B. von Stengel. Optimal lower bounds for projective list update algorithms. *ACM Trans. Algorithms*, 9(4):31:1–31:18, 2013.
6. S. Angelopoulos and P. Schweitzer. Paging and list update under bijective analysis. *J. ACM*, 60(2):7:1–7:18, 2013.
7. A. Borodin and R. El-Yaniv. *Online computation and competitive analysis.* Cambridge University Press, 1998.
8. J. Boyar, S. Kamali, K.S. Larsen, and A. López-Ortiz. On the list update problem with advice. *Inf. Comput.*, 253:411–423, 2017.
9. M. Burrows and D. Wheeler. A block sorting lossless data compression algorithm. *Technical Report 124*, 1994.
10. M. Crochemore, R. Grossi, J. Kärkkäinen, and G.M. Landau. Computing the Burrows-Wheeler transform in place and in small space. *J. Discrete Algorithms*, 32:44–52, 2015.
11. R. Dorrigiv, M.R. Ehmsen, and A. López-Ortiz. Parameterized analysis of paging and list update algorithms. *Algorithmica*, 71(2):330–353, 2015.
12. S. Irani. Two results on the list update problem. *Inf. Process. Lett.*, 38(6):301–306, 1991.
13. S. Kamali, S. Ladra, A. López-Ortiz, and D. Seco. Context-based algorithms for the list-update problem under alternative cost models. In *Proc. 2013 Data Compression Conference (DCC)*, IEEE, pages 361–370, 2013.
14. S. Kamali and A. López-Ortiz. Better compression through better list update algorithms. In *Proc. 2014 Data Compression Conference (DCC)*, IEEE, pages 372–381, 2014.
15. R. Karp and P. Raghavan. Personal communication cited in [22].
16. A. López-Ortiz, M.P. Renault, and A. Rosén. Paid exchanges are worth the price. In *Proc. 32nd International Symposium on Theoretical Aspects of Computer Science (STACS15)*, LIPIcs 30, pages 636–648, 2015.
17. M.S. Manasse, L.A. McGeoch, and D.D. Sleator. Competitive algorithms for on-line problems. In *Proc. 20th Annual ACM Symposium on Theory of Computing*, pages 322–333, 1988.
18. G. Manzini. An analysis of the Burrows-Wheeler transform. *J. ACM*, 48(3):407–430, 2001.
19. C. Martínez and S. Roura. On the competitiveness of the move-to-front rule. *Theor. Comput. Sci.*, 242(1-2):313–325, 2000.
20. J. McCabe. On serial files with relocatable records. *Operations Research*, 12:609–618, 1965.
21. J.I. Munro. On the competitiveness of linear search. In *Proc. 8th Annual European Symposium on Algorithms (ESA00)*, Springer LNCS 1879, pages 338–345, 2000.
22. N. Reingold, J. Westbrook, and D.D. Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica*, 11(1):15–32, 1994.
23. J. Sirén. Burrows-Wheeler transform for terabases. In *Proc. 2016 Data Compression Conference (DCC)*, IEEE, pages 211–220, 2016.
24. D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985.
25. B. Teia. A lower bound for randomized list update algorithms. *IPL*, 47(1):5–9, 1993.
26. A.C.C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proc. 18th IEEE Annual Symposium on Foundations of Computer Science*, pages 222–227, 1977.

Appendix B

Scheduling in the Random-Order Model

Bibliographic Information *Scheduling in the Random-Order Model.* S. Albers, M. Janke. *Algorithmica* (2021): 1-30.
A preliminary version appeared in: The 47th International Colloquium on Automata, Languages and Programming (**ICALP**) 2020

Summary of Contributions We study Online Makespan Minimization in the random-order model. Jobs, defined by their processing times, have to be assigned to parallel and identical machines. Preemption is not allowed. The goal is to minimize the time it takes to process them all, called the makespan. Opposed to the traditional adversarial model that considers worst-case orders, the recently popular random-order model presents jobs to an online algorithm in a uniformly random order.

A novel online algorithm is developed, which has competitive ratio 1.8478 in the random-order model. In the adversarial model such competitive ratio is impossible. The competitive ratio of 1.8478 holds, in fact, not only in expectation but on almost all input permutations. Worst-case sequences are thus extremely rare and unlikely to occur after random permutation.

This upper bound is complemented by first lower bounds. We show that no deterministic online algorithm can be better than 4/3-competitive in the random-order model. Moreover, no deterministic online algorithm can guarantee a competitive ratio below 1.5 with high probability.

Individual Contributions

- Discovered and developed the upper bound including the notion and analysis of stable sequences and the adversarial analysis.
- Discovered and developed the lower bounds.
- Composed the first manuscript excluding abstract and introduction. Included are all other technical and non-technical parts (later significantly revised and edited by S. Albers).

Algorithmica
https://doi.org/10.1007/s00453-021-00841-8

Scheduling in the Random-Order Model

Susanne Albers[1] · **Maximilian Janke**[1]

Received: 7 November 2020 / Accepted: 21 May 2021

Abstract

Makespan minimization on identical machines is a fundamental problem in online scheduling. The goal is to assign a sequence of jobs to m identical parallel machines so as to minimize the maximum completion time of any job. Already in the 1960s, Graham showed that *Greedy* is $(2 - 1/m)$-competitive. The best deterministic online algorithm currently known achieves a competitive ratio of 1.9201. No deterministic online strategy can obtain a competitiveness smaller than 1.88. In this paper, we study online makespan minimization in the popular random-order model, where the jobs of a given input arrive as a random permutation. It is known that *Greedy* does not attain a competitive factor asymptotically smaller than 2 in this setting. We present the first improved performance guarantees. Specifically, we develop a deterministic online algorithm that achieves a competitive ratio of 1.8478. The result relies on a new analysis approach. We identify a set of properties that a random permutation of the input jobs satisfies with high probability. Then we conduct a worst-case analysis of our algorithm, for the respective class of permutations. The analysis implies that the stated competitiveness holds not only in expectation but with high probability. Moreover, it provides mathematical evidence that job sequences leading to higher performance ratios are extremely rare, pathological inputs. We complement the results by lower bounds, for the random-order model. We show that no deterministic online algorithm can achieve a competitive ratio smaller than 4/3. Moreover, no deterministic online algorithm can attain a competitiveness smaller than 3/2 with high probability.

Keywords Scheduling · Makespan minimization · Online algorithm · Competitive analysis · Lower bound · Random-order

A preliminary version of this paper has appeared in the 47th International Colloqium on Automata, Languages and Programming (ICALP), 2020. Work supported by the European Research Council, Grant Agreement No. 691672, project APEG.

✉ Maximilian Janke
maximilian@janke.tech

Extended author information available on the last page of the article

Published online: 09 June 2021

1 Introduction

We study one of the most basic scheduling problems. Consider a sequence of jobs $\mathcal{J} = J_1, \ldots, J_n$ that has to be assigned to m identical parallel machines. Each job J_t has an individual processing time p_t, $1 \leq t \leq n$. Preemption of jobs is not allowed. The goal is to minimize the *makespan*, i.e. the maximum completion time of any job in the constructed schedule. Both the offline and online variants of this problem have been studied extensively, see e.g. [4, 11, 14, 19, 21, 34] and references therein.

We focus on the online setting, where jobs arrive one by one. Whenever a job J_t is presented, its processing time p_t is revealed. The job has to be scheduled immediately on one of the machines without knowledge of any future jobs J_s, with $s > t$. Given a job sequence $\mathcal{J}$, let $A(\mathcal{J})$ denote the makespan of an online algorithm A on $\mathcal{J}$. Let $OPT(\mathcal{J})$ be the optimum makespan. A deterministic online algorithm A is *c-competitive* if $A(\mathcal{J}) \leq c \cdot OPT(\mathcal{J})$ holds for all $\mathcal{J}$ [39]. The best competitive ratio that can be achieved by deterministic online algorithms is in the range [1.88, 1.9201], see 14, 35. No randomized online algorithm is known that beats deterministic ones, for general m.

In this paper we investigate online makespan minimization in the random-order model. Here an input instance/job sequence is chosen by an adversary. Then a random permutation of the input elements/jobs arrives. The random-order model was considered by Dynkin [10] and Lindley [29] for the secretary problem. Over the last years the framework has received quite some research interest and many further problems have been studied. These include generalized secretary problems [2, 3, 13, 28, 29], the knapsack problem [2, 28], bin packing [26], facility location [31], matching problems [17, 22, 30], packing LPs [27] and convex optimization [20].

We present an in-depth study of online makespan minimization in the random-order model. As a main contribution we devise a new deterministic online algorithm that achieves a competitive ratio of 1.8478. After almost 20 years this is the first progress for the pure online setting, where an algorithm does not resort to extra resources in handling a job sequence.

1.1 Previous Work

We review the most important results relevant to our work and first address the standard setting where an online algorithm must schedule an arbitrary, worst-case job sequence. Graham in 1966 showed that the famous *Greedy* algorithm, which assigns each job to a least loaded machine, is $(2 - \frac{1}{m})$-competitive. Using new deterministic strategies the competitiveness was improved in a series of papers. Galambos and Woeginger [15] gave an algorithm with a competitive ratio of $(2 - \frac{1}{m} - \epsilon_m)$, where ϵ_m tends to 0 as $m \to \infty$. Bartal et al. [4] devised a 1.986-competitive algorithm. The bound was improved to 1.945 [23] and 1.923 [1]. Fleischer and Wahl [14] presented an algorithm that attains a competitive ratio of 1.9201 as $m \to \infty$. Chen et al. [7] gave an algorithm whose competitiveness is at most $1 + \varepsilon$ times the best possible factor, but no explicit bound was provided. Lower bounds on the competitive ratio of deterministic online algorithms were shown in [1, 5, 12, 18, 35, 36]. For general m, the bound

was raised from 1.707 [12] to 1.837 [5] and 1.854 [18]. Rudin [35] showed that no deterministic strategy has a competitiveness smaller than 1.88.

For randomized online algorithms, there is a significant gap between the best known upper and lower bounds. For $m = 2$ machines, Bartal et al. [4] presented an algorithm that achieves an optimal competitive ratio of 4/3. To date, there exists no randomized algorithm whose competitiveness is smaller than the deterministic lower bound, for general m. The best known lower bound on the performance of randomized online algorithms tends to $e/(e-1) \approx 1.581$ as $m \to \infty$ [6, 38].

Recent research on makespan minimization has examined settings where an online algorithm is given extra resources when processing a job sequence. Specifically, an algorithm might have a buffer to reorder the incoming job sequence [11, 25] or is allowed to migrate jobs [37]. Alternatively, an algorithm has information on the job sequence [8, 9, 24, 25], e.g. it might know the total processing time of the jobs or even the optimum makespan.

In the random-order model only one result is known for makespan minimization on identical machines. Osborn and Torng [33] showed that *Greedy* does not achieve a competitive ratio smaller than 2 as $m \to \infty$. Recently Molinaro [32] studied online load balancing with the objective to minimize the l_p-norm of the machine loads. He considers a general scenario with machine-dependent job processing times, which are bounded by 1. For makespan minimization he presents an algorithm that, in the worst case, is $O(\log m/\varepsilon)$-competitive and, in the random-order model, has an expected makespan of $(1+\varepsilon)OPT(\mathcal{J}) + O(\log m/\varepsilon)$, for any $\varepsilon \in (0, 1]$. Göbel et al. [16] consider a scheduling problem on one machine where the goal is to minimize the average weighted completion time of all jobs. Under random-order arrival, their competitive ratio is logarithmic in n, the number of jobs, for the general problem and constant if all jobs have processing time 1.

1.2 Our Contribution

We investigate online makespan minimization in the random-order model, a sensible and widely adopted input model to study algorithms beyond the worst case. Specifically, we develop a new deterministic algorithm that achieves a competitive ratio of 1.8478 as $m \to \infty$. This is the first improved performance guarantee in the random-order model. The competitiveness is substantially below the best known ratio of 1.9201 in the worst-case setting and also below the corresponding lower bound of 1.88 in that framework.

A new feature of our algorithm is that it schedules an incoming job on one of three candidate machines in order to maintain a certain load profile. The best strategies in the worst-case setting use two possible machines, and it is not clear how to take advantage of additional machines in that framework. The choice of our third, extra machine is quite flexible: An incoming job is placed either on a least loaded, a heavily loaded or—as a new option—on an intermediate machine. The latter one is the $(h+1)$st least loaded machine, where h may be any integer with $h \in \omega(1)$ and $h \in o(\sqrt{m})$.

When assigning a job to a machine different from the least loaded one, an algorithm has to ensure that the resulting makespan does not exceed c times the optimum makespan, for the targeted competitive ratio c. All previous strategies in the literature lower bound the optimum makespan by the current average load on the machines. Our new algorithm works with a refined lower bound that incorporates the processing times of the largest jobs seen so far. The lower bound is obvious but has not been employed by previous algorithms.

The analysis of our algorithm proceeds in two steps. First we define a class of *stable job sequences*. These are sequences that reveal information on the largest jobs as processing volume is scheduled. More precisely, once a certain fraction of the total processing volume $\sum_{t=1}^{n} p_t$ has arrived, one has a good estimate on the hth largest job and has encountered a certain number of the $m+1$ largest jobs in the input. The exact parameters have to be chosen carefully.

We prove that with high probability, a random permutation of a given input of jobs is stable. We then conduct a worst-case analysis of our algorithm on stable sequences. Using their properties, we show that if the algorithm generates a flat schedule, like *Greedy*, and can be hurt by a huge job, then the input must contain many large jobs so that the optimum makespan is also high. A new ingredient in the worst-case analysis is the processing time of the hth largest job in the input. We will relate it to machine load in the schedule and to the processing time of the $(m+1)$st largest job; twice the latter value is a lower bound on the optimum makespan.

The analysis implies that the competitive ratio of 1.8478 holds with high probability. Input sequences leading to higher performance ratios are extremely rare. We believe that our analysis approach might be fruitful in the study of other problems in the random-order model: Identify properties that a random permutation of the input elements satisfies with high probability. Then perform a worst-case analysis.

Finally in this paper we devise lower bounds for the random-order model. We prove that no deterministic online algorithm achieves a competitive ratio smaller than 4/3. Moreover, if a deterministic online algorithm is c-competitive with high probability, then $c \geq 3/2$.

2 Strong Competitiveness in the Random-Order Model

We define competitiveness in the random-order model and introduce a stronger measure of competitiveness that implies high-probability bounds. Recall that traditionally a deterministic online algorithm A is c-competitive if $A(\mathcal{J}) \leq c \cdot OPT(\mathcal{J})$ holds for all job sequences $\mathcal{J} = J_1, \ldots, J_n$. We will refer to this worst-case model also as the *adversarial model*.

In the *random-order model* a job sequence $\mathcal{J} = J_1, \ldots, J_n$ is given, which may be specified by an adversary. (Alternatively, a set of jobs could be specified.) Then a random permutation of the jobs arrives. We define the expected cost / makespan of a deterministic online algorithm. Let S_n be the permutation group of the integers from 1 to n, which we consider a probability space under the uniform distribution, i.e. each permutation in S_n is chosen with probability $1/n!$. Given $\sigma \in S_n$, let

$\mathcal{J}^\sigma = J_{\sigma(1)}, \ldots, J_{\sigma(n)}$ be the *job sequence permuted by* σ. The expected makespan of A on $\mathcal{J}$ in the random-order model is $A^{\mathrm{rom}}(\mathcal{J}) = \mathbf{E}_{\sigma \sim S_n}[A(\mathcal{J}^\sigma)] = \frac{1}{n!} \sum_{\sigma \in S_n} A(\mathcal{J}^\sigma)$. The algorithm A is *c-competitive in the random-order model* if $A^{\mathrm{rom}}(\mathcal{J}) \leq c \cdot OPT(\mathcal{J})$ holds for all job sequences $\mathcal{J}$.

We next define the notion of a deterministic online algorithm A being *nearly c-competitive*. The second condition in the following definition requires that the probability of A not meeting the desired performance ratio must be arbitrarily small as m grows and a random permutation of a given job sequence arrives. The subsequent Lemma 1 states that a nearly c-competitive algorithm is c-competitive in the random-order model.

Definition 1 A deterministic online algorithm A is called *nearly c-competitive* if the following two conditions hold.

- Algorithm A achieves a constant competitive ratio in the adversarial model.
- For every $\varepsilon > 0$, there exists an $m(\varepsilon)$ such that for all machine numbers $m \geq m(\varepsilon)$ and all job sequences $\mathcal{J}$ there holds $\mathbf{P}_{\sigma \sim S_n}[A(\mathcal{J}^\sigma) \geq (c + \varepsilon)OPT(\mathcal{J})] \leq \varepsilon$.

Lemma 1 *If a deterministic online algorithm is nearly c-competitive, then it is c-competitive in the random-order model as* $m \to \infty$.

Proof Let C be the constant such that A is C-competitive in the adversarial model. We may assume that $C > c$. Given $0 < \delta \leq C - c$, we show that there exists an $m(\delta)$ such that, for all $m \geq m(\delta)$, we have $A^{\mathrm{rom}}(\mathcal{J}) \leq (c + \delta)OPT(\mathcal{J})$ for every job sequences $\mathcal{J}$. Let $\varepsilon = \delta/(C - c + 1)$. Since A is nearly c-competitive, there exists an $m(\varepsilon)$ such that, for all $m \geq m(\varepsilon)$ and all inputs $\mathcal{J}$, there holds $P_\varepsilon(\mathcal{J}) = \mathbf{P}_{\sigma \sim S_n}[A(\mathcal{J}^\sigma) \geq (c + \varepsilon)OPT(\mathcal{J})] \leq \varepsilon$. Set $m(\delta) = m(\varepsilon)$. We obtain

$$\begin{aligned} A^{\mathrm{rom}}(\mathcal{J}) &= \mathbf{E}_{\sigma \sim S_n}[A(\mathcal{J}^\sigma)] \\ &\leq (1 - P_\varepsilon(\mathcal{J}))(c + \varepsilon)OPT(\mathcal{J}) + P_\varepsilon(\mathcal{J}) \cdot C \cdot OPT(\mathcal{J}) \\ &\leq ((1 - \varepsilon)(c + \varepsilon) + \varepsilon C)OPT(\mathcal{J}) \\ &\leq (c + \varepsilon(C - c + 1))OPT(\mathcal{J}) \\ &= (c + \delta)OPT(\mathcal{J}). \end{aligned}$$

□

3 Description of the New Algorithm

The deficiency of *Greedy* is that it tends to generate a flat, balanced schedule in which all the machines have approximately the same load. An incoming large job can then enforce a high makespan relative to the optimum one. It is thus crucial to try to avoid flat schedules and maintain steep schedules that exhibit a certain load imbalance among the machines.

However, in general, this is futile. Consider a sequence of m identical jobs with a processing time of, say, P_{m+1} (referring to the size of the $(m+1)$st largest job in an input). Any online algorithm that is better than 2-competitive must schedule these m jobs on separate machines, obtaining the flattest schedule possible. An incoming even larger job of processing time $p_{\max}$ will now enforce a makespan of $P_{m+1} + p_{\max}$. Observe that $\mathrm{OPT} \geq \max\{2P_{m+1}, p_{\max}\}$ since there must be one machine containing two jobs. In particular $P_{m+1} + p_{\max} \leq 1.5\mathrm{OPT}$. Hence sensible online algorithms do not perform badly on this sequence.

This example summarizes the quintessential strategy of online algorithms that are good on all sequences: Ensure that in order to create a schedule that is very flat, i.e. such that all machines have high load λ, the adversary must present m jobs that all are large relative to λ. In order to exploit this very flat schedule and cause a high makespan the adversary needs to follow up with yet another large job. But with these $m+1$ jobs, the optimum scheduler runs into the same problem as in the example: Of the $m+1$ large jobs, two have to be scheduled on the same machine. Thus the optimum makespan is high, compensating to the high makespan of the algorithm.

Effectively realizing the aforementioned strategy is highly non-trivial. In fact it is the central challenge in previous works on adversarial makespan minimization that improve upon *Greedy* [1, 4, 14, 15, 23]. These works gave us clear notions of how to avoid flat schedules, which form the basis for our approaches. Instead of simply rehashing these ideas, we want to outline next how we profit from random-order arrival in particular.

3.1 How Random-Order Arrival Helps

The first idea to profit from random-order arrival addresses the lower bound on OPT sophisticated online algorithms need. In the literature only the current average load has been considered, but under random-order arrival another bound comes to mind: The largest job seen so far. In order for an algorithm to perform badly, a large job needs to come close to the end of the sequence. Under random-order arrival, it is equally likely for such a job to arrive similarly close to the beginning of the sequence. In this case, the algorithm knows a better lower bound for OPT. The main technical tool will be our *Load Lemma*, which allows us to relate what a job sequence should reveal early from an analysis perspective to the actual fraction of jobs scheduled. This idea does not work for worst-case orders since they tend to order jobs by increasing processing times.

Recall that the general challenge of our later analysis will be to establish that there had to be m large jobs once the schedule gets very flat. In classical analyses, which consider worst-case orders, these jobs appear with increasing density towards the end of the sequence. In random orders this is unlikely, which can be exploited by the algorithm.

The third idea improves upon the first idea. Suppose, that we were to modify our algorithm such that it could handle one very large job arriving close to the end of the sequence. In fact, assume that it could only perform badly when confronted with h very large jobs. We can then disregard any sequence which contains fewer such jobs.

Recall that the first idea requires one very large job to arrive sufficiently close to the beginning. Now, as h grows, the probability of the latter event grows as well and approaches 1. This will not only improve our competitive ratio tremendously, it also allows us to adhere to the stronger notion of nearly competitiveness introduced in Sect. 2. Let us discuss how such a modification is possible: The first step is to design our algorithm in a way that it is reluctant to use the h least loaded machines. Intuitively, if the algorithm tries to retain machines of small load it will require very large jobs to fill them. In order to force these filling jobs to actually be large enough, our algorithm needs to use a very high lower bound for OPT. In fact, here it uses another lower bound for the optimum makespan, $2P^t_{m+1}$, twice the $(m+1)$st largest job seen so far at time t. Common analysis techniques can only make predictions about P^t_{m+1} at the very end of the sequence. It requires very subtle use of the random-order model to work around this.

3.2 Formal Definition

Formally our algorithm *ALG* is nearly c-competitive, where c is the unique real root of the polynomial $Q[x] = 4x^3 - 14x^2 + 16x - 7$, i.e.

$$c = \frac{7 + \sqrt[3]{28 - 3\sqrt{87}} + \sqrt[3]{28 + 3\sqrt{87}}}{6} < 1.8478.$$

Given $\mathcal{J}$, *ALG* schedules a job sequence/permutation $\mathcal{J}^\sigma = J_{\sigma(1)}, \ldots, J_{\sigma(n)}$ that must be scheduled in this order. Throughout the scheduling process *ALG* always maintains a list of the machines sorted in non-increasing order of current *load*. At any time the load of a machine is the sum of the processing times of the jobs already assigned to it. After *ALG* has processed the first $t-1$ jobs $J_{\sigma(1)}, \ldots, J_{\sigma(t-1)}$, we also say *at time t*, let $M^{t-1}_1, \ldots, M^{t-1}_m$ be any ordering of the m machines according to non-increasing load. More specifically, let l^{t-1}_j denote the load of machine M^{t-1}_j. Then $l^{t-1}_1 \geq \ldots \geq l^{t-1}_m$ and l^{t-1}_1 is the makespan of the current schedule.

ALG places each incoming job $J_{\sigma(t)}$, $1 \leq t \leq n$, on one of three candidate machines. The choice of one machine, having an intermediate load, is flexible. Let $h = h(m)$ be an integer with $h(m) \in \omega(1)$ and $h(m) \in o(\sqrt{m})$. We could use e.g. $h(m) = \lfloor \sqrt[3]{m} \rfloor$ or $h(m) = \lfloor \log m \rfloor$. Let[1]

$$i = \lceil (2c-3)m \rceil + h \approx 0.6956m.$$

ALG will assign the incoming job to the machine with the smallest load, the $(h+1)$st smallest load or the ith largest load.

When scheduling a job on a machine that is different from the least loaded one, an algorithm has to ensure that the resulting makespan does not exceed c^* times the optimum makespan, where c^* is the desired competitiveness. All previous algorithms lower bound the optimum makespan by the current average machine load. Algorithm *ALG* works with a refined lower bound that incorporates the

[1] Note that besides rounding $\approx$ also hides a small term in $o(m)$.

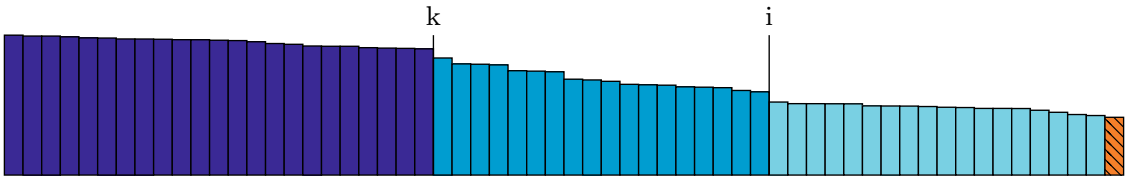


Fig. 1 A steep schedule. *ALG* only considers the least loaded machine

processing time of the largest job and twice the processing time of the $(m+1)$st largest job seen so far. These lower bounds on the optimum makespan are immediate but have not been used in earlier strategies.

Formally, for $j = 1, \ldots, m$, let L_j^t be the *average load* of the $m-j+1$ least loaded machines $M_j^t, \ldots, M_m^t$, i.e. $L_j^t = \frac{1}{m-j+1} \sum_{r=j}^{m} l_r^t$. We let $L^t = L_1^t = \frac{1}{m} \sum_{s=1}^{t} p_s$ be the average load of all the machines. For any $j = 1, \ldots, n$, let P_j^t be the processing time of the jth largest job among the first t jobs $J_{\sigma(1)}, \ldots, J_{\sigma(t)}$ in $\mathcal{J}^\sigma$. If $t < j$, we set $P_j^t = 0$. We let $p_{\max}^t = P_1^t$ be the processing time of the largest job among the first t jobs in $\mathcal{J}^\sigma$. Finally, let $L = L^n$, $P_j = P_j^n$ and $p_{\max} = p_{\max}^n$.

The value $O^t = \max\{L^t, p_{\max}^t, 2P_{m+1}^t\}$ is a common lower bound on the optimum makespan for the first t jobs and hence $OPT(\mathcal{J})$, see Proposition 1 in the next section. Note that immediately before $J_{\sigma(t)}$ is scheduled, *ALG* can compute L^t and hence O^t because L^t is $1/m$ times the total processing time of the jobs that have arrived so far.

We next characterize load imbalance. Let

$$k = 2i - m \approx (4c - 7)m \approx 0.3912m$$

and

$$\alpha = \frac{2(c-1)}{2c-3} \approx 2.7376.$$

The *schedule at time t* is the one immediately before $J_{\sigma(t)}$ has to be assigned. The schedule is *flat* if $l_k^{t-1} < \alpha L_{i+1}^{t-1}$, i.e. if l_k^{t-1}, the load of the kth most loaded machine, does not exceed $L_{i+1}^{t-1} = \frac{1}{m-i} \sum_{r=i+1}^{m} l_r^{t-1}$, the average load of the $m-i$ least loaded machines, by a factor of at least α . Otherwise the schedule is *steep*. Job $J_{\sigma(t)}$ is scheduled *flatly* (*steeply*) if the schedule at time t is flat (steep).

ALG handles each incoming job $J_{\sigma(t)}$, with processing time $p_{\sigma(t)}$, as follows. If the schedule at time t is steep, the job is placed on the least loaded machine M_m^{t-1}. On the other hand, if the schedule is flat, the machines M_i^{t-1}, M_{m-h}^{t-1} and M_m^{t-1} are probed in this order. If $l_i^{t-1} + p_{\sigma(t)} \le c \cdot O^t$, then the new machine load on M_i^{t-1} will not violate the desired competitiveness. The job is placed on this machine M_i^{t-1}. Otherwise, if the latter inequality is violated, *ALG* checks if a placement on M_{m-h}^{t-1} is safe, i.e. if $l_{m-h}^{t-1} + p_{\sigma(t)} \le c \cdot O^t$. If this is the case, the job is put on M_{m-h}^{t-1}. Otherwise, $J_{\sigma(t)}$ is finally scheduled on the least loaded machine M_m^{t-1}. A pseudo-code description of *ALG* is given below in Algorithm 1. The job assignment rules are also illustrated in Figs. 1 and 2.

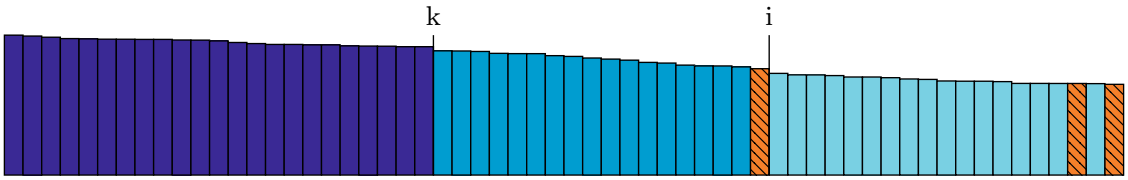


Fig. 2 A flat schedule. The three machines considered by *ALG* are marked for $h = 2$

Algorithm 1 The scheduling algorithm *ALG*

Let $J_{\sigma(t)}$ be the next job to be scheduled.
if the schedule at time t is steep, i.e. $l_k^{t-1} \geq \alpha L_{i+1}^{t-1}$ **then** // see Figure 1
    Assign $J_{\sigma(t)}$ to the least loaded machine M_m^{t-1};
else // the schedule is flat, i.e. $l_k^{t-1} < \alpha L_{i+1}^{t-1}$, see Figure 2
    if $l_i^{t-1} + p_{\sigma(t)} \leq c \cdot O^t$ **then** Assign $J_{\sigma(t)}$ to M_i^{t-1};
    else if $l_{m-h}^{t-1} + p_{\sigma(t)} \leq c \cdot O^t$ **then** Assign $J_{\sigma(t)}$ to M_{m-h}^{t-1};
    else Assign $J_{\sigma(t)}$ to M_m^{t-1};

In the next section we will prove the following theorem, Theorem 1, which uses the notion from Sect. 2. Lemma 1 then immediately gives the main result, Corollary 1.

Theorem 1 *ALG is nearly c-competitive, with* $c < 1.8478$ *defined as above.*

Corollary 1 *ALG is c-competitive in the random-order model as* $m \to \infty$.

4 Analysis of the Algorithm

4.1 Analysis Basics

We present some results for the adversarial model so that we can focus on the true random-order analysis of *ALG* in the next sections. First, recall the three common lower bounds used for online makespan minimization.

Proposition 1 *For any* $\mathcal{J}$, *there holds* $OPT(\mathcal{J}) \geq \max\{L, p_{\max}, 2P_{m+1}\}$. *In particular,* $O^1 \leq O^2 \leq \ldots \leq O^n \leq OPT(\mathcal{J})$.

Proof The optimum makespan $OPT(\mathcal{J})$ cannot be smaller than the average machine load L for the input, even if all the jobs are distributed evenly among the m machines. Moreover, the job with the largest processing time $p_{\max}$ must be scheduled non-preemptively on one of the machines in an optimal schedule. Thus $OPT(\mathcal{J}) \geq p_{\max}$. Finally, among the $m + 1$ largest jobs of the input, two must be placed on the same machine in an optimal solution. Hence $OPT(\mathcal{J}) \geq 2P_{m+1}$. □

For any job sequence $\mathcal{J} = J_1, \ldots, J_n$, let $R(\mathcal{J}) = \min\{\frac{L}{p_{\max}}, \frac{p_{\max}}{L}\}$. Intuitively, this measures the complexity of $\mathcal{J}$.

Proposition 2 *There holds* $Alg(\mathcal{J}) <= \max\{1 + R(\mathcal{J}), c\}OPT(\mathcal{J})$ *for any* $\mathcal{J} = J_1, \ldots, J_n$.

Proof Let $\mathcal{J} = J_1, \ldots, J_n$ be an arbitrary job sequence and let J_t be the job that defines *ALG*'s makespan. If the makespan exceeds $c \cdot OPT(\mathcal{J})$, then it exceeds $c \cdot O^t$. Thus *ALG* placed J_t on machine M_m^{t-1}, cf. lines 4 and 5 of the algorithm. This machine was a least loaded one, having a load of at most L. Hence $ALG(\mathcal{J}) \leq L + p_t \leq L + p_{\max} \leq \frac{L+p_{\max}}{\max\{L,p_{\max}\}} \cdot OPT(\mathcal{J}) = (1 + R(\mathcal{J})) \cdot OPT(\mathcal{J})$. □

Since $R(\mathcal{J}) \leq 1$ we immediately obtain the following result, which ensures that *ALG* satisfies the first condition of a nearly c-competitive algorithm, see Definition 1.

Corollary 2 *ALG is 2-competitive in the adversarial model.*

We next identify a class of *plain* job sequences that we do not need to consider in the random-order analysis because *ALG*'s makespan is upper bounded by c times the optimum on these inputs.

Definition 2 A job sequence $\mathcal{J} = J_1, \ldots, J_n$ is called *plain* if $n \leq m$ or if $R(\mathcal{J}) \leq c - 1$. Otherwise it is called *proper*.

Let $\mathcal{J} = J_1, \ldots, J_n$ be any job sequence that is processed/scheduled in this order. Observe that if it contains at most m jobs, i.e. $n \leq m$, and *ALG* cannot place a job J_t on machines M_i^{t-1} or M_{m-h}^{t-1} because the resulting load would exceed $c \cdot O^t$, then the job is placed on an empty machine. Using Proposition 2 we derive the following fact.

Lemma 2 *There holds* $ALG(\mathcal{J}) \leq c \cdot OPT(\mathcal{J})$ *for any plain job sequence* $\mathcal{J} = J_1, \ldots, J_n$.

If a job sequence $\mathcal{J}$ is plain (proper), then every permutation of it is. Hence, given Lemma 2, we may concentrate on proper job sequences in the remainder of the analysis. We finally state a fact that relates to the second condition of a nearly c-competitive algorithm, see again Definition 1.

Lemma 3 *Let* $\mathcal{J} = J_1, \ldots, J_n$ *be any job sequence that is scheduled in this order and let* J_t *be a job that causes ALG's makespan to exceed* $(c + \varepsilon)OPT(\mathcal{J})$, *for some* $\epsilon \geq 0$. *Then both the load of ALG's least loaded machine at the time of the assignment as well as* p_t *exceed* $(c - 1 + \varepsilon)OPT(\mathcal{J})$.

Proof *ALG* places J_t on machine M_m^{t-1}, which is a least loaded machine when the assignment is done. If l_m^{t-1} or p_t were upper bounded by $(c-1+\varepsilon)OPT(\mathcal{J})$, then the resulting load would be $l_m^{t-1} + p_t \leq (c-1+\varepsilon)OPT(\mathcal{J}) + \max\{L, p_t\} \leq (c-1+\varepsilon)OPT(\mathcal{J}) + OPT(\mathcal{J})$ $= (c+\varepsilon)OPT(\mathcal{J})$. □

4.2 Stable Job Sequences

We define the class of stable job sequences. These sequences are robust in that they will admit an adversarial analysis of *ALG*. Intuitively, the sequences reveal information on the largest jobs when a significant fraction of the total processing volume $\sum_{t=1}^{n} p_t$ has been scheduled. More precisely, one gets an estimate on the processing time of the hth largest job in the entire sequence and encounters a relevant number of the $m+1$ largest jobs. If a job sequence is unstable, large jobs occur towards the very end of the sequence and can cause a high makespan relative to the optimum one.

We will show that *ALG* is adversarially $(c+\varepsilon)$-competitive on stable sequences, for any given $\varepsilon > 0$. Therefore, the definition of stable sequences is formulated for a fixed $\varepsilon > 0$. Given $\mathcal{J}$, let $\mathcal{J}^\sigma = J_{\sigma(1)}, \ldots, J_{\sigma(n)}$ be any permutation of the jobs. Furthermore, for every $j \leq n$ and in particular $j \in \{h, m+1\}$, the *set of the j largest jobs* is a fixed set of cardinality j such that no job outside this set has a strictly larger processing time than any job inside the set.

Definition 3 A job sequence $\mathcal{J}^\sigma = J_{\sigma(1)}, \ldots, J_{\sigma(n)}$ is *stable* if the following conditions hold.

- There holds $n > m$.
- Once $L^t \geq (c-1)\frac{i}{m}L$, there holds $p_{\max}^t \geq P_h$.
- For every $j \geq i$, the sequence ending once we have $L^t \geq (\frac{j}{m} + \frac{\varepsilon}{2})L$ contains at least $j+h+2$ many of the $m+1$ largest jobs in $\mathcal{J}$.
- Consider the sequence ending right before either (a) $L^t \geq \frac{i}{m}(c-1)\varepsilon L$ holds or (b) one of the hth largest jobs of $\mathcal{J}$ arrives; this sequence contains at least $h+1$ many of the $m+1$ largest jobs in $\mathcal{J}$.

Otherwise the job sequence is *unstable*.

Given $\varepsilon > 0$ and m, let $P_\varepsilon(m)$ be the infimum, over all proper job sequences $\mathcal{J}$, that a random permutation of $\mathcal{J}$ is stable, i.e.

$$P_\varepsilon(m) = \inf_{\mathcal{J} \text{ proper}} \mathbf{P}_{\sigma \sim S_n}[\mathcal{J}^\sigma \text{ is stable}].$$

As the main result of this section we will prove that this probability tends to 1 as $m \to \infty$.

Main Lemma 1 *For every* $\varepsilon > 0$, *there holds* $\lim_{m\to\infty} P_\varepsilon(m) = 1$.

The Main Lemma 1 implies that for any $\varepsilon > 0$ there exists an $m(\varepsilon)$ such that, for all $m \geq m(\varepsilon)$ and all $\mathcal{J}$, there holds $\mathbf{P}_{\sigma \sim S_n}[\mathcal{J}^\sigma \text{ is stable}] \geq 1 - \varepsilon$. In Sect. 4.3 we will show that *ALG* is $(c+\varepsilon)$-competitive on stable job sequences. This implies $\mathbf{P}_{\sigma \sim S_n}[ALG(\mathcal{J}^\sigma) \geq (c+\varepsilon)OPT(\mathcal{J})] \leq \varepsilon$ on proper sequences. By Lemma 2 this probability is 0 on plain sequences. We obtain the following corollary to Main Lemma 1.

Corollary 3 *If ALG is adversarially $(c+\varepsilon)$-competitive on stable sequences, for every $\varepsilon > 0$ and $m \geq m(\varepsilon)$ sufficiently large, then it is nearly c-competitive.*

In the remainder of this section we describe how to establish Main Lemma 1. We need some notation. In Sect. 3 the value L_j^t was defined with respect to a fixed job sequence that was clear from the context. We adopt the notation $L_j^t[\mathcal{J}^\sigma]$ to make this dependence visible. We adopt the same notation for the variables L, P_j^t, P_j, $p_{\max}^t$ and $p_{\max}$. For a fixed input $\mathcal{J}$ and variable $\sigma \in S_n$, we use the simplified notation $L_j^t[\sigma] = L_j^t[\mathcal{J}^\sigma]$. Again, we use the same notation for the variables P_j^t and $p_{\max}^t$.

At the heart of the proof of Main Lemma 1 is the Load Lemma. Observe that after t time steps in a random permutation of an input $\mathcal{J}$, each job has arrived with probability t/n. Thus the expected total processing time of the jobs seen so far is $t/n \cdot \sum_{s=1}^{n} p_s$. Equivalently, in expectation L^t equals $t/n \cdot L$. The Load Lemma proves that this relation holds with high probability. We set $t = \lfloor \varphi n \rfloor$.

Load Lemma *Given any $\varepsilon > 0$ and $\varphi \in (0,1]$, there exists an $m(\varepsilon, \varphi)$ such that for all $m \geq m(\varepsilon, \varphi)$ and all proper sequences $\mathcal{J}$, there holds*

$$\mathbf{P}_{\sigma \sim S_n}\left[\left|\frac{L^{\lfloor \varphi n \rfloor}[\mathcal{J}^\sigma]}{\varphi L[\mathcal{J}^\sigma]} - 1\right| \geq \varepsilon\right] \leq \varepsilon.$$

Proof Let us fix a proper job sequence $\mathcal{J}$. We use the shorthand $\hat{L}[\sigma] = \hat{L}[\mathcal{J}^\sigma] = L^{\lfloor \varphi n \rfloor}[\mathcal{J}^\sigma]$ and $L = L[\mathcal{J}]$.

Let $\delta = \frac{\varphi \varepsilon}{2}$. We will first treat the case that we have $p_{\max}[\mathcal{J}] = 1$ and every job size in $\mathcal{J}$ is of the form $(1+\delta)^{-j}$, for some $j \geq 0$. Note that we have in particular $c - 1 \leq L \leq \frac{1}{c-1}$ because we are working with a proper sequence. For $j \geq 0$ let h_j denote the number of jobs J_t of size $(1+\delta)^{-j}$ and, given $\sigma \in S_n$, let h_j^σ denote the number of such jobs J_t that additionally satisfy $\sigma(t) \leq \lfloor \varphi n \rfloor$, i.e. they are among the $\lfloor \varphi n \rfloor$ first jobs in the sequence $\mathcal{J}^\sigma$. We now have

$$L = \frac{1}{m}\sum_{j=0}^{\infty}(1+\delta)^{-j}h_j \quad \text{and} \quad \hat{L}[\sigma] = \frac{1}{m}\sum_{j=0}^{\infty}(1+\delta)^{-j}h_j^\sigma.$$

The random variables h_j^σ are hypergeometrically distributed, i.e. we sample $\lfloor \varphi n \rfloor$ jobs from the set of all n jobs and count the number of times we get one of the h_j many jobs of processing time $(1+\delta)^{-j}$. Hence, we know that the random variable h_j^σ has mean

$$\mathbf{E}[h_j^\sigma] = \frac{\lfloor \varphi n \rfloor}{n} h_j \leq \varphi h_j$$

and variance

$$\mathrm{Var}[h_j^\sigma] = \frac{h_j\big(n - h_j\big)\lfloor \varphi n \rfloor (n - \lfloor \varphi n \rfloor)}{n^2(n-1)}.$$

In particular,

$$\mathrm{Var}[h_j^\sigma] \leq h_j \leq (1+\delta)^j mL \leq (1+\delta)^j \frac{m}{c-1}.$$

By Chebyshev's inequality we have

$$\mathbf{P}\left[\left|h_j^\sigma - \varphi h_j\right| \geq (1+\delta)^{3j/4} m^{3/4}\right] \leq (1+\delta)^{-3j/2}\mathrm{Var}[h_j^\sigma] m^{-3/2} \leq (1+\delta)^{-j/2}\frac{m^{-1/2}}{c-1}.$$

In particular, by the Union Bound, with probability

$$P(m) = 1 - \sum_{j=0}^{\infty}(1+\delta)^{-j/2}\frac{m^{-1/2}}{c-1} = 1 - \frac{m^{-1/2}}{\left(1-\sqrt{1+\delta}\right)(c-1)} = 1 - O\big(m^{-1/2}\big)$$

we have for all j,

$$\left|h_j^\sigma - \varphi h_j\right| < (1+\delta)^{3j/4} m^{3/4}.$$

We conclude that the following holds:

$$\begin{aligned}
\left|\hat{L}[\sigma] - \varphi L\right| &= \left|\frac{1}{m}\sum_{j=0}^{\infty}(1+\delta)^{-j} h_j^\sigma - \frac{\varphi}{m}\sum_{j=0}^{\infty}(1+\delta)^{-j} h_j\right| \\
&\leq \sum_{j=0}^{\infty}(1+\delta)^{-j}\frac{\left|h_j^\sigma - h_j \cdot \varphi\right|}{m} \\
&< \sum_{j=0}^{\infty}(1+\delta)^{-j/4} m^{-1/4} \\
&= \frac{m^{-1/4}}{\left(1-(1+\delta)^{-1/4}\right)}.
\end{aligned}$$

In particular, with probability $P(m)$, we have

$$\left|\frac{\hat{L}[\sigma]}{\varphi L} - 1\right| = \frac{\left|\hat{L}[\sigma] - \varphi L\right|}{\varphi L} \leq \frac{m^{-1/4}}{\varphi(c-1)\big(1-(1+\delta)^{-1/4}\big)} = O\big(m^{-1/4}\big).$$

Hence, if we choose m large enough we can ensure that

$$\mathbf{P}\left[\left|\frac{\hat{L}[\sigma]}{\varphi L} - 1\right| > \frac{\varepsilon}{2}\right] \leq 1 - P(m) \leq \varepsilon. \tag{1}$$

So far we have assumed that $p_{\max}[\mathcal{J}] = 1$ and every job in $\mathcal{J}$ has a processing time of $(1+\delta)^{-j}$, for some $j \geq 0$. Now we drop these assumptions. Given an arbitrary sequence $\mathcal{J}$ with $0 < p_{\max}[\mathcal{J}] \neq 1$, let $\lfloor \mathcal{J} \rfloor$ denote the sequence obtained from $\mathcal{J}$ by first dividing every job processing time by $p_{\max}[\mathcal{J}]$ and rounding every job size down to the next power of $(1+\delta)^{-1}$. We have proven that inequality (1) holds for $\lfloor \mathcal{J} \rfloor$. The values L and $\hat{L}[\sigma]$ only change by a factor lying in the interval $[p_{\max}, (1+\delta)p_{\max})$ when passing over from $\lfloor \mathcal{J} \rfloor$ to $\mathcal{J}$. This implies that

$$\left|\frac{\hat{L}[\mathcal{J}^\sigma]}{\varphi L[\mathcal{J}]} - \frac{\hat{L}[\lfloor \mathcal{J} \rfloor^\sigma]}{\varphi L[\lfloor \mathcal{J} \rfloor]}\right| \leq \delta \frac{\hat{L}[\lfloor \mathcal{J} \rfloor^\sigma]}{\varphi L[\mathcal{J}]}.$$

Since $\hat{L}[\lfloor \mathcal{J} \rfloor^\sigma] \leq L[\mathcal{J}]$ we obtain

$$\left|\frac{\hat{L}[\mathcal{J}^\sigma]}{\varphi L[\mathcal{J}]} - \frac{\hat{L}[\lfloor \mathcal{J} \rfloor^\sigma]}{\varphi L[\lfloor \mathcal{J} \rfloor]}\right| \leq \frac{\delta}{\varphi} = \frac{\varepsilon}{2}.$$

Combining this with inequality (1) for $\lfloor \mathcal{J} \rfloor$ (and the triangle inequality), we obtain

$$\mathbf{P}\left[\left|\frac{\hat{L}[\mathcal{J}^\sigma]}{\varphi L[\mathcal{J}]} - 1\right| > \varepsilon\right] \leq \mathbf{P}\left[\left|\frac{\hat{L}[\lfloor \mathcal{J} \rfloor^\sigma]}{\varphi L[\lfloor \mathcal{J} \rfloor]} - 1\right| > \frac{\varepsilon}{2}\right] \leq \varepsilon.$$

Thus the lemma follows. □

We note that the Load Lemma does not hold for general sequences. A counterexample is a job sequence in which one job carries all the load, while all the other jobs have a negligible processing time. The proof of the Load Lemma relies on a lower bound of $R(\mathcal{J})$, which is $c - 1$ for proper sequences.

We present two consequences of the Load Lemma that will allow us to prove that stable sequences reveal information on the largest jobs when a certain processing volume has been scheduled. Consider a proper $\mathcal{J}$. Given $\mathcal{J}^\sigma = J_{\sigma(1)}, \ldots, J_{\sigma(n)}$ and $\varphi > 0$, let $N(\varphi)[\mathcal{J}^\sigma]$ be the number of jobs $J_{\sigma(t)}$ that are among the $m+1$ largest jobs in $\mathcal{J}$ and such that $L^t \leq \varphi L$.

Lemma 4 *Let $\varepsilon > 0$ and $\varphi \in (0, 1]$. Then there holds*

$$\lim_{m \to \infty} \inf_{\mathcal{J} \text{ proper}} \mathbf{P}_{\sigma \sim S_n}\left[N(\varphi + \varepsilon)[\mathcal{J}^\sigma] \geq \lfloor \varphi m \rfloor + h + 2\right] = 1.$$

Proof Fix any proper job sequence $\mathcal{J}$. For any $\mathcal{J}^\sigma$, let $N(\varphi+\varepsilon)[\sigma] = N(\varphi+\varepsilon)[\mathcal{J}^\sigma]$. Furthermore, let $\tilde{N}\left(\varphi + \frac{\varepsilon}{2}\right)[\sigma]$ denote the number of the $m+1$ largest jobs of $\mathcal{J}$ that appear among the first $\left\lfloor \left(\varphi + \frac{\varepsilon}{2}\right) n \right\rfloor$ jobs in $\mathcal{J}^\sigma$. Then we derive by the inclusion-exclusion principle:

$$\begin{aligned}
&\mathbf{P}_{\sigma\sim S_n}\left[N(\varphi+\varepsilon)[\sigma] \geq \lfloor \varphi m \rfloor + h + 2\right] \\
&\quad \geq \mathbf{P}_{\sigma\sim S_n}\left[\tilde{N}\left(\varphi+\frac{\varepsilon}{2}\right)[\sigma] \geq \lfloor \varphi m \rfloor + h + 2 \text{ and } L^{\left\lfloor\left(\varphi+\frac{\varepsilon}{2}\right)n\right\rfloor}[\sigma] < (\varphi+\varepsilon)L\right] \\
&\quad \geq \mathbf{P}_{\sigma\sim S_n}\left[\tilde{N}\left(\varphi+\frac{\varepsilon}{2}\right)[\sigma] \geq \lfloor \varphi m \rfloor + h + 2\right] + \mathbf{P}_{\sigma\sim S_n}\left[L^{\left\lfloor\left(\varphi+\frac{\varepsilon}{2}\right)n\right\rfloor}[\sigma] < (\varphi+\varepsilon)L\right] - 1.
\end{aligned}$$

By the Load Lemma the second summand can be lower bounded for every proper sequence $\mathcal{J}$ by a term approaching 1 as $m \to \infty$. Hence it suffices to verify that this is also possible for the term

$$\mathbf{P}_{\sigma\sim S_n}\left[\tilde{N}\left(\varphi+\frac{\varepsilon}{2}\right)[\sigma] \geq \lfloor \varphi m \rfloor + h + 2\right].$$

We will upper bound the probability of the opposite event by a term approaching 0 for $m \to \infty$. The random variable $\tilde{N}\left(\varphi+\frac{\varepsilon}{2}\right)[\sigma]$ is hypergeometrically distributed and therefore has expected value

$$E = \frac{\left\lfloor\left(\varphi+\frac{\varepsilon}{2}\right)n\right\rfloor}{n}(m+1) \geq \left(\varphi+\frac{2}{5}\varepsilon\right)(m+1).$$

Recall that for proper sequences $n > m$ holds. For the second inequality we require m and hence in also n to be large enough such that $\frac{1}{n} \leq \frac{\varepsilon}{10}$ holds. Again, the variable $\tilde{N}\left(\varphi+\frac{\varepsilon}{2}\right)[\sigma]$ is hypergeometrically distributed and its variance is thus

$$V = \frac{\left\lfloor\left(\varphi+\frac{\varepsilon}{2}\right)n\right\rfloor\left(n-\left\lfloor\left(\varphi+\frac{\varepsilon}{2}\right)n\right\rfloor\right)(m+1)(n-m-1)}{n^2(n-1)} \leq m+1.$$

Note that we have for m large enough:

$$\lfloor \varphi m \rfloor + h + 2 \leq \left(1+\frac{1}{5}\varepsilon\right)\varphi(m+1) \leq E - \frac{\varepsilon\varphi\sqrt{m+1}}{5}\sqrt{V}.$$

Hence, using Chebyshev's inequality, we have

$$\begin{aligned}
&\mathbf{P}_{\sigma\sim S_n}\left[\tilde{N}\left(\varphi+\frac{\varepsilon}{2}\right)[\sigma] < \lfloor \varphi m \rfloor + h + 2\right] \\
&\quad \leq \mathbf{P}_{\sigma\sim S_n}\left[E - \tilde{N}\left(\varphi+\frac{\varepsilon}{2}\right)[\sigma] > \frac{\varepsilon\varphi\sqrt{m+1}}{5}\sqrt{V}\right] \\
&\quad \leq \frac{25}{\varepsilon^2\varphi^2(m+1)}
\end{aligned}$$

and this term vanishes as $m \to \infty$. □

Lemma 5 *Let $\varepsilon > 0$ and $\varphi \in (0, 1]$. Then there holds*

$$\lim_{m\to\infty} \inf_{\mathcal{J}\text{ proper}} \mathbf{P}_{\sigma\sim S_n}\left[\forall_{\tilde{\varphi}\geq\varphi} N(\tilde{\varphi}+\varepsilon)[\mathcal{J}^\sigma] \geq \lfloor\tilde{\varphi}m\rfloor + h + 2\right] = 1.$$

Proof Let us fix any proper sequence $\mathcal{J}$ and set

$$\Lambda = \left\{1 - \frac{\varepsilon}{2}j \mid j \in \mathbb{N}, \varphi \leq 1 - \frac{\varepsilon}{2}j\right\}$$

which is a finite set whose size only depends on ε and φ. Given $\tilde{\varphi} \geq \varphi$, let $u(\tilde{\varphi})$ be the smallest element in Λ greater or equal to $\tilde{\varphi}$. Then

$$\tilde{\varphi} \leq u(\tilde{\varphi}) \leq \tilde{\varphi} + \frac{\varepsilon}{2}$$

and if we have

$$N(\tilde{\varphi}+\varepsilon)[\mathcal{J}^\sigma] < \lfloor\tilde{\varphi}m\rfloor + h + 2$$

there holds

$$N\left(u(\tilde{\varphi}) + \frac{\varepsilon}{2}\right)[\mathcal{J}^\sigma] < \lfloor\tilde{\varphi}m\rfloor + h + 2 \leq \lfloor u(\tilde{\varphi})m\rfloor + h + 2.$$

In particular, in order to prove the lemma it suffices to verify that

$$\lim_{m\to\infty} \inf_{\mathcal{J}\text{ proper}} \mathbf{P}_{\sigma\sim S_n}\left[\forall_{\tilde{\varphi}\in\Lambda} N\left(\tilde{\varphi} + \frac{\varepsilon}{2}\right)[\mathcal{J}^\sigma] \geq \lfloor\tilde{\varphi}m\rfloor + h + 2\right] = 1.$$

The latter is a consequence of applying Lemma 4 to all $\tilde{\varphi} \in \Lambda$ and the Union Bound. □

We can now conclude the main lemma of this section:

Proof of Main Lemma 1 A proper job sequence is stable if the following four properties hold.

- Once $L^t \geq (c-1)\frac{i}{m} \cdot L$ we have $p^t_{\max} \geq P_h$.
- For every $j \geq i$ the sequence ending once we have $L^t \geq \left(\frac{j}{m} + \frac{\varepsilon}{2}\right)L$ contains at least $j + h + 2$ of the $m + 1$ largest jobs.
- The sequence ending right before $L^t \geq \frac{i}{m}(c-1)\varepsilon L$ holds contains at least $h + 1$ of the $m + 1$ largest jobs.
- The sequence ending right before the first of the h largest jobs contains at least $h + 1$ of the $m + 1$ largest jobs.

By the Union Bound we may consider each property separately and prove that it holds with a probability that tends to 1 as $m \to \infty$.

Let $\varphi = (c-1)\frac{i}{m}$ and choose $\varepsilon > 0$. By the Load Lemma, for $m \geq m(\varepsilon, \varphi)$, after $t = \lfloor\varphi n\rfloor$ jobs of a proper job sequence $\mathcal{J}^\sigma$ have been scheduled, there holds $L^t \leq (c-1)\frac{i}{m} \cdot L$ with probability at least $1 - \varepsilon$. Observe that φ is a fixed problem parameter so that $m(\varepsilon, \varphi)$ is determined by ε. The probability of any particular

job being among the first t jobs in $\mathcal{J}^\sigma$ is $\lfloor \varphi n \rfloor / n$. Thus $p^t_{\max} \geq P_h$ holds with probability at least $1 - (1 - \lfloor \varphi n \rfloor / n)^h$. Since $\mathcal{J}^\sigma$ is proper, we have $n > m$. Furthermore, $h = h(m) \in \omega(1)$. Therefore, the probability that the first property holds tends to 1 as $m \to \infty$.

The second property is a consequence of Lemma 5 with $\varphi = \frac{i}{m}$. The third property follows from Lemma 4. We need to choose the ε in the statement of the lemma to be $\frac{i}{m}(c-1)\varepsilon$. Finally we examine the last property. In $\mathcal{J}^\sigma$ we focus on the positions of the $m+1$ largest jobs. Consider any of the h largest jobs. The probability that it is preceded by less than $h+1$ of the $m+1$ largest jobs is $(h+1)/(m+1)$. Thus the probability of the fourth property not to hold is at most $h(h+1)/(m+1)$. Since $h \in o(\sqrt{m})$, the latter expression tends to 0 as $m \to \infty$. □

4.3 An Adversarial Analysis

In this section we prove the following main result.

Main Lemma 2 *For every $\varepsilon > 0$ and $m \geq m(\varepsilon)$ sufficiently large, ALG is adversarially $(c+\varepsilon)$-competitive on stable job sequences.*

Consider a fixed $\varepsilon > 0$. Given Corollary 2, we may assume that $0 < \varepsilon < 2 - c$. Suppose that there was a stable job sequence $\mathcal{J}^\sigma$ such that $ALG(\mathcal{J}^\sigma) > (c+\varepsilon)OPT(\mathcal{J}^\sigma)$. We will derive a contradiction, given that m is large. In order to simplify notation, in the following let $\mathcal{J} = \mathcal{J}^\sigma$ be the stable job sequence violating the performance ratio of $c+\varepsilon$. Let $\mathcal{J} = J_1, \ldots, J_n$ and $OPT = OPT(\mathcal{J})$.

Let $J_{n'}$ be the first job that causes ALG to have a makespan greater than $(c+\varepsilon)OPT$ and let $b_0 = l_m^{n'-1}$ be the load of the least loaded machine $M_m^{n'-1}$ right before $J_{n'}$ is scheduled on it. The makespan after $J_{n'}$ is scheduled, called the *critical makespan*, is at most $b_0 + p_{n'} \leq b_0 + OPT$. In particular $b_0 > (c - 1 + \varepsilon)OPT$ as well as $p_{n'} > (c - 1 + \varepsilon)OPT$, see Lemma 3. Let

$$\lambda_{\text{start}} = \frac{c-1}{1+2c(2-c)} \approx 0.5426 \quad \text{and} \quad \lambda_{\text{end}} = \frac{1}{2(c-1+\varepsilon)} \approx 0.5898.$$

There holds $\lambda_{\text{start}} < \lambda_{\text{end}}$. The critical makespan of ALG is bounded by $b_0 + OPT < (1 + \frac{1}{c-1+\varepsilon})b_0 = (c+\varepsilon)\frac{b_0}{c-1+\varepsilon} = (c+\varepsilon)2\lambda_{\text{end}}b_0$. Since ALG does not achieve a performance ratio of $c+\varepsilon$ on $\mathcal{J}$ we have

$$P_{m+1} \leq OPT/2 < \lambda_{\text{end}} b_0. \tag{2}$$

Our main goal is to derive a contradiction to this inequality.

The impact of the variable P_h:

A new, crucial aspect in the analysis of ALG is P_h, the processing time of the hth largest job in the sequence $\mathcal{J}$. Initially, when the processing of $\mathcal{J}$ starts, we have no information on P_h and can only infer $P_{m+1} \geq \lambda_{\text{start}} b_0$. The second property in the definition of stable job sequences ensures that $p^t_{\max} \geq P_h$ once the load

ratio L^t/L is sufficiently large. Note that *ALG* then also works with this estimate because $P_h \leq p^t_{\max} \leq O^t$. This will allow us to evaluate the processing time of flatly scheduled jobs. In order prove that P_{m+1} is large, we will relate P_{m+1} and P_h, i.e. we will lower bound P_{m+1} in terms of P_h and vice versa. Using the relation we can then conclude $P_{m+1} \geq \lambda_{\text{end}} b_0$. In the analysis we repeatedly use the properties of stable job sequences and will explicitly point to it when this is the case.

We next make the relationship between P_h and P_{m+1} precise. Given $0 < \lambda$, let $f(\lambda) = 2c\lambda - 1$ and given $w > 0$, let $g(w) = (c(2c-3)-1)w + 4 - 2c \approx 0.2854 \cdot w + 0.3044$. We set $g_b(\lambda) = g\left(\frac{\lambda}{b}\right)b$ and $f_b(w) = f\left(\frac{w}{b}\right)b$, for any $b > 0$. Then we will lower bound P_{m+1} by $g_{b_0}(P_h)$ and P_h by $f_{b_0}(P_{m+1})$. We state two technical propositions.

Proposition 3 *For $\lambda > \lambda_{\text{start}}$, we have $g(f(\lambda)) > \lambda$.*

Proof Consider the function

$$\begin{aligned} F(\lambda) = g(f(\lambda)) - \lambda &= (c(2c-3)-1)(2c\lambda - 1) + 4 - 2c - \lambda \\ &= (4c^3 - 6c^2 - 2c - 1)\lambda - 2c^2 + c + 5 \\ &\approx 0.05446 \cdot \lambda + 0.01900. \end{aligned}$$

The function F is linear and strictly increasing in λ. Hence for the proposition to hold it suffices to verify that $F(\lambda_{\text{start}}) \geq 0$. We can now compute that $F(\lambda_{\text{start}}) \approx 0.04865 > 0$. □

Proposition 4 *For $0 < \varepsilon \leq 1$, we have $g(1-\varepsilon) > \lambda_{\text{end}}$.*

Note that the following proof determines the choice of our competitive ratio c, which was chosen minimal such that $Q[c] = 4c^3 - 14c^2 + 16c - 7 \geq 0$.

Proof We calculate that

$$\begin{aligned} g(1-\varepsilon) - \lambda_{\text{end}} &= (c(2c-3)-1)(1-\varepsilon) + 4 - 2c - \frac{1}{2(c-1+\varepsilon)} \\ &= \frac{2(c-1+\varepsilon)(2c^2 - 5c + 3 - (2c^2 - 3c - 1)\varepsilon) - 1}{2(c-1+\varepsilon)} \\ &= \frac{4c^3 - 14c^2 + 16c - 7 + (4-2c)\varepsilon - 2(2c^2 - 3c - 1)\varepsilon^2}{2(c-1+\varepsilon)}. \end{aligned}$$

Recall that $Q[c] = 4c^3 - 14c^2 + 16c - 7 = 0$. For $0 < \varepsilon \leq 1$ we have

$$(4-2c)\varepsilon - (2c^2 - 3c - 1)\varepsilon^2 \approx 0.3044 \cdot \varepsilon - 0.2854 \cdot \varepsilon^2 > 0.$$

Thus we see that $g(1-\varepsilon) - \lambda_{\text{end}} > 0$ and can conclude the lemma. □

4.3.1 Analyzing Large Jobs Towards Lower Bounding P_h and P_{m+1}

Let $b > (c - 1 + \varepsilon)OPT$ be a value such that immediately before $J_{n'}$ is scheduled at least $m - h$ machines have a load of at least b. Note that $b = b_0$ satisfies this condition but we will be interested in larger values of b as well. We call a machine *b-full* once its load is at least b; we call a job J a *b-filling job* if it causes the machine it is scheduled on to become b-full. We number the b-filling jobs according to their order of arrival $J^{(1)}, J^{(2)}, \ldots$ and let $t(j)$ denote the time of arrival of the jth filling job $J^{(j)}$.

Recall that our main goal is to show that $P_{m+1} \geq \lambda_{\text{end}} b_0$ holds. To this end we will prove that the *b_0-filling jobs* have a processing time of at least $\lambda_{\text{end}} b_0$. As there are m such jobs, the bound on P_{m+1} follows by observing that $J_{n'}$ arrives after all *b_0-filling jobs* are scheduled and that its processing time exceeds $\lambda_{\text{end}} b_0$ as well. In fact, since $OPT \geq b_0$, we have

$$p_{n'} > (c - 1)OPT > 0.847 \cdot OPT > \lambda_{\text{end}} b_0 \approx 0.5898 \cdot b_0. \tag{3}$$

We remark that different to previous analyses in the literature we do not solely rely on lower bounding the processing time of filling jobs. By using the third property of stable job sequences, we can relate load and the size of the $(m + 1)$st largest job at specific points in the time horizon, formally this is done later in Lemma 8.

In the following we regard b as fixed and omit it from the terms *filling job* and *full*. Let $\lambda = \max\{\lambda_{\text{start}} b, \min\{g_b(P_h), \lambda_{\text{end}} b\}\}$. We call a job *large* if it has a processing time of at least λ. Let $\tilde{t} = t(m - h)$ be the time when the $(m - h)$th filling job arrived. The remainder of this section is devoted to showing the following important Lemma 6. Some of the underlying lemmas, but not all of them, hold if $m \geq m(\varepsilon)$ is sufficiently large. We will make the dependence clear.

Lemma 6 *At least one of the following statements holds:*

- *All filling jobs are large.*
- *If $m \geq m(\varepsilon)$, there holds $P^{\tilde{t}}_{m+1} \geq \lambda = \max\{\lambda_{\text{start}} b, \min\{g_b(P_h), \lambda_{\text{end}} b\}\}$, i.e. there are at least $m + 1$ large jobs once the $(m - h)$-th filling job is scheduled.*

Before we prove the lemma we derive two important implications towards a lower bound of P_{m+1}.

Corollary 4 *We have $P_{m+1} \geq \lambda = \max\{\lambda_{\text{start}} b_0, \min\{g_{b_0}(P_h), \lambda_{\text{end}} b_0\}\}$.*

Proof Apply the previous lemma, taking into account that $b \geq b_0$, and use that there are m many b_0-filling jobs followed by $J_{n'}$. The latter has size at least λ by inequality (3). □

We also want to lower bound the processing time of the $(m + 1)$st largest job at time $\tilde{t}$. However, at that time only $m - h$ filling jobs have arrived. The next lemma ensures that, if additionally P_h is not too large, this is not a problem.

Corollary 5 *If $P_h \leq (1-\varepsilon)b$ and $m \geq m(\varepsilon)$, the second statement in Lemma 6 holds, i.e. $P^{\tilde{t}}_{m+1} \geq \lambda = \max\{\lambda_{\text{start}} b, \min\{g_b(P_h), \lambda_{\text{end}} b\}\}$.*

The proof of the lemma makes use of the fourth property of stable job sequences. In particular we would not expect such a result to hold in the adversarial model.

Proof We will show that the first statement in Lemma 6 implies the second one if $P_h \leq (1-\varepsilon)b$ holds. In order to conclude the second statement it suffices to verify that at least $m+1$ jobs of processing time λ have arrived until time $\tilde{t}$. By the first statement we know that there were $m-h$ large filling jobs coming before time $\tilde{t}$. Hence it is enough to verify that $h+1$ large jobs arrive (strictly) before the first filling job J.

To show that there are $h+1$ jobs with a processing time of at least P_{m+1} before the first filling job J, we use the last property of stable job sequences. If J is among the h largest jobs, we are done immediately by the condition. Else J had size at most $P_h \leq (1-\varepsilon)b$. Assume $J = J_t$ was scheduled on the machine M^{t-1}_j, for $j \in \{i, m-h, m\}$, and let $l = l^{t-1}_j$ be its load before J was scheduled. Because J is a filling job we have

$$l \geq b - P_h \geq \varepsilon b \geq \varepsilon(c-1)OPT.$$

In particular, before J was scheduled, the average load at that time was at least

$$\frac{jl}{m} \geq \frac{il}{m} \geq \varepsilon \frac{i}{m}(c-1)OPT.$$

Again, by the last property of stable job sequences, at least $h+1$ jobs of processing time at least P_{m+1} were scheduled before this was the case. □

We introduce late and early filling jobs. We later need a certain condition to hold, namely the ones stated in Lemma 8, in order to show that the early filling jobs are large. We show that if this condition is not met, the fact that the given job sequence is stable ensures that $P^{\tilde{t}}_m \geq \lambda$.

Let s be chosen maximal such that the sth filling job is scheduled steeply. If $s \leq i$, then set $s = i+1$ instead. We call all filling jobs $J^{(j)}$ with $j > i$ that are scheduled flatly *late filling jobs*. All other filling jobs are called *early filling jobs*. In particular the job $J^{(s+1)}$ and the filling jobs afterwards are late filling jobs. The following proposition implies that the fillings jobs after $J^{(m-h)}$, if they exist, are all late, i.e. scheduled flatly.

Proposition 5 *We have $s \leq m-h$ if $m \geq m(\varepsilon)$.*

Proof of Proposition 5 Let $\tilde{h} < h$ and $t = t(m-\tilde{h})$ be the time the $(m-\tilde{h})$th filling job J arrived. We need to see that J was scheduled flatly. Assume that was not the case.

We know that for $j \leq m - \tilde{h}$ we have $l_j^{t-1} \geq b > (c - 1 + \varepsilon)OPT$. In particular we have

$$L_{i+1}^{t-1} = \frac{1}{m-i} \sum_{j=i+1}^{m} l_j^{t-1} > \frac{m-i-h-1}{m-i}(c - 1 + \varepsilon)OPT \geq (c-1)OPT.$$

For the last inequality we need to choose m large enough. If the schedule was steep at time t, then we had for every $j \leq k$

$$l_j^{t-1} \geq l_k^{t-1} \geq \alpha(c-1)OPT = \frac{2(c-1)^2}{2c-3}OPT.$$

But then the average load at time $t-1$ would be:

$$\begin{aligned} L^{t-1} &= \frac{1}{m} \sum_{j=1}^{m} l_j^{t-1} \\ &> \frac{k\frac{2(c-1)^2}{2c-3} + (m-h-k)(c-1)}{m} OPT \\ &\geq \frac{((4c-7)m + 2h)\frac{2(c-1)^2}{2c-3} + (m-(4c-7)m-h)(c-1)}{m} OPT \\ &\approx 1.3247 \cdot OPT. \end{aligned}$$

For the second inequality we need to observe that we have $k \geq (4c-7)m + 2h$ and that the previous term decreases if we decrease k. One also can check that the second last term is minimized if $h = 0$.

But now we have shown $L^{t-1} > OPT$, which is a contradiction. Hence the schedule could not have been steep at time $t-1$. □

We need a technical lemma. For any time t, let $\overline{L}_s^t = \frac{1}{m-h-s+1} \sum_{j=s}^{m-h} l_j^t$ be the average load on the machines numbered s to $m-h$.

Lemma 7 *If $\overline{L}_s^{t(s)-1} \geq \alpha^{-1} b$ holds and $m \geq m(\varepsilon)$ then $L^{t(s)-1} > \left(\frac{s}{m} + \frac{\varepsilon}{2}\right) \cdot L$.*

This lemma comes down to a mere computation. While being simple at its core, we have to account for various small error terms. These arise in three ways. Some are inherent to the properties of stable sequences. Others arise from the rounding involved in the definition of certain numbers, i in particular. Finally, the small number h introduces such an error. While all these errors turn out to be negligible, rigorously showing so is technical. Note that the following proof also determines our choice of the value i. For larger values the proof would not hold.

Proof of Lemma 7 Let $t = t(s) - 1$. We have $l_j^t \geq b$ for $j \leq s-1$ as the first $s-1$ machines are full. Considering the load on the machines numbered up to $m-h$ we obtain

$$\begin{aligned} L_{i+1}^t &\geq \frac{\sum_{j=i+1}^{s-1} l_j^t + (m-h-s+1)\overline{L}_s^t}{m-i} \\ &\geq \frac{(s-i-1)b + (m-h-s+1)\alpha^{-1}b}{m-i} \\ &\geq \alpha^{-1}b + \frac{s-i-1}{m-i}(1-\alpha^{-1})b - \frac{h}{m-i}\alpha^{-1}b \\ &= \alpha^{-1}b + \frac{s-i-1}{m-i}\frac{b}{2(c-1)} - \frac{h}{m-i}\alpha^{-1}b. \end{aligned}$$

If $s > i+1$, the schedule was steep at time $t = t(s) - 1$ and hence

$$l_k^t \geq \alpha L_{i+1}^t > b + \frac{s-i-1}{m-i}\frac{\alpha \cdot b}{2(c-1)} - \frac{h \cdot b}{m-i}.$$

Since $l_k^t \geq l_i^t \geq b$, the previous inequality holds for $s = i+1$, too, no matter whether $J^{(s)} = J^{(i+1)}$ was scheduled flatly or steeply. We hence get, for all $s \geq i+1$,

$$\begin{aligned} L^t &\geq \frac{k l_k^t + (s-k-1)l_{s-1}^t + (m-h-s+1)\overline{L}_s^t}{m} \\ &> \frac{k\left(b + \frac{s-i-1}{m-i} \cdot \frac{\alpha \cdot b}{2(c-1)} - \frac{h \cdot b}{m-i}\right) + (s-k-1)b + (m-h-s+1)\left(b - \frac{b}{2(c-1)}\right)}{m} \\ &= \left(1 + \frac{1}{2(c-1)}\left(\frac{k}{m}\frac{s-i-1}{m-i} \cdot \alpha - \frac{m-s+1}{m}\right)\right)b - \left(\frac{k \cdot h}{m(m-i)} + \frac{h \cdot \alpha^{-1}}{m}\right)b. \end{aligned}$$

In the above difference, we first examine the first term, which is minimized if $s = i+1$. With this setting it is still lower bounded by

$$\left(1 - \frac{m-i}{2(c-1)m}\right)b > \left(1 - \frac{2(2-c)}{2(c-1)}\right)b \approx 0.8205 \cdot b > \frac{3b}{4}.$$

In the second term of the above difference $\frac{k}{m-i} = \frac{2i-m}{m-i}$ is increasing in i, where $i \leq (2c-3)m + h + 1$. We choose m large enough such that

$$\frac{k}{m-i} \leq \frac{(4c-7)m + 2(h+1)}{2(2-c)m - (h+1)} \leq 1.5.$$

There holds $\alpha^{-1} < 0.5$. Thus the second term in the difference is upper bounded by $\frac{2hb}{m}$.

Recall that $b > (c-1+\varepsilon)OPT$. Furthermore, $0 < \varepsilon < 2-c$ such that $c-1+\varepsilon < 1$. Therefore, we obtain

$$L^t > \left(c - 1 + \frac{1}{2}\left(\frac{\mathbf{k}}{m}\frac{s-i-1}{m-\mathbf{i}} \cdot \alpha - \frac{m-\mathbf{i}}{m} + \frac{s-i+1}{m}\right) + \frac{3}{4}\varepsilon - \frac{2h}{m}\right)OPT.$$

In the previous term we intentionally highlighted three variables. It is easy to check that if we decrease these variables, the term decreases, too. We do this by setting $\mathbf{k} = (4c-7)m$ and $\mathbf{i} = (2c-3)m$ (while ignoring the non-highlighted occurrences

of i). We also assume that m is large enough such that $\frac{\varepsilon}{4} \geq \frac{3h+2}{m}$. Then the previous lower bound on L^t can be brought to the following form:

$$L^t > \left(2c - 3 + \frac{1}{2}\left(\frac{4c-7}{2(2-c)} \cdot \alpha + 1\right)\frac{s-i-1}{m} + \frac{h+2}{m} + \frac{\varepsilon}{2}\right)OPT$$

Using that $\frac{i+1}{m} < 2c - 3 + \frac{h+2}{m}$ and evaluating the term in front of $\frac{s-i-1}{m}$ we get

$$L^t > \left(\frac{i+1}{m} + 1.0666 \cdot \frac{s-i-1}{m} + \frac{\varepsilon}{2}\right)OPT > \left(\frac{s}{m} + \frac{\varepsilon}{2}\right)OPT.$$

The lemma follows by noting that $OPT \geq L$. □

Lemma 8 *If the late filling jobs are large,* $\overline{L}_s^{t(s)-1} \geq \alpha^{-1}b$ *and* $m \geq m(\varepsilon)$*, we have* $P^{\tilde{t}}_{m+1} \geq \lambda$.

Proof Assume that the conditions of the lemma hold. By Lemma 7 we have $L^{t(s)-1} > \left(\frac{s}{m} + \frac{\varepsilon}{2}\right) \cdot L$. By the third property of stable sequences, at most $m + 1 - (s + h + 2) = m - s - h - 1$ of the largest $m + 1$ jobs appear in the sequence starting after time $t(s) - 1$. However, this sequence contains $m - h - s$ late filling jobs. Thus there exists a late filling job that is not among the $m + 1$ largest jobs. As it has a processing time of at least λ, by the assumption of the lemma, $P_{m+1} \geq \lambda$ holds.

Now consider the $m + 1$ largest jobs of the entire sequence that arrive before $J^{(s)}$ as well as the jobs $J^{(s+1)}, \ldots, J^{(m-h)}$. There are at least $s + h + 2$ of the former and $m - h - s$ of the latter. Thus we have found a set of at least $m + 1$ jobs arriving before (or at) time $\tilde{t} = t(m - h)$. Moreover, we argued that all these jobs have a processing time of at least λ. Hence $P^{\tilde{t}}_{m+1} \geq \lambda$ holds true. □

We are ready to evaluate the processing time of filling jobs to prove Lemma 6, which we will do in the following two lemmas.

Lemma 9 *Any late filling job's processing time exceeds* $\max\{\lambda_{\text{start}}b, g_b(P_h)\}$.

Proof Let $j \geq i + 1$ such that $J^{(j)}$ was scheduled flatly. Set $t = t(j) - 1$ and $l = l_i^t$. Because at least i machines were full, we have have $L^t \geq b \cdot \frac{i}{m} \geq (c-1)\frac{i}{m}OPT \geq (c-1)\frac{i}{m}L$. Hence by Definition 3 we have $p^t_{\max} \geq P_h$.

Let $\tilde{\lambda} = \max\{\lambda_{\text{start}}b, g_b(P_h)\}$. We need to show that $J^{(j)}$ has a processing time strictly greater than $\tilde{\lambda}$. If we have $l^t_{m-h} < b - \tilde{\lambda}$, then this was the case because $J^{(j)}$ increased the load of some machine from a value smaller than $b - \tilde{\lambda}$ to b. Hence let us assume that we have $l^t_{m-h} \geq b - \tilde{\lambda}$. In particular we have

$$L^t \geq \frac{(j-1)l + (m-j-h+1)(b-\tilde{\lambda})}{m}.$$

By the definition of a late filling job, $J^{(j)}$ was scheduled flatly. In particular, it would have been scheduled on machine M_i^t (which was not the case) if any of the following two inequalities did not hold:

– $p_t + l > cp^t_{\max} \geq cP_h$
– $p_t + l > cL^t$

If $l \leq cP_h - \tilde{\lambda}$ held true, we get $p_t > \tilde{\lambda}$ from the first inequality. Thus we only need to treat the case that $l > cP_h - \tilde{\lambda}$ held true. We also know that we have $l \geq b$, because the ith machine is full. Hence we may assume that

$$l \geq \max\{b, cP_h - \tilde{\lambda}\}.$$

In order to derive the lemma we need to prove that $p_t - \tilde{\lambda} > 0$ holds. Using the second inequality we get

$$p_t - \tilde{\lambda} > cL^t - l - \tilde{\lambda} \geq c\frac{(j-1)l + (m-j-h+1)(b-\tilde{\lambda})}{m} - l - \tilde{\lambda}.$$

Using that $(2c-3)m + h < i < j - 1$ and $b - \tilde{\lambda} < b \leq l$ hold, the previous term does not increase if we replace $j - 1$ by $(2c-3)m + h$. The resulting term is

$$p_t - \tilde{\lambda} > c\frac{((2c-3)m+h)l + (m-(2c-3)m-2h+1)(b-\tilde{\lambda})}{m} - l - \tilde{\lambda}.$$

Now let us observe that we have $l \geq b \geq 2(b - \lambda_{\text{start}}b) \geq 2(b - \tilde{\lambda})$. Hence the previous term is minimized if we set $h = 0$. We get

$$p_t - \tilde{\lambda} > c\left[(2c-3)l + (1-(2c-3))(b-\tilde{\lambda})\right] - l - \lambda.$$

As $c(2c-3) - 1 \approx 0.2584 > 0$ the above term does not increase if we replace l by either value: b or $cP_h - \tilde{\lambda}$.

If we have $\tilde{\lambda} = \lambda_{\text{start}}b$, we choose $l = b$ and get

$$\begin{aligned} p_t - \lambda_{\text{start}}b &> c\left[(2c-3)b + (1-(2c-3))(b-\lambda_{\text{start}}b)\right] - b - \lambda_{\text{start}}b \\ &= (c-1)b - (1+2c(2-c))\lambda_{\text{start}}b \\ &= (c-1)b - (c-1)b \\ &= 0. \end{aligned}$$

The third equality uses the definition of λ_{start}. The lemma follows if $\tilde{\lambda} = \lambda_{\text{start}}b$.

Otherwise, if $\tilde{\lambda} = g_b(P_h)$, we choose $l = cP_h - \tilde{\lambda}$ and get

$$\begin{aligned} p_t - \tilde{\lambda} &> c\left[(2c-3)(cP_h - \tilde{\lambda}) + (1-(2c-3))(b-\tilde{\lambda})\right] - (cP_h - \tilde{\lambda}) - \tilde{\lambda} \\ &= (c^2(2c-3) - c)P_h + (c(4-2c))b - cg_b(P_h) \\ &= 0. \end{aligned}$$

Here the last equality follows from the definition of g_b. The lemma follows in the case $\tilde{\lambda} = g_b(P_h)$. □

Lemma 10 *If $\overline{L}_s^{t(s)-1} < \alpha^{-1}b$ holds, the early filling jobs have a processing time of at least $\lambda_{\text{end}}b$.*

Before proving Lemma 10 let us observe the following, strengthening its condition.

Lemma 11 *We have*

$$L_{i+1}^{t(i+1)-1} \leq L_{i+2}^{t(i+2)-1} \leq \ldots L_{s}^{t(s)-1}.$$

Proof Let $i+1 \leq j < s$. It suffices to verify that

$$L_{j}^{t(j)-1} \leq L_{j+1}^{t(j)} \leq L_{j+1}^{t(j+1)-1}.$$

The second inequality is obvious because for every r the loads l_r^t can only increase as t increases. For the first inequality we note that by definition the job $J^{(j)}$ was scheduled steeply and hence on a least loaded machine. This machine became full. Thus it is not among the $m-j$ least loaded machines at time $t(j)$. In particular $L_{j+1}^{t(j)}$, the average over the $m-j$ smallest loads at time $t(j)$, is also the average of the $m-j+1$ smallest loads excluding the smallest load at time $t(j)-1$. Therefore it cannot be less than $L_j^{t(j)-1}$. □

Proof of Lemma 10 Let $i < j \leq s$ such that $J^{(j)}$ was an early filling job. By Lemma 11 we have $L_j^{t(j)-1} \leq L_s^{t(s)-1} < \alpha^{-1}b = b - \frac{b}{2(c-1)} < b - \lambda_{\text{end}}b$. By definition $J^{(j)}$ was scheduled on a least loaded machine $M_m^{t(j)-1}$ which had load less than $L_j^{t(j)-1} < b - \lambda_{\text{end}}b$ before and at least b afterwards because it became full. In particular $J^{(j)}$ had size $\lambda_{\text{end}}b$.

For $k < j \leq i$ the job $J^{(j)}$ is scheduled steeply because we have by Lemma 11

$$l_k^{t(j)-1} \geq b > \alpha L_s^{t(s)-1} \geq \alpha L_{i+1}^{t(i+1)-1} \geq \alpha L_{i+1}^{t(j)-1}.$$

Thus for $k < j \leq i$ the job $J^{(j)}$ is scheduled on the least loaded machine $M_m^{t(j)-1}$, whose load $l_m^{t(j)-1}$ is bounded by

$$l_m^{t(j)-1} \leq L_{i+1}^{t(j)-1} \leq L_s^{t(s)-1} < \alpha^{-1}b = b - \frac{b}{2(c-1)} < b - \lambda_{\text{end}}b.$$

Hence the job $J^{(j)}$ had a size of at least $\lambda_{\text{end}}b$. We also observe that we have

$$l_i^{t(k)-1} \leq l_{i+1}^{t(k+1)-1} \leq \ldots \leq l_{i+(m-i)}^{t(k+(m-i))-1} = l_m^{t(i)-1} < b - \lambda_{\text{end}}b.$$

In particular for $1 \leq j \leq k$ any filling job $J^{(j)}$ filled a machine with a load of at most $\max\{l_m^{t(k)}, l_i^{t(k)}\} = l_i^{t(k)} < b - \lambda_{\text{end}}b$. Hence it had a size of at least $\lambda_{\text{end}}b$. □

We now conclude the main lemma of this subsection, Lemma 6.

Proof of Lemma 6 By Lemma 9, all late filling jobs are large. We distinguish two cases depending on whether or not $L_s^{t(s)-1} < \alpha^{-1}b$ holds. If it does, all filling jobs are large by Lemma 10 and the first statement in Lemma 6 holds. Otherwise, the second statement in Lemma 6 holds by Lemma 8. □

4.3.2 Lower Bounding P_h and P_{m+1}

In this section we establish the following relations on P_h and P_{m+1}.

Lemma 12 *There holds $P_h > (1-\varepsilon)b_0$ or $P_{m+1} \geq \lambda_{\text{end}} b_0$ if $m \geq m(\varepsilon)$.*

For the proof we need a way to lower bound the processing time of a job J_t depending on P^t_{m+1}:

Lemma 13 *Let J_t be any job scheduled flatly on the least loaded machine and let $b = l^{t-1}_{m-h}$ be the load of the $(h+1)$-th least loaded machine. Then J_t has a processing time of at least $f_b(P^t_{m+1})$.*

Proof From the fact that J_t was not scheduled on the $(h+1)$th least loaded machine M^t_{m-h} we derive that $p_t > c \cdot O^t - b \geq c \cdot P^t_{m+1} - b = f_b(P^t_{m+1})$ holds. □

Proof of Lemma 12 Assume for a contradiction that we had $P_h \leq (1-\varepsilon)b_0$. Let $J = J_t$ be the smallest among the h last b_0-filling jobs. Then J has a processing time $p \leq P_h$. We want to derive a contradiction to that. Let $b_1 = l^{t-1}_{m-h}$ be the load of the $(m-h)$th machine right before J was scheduled. Because this machine was b_0-full at that time we know that $b_1 \geq b_0 > (c-1+\varepsilon)OPT$ holds and it makes sense to consider b_1-filling jobs. Let $\tilde{t}$ be the time the $(m-h)$th b_1-filling job arrived. By Lemma 6 we have $P^{\tilde{t}}_{m+1} \geq \lambda = \max\{\lambda_{\text{start}} b_1, \min\{g_{b_1}(P_h), \lambda_{\text{end}} b_1\}\}$.

If we have $\lambda = \lambda_{\text{end}} b_1 \geq \lambda_{\text{end}} b_0$ we have already proven $P_{m+1} \geq \lambda_{\text{end}} b_0$ and the lemma follows. So we are left to treat the case that we have $P^{\tilde{t}}_{m+1} \geq \lambda = \max\{\lambda_{\text{start}} b_1, g_{b_1}(P_h)\}$.

Now we can derive the following contradiction:

$$P^{\tilde{t}}_{m+1} \geq g_{b_1}(P_h) \geq g_{b_1}(p) \geq g_{b_1}\left(f_{b_1}\left(P^{\tilde{t}}_{m+1}\right)\right) = g\left(f\left(\frac{P^{\tilde{t}}_{m+1}}{b_1}\right)\right) b_1 > P^{\tilde{t}}_{m+1}.$$

For the second inequality, we use the monotonicity of $g_{b_1}(-)$. The third inequality follows from Lemma 13 and the last one from Proposition 3. □

4.3.3 Establishing Main Lemma 2.

Let $m \geq m(\varepsilon)$ be sufficiently large. The machine number $m(\varepsilon)$ is determined by the proofs of Proposition 5 and Lemma 7, and then carries over to the subsequent lemmas. Let us assume for a contradiction sake that there was a stable sequence $\mathcal{J}$ such that $ALG(\mathcal{J}) > (c+\varepsilon)OPT(\mathcal{J})$. As argued in the beginning of Sect. 4.3, see (2), it suffices to show that $P_{m+1} \geq \lambda_{\text{end}} b_0$. If this was not the case, we would have $P_h \geq (1-\varepsilon)b_0$ by Lemma 12. In particular by Proposition 4 we had $g_{b_0}(P_h) = g(1-\varepsilon)b_0 > \lambda_{\text{end}} b_0$. But now Lemma 4 shows that $P_{m+1} \geq \max\{\lambda_{\text{start}} b_0, \min\{g_{b_0}(P_h), \lambda_{\text{end}} b_0\}\} = \lambda_{\text{end}} b_0$.

We conclude, by Corollary 3, that *ALG* is nearly c-competitive.

5 Lower Bounds

We present lower bounds on the competitive ratio of any deterministic online algorithm in the random-order model. Theorem 3 implies that if a deterministic online algorithm is c-competitive with high probability as $m \to \infty$, then $c \geq 3/2$.

Theorem 2 *Let A be a deterministic online algorithm that is c-competitive in the random-order model. Then $c \geq 4/3$ if $m \geq 8$.*

Theorem 3 *Let A be a deterministic online algorithm that is nearly c-competitive. Then $c \geq 3/2$.*

A basic family of inputs are job sequences that consist of jobs having an identical processing time of, say, 1. We first analyze them and then use the insight to derive our lower bounds. Let $m \geq 2$ be arbitrary. For any deterministic online algorithm A, let $r(A, m)$ be the maximum number in $\mathbb{N} \cup \{\infty\}$ such that A handles a sequence consisting of $r(A, m) \cdot m$ jobs with an identical processing time of 1 by scheduling each job on a least loaded machine.

Lemma 14 *Let $m \geq 2$ be arbitrary. For every deterministic online algorithm A, there exists a job sequence $\mathcal{J}$ such that $A^{\text{rom}}(\mathcal{J}) \geq (1 + \frac{1}{r(A,m)+1})OPT(\mathcal{J})$. We use the convention that $\frac{1}{\infty+1} = 0$.*

Proof For $r(A, m) = \infty$ there is nothing to show. For $r(A) < \infty$, consider the sequence $\mathcal{J}$ consisting of $(r(A, m) + 1) \cdot m$ identical jobs, each having a processing time of 1. It suffices to analyze the algorithm adversarially as all permutations of the job sequence are identical. After having handled the first $r(A, m) \cdot m$ jobs, the algorithm A has a schedule in which every machine has load of $r(A, m)$. By the maximality of $r(A, m)$, the algorithm A schedules one of the following m jobs on a machine that is not a least loaded one. The resulting makespan is $r(A, m) + 2$. The lemma follows since the optimal makespan is $r(A, m) + 1$. □

Proof of Theorem 2 Let $m \geq 8$ be arbitrary. Consider any deterministic online algorithm A. If $r(A, m) \leq 2$, then, by Lemma 14, there exists a sequence $\mathcal{J}$ such that $A^{\text{rom}}(\mathcal{J}) \geq \frac{4}{3} \cdot OPT(\mathcal{J})$. Therefore, we may assume that $r(A, m) \geq 3$. Consider the input sequence $\mathcal{J}$ consisting of $4m - 4$ identical small jobs of processing time 1 and one large job of processing time 4. Obviously $\text{OPT}(\mathcal{J}) = 4$.

Let i be the number of small jobs preceding the large job in $\mathcal{J}^\sigma$. The random variable i takes any (integer) value between 0 and $4m - 4$ with probability $\frac{1}{4m-3}$. Since $r(A, m) \geq 3$ the least loaded machine has load of at least $l = \left\lfloor \frac{i}{m} \right\rfloor$ when the large job arrives. Thus $A(\mathcal{J}^\sigma) \geq l + 4$. The load l takes the values 0, 1 and 2 with probability

$\frac{m}{4m-3}$ and the value 3 with probability $\frac{m-3}{4m-3}$. Hence the expected makespan of algorithm A is at least

$$A^{\text{rom}}(\mathcal{J}) \geq \frac{m}{4m-3} \cdot (0+1+2) + \frac{m-3}{4m-3} \cdot 3 + 4 = \frac{6m-9}{4m-3} + 4 > \frac{16}{3} = \frac{4}{3}\text{OPT}(\mathcal{J}).$$

For the last inequality we use that $m \geq 8$. □

Proof of Theorem 3 Let $m \geq 2$ be arbitrary and let A be any deterministic online algorithm. If $r(A, m) = 0$, then consider the sequence $\mathcal{J}$ consisting of m jobs with a processing time of 1 each. On every permutation of $\mathcal{J}$ algorithm A has a makespan of 2, while the optimum makespan is 1. If $r(A, m) \geq 1$, then consider the sequence $\mathcal{J}$ consisting of $2m-2$ small jobs having a processing time of 1 and one large job with a processing time of 2. Obviously $OPT(\mathcal{J}) = 2$. If the permuted sequence starts with m small jobs, the least loaded machine has load 1 once the large job arrives. Under such permutations $A(\mathcal{J}^{\sigma}) \geq 3 = \frac{3}{2} \cdot \text{OPT}(\mathcal{J})$ holds true. The probability of this happening is $\frac{m-1}{2m-1}$. The probability approaches $\frac{1}{2}$ and in particular does not vanish, for $m \to \infty$. Thus, if A is nearly c-competitive, then $c \geq 3/2$. □

Funding Open Access funding enabled and organized by Projekt DEAL.

References

1. Albers, S.: Better bounds for online scheduling. SIAM J. Comput. **29**(2), 459–473 (1999)
2. Babaioff, M., Immorlica, N., Kempe, D., Kleinberg, R.: A knapsack secretary problem with applications. In: Proceedings of 10th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX). Springer, pp. 16–28 (2007)
3. Babaioff, M., Immorlica, N., Kempe, D., Kleinberg, R.: Matroid secretary problems. J. ACM **65**(6), 1–26 (2018)
4. Bartal, Y., Fiat, A., Karloff, H., Vohra, R.: New algorithms for an ancient scheduling problem. In: Proceedings of 24th ACM Symposium on Theory of Computing (STOC), pp. 51–58 (1992)
5. Bartal, Y., Karloff, H., Rabani, Y.: A better lower bound for on-line scheduling. Inf. Process. Lett. **50**(3), 113–116 (1994)
6. Chen, B., van Vliet, A., Woeginger, G.: A lower bound for randomized on-line scheduling algorithms. Inf. Process. Lett. **51**(5), 219–222 (1994)

7. Chen, L., Ye, D., Zhang, G.: Approximating the optimal algorithm for online scheduling problems via dynamic programming. Asia Pac. J. Oper. Res. **32**(01), 1540011 (2015)
8. Cheng, T., Kellerer, H., Kotov, V.: Semi-on-line multiprocessor scheduling with given total processing time. Theor. Comput. Sci. **337**(1–3), 134–146 (2005)
9. Dohrau, J.: Online makespan scheduling with sublinear advice. In: 41st International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM). Springer, pp. 177–188 (2015)
10. Dynkin, E.: The optimum choice of the instant for stopping a Markov process. Sov. Math. **4**, 627–629 (1963)
11. Englert, M., Özmen, D., Westermann, M.: The power of reordering for online minimum makespan scheduling. In: Proceedings of 49th 676 IEEE Annual Symposium on Foundations of Computer Science (FOCS). IEEE, pp. 603–612 (2008)
12. Faigle, U., Kern, W., Turán, G.: On the performance of on-line algorithms for partition problems. Acta Cybern. **9**(2), 107–119 (1989)
13. Feldman, M., Svensson, O., Zenklusen, R.: A simple O (log log (rank))-competitive algorithm for the matroid secretary problem. In: Proceedings of 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). SIAM, pp. 1189–1201 (2014)
14. Fleischer, R., Wahl, M.: On-line scheduling revisited. J. Sched. **3**(6), 343–353 (2000)
15. Galambos, G., Woeginger, G.: An on-line scheduling heuristic with better worst-case ratio than Graham's list scheduling. SIAM J. Comput. **22**(2), 349–355 (1993)
16. Göbel, O., Kesselheim, T., Tönnis, A.: Online appointment scheduling in the random order model. In: Algorithms-ESA 2015. Springer, pp. 680–692 (2015)
17. Goel, G., Mehta, A.: Online budgeted matching in random input models with applications to Adwords. SODA **8**, 982–991 (2008)
18. Gormley, T., Reingold, N., Torng, E., Westbrook, J.: Generating adversaries for request-answer games. In: Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 564–565 (2000)
19. Graham, R.: Bounds for certain multiprocessing anomalies. Bell Syst. Tech. J. **45**(9), 1563–1581 (1966)
20. Gupta, A., Mehta, R., Molinaro, M.: Maximizing Profit with Convex Costs in the Random-Order Model. arXiv preprint arXiv:1804.08172 (2018)
21. Hochbaum, D., Shmoys, D.: Using dual approximation algorithms for scheduling problems theoretical and practical results. J. ACM **34**(1), 144–162 (1987)
22. Karande, C., Mehta, A., Tripathi, P.: Online bipartite matching with unknown distributions. In: Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing, pp. 587–596 (2011)
23. Karger, D., Phillips, S., Torng, E.: A better algorithm for an ancient scheduling problem. J. Algorithms **20**(2), 400–430 (1996)
24. Kellerer, H., Kotov, V.: An efficient algorithm for bin stretching. Oper. Res. Lett. **41**(4), 343–346 (2013)
25. Kellerer, H., Kotov, V., Speranza, M.G., Tuza, Z.: Semi on-line algorithms for the partition problem. Oper. Res. Lett. **21**(5), 235–242 (1997)
26. Kenyon, C.: Best-fit bin-packing with random order. SODA **96**, 359–364 (1996)
27. Kesselheim, T., Tönnis, A., Radke, K., Vöcking, B.: Primal beats dual on online packing LPs in the random-order model. In: Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing, pp. 303–312 (2014)
28. Kleinberg, R.: A multiple-choice secretary algorithm with applications to online auctions. SODA **5**, 630–631 (2005)
29. Lachish, O.: O (log log rank) competitive ratio for the matroid secretary problem. In: 2014 IEEE 55th Annual Symposium on Foundations of Computer Science. IEEE, pp. 326–335 (2014)
30. Mahdian, M., Yan, Q.: Online bipartite matching with random arrivals: an approach based on strongly factor-revealing lps. In: Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing, pp. 597–606 (2011)
31. Meyerson, A.: Online facility location. In: Proceedings 42nd IEEE Symposium on Foundations of Computer Science. IEEE, pp. 426–431 (2001)
32. Molinaro, M.: Online and random-order load balancing simultaneously. In: Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms. SIAM, pp. 1638–1650 (2017)

33. Osborn, C., Torng, E.: List's worst-average-case or WAC ratio. J. Sched. **11**(3), 213–215 (2008)
34. Pruhs, K., Sgall, J., Torng, E.: Online scheduling. (2004)
35. Rudin, J., III.: Improved bounds for the on-line scheduling problem (2001)
36. Rudin, J., III., Chandrasekaran, R.: Improved bounds for the online scheduling problem. SIAM J. Comput. **32**(3), 717–735 (2003)
37. Sanders, P., Sivadasan, N., Skutella, M.: Online scheduling with bounded migration. Math. Oper. Res. **34**(2), 481–498 (2009)
38. Sgall, J.: A lower bound for randomized on-line multiprocessor scheduling. Inf. Process. Lett. **63**(1), 51–55 (1997)
39. Sleator, D., Tarjan, R.: Amortized efficiency of list update and paging rules. Commun. ACM **28**(2), 202–208 (1985)

Authors and Affiliations

Susanne Albers[1] · **Maximilian Janke**[1]

Susanne Albers
albers@in.tum.de

[1] Lehrstuhl für Algorithmen und Komplexität, Institut für Informatik, Technische Universität München, Boltzmannstr. 3, 85748 Garching, Germany

Appendix C

Scheduling in the Secretary Model

Bibliographic Information *Makespan Minimization in the Secretary Model*. S. Albers, M. Janke. To appear in: 41st Annual Conference on Foundations of Software Technology and Theoretical Computer Science (**FSTTCS**) 2021

Summary of Contributions We study Online Makespan Minimization in the secretary model. Online algorithms treat jobs in a uniformly random order and know their number n in advance. This knowledge has been implicitly used by any online algorithm developed for scheduling under random-order arrival excluding ours in Appendix B. The arriving jobs have to be scheduled onto parallel and identical machines. Preemption is not allowed. The goal is to minimize the makespan, which is the time it takes to process all jobs.

We analyze two deterministic online algorithms. First, a straightforward adaptation of the algorithm LightLoad [10] from the literature has a competitive ratio of 1.75. This ratio is not only obtained in expectation but on almost all input permutations. We also analyze LightLoad for small values of m. A more sophisticated deterministic 1.581-competitive algorithm then outcompetes even the pessimistic randomized lower bound for adversarial settings [51, 151]. This competitive ratio holds again for nearly all input permutations. It provides formal evidence that job sequences leading to worse competitive ratios are extremely rare. We complement these results by first lower bounds. No online algorithm, deterministic or randomized, can be better than 1.043-competitive in the secretary model. Moreover, it cannot achieve a competitive ratio below 1.257 with high probability.

Individual Contributions

- Initial proposal to study the secretary variant of the random-order model, where input size n is known in advance.
- Analysis and adaptation of LightLoad, as well as development and analysis of the improved algorithm and the lower bounds.
- Composition of the manuscript including graphics, graphs, technical and non-technical parts.

Scheduling in the secretary model*

Susanne Albers
Department of Computer Science, Technical University of Munich

Maximilian Janke
Department of Computer Science, Technical University of Munich

Abstract

This paper studies online makespan minimization in the secretary model. Jobs, specified by their processing times, are presented in a uniformly random order. The input size n is known in advance. An online algorithm has to non-preemptively assign each job permanently and irrevocably to one of m parallel and identical machines such that the expected time it takes to process them all, the makespan, is minimized.

We give two deterministic algorithms. First, a straightforward adaptation of the semi-online strategy LightLoad [4] provides a very simple approach retaining its competitive ratio of 1.75. A new and sophisticated algorithm is 1.535-competitive. These competitive ratios are not only obtained in expectation but, in fact, for all but a very tiny fraction of job orders.

Classically, online makespan minimization only considers the worst-case order. Here, no competitive ratio below 1.885 for deterministic algorithms and 1.581 using randomization is possible. The best randomized algorithm so far is 1.916-competitive. Our results show that classical worst-case orders are quite rare and pessimistic for many applications.

We complement our results by providing first lower bounds. A competitive ratio obtained on nearly all possible job orders must be at least 1.257. This implies a lower bound of 1.043 for both deterministic and randomized algorithms in the general model.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Scheduling, makespan minimization, online algorithm, competitive analysis, lower bound, random-order, secretary problem

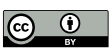
Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

We study one of the most basic scheduling problems, the classic problem of makespan minimization. For the classic makespan minimization problem, one is given an input set $\mathcal{J}$ of n jobs, which have to be scheduled onto m identical and parallel machines. Preemption is not allowed. Each job $J \in \mathcal{J}$ runs on precisely one machine. The goal is to find a schedule minimizing the *makespan*, i.e. the completion time of the last job. This problem admits a long line of research and countless practical applications in both, its offline variant see e.g. [31, 34] and references therein, as well as in the online setting studied in this paper.

In the online setting, jobs are revealed one by one and each has to be scheduled by an online algorithm A immediately and irrevocably without knowing the sizes of future jobs. The makespan of online algorithm A, denoted by $A(\mathcal{J}^\sigma)$, may depend on both the job set $\mathcal{J}$ and the job order σ. The optimum makespan $\mathrm{OPT}(\mathcal{J})$ only depends on the former. Traditionally, one measures the performance of A in terms of competitive analysis. The input set $\mathcal{J}$ as well as the job order σ are chosen by an adversary whose goal is to maximize the ratio $\frac{A(\mathcal{J}^\sigma)}{\mathrm{OPT}(\mathcal{J})}$. The maximum ratio, $c = \sup_{\mathcal{J},\sigma} \frac{A(\mathcal{J}^\sigma)}{\mathrm{OPT}(\mathcal{J})}$, is the *(adversarial) competitive ratio*. The goal is to find online algorithms obtaining small competitive ratios.

* Work supported by the European Research Council, Grant Agreement No. 691672, project APEG.



In the classical secretary problem, the goal is to hire the best secretary out of a linearly ordered set S of candidates. Its size n is known. Secretaries appear one by one in a uniformly random order. An online algorithm can only compare secretaries it has seen so far. It has to decide irrevocably for each new arrival whether this is the single one it wants to hire. Once a candidate is hired, future ones are automatically rejected even if they are better. The algorithm fails unless it picks the best secretary. Similar to makespan minimization this problem has been long studied, see [21, 24, 25, 35, 44, 46, 47] and references therein.

This paper studies makespan minimization under the input model of the secretary problem. The adversary determines a job set of known size n. Similar to the secretary problem, these jobs are presented to an online algorithm A one by one in a uniformly random order. Again, A has to schedule each job without knowledge of the future. The expected makespan is considered. The *competitive ratio in the secretary (or random-order) model* is $c = \sup_{\mathcal{J}} \mathbf{E}_\sigma \left[\frac{A(\mathcal{J}^\sigma)}{\mathrm{OPT}(\mathcal{J})}\right] = \sup_{\mathcal{J}} \frac{1}{n!} \sum_{\sigma \in S_n} \frac{A(\mathcal{J}^\sigma)}{\mathrm{OPT}(\mathcal{J})}$, the maximum ratio between the expected makespan of A and the optimum makespan. The goal is again to obtain small competitive ratios.

We propose the term *secretary model* to set this result apart from [6] where we provide a 1.8478-competitive where n, the number of jobs, is not known in advance. Not knowing n is quite restrictive and has never been considered in any other scheduling algorithm designed with random-order arrival in mind [3, 28, 51, 52]. We hope to raise attention to these two surprisingly different models. Even though for the adversarial model such information is useless; the secretary-model requires novel and significantly different approaches and leads to, as our results show, vastly better performance guarantees.

Frameworks similar to the secretary model received a lot of recent attention in the research community sparking the area of random-order analysis. Random-order analysis has been successfully applied to numerous problems such as matching [29, 36, 38, 48], various generalizations of the secretary problem [9, 24, 25, 33, 35, 44, 46], knapsack problems [10], bin packing [42], facility location [49], packing LPs [43], convex optimization [32], welfare maximization [45], budgeted allocation [50] and recently scheduling [3, 6, 28, 51, 52]. We refer to the chapter [8] for a general overview over random-order models.

For makespan minimization, the role of randomization is poorly understood. The lower bound of 1.581 from [14, 55] is considered pessimistic and exhibits quite a big gap towards the best randomized ratio of 1.916 from [2]. A main consequence of the paper is that random-order arrival allows to beat the lower bound of 1.581. This formally sets the secretary model apart from the classical adversarial setting even if randomization is involved.

Previous work: Online makespan minimization and variants of the secretary problem have been studied extensively. We only review results most relevant to this work, beginning with the traditional deterministic adversarial setting. For m identical machines, Graham [31] showed 1966 that the greedy strategy, which schedules each job onto a least loaded machine, is $\left(2 - \frac{1}{m}\right)$-competitive. This was subsequently improved in a long line of research [27, 11, 37, 1] leading to the currently best competitive ratio by Fleischer and Wahl [26], which approaches 1.9201 for $m \to \infty$. Chen et al. [15] presented a deterministic algorithm whose competitive ratio is at most $(1 + \varepsilon)$-times the optimum one, although the actual ratio remains to be determined. For general m, lower bounds are provided in [23, 12, 30, 53]. The currently best bound is due to Rudin III [53] who shows that no deterministic online algorithm can be better than 1.88-competitive.

The role of randomization in this model is not well understood. The currently best randomized ratio of 1.916 [2] barely beats deterministic guarantees. In contrast, the best lower bound approaches $\frac{e}{e-1} > 1.581$ for $m \to \infty$ [14, 55]. There has been considerable

research interest in tightening these bounds.

Recent results for makespan minimization consider variants where the online algorithm obtains extra resources. In semi-online settings, additional information on the job sequence is given in advance, such as the optimum makespan [13, 39] or the total processing time of jobs [4, 16, 41, 40]. In the former model, the optimum competitive ratio lies in the interval $[1.333, 1.5]$, see [13], while for the latter the optimum competitive ratio is known to be 1.585 cf. [4, 40]. Taking this further, the advice complexity setting allows the algorithm to receive a certain number of advice bits from an offline oracle [5, 20, 41]. Other algorithms can migrate jobs [54] or offer a buffer, which they use to reorder the job sequence [22, 41].

The secretary problem is even older than scheduling [25]. We only summarize the work most relevant to this paper. Lindley [47] and Dynkin [21] first show that the optimum strategy finds the best secretary with probability $1/e$ for $n \to \infty$. Recent research focusses on many variants, among others generalizations to several secretaries [7, 44] or even matroids [9, 24, 46]. A modern version considers adversarial orders but allows prior sampling [18, 35, 8, 33]. Related models are prophet inequalities and the game of googol [17, 19].

So far, little is known for scheduling in the secretary model. Osborn and Torng [52] prove that Graham's greedy strategy is still not better than 2-competitive for $m \to \infty$. We study makespan minimization in the restricted random-order model where n is not known in advance [6] and the dual problem, Machine Covering, in the secretary model [3]. Molinaro [51] studies a very general scheduling problem. His algorithm uses n to restart itself after half the jobs are seen and has expected makespan $(1+\varepsilon)\mathrm{OPT} + O(\log(m)/\varepsilon)$. Göbel et al. [28] study scheduling on a single machine where the goal is to minimize weighted completion times. Their competitive ratio is $O(\log(n))$ whereas they show that adversarial models allow no sublinear competitive ratios.

Our contribution: We study makespan minimization for the secretary (or random-order) model in depth. We show that basic sampling ideas allow to adapt a fairly simple algorithm from the literature [4] to be 1.75-competitive. A more sophisticated algorithm vastly improves this competitive ratio to 1.535. Both algorithms are deterministic. This ratio of 1.535 beats all lower bounds for adversarial scheduling, including the bound of 1.582 for randomized algorithms. [14, 55]

Our main results focus on large number of machines, $m \to \infty$. This is in line with most recent adversarial results [3, 2, 26] and all random-order scheduling results [6, 28, 51, 52], excluding [28] who study scheduling on one machine. While adversarial guarantees are known to improve for small numbers of machines, nobody has ever, to the best of our knowledge, explored guarantees for random-order arrival on small number of machines. We prove that our simple algorithm is $\left(1.75 + O(\frac{1}{\sqrt{m}})\right)$-competitive. Explicit bounds on the term hidden in the O-notation are provided. This result indicates that the focus of contemporary analyses on the limit case is sensible and does not hide unreasonably large terms.

All upper bounds in this paper abide to the stronger measure of *nearly competitiveness* from [6]. An algorithm is required to achieve its competitive ratio not only in expectation but on nearly all input permutations. Thus, input sequences where the competitive ratios are not obtained can be considered extremely rare and pathological. Moreover, we require worst-case guarantees even for such pathological inputs. This is relevant to practical applications, where we do not expect fully random inputs. Both algorithms hold up to this stronger measure of nearly competitiveness.

A basic approach in secretary models uses sampling statistics; a small part of the input allows to predict the rest. Sampling lets us include techniques from semi-online and advice settings with two further challenges. On the one hand, the advice is imperfect and may be,

albeit with low probability, totally wrong. On the other hand, the advice has to be learned, rather than being available right from the start. In the beginning "mistakes" cannot be avoided. This makes it impossible to adapt better semi-online algorithms than LightLoad, namely [33, 16, 41, 40] to our model. These algorithms need to know the total processing volume right from the start. The advanced algorithm in this paper out-competes the optimum competitive ratio of 1.585 these semi-online algorithms can achieve [1, 40]. We conjecture that this is not possible for order oblivious algorithms that solely use sampling. Order oblivious algorithms first observe a random sample and then treat the input sequence in an adversarial order [8, 33]. Our analysis indicates that LightLoad can be adapted as an order-oblivious algorithm. The 1.535-competitive algorithm does not maintain its competitive ratio in such a setting.

The 1.535-competitive main algorithm is based on a modern point of view, which, analogous to kernelization, reduces complex inputs to sets of critical jobs. A set of critical jobs is estimated using sampling. Critical jobs impose a lower bound on the optimum makespan. If the bound is high, an enhanced version of Graham's greedy strategy suffices; called the Least-Loaded-Strategy. Else, it is important to schedule critical jobs correctly. The Critical-Job-Strategy, based on sampling, estimates the critical jobs and schedules them ahead of time. An easy heuristic suffices due to uncertainty involved in the estimates. Uncertainty poses not only the main challenge in the design of the Critical-Job-Strategy. On a larger scale, it also makes it hard to decide, which of the two strategies to use. Sometimes the Critical-Job-Strategy is chosen wrongly. These cases comprise the crux of the analysis and require using random-order arrival in a novel way beyond sampling.

The analyses of both algorithms follow three steps, which leads to the situation depicted in Figure 1. In the first step, adversarial analyses give worst-case guarantees and take care of *simple job sets*. These simple sets lack structure to be exploited via random reordering but do not pose problems to online algorithms. We thus are reduced to non-simple inputs. Non-simple random sequences have useful properties with high probability. They are 'sampleable' and do not have too many problematic jobs clustered at the end of the sequence. A second step formalizes this, introducing *stable sequences*. Non-stable sequences are rare and negligible, we are thus reduced to stable sequences. The third step is a classical adversarial analysis that uses the properties of stable sequences to again establish worst-case guarantees.

The paper concludes with lower bounds. We show that no algorithm, deterministic or randomized, is better than nearly 1.257-competitive. This immediately implies a lower bound of 1.043 in the general secretary model.

Notation: We use the notation $[\mathcal{J}]$ or $[\mathcal{J}^\sigma]$ to highlight values that depend on the job set $\mathcal{J}$ or the ordered job sequence $\mathcal{J}^\sigma$. Such appendage is omitted when the dependency needs not be highlighted. In similar vein, we may write OPT for $\mathrm{OPT}(\mathcal{J})$.

Figure 1 The lay of the land in our analysis. The algorithm is $(c+\varepsilon)$-competitive on simple and stable sequences. Only the small unstable remainder (hashed) is problematic. Dashed lines mark orbits under the action of the permutation group S_n. Simple sequences stay simple under permutation. Non-simple orbits have at most an ε-fraction, which is unstable (hashed). Thus, the algorithm is $(c+\varepsilon)$-competitive with probability at least $1-\varepsilon$ after random permutation.

2 A strong measure of random-order competitiveness

Consider a job set $\mathcal{J} = \{J_1, \ldots, J_n\}$ of known size n. Each job is fully defined[1] by its non-negative size (or processing time) $p_1, \ldots, p_n$. Let S_n be the group of permutations of the integers from 1 to n, which we consider a probability space under the uniform distribution. We pick each permutation with probability $1/n!$. Each permutation $\sigma \in S_n$, called an *order*, gives us a *job sequence* $\mathcal{J}^\sigma = J_{\sigma(1)}, \ldots, J_{\sigma(n)}$. Recall that traditionally an online algorithm A is called *c-competitive* for some $c \geq 1$ if we have for all job sets $\mathcal{J}$ and job orders σ that $A(\mathcal{J}^\sigma) \leq c\mathrm{OPT}(\mathcal{J})$. We call this the *adversarial model*.

In the *secretary model* we consider the expected makespan of A under a uniformly random job order, i.e. $\mathbf{E}_{\sigma\sim S_n}[A(\mathcal{J}^\sigma)] = \frac{1}{n!}\sum_{\sigma\in S_n} A(\mathcal{J}^\sigma)$. We use the term *secretary model*, to distinguish this setting from the *random-order model* in [6] where the input size n is not known in advance. The algorithm A is *c-competitive in the secretary model* if we have $\mathbf{E}_{\sigma\sim S_n}[A(\mathcal{J}^\sigma)] \leq c\mathrm{OPT}(\mathcal{J})$ for all input sets $\mathcal{J}$.

The secretary model tries to lower the impact of particularly badly ordered sequences by looking at competitive ratios only in expectation. Interestingly, the scheduling problem allows for a stronger measure of random-order competitiveness for large m, called *nearly competitiveness* [6]. One requires the given competitive ratio to be obtained on nearly all sequences—not only in expectation—as well as a bound on the adversarial competitive ratio as well. We recall the definition and the main fact, that an algorithm is already c-competitive in the secretary model if it is nearly c-competitive.

▶ **Definition 1.** *A deterministic online algorithm A is called* nearly c-competitive *if the following two conditions hold.*

- *The algorithm A achieves a constant competitive ratio in the adversarial model.*
- *For every $\varepsilon > 0$, we can find $m(\varepsilon)$ such that for all machine numbers $m \geq m(\varepsilon)$ and all job sets $\mathcal{J}$ there holds $\mathbf{P}_{\sigma\sim S_n}[A(\mathcal{J}^\sigma) \geq (c+\varepsilon)OPT(\mathcal{J})] \leq \varepsilon$.*

▶ **Lemma 2.** *If a deterministic online algorithm is nearly c-competitive, then it is c-competitive in the secretary model for $m \to \infty$, i.e. for its competitive ratio c_m on m machines holds $\lim_{m\to\infty} c_m = c$.*

3 Basic properties

Let us fix an input set $\mathcal{J}$. Graham [31] establishes that his greedy strategy is 2-competitive. He considers the *average load* $L = L[\mathcal{J}] = \frac{1}{m}\sum_{i=1}^{m} p_i$, which is the same for any schedule of the jobs in $\mathcal{J}$, and the maximum size of any job $p_{\max} = \max_i p_i$. Both are lower bounds for OPT. Indeed, even the best schedule cannot have all machines loads below average, i.e. smaller than L, and the machine containing the largest job has load at least $p_{\max}$. Now, Graham observes that the smallest load in any schedule cannot exceed the average load L. Greedily using the least loaded machine causes makespan at most $L + p_{\max} \leq 2\mathrm{OPT}$. The greedy strategy is thus 2-competitive.

Graham's argument builds the foundation for subsequent work on scheduling problems. The following proposition guarantees a (small) constant adversarial ratio for almost every sensible random-order algorithm, which is necessary for obtaining nearly competitiveness.

[1] We propose for completeness that jobs of similar size are indistinguishable. A unique identification, say the index or a hash value, could in theory be used to derandomize a randomized algorithm. All of the results in this paper hold independently of whether such identification is possible.

▶ **Proposition 3.** *Assume job J is scheduled on a machine M such that at most $i-1$ machines have strictly smaller load than M. Then load of M is at most $\left(\frac{m}{m-i}+1\right) OPT$ afterwards.*

Proof. Let l be the load of M prior to receiving job J. By assumption at least $m-i$ machines have load l. Thus $L \geq \frac{m-i}{m} l$. We schedule job J of size at most OPT on machine M of load at most $l \leq \frac{m}{m-i} L \leq \frac{m}{m-i}\text{OPT}$. The resulting load is at most $\left(\frac{m}{m-i}+1\right) OPT$. ◀

The previous result cannot be improved in general. The most difficult adversarial sequences have $L \approx p_{\max} \approx \text{OPT}$. Random-order arrival faces further challenges. Certain degenerate sequences, where few jobs carry all the load, are not suited for reordering arguments. See Figure 2. This "degeneracy" is measured by $R(\mathcal{J}) = \min(\frac{L}{p_{\max}}, 1)$. Adapting the previous arguments we obtain the following result, which indicates good performance in almost all situations if $R(\mathcal{J})$ is small.

▶ **Proposition 4.** *Let M be a machine such that at most $i-1$ machines have strictly smaller load than M. If M receives a job, its load is at most $\left(\frac{m}{m-i} R(\mathcal{J})+1\right) OPT$ afterwards.*

Proof. Adapt the previous proof using that $L \leq R(\mathcal{J})\text{OPT}$. ◀

Proposition 3 and 4 form the basis of our analyses. They give conditions when to use the Least-Loaded-Strategy in the main algorithm, establish most of our worst-case guarantees and explain why we can exclude simple sequences like the one in Figure 2. In the full paper, we generalize these propositions further, which is required for the main algorithm.

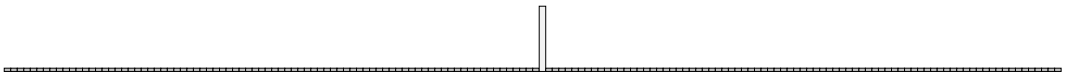


Figure 2 A surprisingly difficult sequence for random-order arguments. The big job carries most of the processing volume. Other jobs are negligible. Thus, all permutations look basically the same. Such 'simple' job sets need to be excluded before the main analysis.

3.1 Sampling for Scheduling Problems

We now explain how we use sampling in the secretary model. Consider any input permutation $\mathcal{J}^\sigma = J_{\sigma(1)} \dots J_{\sigma(n)}$. A standard technique is to sample the φ-fraction of jobs, $J_{\sigma(1)} \dots J_{\sigma(\lceil \varphi n \rceil)}$, to make predictions about $\mathcal{J}^\sigma$. The previous section gives two prime candidates for sampling which relate to OPT, namely L and $p_{\max}$. Directly 'sampling' OPT is futile.

The size $p_{\max}$ is best estimated by $p^{\varphi t}_{\max} = \max(p_{\sigma(t')} \mid \sigma(t') < \varphi n + 1)$. This corresponds to how we try to estimate the best secretary in the secretary-problem. Of course, $p^{\varphi t}_{\max}$ may vastly underestimate $p_{\max}$. If the sequence contains only a single huge job, this job is unlikely to be observed in the sample. Still, only very few jobs can have size exceeding $p^{\varphi t}_{\max}$ on random-order sequences; only $1/\varphi$ in expectation. The main algorithm uses reserve machines to catch these "exceptional" jobs.

For L we can get an unbiased[2] estimator from the sample: $L_\varphi = \frac{1}{\varphi m} \sum_{\sigma(t) \leq \varphi n} p_i$. Of course, we still need to determine how close L_φ is to L. Can we say that with high probability

[2] The estimator is unbiased, i.e. $E[L_\varphi] = L$, if φn is a natural number. For general n, we could have replaced the factor $\frac{1}{\varphi m}$ in the definition of L_φ by the more complicated expression $\frac{n}{\lceil \varphi n \rceil m}$.

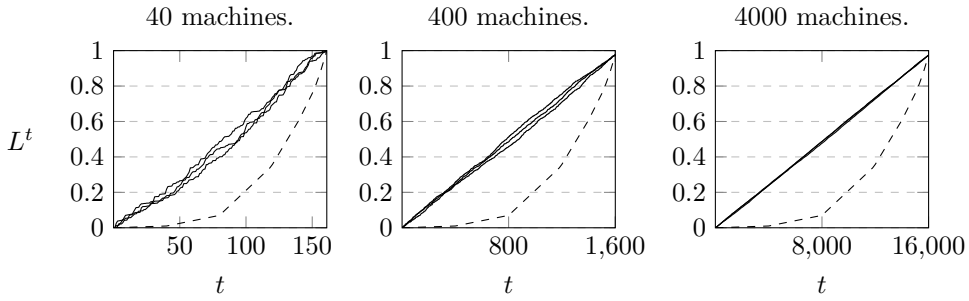


Figure 3 A graphic depicting the average load over time on the classical lower bound sequence from [1] for 40, 400 and 4000 machines. The dashed line corresponds to the original adversarial order. The three solid lines, corresponding to random permutations, clearly approximate a straight line. Thus, sampling allows to predict the (final) average load.

$L_\varphi \approx L$? For the sequence in Figure 2 such a statement cannot be true. The main observation is that these counterexamples tend to have a small value $R(\mathcal{J})$. Given a lower bound $R_{\text{low}} > 0$ on $R(\mathcal{J})$ the following Load Lemma establishes $L_\varphi \approx L$. We have seen in the previous section that sequences with $R(\mathcal{J}) < R_{\text{low}}$ pose no major obstruction. The results in the previous section guarantee arbitrarily good performance if we choose $R_{\text{low}} > 0$ small enough.

The Load Lemma is quite potent and thus fundamental to random-order makespan minimization. It may be somewhat surprising to researchers on related problems since it makes implicit use of having non-small input sizes. Note that for our problem small inputs of size less than m are trivially scheduled optimally.

▶ **Lemma 5.** *[Load Lemma [6]] Let $R_{\text{low}} = R_{\text{low}}(m) > 0$, $1 \geq \varphi = \varphi(m) > 0$ and $\varepsilon = \varepsilon(m) > 0$ be three functions in m such that $\varepsilon^{-4}\varphi^{-1}R_{\text{low}}^{-1} = o(m)$. Then there exists a variable $m(R_{\text{low}}, \varphi, \varepsilon)$, depending on these three functions, such that for $m \geq m(R_{\text{low}}, \varphi, \varepsilon)$ machines and all job sets $\mathcal{J}$ with $R(\mathcal{J}) \geq R_{\text{low}}$ and $|\mathcal{J}| \geq m$:*

$$\mathbf{P}_{\sigma \sim S_n}\left[\left|\frac{L_\varphi[\mathcal{J}^\sigma]}{L[\mathcal{J}]} - 1\right| \geq \varepsilon\right] < \varepsilon.$$

A less general version of the Load Lemma already appeared in [6]. While the Load Lemma gives only asymptotic guarantees simulations show that it requires not very large numbers of machines. Figure 4 shows the expected value of $\left|\frac{L_{1/4}[\mathcal{J}^\sigma]}{L[\mathcal{J}]} - 1\right|$ on a suitable benchmark sequence.

For our more sophisticated algorithm we also use sampling to estimate the size of critical jobs. Consider a job class $\mathcal{C}$ of size $n_{\mathcal{C}} \in O(m)$. A consequence of Chebyshev's inequality, detailed in the full version, shows that we can estimate $n_{\mathcal{C}}$ up to an additive summand of $m^{3/4}$ after sampling a $\frac{1}{\log(m)}$-fraction of the sequence. In fact the load lemma is proven by sampling job classes obtained through geometric rounding.

4 A simple 1.75-competitive algorithm

We modify the semi-online algorithm LightLoad from the literature to obtain a very simple nearly 1.75-competitive algorithm. For any $0 \leq t \leq n$, let M_{mid}^t be a machine having the $\lfloor m/2 \rfloor$-lowest load at time t, i.e. right before job J_{t+1} is scheduled. Let l_{mid}^t be its load and let l_{low}^t be the smallest load of any machine.

Let $\delta = \delta(m)$ be a certain *margin of error* our algorithm allows. It is optimal to set $\delta = 0$ but then the analysis requires a generalization of the result in [4]. In order for our main

result to be self-contained one may set $\delta = \frac{1}{\log(m)}$, which allows to use results from [4] as a black box.

Given an input sequence $\mathcal{J}^\sigma$ we know from Section 3 that $\hat{L}_{\text{pre}} = \hat{L}_{\text{pre}}[\mathcal{J}^\sigma] = \frac{L_{1/4}[\mathcal{J}^\sigma]}{1-\delta}$ provides a good estimate of the (final) average load $L = \frac{1}{m}\sum_{i=1}^{m} p_i$. We use the index 'pre' since our main algorithm later will use a slightly different guess $\hat{L}$. Consider the following adaptation LightLoadROM of the algorithm LightLoad from Albers and Hellwig [4].

Algorithm 1 The algorithm LightLoadROM

Let J_t be the job to be scheduled and let p_t be its size.
if $t < n/4$ **or** $l_{\text{low}}^{t-1} \leq 0.25\hat{L}_{\text{pre}}$ **or** $l_{\text{mid}}^{t-1} + p_t > 1.75\hat{L}_{\text{pre}}$ **then**
    Schedule J_t on any least loaded machine;
else schedule J_t on M_{mid}^{t-1};

▶ Remark 6. The first condition in the **if**-statement, $t < n/4$, already implies $l_{\text{low}}^{t-1} \leq 0.25\hat{L}_{\text{pre}}$ and is thus technically superfluous. We added it to clarify that LightLoadROM can be implemented as an online algorithm and only needs to know $\hat{L}_{\text{pre}}$ once $t \geq n/4$.

If we replace $\hat{L}_{\text{pre}}$ in the previous pseudocode by the average load L, we recover the semi-online algorithm LightLoad for makespan minimization, which has been analyzed by Albers and Hellwig [4]. They show that the algorithm is 1.75-competitive for $L = \hat{L}_{\text{pre}}$. We can show that the algorithm can also be used for general values $\hat{L} \approx L$. The performance gracefully decreases with $|L - \hat{L}_{\text{pre}}|$.

▶ **Theorem 7.** *Let $\mathcal{J}^\sigma$ be any (ordered) input sequence. The makespan of* LightLoadROM *on $\mathcal{J}^\sigma$ is at most* $1.75\left(1 + \frac{|\hat{L}_{\text{pre}}[\mathcal{J}^\sigma] - L|}{L}\right)\text{OPT}$.

Proof Sketch. For $\hat{L}_{\text{pre}} = L$, this is the main result in [4].

ItFor $\hat{L}_{\text{pre}} \geq L$, we can reduce ourselves to the case $\hat{L}_{\text{pre}} = L$. Consider any machine M in the optimum schedule of $\mathcal{J}$ that has load $l_M < \max(\hat{L}_{\text{pre}}, \text{OPT})$. We assign an additional job J_M of size $p_M = \max(\hat{L}_{\text{pre}}, \text{OPT}(\mathcal{J})) - l_M$ to this machine. For the resulting job set $\mathcal{J}'$ clearly $\text{OPT}(\mathcal{J}') = L(\mathcal{J}') = \max(\hat{L}_{\text{pre}}, \text{OPT})$. We can apply the main result of [4] to see that LightLoad has makespan at most $1.75\max(\hat{L}_{\text{pre}}, \text{OPT}(\mathcal{J}))$ if it first schedules the jobs $\mathcal{J}^\sigma$ (in order σ) followed by the additional jobs. But on the prefix $\mathcal{J}^\sigma$ LightLoad behaves precisely like LightLoadROM on input $\mathcal{J}^\sigma$. Thus, LightLoadROM has makespan at most $1.75\ \max(\hat{L}_{\text{pre}}[\mathcal{J}^\sigma], \text{OPT}(\mathcal{J}))$. Then, the theorem follows for $\hat{L}_{\text{pre}} \geq L$ since $\hat{L}_{\text{pre}} \leq \big(1 + \frac{\hat{L}_{\text{pre}} - L}{L}\big)L \leq \big(1 + \frac{|\hat{L}_{\text{pre}}[\mathcal{J}^\sigma] - L|}{L}\big)\text{OPT}$.

If $\hat{L}_{\text{pre}} \leq L$, the statement of the theorem still holds. can be derived similar to the analysis in [4]. Unfortunately, it cannot be immediately deduced from their results. Instead, their proofs need to be adapted. We sketch the necessary adaptations in the full version. ◀

The previous theorem already establishes a constant adversarial competitive ratio of 7. Use that $0 \leq \hat{L}_{\text{pre}} \leq L_{1/4} \leq 4L$ implies $|\hat{L}_{\text{pre}}[\mathcal{J}^\sigma] - L| \leq 3L$. We can improve this result, most importantly, if $R(\mathcal{J})$ is small.

▶ **Lemma 8.** *For any (ordered) job sequence $\mathcal{J}^\sigma$ the makespan of* LightLoadROM *is at most* $(1 + 2R(\mathcal{J}))\text{OPT}(\mathcal{J})$. *In particular, it is at most* $3\,\text{OPT}(\mathcal{J})$ *in general and at most* $1.75\,\text{OPT}(\mathcal{J})$ *for* $R(\mathcal{J}) < 3/8$.

Proof. Since LightLoadROM only considers the least or the $\lfloor m/2 \rfloor$-th least loaded machine, the lemma follows from Proposition 4. ◀

We now establish the competitive ratio of LightLoadROM in the strong model of nearly competitiveness. Corollary 10 follows immediately by Lemma 2.

▶ **Theorem 9.** *The algorithm* LightLoadROM *is nearly* 1.75*-competitive.*

▶ **Corollary 10.** LightLoadROM *is* 1.75*-competitive in the secretary model for* $m \to \infty$.

Proof of Theorem 9. Our analysis forms a triad outlining how we analyze the more sophisticated 1.535-competitive main algorithm. See Figure 1 for an illustration. Since we only prove the case $\hat{L} \geq L$ of Theorem 7, we will not rely on the case $L \leq \hat{L}$ in this proof. For this, we need to set $\delta(m) = \frac{1}{\log(m)}$.

Analysis basics. By Lemma 8 algorithm LightLoadROM is 3-competitive in the adversarial model. The first condition of nearly competitiveness is satisfied. We call input set $\mathcal{J}$ *simple* if $|\mathcal{J}| \leq m$ or $R[\mathcal{J}] < \frac{3}{8}$. Observe that LightLoadROM is (adversarially) 1.75-competitive on simple job sets. Indeed, if $|\mathcal{J}| < m$ LightLoadROM assigns every job to an empty least-loaded machine, which is obviously optimal. If $R[\mathcal{J}] < \frac{3}{8}$, Lemma 8 bounds the competitive ratio by $1 + 2R[\mathcal{J}] < 1.75$. We thus are left to consider non-simple, so called *proper*, job sets.

Stable job sequences. We call a sequence $\mathcal{J}^\sigma$ *stable* if $L \leq \hat{L}_{\text{pre}} \leq \frac{1+\delta(m)}{1-\delta(m)} L$. If a sequence is proper, it fulfills the conditions of the Load Lemma with $\varphi = 1/4$, $R_{\text{low}} = \frac{3}{8}$ and $\varepsilon(m) = \delta(m) = 1/\log(m) \in \omega(m^{-1/4})$. The Load Lemma guarantees that for m large enough, $\mathbf{P}_{\sigma \sim S_n}\left[\left|\frac{L_\varphi[\mathcal{J}^\sigma]}{L[\mathcal{J}]} - 1\right| \geq \delta\right] < \delta$. Note that $\left|\frac{L_\varphi[\mathcal{J}^\sigma]}{L[\mathcal{J}]} - 1\right| < \delta$ is equivalent to $(1-\delta)L < L_\varphi[\mathcal{J}^\sigma] < (1+\delta)L$, which in turn implies that $L \leq \hat{L}_{\text{pre}} \leq \frac{1+\delta(m)}{1-\delta(m)} L$. Thus, the probability of the sequence $\mathcal{J}^\sigma$ being stable is at least $1 - \delta$ for m large enough and $\mathcal{J}$ proper.

Adversarial Analysis. By Theorem 7, the makespan of LightLoadROM on stable sequences with $L \leq \hat{L}_{\text{pre}} \leq \frac{1+\delta(m)}{1-\delta(m)} L$ is at most $1.75 \cdot \frac{1+\delta(m)}{1-\delta(m)} \text{OPT} = \left(1.75 + \frac{3.5 \cdot \delta(m)}{1-\delta(m)}\right) \text{OPT}(\mathcal{J})$. We only require the easy case, $L \leq \hat{L}_{\text{pre}}$ of Theorem 7, which is fully proven in this paper.

Conclusion. Let $\varepsilon > 0$. Since $\delta(m) \to 0$, we can choose m large enough such that $\frac{3.5\delta(m)}{1-\delta(m)} \leq \varepsilon$. In particular $\mathbf{P}_{\sigma \sim S_n}[\text{LightLoadROM}(\mathcal{J}^\sigma) \geq (1.75+\varepsilon) OPT(\mathcal{J})] \leq \delta(m) \leq \varepsilon$ since the only sequences where the inequality does not hold are proper but not stable. This concludes the second condition of nearly competitivity.

The δ-term. Setting $\delta = 0$ can increase the $\frac{|\hat{L}_{\text{pre}}[\mathcal{J}^\sigma] - L|}{L}$-term in Theorem 7 by at most $1/\log(m)$, which vanishes for $m \to \infty$. Of course, in reality LightLoadROM improves for $\delta = 0$. ◀

Analyzing the algorithm LightLoadROM on small numbers of machines.

From now on, we consider LightLoadROM with $\delta = 0$. Thus, the average load L is estimated by $\hat{L}_{\text{pre}} = L_{1/4}$. The *normalized absolute mean deviation* of $\hat{L}_{\text{pre}} = L_{1/4}$ is defined as $\text{NMD}(\hat{L}_{\text{pre}}) = \mathbf{E}_{\sigma \sim S_n}\left[\frac{|\hat{L}_{\text{pre}}[\mathcal{J}^\sigma] - L|}{L}\right]$. The following is a consequence of Theorem 7 .

▶ **Theorem 11.** *On input set* $\mathcal{J}$ *the competitive ratio of* LightLoadROM *in the secretary model is at most* $1.75(1 + \text{NMD}(\hat{L}_{\text{pre}}))$.

In the full version we give an estimation on $\text{NMD}(\hat{L}_{\text{pre}})$, which leads to the following result.

▶ **Theorem 12.** *The competitive ratio of* LightLoadROM *is* $1.75 + \frac{18}{\sqrt{m}} + O\left(\frac{1}{m}\right)$.

The techniques presented in this section can, in theory, be extended to analyze the main algorithm in the next section. This is impractical due to the complexity of the analysis at hand. We are certain that the error term involved will be of the form $m^{-1/a}$ for a small.

The constant summand $\frac{18}{\sqrt{m}}$ in Theorem 12 is pessimistic. We discuss several avenues of further improvement in the full version. The best we are aware of allows for a competitive ratio as small as $\frac{4.4}{\sqrt{m}} + \frac{7}{m} + O\left(\frac{1}{m^{3/2}}\right)$ but there are ways to improve even further. The terms still hidden in the O-notation result from the Stirling-approximation and are known to be tiny. Figure 4 shows $\mathrm{NMD}(\hat{L}_{\mathrm{pre}})$ on the lower bound from [1], which is a sensible benchmark.

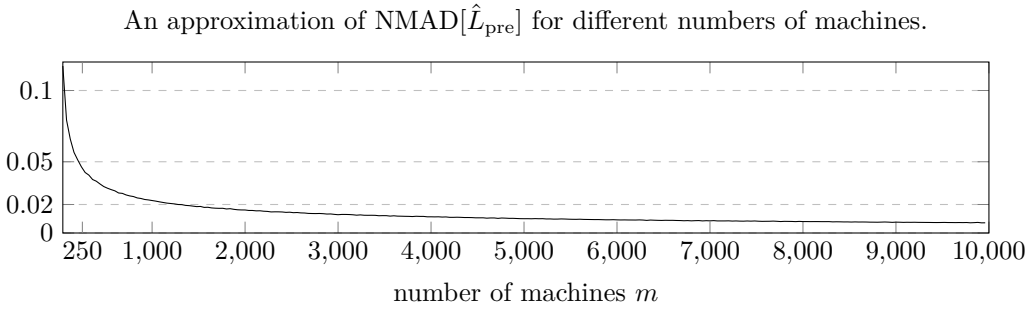


Figure 4 The extra cost for small numbers of machines. The graph shows an estimation of $\mathrm{NMD}(\hat{L}_{\mathrm{pre}})$ on the lower bound sequence from [1] based on 10,000 random samples. Theorem 11 this indicates good performance of LightLoadROM in practice.

5 The nearly 1.535-competitive algorithm

The new main algorithm achieves a competitive ratio of $c = \frac{1+\sqrt{13}}{3} \approx 1.535$. It consists of three components: a sampling phase, the Least-Loaded-Strategy and the Critical-Job-Strategy. We now give a simplified description of the algorithm.

The sampling phase. A few jobs are sampled to predict the whole sequence. These Jobs are scheduled greedily with a some machines kept in reserve. This phase is uninformed and "mistakes" are unavoidable. Such mistakes are few, since the processing volume scheduled is small—at least if we exclude worst-case sequences. First, we sample B, which tries to estimate $\max(p_{\max}, L) \leq \mathrm{OPT}$. We then use sampling to predict *critical* jobs of size in between $(c-1)B$ and B. Intuitively, jobs smaller than $(c-1)B$ are too small to pose a problem. Jobs larger than B are also critical but cannot be predicted since they did not appear during sampling. This in turn means that they are rare. We keep a few reserve machines to safely process them.

The Critical-Job-Strategy. Our plan is to assign critical jobs ahead of time. Formally, placeholder jobs are used to reserve space for jobs yet to come. Critical jobs are assigned according to an easy heuristic: Each machine gets either one big or two medium jobs. Reserve machines handle errors in the predictions and unexpected huge jobs.

The Least-Loaded-Strategy. Sometimes the Critical-Job-Strategy is not feasible; there simply are too many critical jobs. This may already by apparent from sampling predictions, but for some job sets this cannot be predicted. The latter input sets form the crux of the analysis. Once we find out, we pick the Least-Loaded-Strategy, which enhances a Graham's greedy approach by still maintaining reserve machines for particularly large jobs. Intuitively,

many critical jobs make it even for OPT impossible to schedule all jobs efficiently, which is why we rely on this less sophisticated strategy.

Further challenges. Algorithm design and analysis have to deal with three further issues. First, the Critical-Job-Strategy needs to take scheduling decisions made during sampling into account. Second, a consequence of sampling is that no value is exact, small sources of errors are imminent. Third, we need a constant competitive ratio against an adversary. All these challenges impact details of the algorithm design in rather subtle ways.

5.1 Formal Description

Let $\delta = \delta(m) = \frac{1}{\log(m)}$ be the *margin of error our algorithm allows*. Most of the time, it is sensible to treat δ as a constant and forget about its dependency on m. Our algorithm maintains a set of $\lceil \delta m \rceil$ *reserve machines*. Their complement are the *principal machines*. Let us fix an input sequence $\mathcal{J}^\sigma$. Let $\hat{L} = \hat{L}[\mathcal{J}^\sigma] = L_{\delta^2}[\mathcal{J}^\sigma]$. For simplicity, we hide the dependency on $\mathcal{J}^\sigma$ whenever possible. Our online algorithm uses $B = \max\left(p_{\max}^{\delta^2 n}, \hat{L}\right)$ as an *estimated lower bound for* OPT. This bound is known after the first $\lfloor \delta^2 n \rfloor$ jobs are treated. Our algorithm uses geometric rounding implicitly. Given a job J_t of size p_t let $f(p_t) = (1+\delta)^{\lfloor \log_{1+\delta} p_t \rfloor}$ be its *rounded size*. We also call J_t an $f(p_t)$-job.

Using rounded sizes, we introduce job classes. Let $p_{\text{small}} = c - 1 = \frac{\sqrt{13}-2}{3} \approx 0.535$ and $p_{\text{big}} = \frac{c}{2} = \frac{1+\sqrt{13}}{6} \approx 0.768$. We call job J_t

- *small* if $f(p_t) \leq p_{\text{small}} B$ and *critical* else,
- *big* if $f(p_t) > p_{\text{big}} B$,
- *medium* if J is neither small nor big, i.e. $p_{\text{small}} B \leq f(p_t) \leq p_{\text{big}} B$,
- *huge* if its (not-rounded) size exceeds B, i.e. $B < p_t$, and *normal* else.

Consider the set $\mathcal{P} = \{(1+\delta)^i \mid p_{\text{small}} B \leq (1+\delta)^i \leq B\}$ corresponding to rounded sizes of critical jobs. Given $p \in \mathcal{P}$ let n_p be the total number of p-jobs. We could estimate n_p by $\delta^{-2}\hat{n}_p$ after sampling where $\hat{n}_p = |\{J_{\sigma(j)} \mid \sigma(j) \leq \delta^2 n \wedge J_{\sigma(j)} \text{ is a } p\text{-job}\}|$ after sampling. In practice we need a more complicated guess: $c_p = \max\left(\left\lfloor \left(\delta^{-2}\hat{n}_p - m^{3/4}\right) w(p) \right\rfloor, \hat{n}_p\right) w(p)^{-1}$. It has two advantages. The value c_p is close to n_p with high probability, but unlikely to exceed it. Overestimating n_p turns out to be far worse than underestimating it. It also simplifies the description of the algorithm allowing medium jobs to always "pair up".

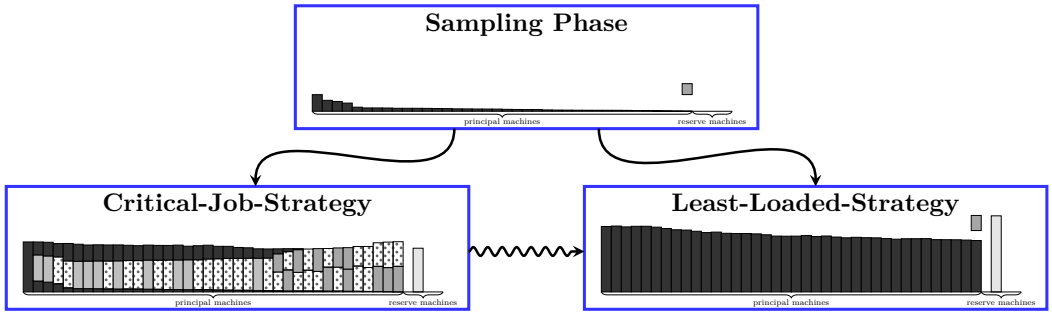


Figure 5 The 1.535-competitive algorithm. First, few jobs are sampled. Then, the algorithm decides between two strategies. The Critical-Job-Strategy tries to schedule critical jobs ahead of time. The Least-Loaded-Strategy follows a greedy approach, which reserves some machines for large jobs. Sometimes, we realize very late that the Critical-Job-Strategy does not work and have to switch to the Least-Loaded-Strategy "on the fly". We never switch in the other direction.

Statement of the algorithm: If there are less jobs than machines, each job is placed onto a separate machine. This is optimal. Else, a short sampling phase greedily assigns each of the first $\lfloor \delta^2 n \rfloor$ jobs to the least loaded principal machine. Now, B and $(c_p)_{p\in\mathcal{P}}$ are known. We choose the Least-Loaded-Strategy if we predict the Critical-Job-Strategy to be infeasible. Formally, if $\sum_{p\in\mathcal{P}} w(p)c_p > m$, where $w(p) = 1/2$ for $p \leq p_{\text{big}}$ and $1 > p \leq p_{\text{big}}$. If $\sum_{p\in\mathcal{P}} w(p)c_p \leq m$, we choose the Critical-Job-Strategy. The Critical-Job-Strategy requires a one-time preparation. It may later switch to the Least-Loaded-Strategy but it never switches the other way around.

The Least-Loaded-Strategy

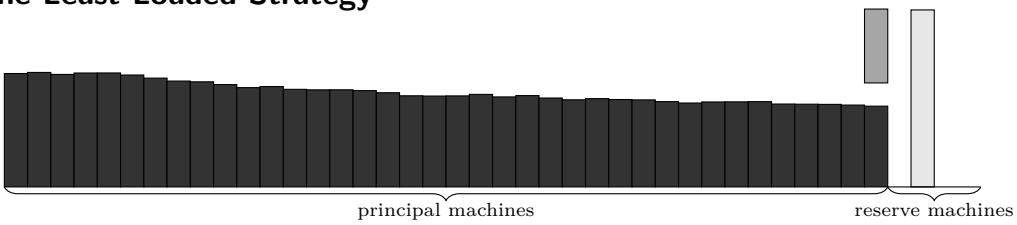


Figure 6 The Least-Loaded-Strategy schedules jobs greedily. A few machines are reserved for unexpected huge jobs. For example the largest job, which is unlikely to arrive in the sampling phase.

The Least-Loaded-Strategy places any normal job on a least loaded principal machine. Huge jobs are scheduled on a least loaded reserve machine. This reserve machine will be empty, unless we consider rare and pathological worst-case orders.

The Critical-Job-Strategy

For the Critical-Job-Strategy we introduce *p-placeholder-jobs* for every size $p \in \mathcal{P}$. Sensibly, the size of a p-placeholder-job is p. During the Critical-Job-Strategy we treat placeholder-jobs similar to *real* jobs. They are assigned in the **Preparation for the Critical-Job-Strategy**. We try to pair off medium jobs, some of which already arrived during sampling. Moreover it is important to assign fewer processing volume to those machines, which have a higher load after the sampling phase.

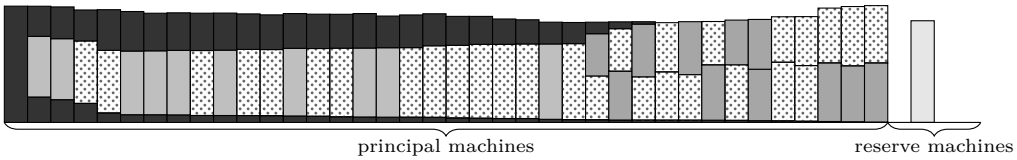


Figure 7 The Critical-Job-Strategy. Each machine gets either two medium, one large or no critical job. Placeholder jobs (dotted) reserve space for critical jobs yet to come. Processing volume of small jobs (dark) on the bottom arrived during the sampling phase. Reserve machines accommodate huge jobs or, possibly, jobs without matching placeholders.

The **Critical-Job-Strategy** places small jobs on least-loaded principal machines taking placeholders into account. Critical jobs of rounded size $p \in \mathcal{P}$ replace p-placeholder-jobs whenever possible. If no matching placeholder is found or if the current job is exceptional, the reserve machines are used. Again, medium jobs are paired up. If the reserve machines are full, the algorithm *fails*. It switches to the Least-Loaded-Strategy.

The full description of the main algorithm is provided in Appendix B.

6 Analysis of the algorithm

Theorem 13 is main result of the paper.

▶ **Theorem 13.** *Our algorithm is nearly c-competitive. Recall that* $c = \frac{1+\sqrt{13}}{3} \approx 1.535$.

Due to Lemma 2 this competitive ratio also holds in the general secretary model.

▶ **Corollary 14.** *Our algorithm is c-competitive in the secretary model as* $m \to \infty$.

The analysis of the algorithm proceeds along the same three reduction steps used in the proof of Theorem 9. First, we assert that our algorithm has a constant adversarial competitive ratio, which approaches 1 as $R(\mathcal{J}) \to 0$. Not only does this lead to the first condition of nearly competitiveness, it also enables us to introduce *simple* job sets on which we perform well due to basic considerations resulting from Section 3.

▶ **Definition 15.** *A job set* $\mathcal{J}$ *is called* simple *if* $R(\mathcal{J}) \leq \frac{(1-\delta)\delta^2}{\delta^2+1}(c-1)$ *or if it consists of at most m jobs. Else, we call it* proper. *We call any ordered input sequence* $\mathcal{J}^\sigma$ simple *respectively* proper *if the underlying set* $\mathcal{J}$ *has this property.*

▶ **Main Lemma 1.** In the adversarial model our algorithm has competitive ratio $4 + O(\delta)$ on general input sequences and $c + O(\delta)$ on simple sequences.

We are thus reduced to treating *proper* job sets. In the second reduction we introduce *stable* sequences. These have many desirable properties. Most notably, they are suited to sampling. Their formal definition can be found later in Definition 19. The second reduction shows that stable sequences arise with high probability if the order of a proper job set $\mathcal{J}$ is picked uniformly randomly.

Formally, for m the number of machines, let $P(m)$ be the maximum probability by which the permutation of any proper sequence may not be stable, i.e.

$$P(m) = \sup_{\mathcal{J} \text{ proper}} \mathbf{P}_{\sigma \sim S_n} \left[\mathcal{J}^\sigma \text{ is not stable} \right].$$

The second main lemma asserts that this probability vanishes as $m \to \infty$.

▶ **Main Lemma 2.** $\lim_{m \to \infty} P(m) = 0$.

In other words, non-stable sequences are very rare and of negligible impact in random-order analyses. Thus, we only need to consider stable sequences. In the final, third, step we analyze our algorithm on stable sequences. This analysis is quite general. In particular, it does not rely further on random-order arrival. Instead, we work with worst-case stable inputs, i.e. we allow an adversary to present any stable input sequence.

▶ **Main Lemma 3.** Our algorithm is adversarially $(c + O(\delta))$-competitive on stable sequences.

These three main lemmas allow us to conclude the proof of Theorem 13.

Proof of Theorem 13. Recall that $\delta(m) \to 0$ for $m \to \infty$. By Main Lemma 1, the first condition of nearly competitiveness holds, i.e. our algorithm has a constant competitive ratio. Moreover, by Main Lemma 1 and Main Lemma 3, given $\varepsilon > 0$, we can pick $m_0(\varepsilon)$ such that our algorithm is $(c + \varepsilon)$-competitive on all sequences that are stable or simple, if there are at least $m_0(\varepsilon)$ machines. This implies that for $m \geq m_0(\varepsilon)$ the probability of our algorithm not being $(c + \varepsilon)$-competitive is at most $P(m)$, the maximum probability with which a random permutation of a proper input sequence is not stable. By Main Lemma 2, we can find $m(\varepsilon) \geq m_0(\varepsilon)$ such that $P(m) \leq \varepsilon$ for $m \geq m(\varepsilon)$. This choice of $m(\varepsilon)$ satisfies the second condition of nearly competitiveness. ◀

Proof sketch of Main Lemma 1

The *anticipated load* $\tilde{l}_M^t$ of a machine M at time t denotes its load including placeholder-jobs. We can obtain the following two bounds on the average anticipated load $\tilde{L} = \sup_t \frac{1}{m} \sum_M \tilde{l}_M^t$.

▶ **Lemma 16.** *We have* $\tilde{L} \leq L + 2p_{\max}$*, as well as* $\tilde{L} \leq \left(1 + \frac{1}{\delta^2}\right) L$*.*

Thus the average anticipated load $\tilde{L}$ relates to the original values L, $p_{\max}$. In the full version, we generalize Proposition 4 to anticipated loads. We can then use Lemma 16 to conclude Main Lemma 1. The only exception are reserve machines, which receives its last job using the Critical-Job-Strategy. Their load needs to be bounded using different techniques.

Formally, we can prove the following two statements, which imply Main Lemma 1.

▶ **Proposition 17.** *The main algorithm is adversarially* $\left(1 + \frac{3}{1-\delta} + 2\delta\right)$*-competitive.*

▶ **Proposition 18.** *The main algorithm has makespan at most* $(c + 2\delta)\mathrm{OPT}$ *on simple sequences* $\mathcal{J}^\sigma$*.*

Stable job sequences and a proof sketch of Main Lemma 2

We introduce the class of *stable* job sequences. The first two conditions state that all estimates our algorithm makes are *accurate*, i.e. sampling works. By the third condition there are less exceptional jobs than reserve machines and the fourth condition states that these jobs are distributed evenly. The final condition is a technicality. Stable sequences are useful since they occur with high probability if we randomly order a proper job set.

▶ **Definition 19.** *A job sequence* $\mathcal{J}^\sigma$ *is* stable *if the following conditions hold:*

- *The estimate* $\hat{L}$ *for* L *is accurate, i.e.* $(1-\delta)L \leq \hat{L} \leq (1+\delta)L$*.*
- *The estimate* c_p *for* n_p *is accurate, i.e.* $c_p \leq n_p \leq c_p + 2m^{3/4}$ *for all* $p \in \mathcal{P}$*.*
- *There are at most* $\lceil \delta m \rceil$ *exceptional jobs in* $\mathcal{J}^\sigma$*.*
- *Let* $\tilde{t}$ *be the time the last exceptional job arrived and let* $n_{p,\tilde{t}}$ *be the number of* p*-jobs scheduled at that time for a given* $p \in \mathcal{P}$*. Then* $n_{p,\tilde{t}} \leq \left(1 - \delta^3\right) n_p$ *for every* $p \in \mathcal{P}$*.*
- $\delta^3 \left\lfloor \left(1 - \delta - 2\delta^2\right) m / |\mathcal{P}| \right\rfloor \geq 2|\mathcal{P}| m^{3/4}$*.*

Proof sketch of Main Lemma 2. The first condition follows from Lemma 5. The second condition can be derived using Chebyshev's inequality as discussed at the end of Section 3.1. Both conditions require that only proper sequences are considered. The third condition is equivalent to demanding one of the $\lceil \delta m \rceil$ largest jobs to occur during the sampling phase. This is extremely likely. In expectation the rank of the largest job occurring in the sampling phase is δ^{-2}, a constant. The fourth condition states that the exceptional jobs are evenly spread throughout the sequence compared to the p-jobs for any $p \in \mathcal{P}$. Again, this is expected of a random sequence and corresponds to how one would determine randomness statistically. For the final condition it suffices to choose the number of machines m large enough. One technical problem arises since the set of rounded critical job sizes $\mathcal{P} = \mathcal{P}[\mathcal{J}^\sigma]$ is defined using the value $B[\mathcal{J}^\sigma]$. It thus highly depends on the input permutation σ. We rectify this by passing over to a larger class $\hat{\mathcal{P}}$ such that $\mathcal{P} \subset \hat{\mathcal{P}}$ with high probability. ◀

Proof sketch of Main Lemma 3

We first consider the Critical-Job-Strategy. Main Lemma 3 holds as long as it is employed.

▶ **Lemma 20.** *The makespan of our algorithm is at most* $(c + O(\delta)) \max(B, L, p_{\max})$ *on stable sequences till it employs the Least-Loaded-Strategy (or till the end of the sequence).*

Proof sketch. Let us only consider critical jobs at any time the Least-Loaded-Strategy is not employed. We can then show that a machine contains either one big job or at most two medium jobs. In the first case we bound the size of this big, possibly exceptional, job by $p_{\max}$. Else, if the machine contains two medium jobs their total weight is at most $2(1+\delta)p_{\text{big}}B = (1+\delta)cB$. The factor $(1+\delta)$ arises since we use rounded sizes in the definition of medium jobs. Thus, critical jobs may cause a load of at most $\max(p_{\max}, (c+O(\delta))B)$.

Analyzing the load increase caused by small jobs requires techniques similar to the proof of Main Lemma 1. ◀

Note that for stable sequences $\hat{L} \leq (1+\delta)L \leq (1+\delta)\text{OPT}$, in particular $\max(B, L, p_{\max}) = \max\left(p_{\max}^{\delta^2 n}, \hat{L}, L, p_{\max}\right) \leq (1+\delta)\text{OPT}$. This proves the following corollary to Lemma 20.

▶ **Corollary 21.** *Till our algorithm uses the Least-Loaded-Strategy its makespan is less than* $(c+O(\delta))\text{OPT}$ *on stable sequences.*

Hence, we are left to consider the Least-Loaded-Strategy. We say the algorithm *fails* if it has to switch from the Critical-Job-Strategy to the Least-Loaded-Strategy. The following lemma is crucial and relies deeply on the properties of stable sequences, in particular the fourth one.

▶ **Lemma 22.** *If the algorithm fails, every exceptional job has been scheduled.*

The lemma shows that the Least-Loaded-Strategy only needs to deal with exceptional jobs if it is picked immediately. In this case, all reserve machines are empty. The third property of stable sequences ensures that there are enough reserve machines so that every exceptional job is assigned to an empty machine.

Non-exceptional jobs, i.e. jobs of size at most B, are scheduled onto a least loaded principal machine. This machine was among the $\delta m + 1$ least loaded machines and had load at most $mL/(m-\delta m+1)$ by Proposition 4. Afterwards, its load was at most $mL/(m-\delta m+1)+B \leq (2+O(\delta))B$ since $(1-\delta)L \leq B$ for stable sequences. The following lemma concludes the proof of Main Lemma 3 since it implies that $(2+O(\delta))B \leq (c+O(\delta))\text{OPT}$.

▶ **Lemma 23.** *If the Least-Loaded-Strategy is applied on a stable sequence,* $B \leq \frac{c}{2}\text{OPT}$.

The proof of Lemma 23 is left to the full version.

7 Lower bounds

We establish the following theorem using two lower bound sequences. These results generalize to randomized algorithms using appropriate notions of (nearly) competitiveness.

▶ **Theorem 24.** *For every online algorithm A, deterministic or randomized, there exists a job set $\mathcal{J}$ such that* $\mathbf{P}_{\sigma\sim S_n}\left[A(\mathcal{J}^\sigma) \geq \frac{\sqrt{73}-1}{6}\text{OPT}(\mathcal{J})\right] \geq \frac{1}{6}$. *If A is randomized the previous probability also includes its random choices.*

The lower bound highlights the inability of the main algorithm to decide between the Least-Loaded-Strategy and the Critical-Job-Strategy. If we could communicate this decision, say through a single advice bit, our main algorithm would become nearly optimal, i.e. nearly 1-competitive, on the lower bound sets. Theorem 24 implies the following lower bounds.

▶ **Corollary 25.** *If an online algorithm A is nearly c-competitive, then* $c \geq \frac{\sqrt{73}-1}{6} \approx 1.257$.

▶ **Corollary 26.** *The best competitive ratio possible in the secretary model is* $\frac{\sqrt{73}+29}{36} \approx 1.043$.

The lower bounds are proven in the appendix.

References

1. S. Albers. Better bounds for online scheduling. *SIAM Journal on Computing*, 29(2):459–473, 1999. Publisher: SIAM.
2. S. Albers. On randomized online scheduling. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 134–143, 2002.
3. S. Albers, W. Gálvez, and M. Janke. Machine covering in the random-order model. In *32nd International Symposium on Algorithms and Computation (ISAAC 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.
4. S. Albers and M. Hellwig. Semi-online scheduling revisited. *Theoretical Computer Science*, 443:1–9, 2012. Publisher: Elsevier.
5. S. Albers and M. Hellwig. Online makespan minimization with parallel schedules. *Algorithmica*, 78(2):492–520, 2017. Publisher: Springer.
6. S. Albers and M. Janke. Scheduling in the Random-Order Model. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*. unpublished, 2020.
7. S. Albers and L. Ladewig. New results for the k-secretary problem. *arXiv preprint arXiv:2012.00488*, 2020.
8. Pablo Azar, Robert Kleinberg, and S. Weinberg. Prophet inequalities with limited information. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, 07 2013. doi:10.1137/1.9781611973402.100.
9. M. Babaioff, N. Immorlica, D. Kempe, and R. Kleinberg. Matroid Secretary Problems. *Journal of the ACM (JACM)*, 65(6):1–26, 2018. Publisher: ACM New York, NY, USA.
10. M. Babaioff, N. Immorlica, D. Kempe, and Robert Kleinberg. A knapsack secretary problem with applications. In *Approximation, randomization, and combinatorial optimization. Algorithms and techniques*, pages 16–28. Springer, 2007.
11. Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 51–58, 1992.
12. Y. Bartal, H. J. Karloff, and Y. Rabani. A better lower bound for on-line scheduling. *Inf. Process. Lett.*, 50(3):113–116, 1994.
13. Martin Böhm, Jiří Sgall, Rob Van Stee, and Pavel Veselỳ. A two-phase algorithm for bin stretching with stretching factor 1.5. *Journal of Combinatorial Optimization*, 34(3):810–828, 2017.
14. B. Chen, A. van Vliet, and G. J. Woeginger. A lower bound for randomized on-line scheduling algorithms. *Information Processing Letters*, 51(5):219–222, 1994. Publisher: Elsevier.
15. L. Chen, D. Ye, and G. Zhang. Approximating the optimal algorithm for online scheduling problems via dynamic programming. *Asia-Pacific Journal of Operational Research*, 32(01):1540011, 2015. Publisher: World Scientific.
16. T.C.E. Cheng, H. Kellerer, and V. Kotov. Semi-on-line multiprocessor scheduling with given total processing time. *Theoretical computer science*, 337(1-3):134–146, 2005. Publisher: Elsevier.
17. J. Correa, A. Cristi, B. Epstein, and J. Soto. The two-sided game of googol and sample-based prophet inequalities. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2066–2081. SIAM, 2020.
18. J. Correa, A. Cristi, L. Feuilloley, T. Oosterwijk, and A. Tsigonias-Dimitriadis. The secretary problem with independent sampling. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2047–2058. SIAM, 2021.
19. J. Correa, P. Dütting, F. Fischer, and K. Schewior. Prophet inequalities for iid random variables from an unknown distribution. In *Proceedings of the 2019 ACM Conference on Economics and Computation*, pages 3–17, 2019.
20. J. Dohrau. Online makespan scheduling with sublinear advice. In *International Conference on Current Trends in Theory and Practice of Informatics*, pages 177–188. Springer, 2015.

21 E. B. Dynkin. The optimum choice of the instant for stopping a Markov process. *Soviet Mathematics*, 4:627–629, 1963.

22 M. Englert, D. Özmen, and M. Westermann. The power of reordering for online minimum makespan scheduling. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 603–612. IEEE, 2008.

23 U. Faigle, W. Kern, and G. Turán. On the performance of on-line algorithms for partition problems. *Acta cybernetica*, 9(2):107–119, 1989.

24 M. Feldman, O. Svensson, and R. Zenklusen. A simple O (log log (rank))-competitive algorithm for the matroid secretary problem. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 1189–1201. SIAM, 2014.

25 T. S. Ferguson. Who solved the secretary problem? *Statistical science*, 4(3):282–289, 1989. Publisher: Institute of Mathematical Statistics.

26 R. Fleischer and M. Wahl. On-line scheduling revisited. *Journal of Scheduling*, 3(6):343–353, 2000. Publisher: Wiley Online Library.

27 G. Galambos and G. J. Woeginger. An on-line scheduling heuristic with better worst-case ratio than Graham's list scheduling. *SIAM Journal on Computing*, 22(2):349–355, 1993. Publisher: SIAM.

28 O. Göbel, T. Kesselheim, and A. Tönnis. Online appointment scheduling in the random order model. In *Algorithms-ESA 2015*, pages 680–692. Springer, 2015.

29 G. Goel and A. Mehta. Online budgeted matching in random input models with applications to Adwords. In *SODA*, volume 8, pages 982–991, 2008.

30 T. Gormley, N. Reingold, E. Torng, and J. Westbrook. Generating adversaries for request-answer games. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 564–565, 2000.

31 R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9):1563–1581, 1966. Publisher: Wiley Online Library.

32 A. Gupta, R. Mehta, and M. Molinaro. Maximizing Profit with Convex Costs in the Random-order Model. *arXiv preprint arXiv:1804.08172*, 2018.

33 A. Gupta and S. Singla. Random-order models. In Tim Roughgarden, editor, *Beyond worst-case analysis*. Cambridge University Press, 2020.

34 D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM (JACM)*, 34(1):144–162, 1987. Publisher: ACM New York, NY, USA.

35 H. Kaplan, D. Naori, and D. Raz. Competitive Analysis with a Sample and the Secretary Problem. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2082–2095. SIAM, 2020.

36 C. Karande, A. Mehta, and P. Tripathi. Online bipartite matching with unknown distributions. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 587–596, 2011.

37 D. R. Karger, S. J. Phillips, and E. Torng. A better algorithm for an ancient scheduling problem. *Journal of Algorithms*, 20(2):400–430, 1996. Publisher: Elsevier.

38 R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 352–358, 1990.

39 H. Kellerer and V. Kotov. An efficient algorithm for bin stretching. *Operations Research Letters*, 41(4):343–346, 2013. Publisher: Elsevier.

40 H. Kellerer, V. Kotov, and M. Gabay. An efficient algorithm for semi-online multiprocessor scheduling with given total processing time. *Journal of Scheduling*, 18(6):623–630, 2015.

41 H. Kellerer, V. Kotov, M. Grazia Speranza, and Z. Tuza. Semi on-line algorithms for the partition problem. *Operations Research Letters*, 21(5):235–242, 1997. Publisher: Elsevier.

42 C. Kenyon. Best-Fit Bin-Packing with Random Order. In *SODA*, volume 96, pages 359–364, 1996.

43 T. Kesselheim, A. Tönnis, K. Radke, and B. Vöcking. Primal beats dual on online packing LPs in the random-order model. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 303–312, 2014.

44 R. D. Kleinberg. A multiple-choice secretary algorithm with applications to online auctions. In *SODA*, volume 5, pages 630–631, 2005.

45 N. Korula, V. Mirrokni, and M. Zadimoghaddam. Online submodular welfare maximization: Greedy beats 1/2 in random order. *SIAM Journal on Computing*, 47(3):1056–1086, 2018. Publisher: SIAM.

46 O. Lachish. O (log log rank) competitive ratio for the matroid secretary problem. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 326–335. IEEE, 2014.

47 D. V. Lindley. Dynamic programming and decision theory. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 10(1):39–51, 1961. Publisher: Wiley Online Library.

48 M. Mahdian and Q. Yan. Online bipartite matching with random arrivals: an approach based on strongly factor-revealing lps. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 597–606, 2011.

49 A. Meyerson. Online facility location. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 426–431. IEEE, 2001.

50 V.S. Mirrokni, S. O. Gharan, and M. Zadimoghaddam. Simultaneous approximations for adversarial and stochastic online budgeted allocation. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1690–1701. SIAM, 2012.

51 M. Molinaro. Online and random-order load balancing simultaneously. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1638–1650. SIAM, 2017.

52 C. J. Osborn and E. Torng. List's worst-average-case or WAC ratio. *Journal of Scheduling*, 11(3):213–215, 2008. Publisher: Springer.

53 J. Rudin III. *Improved bounds for the on-line scheduling problem*. PhD thesis, University of Phoenix, 2001.

54 P. Sanders, N. Sivadasan, and M. Skutella. Online scheduling with bounded migration. *Mathematics of Operations Research*, 34(2):481–498, 2009. Publisher: INFORMS.

55 J. Sgall. A lower bound for randomized on-line multiprocessor scheduling. *Information Processing Letters*, 63(1):51–55, 1997. Publisher: Citeseer.

A Lower bounds

We establish the following theorem using two lower bound sequences.

▶ **Theorem 24.** *For every online algorithm A, deterministic or randomized, there exists a job set $\mathcal{J}$ such that $\mathbf{P}_{\sigma\sim S_n}\left[A(\mathcal{J}^\sigma) \geq \frac{\sqrt{73}-1}{6}\mathrm{OPT}(\mathcal{J})\right] \geq \frac{1}{6}$. If A is randomized the previous probability also includes its random choices.*

Theorem 24 implies the following lower bounds.

▶ **Corollary 25.** *If an online algorithm A is nearly c-competitive, then $c \geq \frac{\sqrt{73}-1}{6} \approx 1.257$.*

▶ **Corollary 26.** *The best competitive ratio possible in the secretary model is $\frac{\sqrt{73}+29}{36} \approx 1.043$.*

Let us now prove these results. For this section let $c = \frac{\sqrt{73}-1}{6}$ be our main lower bound on the competitive ratio. We consider three types of jobs:

1. *negligible jobs* of size 0 (or a tiny size $\varepsilon > 0$ if one were to insist on positive sizes).
2. *big jobs* of size $1 - \frac{c}{3} = \frac{17-\sqrt{37}}{18} \approx 0.581$.
3. *small jobs* of size $\frac{c}{3} = \frac{1+\sqrt{37}}{18} \approx 0.419$

Let $\mathcal{J}$ be the job set consisting of m jobs of each type.

▶ **Lemma 27.** *There exists a schedule of $\mathcal{J}$ where every machine has load 1. Every schedule that has a machine with smaller load has makespan at least c.*

Proof. This schedule is achieved by scheduling a type 2 and a type 3 job onto each machine. The load of each machine is then 1. Every schedule which allocates these jobs differently must have at least one machine M which contains at least three jobs of type 2 or 3 by the pigeonhole principle. The load of M is then at least $3\frac{c}{3} = c$. ◀

Given a permutation $\mathcal{J}^\sigma$ of $\mathcal{J}$ and an online algorithm A, which expects $3m+1$ jobs to arrive in total. Let $A(\mathcal{J}^\sigma, 3m+1)$ denote its makespan after it processes $\mathcal{J}^\sigma$ expecting yet another job to arrive. Let $P = \mathbf{P}[A(\mathcal{J}^\sigma, 3m+1) = 1]$ be the probability that A achieves the optimal schedule where every machine has load 1 under these circumstances. Depending on P we pick one out of two input sets on which A performs bad.

Let $j \in \{1, 2\}$. We now consider the job set $\mathcal{J}_j$ consisting of m jobs of each type plus one additional job of type j, i.e. a negligible job if $j = 1$ and a big one if $j = 2$. We call an ordering $\mathcal{J}_j^\sigma$ of $\mathcal{J}_j$ *good* if it ends with a job of type j or, equivalently, if its first $3m$ jobs are a permutation of $\mathcal{J}$. Note that the probability of $\mathcal{J}^\sigma$ being good is $\frac{m+1}{3m+1} \geq \frac{1}{3}$ for $\sigma \sim S_{3m+1}$.

▶ **Lemma 28.** *For job set $\mathcal{J}_1$ we have $\mathbf{P}_{\sigma\sim S_n}[A(\mathcal{J}_1^\sigma) \geq c\mathrm{OPT}(\mathcal{J})] \geq \frac{1-P}{3}$ and for job set $\mathcal{J}_2$ furthermore $\mathbf{P}_{\sigma\sim S_n}[A(\mathcal{J}_2^\sigma) \geq c\mathrm{OPT}(\mathcal{J})] \geq \frac{P}{3}$.*

Proof. Consider a good permutation of $\mathcal{J}_1$. Then with probability $1 - P$ the algorithm A does have makespan c even before the last job is scheduled. On the other hand $\mathrm{OPT}(\mathcal{J}_1) = 1$. Thus with probability $\frac{1-P}{3}$ we have $A(\mathcal{J}_1^\sigma) = c = c\mathrm{OPT}(\mathcal{J}_1)$.

Now consider a good permutation of $\mathcal{J}_2$. Then, with probability P, algorithm A has to schedule the last job on a machine of size 1. Its makespan is thus $2 - \frac{c}{3} = c^2$ by our choice of c. The optimum algorithm may schedule two big jobs onto one machine, incurring load $2 - \frac{2c}{3} < c$, three small jobs onto another one, incurring load c and one job of each type onto the remaining machines, causing load $1 < c$. Thus $\mathrm{OPT}(\mathcal{J}_2) = c$. In particular we have with probability $\frac{P}{3}$ that $A(\mathcal{J}_2^\sigma) = c^2 = c\mathrm{OPT}(\mathcal{J}_2)$. ◀

We now conclude the main three lower bound results.

▶ **Theorem 24.** *For every online algorithm A, deterministic or randomized, there exists a job set $\mathcal{J}$ such that $\mathbf{P}_{\sigma\sim S_n}\left[A(\mathcal{J}^\sigma) \geq \frac{\sqrt{73}-1}{6}\mathrm{OPT}(\mathcal{J})\right] \geq \frac{1}{6}$. If A is randomized the previous probability also includes its random choices.*

Proof. By the previous lemma we get that

$$\max_{j=1,2}\left(\mathbf{P}_{\sigma\sim S_n}\left[A(\mathcal{J}_j^\sigma) \geq c\mathrm{OPT}(\mathcal{J})\right]\right) = \max\left(\frac{1-P}{3}, \frac{P}{3}\right) \geq \frac{1}{6}. \quad ◀$$

▶ **Corollary 25.** *If an online algorithm A is nearly c-competitive, then $c \geq \frac{\sqrt{73}-1}{6} \approx 1.257$.*

Proof. This is immediate by the previous theorem. ◀

▶ **Corollary 26.** *The best competitive ratio possible in the secretary model is $\frac{\sqrt{73}+29}{36} \approx 1.043$.*

Proof. Let A be any online algorithm. Pick a job set $\mathcal{J}$ according to Theorem 24. Then

$$A^{\mathrm{rom}}(\mathcal{J}) = \mathbf{E}_{\sigma\sim S_n}[A(\mathcal{J}^\sigma)] \geq \frac{1}{6}\cdot\frac{\sqrt{73}-1}{6}\mathrm{OPT}(\mathcal{J}) + \frac{5}{6}\mathrm{OPT}(\mathcal{J}) = \frac{\sqrt{73}+29}{36}\mathrm{OPT}(\mathcal{J}). \quad ◀$$

B Full description of the main algorithm

Our new algorithm achieves a competitive ratio of $c = \frac{1+\sqrt{13}}{3} \approx 1.535$. Let $\delta = \delta(m) = \frac{1}{\log(m)}$ be the *margin of error our algorithm allows*. Throughout the analysis it is mostly sensible to treat δ as a constant and forget about its dependency on m. Our algorithm maintains a certain set $\mathcal{M}_{\text{res}}$ of $\lceil \delta m \rceil$ *reserve machines*. Their complement, the *principal machines*, are denoted by $\mathcal{M}$. Let us fix an input sequence $\mathcal{J}^\sigma$. Let $\hat{L} = \hat{L}[\mathcal{J}^\sigma] = L_{\delta^2}[\mathcal{J}^\sigma]$. For simplicity, we hide the dependency on $\mathcal{J}^\sigma$ whenever possible. Our online algorithm uses $B = \max\left(p_{\max}^{\delta^2 n}, \hat{L}\right)$ as an *estimated lower bound for* OPT. This bound B is known after the first $\lfloor \delta^2 n \rfloor$ jobs are treated. Our algorithm uses geometric rounding implicitly. Given a job J_t of size p_t let $f(p_t) = (1+\delta)^{\lfloor \log_{1+\delta} p_t \rfloor}$ be its *rounded size*. We also call J_t an $f(p_t)$-job. Using rounded sizes, we introduce job classes. Let $p_{\text{small}} = c - 1 = \frac{\sqrt{13}-2}{3} \approx 0.535$ and $p_{\text{big}} = \frac{c}{2} = \frac{1+\sqrt{13}}{6} \approx 0.768$. Then we call job J_t

- *small* if $f(p_t) \le p_{\text{small}} B$ and *critical* else,
- *big* if $f(p_t) > p_{\text{big}} B$,
- *medium* if J is neither small nor big, i.e. $p_{\text{small}} B \le f(p_t) \le p_{\text{big}} B$,
- *huge* if its (not-rounded) size exceeds B, i.e. $B < p_t$, and *normal* else.

Consider the sets $\mathcal{P}_{\text{med}} = \{(1+\delta)^i \mid (1+\delta)^{-1} p_{\text{small}} B < (1+\delta)^i \le p_{\text{big}} B\}$ and $\mathcal{P}_{\text{big}} = \{(1+\delta)^i \mid p_{\text{big}} B < (1+\delta)^i \le B\}$ corresponding to all possible rounded sizes of medium respectively big jobs, excluding huge jobs. Let $\mathcal{P} = \mathcal{P}_{\text{med}} \cup \mathcal{P}_{\text{big}}$. This subdivision gives rise to a *weight function*, which will be important later. Let $w(p) = 1/2$ for $p \in \mathcal{P}_{\text{med}}$ and $w(p) = 1$ for $p \in \mathcal{P}_{\text{big}}$. The elements $p \in \mathcal{P}$ define job classes $\mathcal{C}_p \subseteq \mathcal{J}$ consisting of all p-jobs, i.e. jobs of rounded size p. By some abuse of notation, we call the elements in $\mathcal{P}$ "job classes", too. We let $n_p = |\mathcal{C}_p|$ and $\hat{n}_p = |\{J_{\sigma(j)} \mid \sigma(j) \le \delta^2 n \wedge J_{\sigma(j)} \text{ is a } p\text{-job}\}|$. We want to use the values $\hat{n}_p$, which are available to an online algorithm quite early, to estimate the values n_p, which accurately describe the set of critical jobs. First, $\delta^{-2}\hat{n}_p$ comes to mind as an estimate for n_p. Yet, we need a more complicated guess: $c_p = \max\left(\left\lfloor \left(\delta^{-2}\hat{n}_p - m^{3/4}\right) w(p) \right\rfloor, \hat{n}_p\right) w(p)^{-1}$. It has three desirable advantages. First, for every $p \in \mathcal{P}$ the value c_p is close to n_p with high probability, but, opposed to $\delta^{-2}\hat{n}_p$, unlikely to exceed it. Overestimating n_p turns out to be far worse than underestimating it. Second, $w(p)c_p$ is an integer and, third, we have $c_p \ge \hat{n}_p w(p)^{-1}$. A fundamental fact regarding the values $(c_p)_{p\in\mathcal{P}}$ and B is, of course, that they are known to the online algorithm once $\lfloor \delta^2 n \rfloor$ jobs are scheduled.

Algorithm 2 The complete algorithm: How to schedule job J_t.

```
STRAT is initialized to CRITICAL, J_t is the job to be scheduled.
if n ≤ m then Schedule J_t on any empty machine;
else if t ≤ φn then schedule J_t on a least loaded machine in M;      ▷ Sampling phase
else
    if we have t = ⌊φn⌋ + 1 then
        if Σ_{p∈P} w(p)c_p > m then STRAT ← LEAST-LOADED
        else proceed with the Preparation for the Critical-Job-Strategy (Algorithm 4);
    if STRAT = CRITICAL then proceed with the Critical-Job-Strategy (Algorithm 5);
    else proceed with the Least-Loaded-Strategy (Algorithm 3);
```

Statement of the algorithm: If there are less jobs than machines, i.e. $n \le m$, it is optimal to put each job onto a separate machine. Else, a short sampling phase greedily schedules each of the first $\lfloor \delta^2 n \rfloor$ jobs to the least loaded principal machine $M \in \mathcal{M}$. Now, the values

B and $(c_p)_{p\in\mathcal{P}}$ are known. Our algorithm has to choose between two strategies, the Least-Loaded-Strategy and the Critical-Job-Strategy, which we will both introduce subsequently. It maintains a variable STRAT, initialized to CRITICAL, to remember its choice. If it chooses the Critical-Job-Strategy, some additional preparation is required. It may at any time discover that the Critical-Job-Strategy is not feasible and switch to the Least-Loaded-Strategy but it never switches the other way around.

The **Least-Loaded-Strategy** places any normal job on a least loaded principal machine. Huge jobs are scheduled on any least loaded reserve machine. This machine will be empty, unless we consider rare worst-case orders.

Algorithm 3 The Least-Loaded-Strategy: How to schedule job J_t.

if J_t is huge **then** schedule J_t on any least loaded reserve machine;
else schedule J_t on any least loaded principal machine;

For the Critical-Job-Strategy we introduce *p-placeholder-jobs* for every size $p \in \mathcal{P}$. Sensibly, the size of a p-placeholder-job is p. During the Critical-Job-Strategy we treat placeholder-jobs similar to *real* jobs. The *anticipated load* $\tilde{l}^t_M$ of a machine M at time t is the sum of all jobs on it, including placeholder-job, opposed to the common load l^t_M, which does not take the latter into account. Note that $\tilde{l}^t_M$ defines a pseudo-load as introduced in Section 3.

During the **Preparation for the Critical-Job-Strategy** the algorithm maintains a counter c'_p of all p-jobs scheduled so far (including placeholders). A job class $p \in \mathcal{P}$ is called *unsaturated* if $c'_p \le c_p$. First, we add unsaturated medium placeholder-jobs to any principal machine that already contains a medium real job from the sampling phase. We will see in Lemma 29 that such an unsaturated medium job class always exists. Now, let m_{empty} be the number of principal machines which do not contain critical jobs. We prepare a set $\mathcal{J}_{\text{rep}}$ of cardinality at most m_{empty}, which we will then schedule onto these machines. The set $\mathcal{J}_{\text{rep}}$ may contain single big placeholder-jobs or pairs of medium placeholder-jobs. We greedily pick any unsaturated job class $p \in \mathcal{P}$ and add a p-placeholder-job to $\mathcal{J}_{\text{rep}}$. If p is medium, we pair it with a job belonging to any other, not necessarily different, unsaturated medium job class. Such a job class always exists by Lemma 29. We stop once all job classes are saturated or if $|\mathcal{J}_{\text{rep}}| = m_{\text{empty}}$. We then assign the elements in $\mathcal{J}_{\text{rep}}$ to machines. We iteratively pick the element $e \in \mathcal{J}_{\text{rep}}$ of maximum size and assign the corresponding jobs to the least loaded principal machine, which does not contain critical jobs yet. Sensibly, the size of a pair of jobs in $\mathcal{J}_{\text{rep}}$ is the sum of their individual sizes. We repeat this until all jobs and job pairs in $\mathcal{J}_{\text{rep}}$ are assigned to some principal machine.

Algorithm 4 Preparation for the Critical-Job-Strategy.

while there is a machine M containing a single medium job **do**
    Add a placeholder p-job for an unsaturated size class $p \in \mathcal{P}_{\text{med}}$ to M; $c'_p \leftarrow c'_p + 1$;
while there is an unsaturated size class $p \in \mathcal{P}$ and $|\mathcal{J}_{\text{rep}}| < m_{\text{empty}}$ **do**
    Pick an unsaturated size class $e = p \in \mathcal{P}$ with c'_p minimal; $w(e) \leftarrow p$; $c'_p \leftarrow c'_p + 1$;
    if p is medium **then** pick $q \in \mathcal{P}_{\text{med}}$ unsaturated. $e \leftarrow (p, q)$; $w(e) \leftarrow p+q$; $c'_q \leftarrow c'_q + 1$;
    Add e to $\mathcal{J}_{\text{rep}}$;
while $\mathcal{J}_{\text{rep}} \neq \emptyset$ **do**
    Pick a least loaded machine $M \in \mathcal{M}$, which does not contain a critical job yet;
    Pick $e \in \mathcal{J}_{\text{rep}}$ of maximum size $w(e)$ and add the jobs in e to M;
    $\mathcal{J}_{\text{rep}} \leftarrow \mathcal{J}_{\text{rep}} \setminus \{e\}$;

▶ **Lemma 29.** *In line 2 and 5 of Algorithm 4 there is always an unsaturated medium size class available. Thus, Algorithm 4, the Preparation for the Critical-Job-Strategy, is well defined.*

Proof. Concerning line 2, there are precisely $\sum_{p\in\mathcal{P}_{\text{med}}} \hat{n}_p$ machines with critical jobs while there are at least $\sum_{p\in\mathcal{P}_{\text{med}}}(c_p - \hat{n}_p) \geq \sum_{p\in\mathcal{P}_{\text{med}}} \hat{n}_p$ placeholder-jobs available to fill them. Here we make use of the fact that for medium jobs $p \in \mathcal{P}_{\text{med}}$ we have $c_p \geq \hat{n}_p w(p)^{-1} = 2\hat{n}_p$.

Concerning line 5, observe that so far every machine and every element in $\mathcal{J}_{\text{rep}}$ contains an even number of medium jobs. If the placeholder picked in line 4 was the last medium job remaining, $\sum_{p\in\mathcal{P}_{\text{med}}} c_p$ would be odd. But this is not the case since every c_p for $p \in \mathcal{P}_{\text{med}}$ is even. ◀

Algorithm 5 The Critical-Job-Strategy.

1: **if** J_t is medium or big **then** *let p denote its rounded size;*
2: **if** there is a machine M containing a p-placeholder-job J **then**
3: Delete the p-placeholder-job J and assign J_t to M;
4: **else if** J_t is medium and there exists $M \in \mathcal{M}_{\text{res}}$ containing a single medium job **then**
5: Schedule J_t on M;
6: **else if** there exists an empty machine $M \in \mathcal{M}_{\text{res}}$ **then** schedule J_t on M;
7: **else** STAT ← LEAST-LOADED; ▷ We say the algorithm *fails*.
8: **use** the Least-Loaded-Strategy (Algorithm 3) from now on;
9: **else** assign J_t to the least loaded machine in $\mathcal{M}$ (take placeholder jobs into account);

After the Preparation is done, the **Critical-Job-Strategy** becomes straightforward. Each small job is scheduled on a principal machines with least anticipated load, i.e. taking placeholders into account. Critical jobs of rounded size $p \in \mathcal{P}$ replace p-placeholder-jobs whenever possible. If no such placeholder exists anymore, critical jobs are placed onto the reserve machines. Again, we try pair up medium jobs whenever possible. If no suitable machine can be found among the reserve machines, we have to switch to the Least-Loaded-Strategy. We say that the algorithm *fails* if it ever reaches this point. In this case, it should rather have chosen the Least-Loaded-Strategy to begin with. Since all reserve machines are filled at this point, the Least-Loaded-Strategy is impeded, too. The most difficult part of our analysis shows that, excluding worst-case orders, this is not a problem on job sets that are prone to cause failing.

Appendix D

Online Makespan Minimization with Budgeted Uncertainty

Bibliographic Information *Online Makespan Minimization with Budgeted Uncertainty.* S. Albers, M. Janke. 17th Algorithms and Data Structures Symposium (**WADS**) 2021

Summary of Contributions We study Online Makespan Minimization under budgeted uncertainty assumptions. Jobs have to be assigned to parallel and identical machines. Preemption is not allowed. Each job has both a regular processing time, as well as an additional processing time. Commonly, jobs only require the regular time but up to Γ jobs fail and require the sum of both processing times. One then considers the maximum makespan that may arise if up to Γ jobs fail, called the *uncertain makespan*. This uncertain makespan should be minimized.

We initiate the study of online algorithms for Makespan Minimization with budgeted uncertainty. We first analyze Graham's *Greedy* strategy and establish that it is $(3 - \frac{2}{m})$-competitive. This ratio is tight. We then provide a superior strategy whose competitive ratio approaches 2.9052. Our lower bound of 2 shows that the general model including budgeted uncertainty is strictly harder than its special case of classical Online Makespan Minimization.

Individual Contributions

- Initial proposal of the problem.
- Analysis of the Greedy strategy; development and analysis of the improved algorithm and the lower bound.
- Composition of the manuscript, including graphics, graphs, technical and non-technical parts.

Online Makespan Minimization With Budgeted Uncertainty

Susanne Albers and Maximilian Janke

Department of Computer Science, Technical University of Munich

Abstract. We study Online Makespan Minimization with uncertain job processing times. Jobs are assigned to m parallel and identical machines. Preemption is not allowed. Each job has a regular processing time while up to Γ jobs fail and require additional processing time. The goal is to minimize the makespan, the time it takes to process all jobs if these Γ failing jobs are chosen worst possible. This models real-world applications where acts of nature beyond control have to be accounted for.
So far Makespan Minimization With Budgeted Uncertainty has only been studied as an offline problem. We are first to provide a comprehensive analysis of the corresponding online problem.
We provide a lower bound of 2 for general deterministic algorithms showing that the problem is more difficult than its special case, classical Online Makespan Minimization. We further analyze Graham's Greedy strategy and show that it is precisely $\left(3 - \frac{2}{m}\right)$-competitive. This bound is tight. We finally provide a more sophisticated deterministic algorithm whose competitive ratio approaches 2.9052.

Keywords: Scheduling · Makespan Minimization · Online Algorithm · Competitive Analysis · Lower Bound · Uncertainty · Budgeted Uncertainty.

1 Introduction

Scheduling is universal in countless areas of computing, decision making and management: Machines simultaneously produce diverse specialized equipment; server hubs execute numerous types of programs at the same time; employees need to perform various tasks in parallel. Innumerous scheduling problems in the literature are derived from such applications and model fuzzy real-world problems using precise mathematical language and problem specifications. This unnatural precision leads to failure when classical approaches are applied to real-world environments. Real-world settings are less clear-cut with multiple sources of uncertainty that vastly affect results. Consider acts of nature beyond control and predictability occurring rarely but regularly: For example, machines malfunction and need to be repaired; programs exhibit bugs, which require restarts and debugging; changes in working environment — such as a global pandemic — require new solutions such as working from home. Most tasks are eventually adapted to the situation and performed as efficiently as before. Still, a few remaining ones suddenly require a lot more time and effort in our daily schedules.

Such errors in the expected difficulty of tasks easily render theoretical predictions void and motivated various models incorporating uncertainty in the literature.

A prolific line of research on online algorithms, [14,15,17,19,23,26,30] and references therein, considers explorable uncertainty. This type of uncertainty can be queried and explored by the online algorithm. Such approaches are sensible for many practical applications but fail if uncertainty is inherently unpredictable — or unexplorable — even to the offline algorithm.

Offline approaches, dating back to the 1950s [13], propose stochastic models; more recent approaches, particularly Budgeted Uncertainty, consider worst-case scenarios, which accustom risk-averse decision makers. To date, many offline problems, Scheduling [8,9,38], Bin Packing [34] and Linear Optimization [6,7] among them, have been analyzed under Budgeted Uncertainty assumptions.

Surprisingly, these studies never extended to online settings. Current online analyses measure the price of not having information regarding the future — an online algorithm is prepared for whatever may come — but are not the least skeptical about information once it is "obtained".

This paper studies the most basic scheduling problem of *Online Makespan Minimization* under *Budgeted Uncertainty* assumptions: Jobs have to be assigned to m identical and parallel machines. Preemption is not allowed. The goal is to minimize the time it takes to process them all, the *makespan*. Each job J_i is defined by two processing times. Its regular processing time $\tilde{p}_i$ is its time required under normal circumstances. Its additional time Δp_i has to be added in rare cases of failure for reparation, potential reduced performance or other application-specific slowdown. Since failures are the exception, it would be extremely pessimistic to assume that jobs in general take time $\tilde{p}_i + \Delta p_i$. Instead, one only accounts for at most Γ such failures. Given an assignment of jobs, we consider the maximum time required for processing these jobs in parallel if up to Γ failures occurred in total. This time is called the *uncertain makespan* or simply the *makespan*. The objective is to minimize this quantity.

For $\Gamma = 0$ the problem reduces to classical makespan minimization. Only regular processing times $\tilde{p}_i$ matter. Similarly, the "paranoid case" $\Gamma = \infty$, where every program has a bug and every machine constantly malfunctions, yields classical makespan minimization using *worst-case processing times* $p_i = \tilde{p}_i + \Delta p_i$.

To an *online algorithm* A jobs are revealed one by one and each has to be scheduled immediately and irrevocably before the next one is revealed. In particular, A never "learns' " which jobs fail and cannot perform optimally on arbitrary input sequences. The (uncertain) makespan of A, denoted by $A(\mathcal{J})$, is then compared to the optimum makespan $\mathrm{OPT}(\mathcal{J})$. Online algorithm A is c-competitive, $c \geq 1$, if $A(\mathcal{J})$ exceeds $\mathrm{OPT}(\mathcal{J})$ by at most a factor c on all input sequences $\mathcal{J}$. The smallest such factor is the *competitive ratio* $c = \sup_{\mathcal{J}} \frac{A(\mathcal{J})}{\mathrm{OPT}(\mathcal{J})}$. The goal is to design online algorithms achieving small competitive ratios.

Related work: Scheduling is a fundamental problem in theoretical computer science and has been studied extensively in both offline and online variants, see e.g. [4,16,20,22,24,31] and references therein. We thus only focus on results most relevant to our work beginning with robust scheduling. Early work studied arbitrary, mostly finite sets of scenarios, see e.g. [3,28,29,32]. More modern work [8,9,38] has adapted to the model of Budgeted Uncertainty from [6]. In particular, Bougeret et. al. [9] habe provided a first 3-approximation for the offline version of the problem studied in this paper and a PTAS for constant Γ. This result has been recently improved by the same authors and Jansen [8]. They provide an EPTAS for general Γ and show that, assuming $P \neq NP$, the best possible approximation ratio for unrelated machines lies in the interval $[2, 3]$.

Online Makespan minimization is very thoroughly researched. We again only review results most relevant for our work. Already in the 1960s Graham [22] established that his famed Greedy strategy obtains a strong competitive ratio of precisely $2 - \frac{1}{m}$. It took nearly thirty years till a breakthrough of Galambos and Woeginger [21] sparked a fruitful line of research [4,27,1] leading to the currently best competitive ratio of 1.9201 from [20]. Chen et al. [10] have given an online algorithm whose competitiveness is at most $1 + \varepsilon$ times the best possible ratio but no explicit bounds on this ratio are obtained. For general m, lower bounds are given in [18,5,1,35]. The currently best lower bound is 1.885 due to [35].

More recent results on Online Makespan Minimization consider semi-online settings, which equip the online algorithm with additional capabilities or information ahead of time [11,36,16]; different approaches weaken the adversary's ability to determine the job-order [2,16]; yet other settings analyze more involved objective functions: particularly the model of vector scheduling [12,25] also considers jobs that have two or even more "processing times". Unlike in our model, these "times" represent multidimensional resource requirements.

A related line of research on online algorithms considers "explorable uncertainty". In 1991, Kahan [26] has investigated the number of queries necessary to determine median and maximum of a set of "uncertain" numbers. This sparked a long line of research covering many problems, such as finding the median or general rank-k-elements [19,23,26,30], caching [33] or most recently scheduling [14,15]. The latter work, in fact, adds the cost of querying to the general cost paid by the algorithm for performing its task at hand. We refer to the survey by Erlebach and Hoffmann [17] and references therein for more results on this topic.

Modern work of Singla [37] introduces the 'price of information' into classical stochastic uncertainty settings, which results in a model highly related to the Budgeted Uncertainty model.

Our contribution: We study online makespan minimization with Budgeted Uncertainty in depth. First, we give tight bounds on the competitive ratio of the Greedy strategy under Budgeted Uncertainty, which shows surprising parallels to the traditional result. It is precisely $\left(3 - \frac{2}{m}\right)$-competitive. This already beats the oldest published offline approximation ratio [9], while being a much simpler

approach at the same time. The lower bound showing that Greedy is not better than $\left(3-\frac{2}{m}\right)$-competitive can be chosen such that $\tilde{p}_i = 0$ for all i raising the question whether this important special case is as hard the general problem.

Next, we provide a better deterministic algorithm particularly suited for large values of m and Γ. Our algorithm adapts the proven strategy from [20] and earlier work of prioritizing schedules exhibiting a steep load profile. These profiles are highly desirable and generally pose no problem to the online algorithm. It is then shown that difficult input sequences leading to less desirable profiles cannot be efficiently scheduled by any algorithm, including the optimum offline algorithm. This ameliorates the possibly higher makespan of the online algorithm on these difficult sequences.

Precise competitive ratios for small m and Γ, Figure 2, attest a strong performance unless Γ is extremely small and m is big. The latter setting is rather unnatural since one expects the number of errors to scale with the number of machines (and jobs). For large m and Γ the competitive ratio rapidly approaches 2.9052. In general, the algorithm outperforms the Greedy strategy.

We end with a lower bound of 2 for the competitive ratio achievable by any deterministic algorithm. The general model of Budgeted Uncertainty is therefore strictly more difficult than the classical model without uncertainty where even Graham's Greedy strategy is $\left(2-\frac{1}{m}\right)$-competitive.

2 Problem definition

Consider any input sequence $\mathcal{J} = J_1, \ldots J_n$. Job J_i is defined by a pair $(\tilde{p}_i, \Delta p_i)$ of non-negative real numbers where $\tilde{p}_i$ is the *regular processing time* of J_i, while Δp_i is its *additional time*. The time job J_i takes to be processed in the worst case is $p_i = \tilde{p}_i + \Delta p_i$, its *robust time*.

A schedule is simply a function $\sigma\colon \mathcal{J} \to \mathcal{M}$ mapping job $J \in \mathcal{J}$ to the machine $\sigma(J) = M \in \mathcal{M}$ processing it. The *regular load* of a machine $M \in \mathcal{M}$ is $\tilde{l}_M = \sum_{\sigma(J_i)=M} \tilde{p}_i$, the time it takes said machine to process all jobs in the best-case. The additional times we have to account for in the worst-case is the *additional load* Δl_M, the sum of the Γ largest additional times of jobs (or all additional times, if less than Γ jobs are scheduled on M). Formally $\Delta l_M = \max\left(\sum_{J_i \in \mathcal{J}'} \Delta p_i \mid \mathcal{J}' \subseteq \sigma^{-1}(M), |\mathcal{J}'| \leq \Gamma\right)$. Let us fix any set $\mathcal{J}'(M)$ where the maximum in the previous term is obtained. We break ties by preferring jobs J_i that came later, i.e. with larger indices i, and by choosing $\mathcal{J}'(M)$ of minimal size. We say for each job $J_i \in \mathcal{J}'(M)$ that J_i *fails in* σ. This allows us to write $\Delta l_M = \sum_{J_i \text{ fails } M} \Delta p_i$. Finally, the *robust load* l_M *of a machine* M is the maximum time machine M may require if up to Γ jobs fail: Formally $l_M = \tilde{l}_M + \Delta l_M = \sum_{J_i \in \sigma^{-1}(M)} \tilde{p}_i + \sum_{J_i \text{ fails } M} \Delta p_i$.

Given any algorithm A, which outputs the schedule σ, its *(robust) makespan* is then $A(\mathcal{J}) = \max_M l_M$. The goal is to design algorithms exhibiting low makespans. Technically, in the classical problem only Γ jobs fail in total. For the analysis a more general, equivalent version leads to a better intuition: Since the definition of each Δl_M only makes use of the fact that at most Γ jobs fail on

M, the problem stays equivalent if we allowed up to Γ jobs to fail per machine. Let OPT be any (fixed) offline algorithm that on any input sequence $\mathcal{J}$ outputs an optimum schedule. By some abuse of notation we also denote the optimum makespan $\mathrm{OPT}(\mathcal{J})$ by OPT, if the corresponding sequence $\mathcal{J}$ is clear.

An Algorithm A is called an *online algorithm*, if it assigns each job J_i in $\mathcal{J} = J_1, \ldots, J_n$ independent of future jobs, i.e. $J_{i+1}, \ldots, J_n$. It's *competitive ratio* is then $c = \sup_{\mathcal{J}} \frac{A(\mathcal{J})}{\mathrm{OPT}(\mathcal{J})}$, the quantity we wish to minimize.

3 Graham's Greedy Strategy

In his seminal work [22] Graham analyzed the greedy strategy and showed that it is $\left(2 - \frac{1}{m}\right)$-competitive. In general, the scheduling literature differentiates between pre-greedy and post-greedy strategies. The former simply choose a least loaded machine; the latter choose a machine such that the resulting load is minimal. Ties can be broken arbitrarily. For classical Makespan Minimization these notions coincide and all greedy strategies are identical up to permutations of machines. For our problem, the pre-greedy strategies perform significantly worse, which is why we focus on post-greedy strategies.

We then are going to establish the following theorem.

Theorem 1. *The competitive ratio of the post-greedy strategy is precisely* $3 - \frac{2}{m}$.

In particular, we will also present a lower bound on which any neither the post-greedy nor the pre-greedy strategy does perform better. This lower bound, interestingly, can be chosen such that $\tilde{p} = 0$ for all jobs, which might be evidence that this case is not easier than the general case. A bound on which the pre-greedy strategy performs worse is omitted.

3.1 Upper Bound

Let us consider the case of classical makespan minimization, i.e. we assume that $\Delta p_i = 0$ for all jobs J_i. The core idea of Graham [22] was to consider the *average load* of any schedule, that is $\tilde{L} = \frac{1}{m} \sum_M \tilde{l}_M = \frac{1}{m} \sum_{J_i \in \mathcal{J}} \tilde{p}_i$. The second term shows that this average load is independent of the schedule considered. This has two important consequences. First, $\mathrm{OPT} \geq \tilde{L}$ since even the optimal schedule cannot have all machine loads below average. On the other hand, no scheduler, not even the worst one, can bring all machine loads above the average load $\tilde{L}$. Graham thus argues that the least loaded machine considered by his greedy strategy has load at most $\tilde{L} \leq \mathrm{OPT}$. Since the job placed on it cannot have processing time greater than OPT either, it thus cannot cause a load exceeding 2OPT. In other words, for classical makespan minimization the greedy strategy is 2-competitive.

In our setting, the core argument of Graham does not work anymore. For $\Delta p_i \neq 0$, the average robust load L is far from being independent of the schedule in question. In fact, it may differ by a factor of m. Before giving an example let us introduce the required notation. Consider the schedule computed by any

algorithm A on input sequence $\mathcal{J}$ and let $L[A] = L[A, \mathcal{J}] = \frac{1}{m}\sum_M l_M$ denote its *average (robust) load*. Similarly, let $\Delta L[A] = \Delta L[A, \mathcal{J}] = \frac{1}{m}\sum_M \Delta l_M = L[A] - \tilde{L}$. Consider $m \cdot \Gamma$ (or more) jobs with processing vector $(\tilde{p}_i, \Delta p_i) = (0, 1/\Gamma)$. Let A be the strategy that always uses a machine of least load and let B be the algorithm which only uses one single machine. Then $L[A] = 1$ while $L[B] = \frac{1}{m}$. The average robust load thus highly depends upon the algorithm considered. Interestingly, we can bound said average load using the optimum makespan OPT.

Lemma 1. *The average (robust) load $L[A, \mathcal{J}]$ of any algorithm A on input $\mathcal{J} = J_1, \ldots, J_n$ is at most $L[\mathrm{OPT}, \mathcal{J}] + \left(1 - \frac{1}{m}\right)\mathrm{OPT}(\mathcal{J})$.*

Proof. Let us fix the sequence $\mathcal{J}$ and omit it from the notation. Let T be the set of jobs that fail A but not OPT. If T is empty, all jobs that fail A also fail OPT and thus $L[A, \mathcal{J}] \leq L[\mathrm{OPT}, \mathcal{J}]$. The lemma follows. Else, consider $J_{\max} \in T$ of maximum additional processing time $\Delta p_{\max}$. Consider the machine M that contains $J_{\max}$ in the optimum schedule. Since $J_{\max}$ does not fail OPT there are Γ different jobs of additional processing time at least $\Delta p_{\max}$ assigned to M which fail OPT. Let G be the set of these jobs. We obtain

$$\begin{aligned}\Delta L[A, \mathcal{J}] - \Delta L[\mathrm{OPT}, \mathcal{J}] &\leq \frac{1}{m}\sum_{J_i \in T} \Delta p_i - \frac{1}{m}\sum_{J_i \in G} \Delta p_i \\ &\leq \frac{1}{m} \cdot (|T| - |G|)\Delta p_{\max}.\end{aligned}$$

At most Γ jobs can fail any machine, therefore $|T| \leq \Gamma m$. By definition $|G| = \Gamma$. Finally, $\Delta p_{\max} \leq \frac{\mathrm{OPT}}{\Gamma}$ since the Γ jobs in G all have additional processing time at least $\Delta p_{\max}$ while their total additional processing is at most OPT. Now the previous inequality yields

$$\Delta L[A, \mathcal{J}] - \Delta L[\mathrm{OPT}, \mathcal{J}] \leq \frac{\Gamma m - \Gamma}{m} \cdot \frac{\mathrm{OPT}}{\Gamma} = \left(1 - \frac{1}{m}\right)\mathrm{OPT}(\mathcal{J}).$$

Recall that the average regular load $\tilde{L} = \frac{1}{m}\sum_{J_i} \tilde{p}_i$ is the same for every algorithm. Therefore $L[A, \mathcal{J}] - L[\mathrm{OPT}, \mathcal{J}] = \Delta L[A, \mathcal{J}] - \Delta L[\mathrm{OPT}, \mathcal{J}]$. Together with the previous inequality this implies $L[A, \mathcal{J}] \leq L[\mathrm{OPT}, \mathcal{J}] + \left(1 - \frac{1}{m}\right)\mathrm{OPT}(\mathcal{J})$. □

Lemma 2. *The post-greedy strategy incurs makespan at most $\left(3 - \frac{2}{m}\right)\mathrm{OPT}(\mathcal{J})$.*

Proof. Let $\mathcal{J} = J_1, \ldots, J_n$. Using induction over n we may assume that the statement of the lemma holds right before job J_n is scheduled. By definition of a post-greedy assignment it then suffices to see that there exists a machine M whose load will not exceed $(3 - 2/m)\,\mathrm{OPT}(\mathcal{J})$ if we assign J_n to it.

Let $(p_n, \Delta p_n)$ be the processing vector of J_n. Consider the greedy schedule of the first $n-1$ jobs and preliminarily assign job J_n to any machine that causes it to fail, i.e. contains less than Γ jobs of processing time strictly exceeding Δp_n.

If no such machine exists, schedule job J_n on an arbitrary machine. Let L be the average robust load of this schedule. We replace each job J_i by a job $\hat{J}_i$ whose regular processing time is $\hat{p}_i = \tilde{p}_i$ if the job does not fail this preliminary schedule and $\hat{p}_i = p_i = \tilde{p}_i + \Delta p_i$ else. Per definition that does not change the load of any machine. Also, $L = \frac{1}{m}\sum_i \hat{p}_i$. Now, we remove the last job $\hat{J}_n$ from the machine it was scheduled on and assign it to a least loaded machine M (with regards to the processing times $\hat{p}_i$). After removing the job $\hat{J}_n$ the average load of the schedule is $L - \frac{1}{m} \cdot \hat{p}_n$. Hence, after assigning $\hat{J}_n$ to the least loaded machine the makespan is at most $L + \left(1 - \frac{1}{m}\right)\hat{p}_n$. Now, replace the jobs $\hat{J}_i$ by their original variants J_i. This can only cause the load of M to decrease. Indeed, by our choice of preliminary assignment we had $\hat{p}_n = p_n$ unless there was no machine on which J_n could fail, so job J_n contributes at most $\hat{p}_n$ to the load of M. For other jobs that fail M it is clear that they continue to do so if we remove J_n. Thus, these jobs contribute processing time $p_i = \hat{p}_i$. Jobs that do not fail only contribute processing time $\tilde{p}_i \leq \hat{p}_i$.

We have shown that assigning J_n to machine M causes its load to be at most $L + \left(1 - \frac{1}{m}\right)\tilde{p}_n \leq \tilde{L}[\mathcal{J}, \mathrm{OPT}] + \left(1 - \frac{1}{m}\right)\mathrm{OPT} + \left(1 - \frac{1}{m}\right)\tilde{p}_n \leq \left(3 - \frac{2}{m}\right)\mathrm{OPT}$. The first inequality is due to Lemma 1 (there is some algorithm computing the preliminary schedule), the second due to the fact that the average load $\tilde{L}[\mathcal{J}, \mathrm{OPT}]$ of the optimum schedule as well as $\tilde{p}_n$, the robust processing time of any job, are both lower bounds for OPT. By definition a post-greedy strategy will not cause a makespan exceeding the one it could obtain by choosing M. □

3.2 Lower Bound

Lemma 3. *Neither pre-greedy nor post-greedy strategy can be better than* $\left(3 - \frac{2}{m}\right)$*-competitive, even when all jobs have regular processing time* $\tilde{p} = 0$.

Figure 1 illustrates the lower bound; the formal proof is left to the full version.

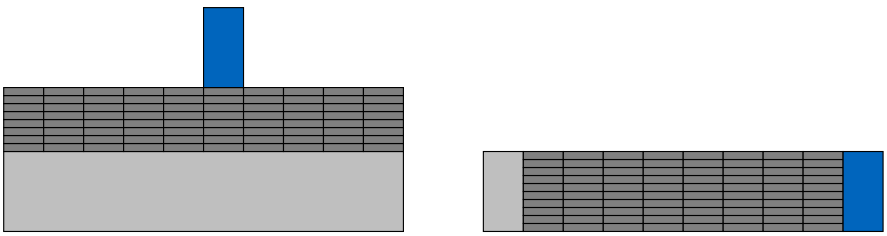


Fig. 1. The lower bound for greedy strategies, where all jobs have regular processing time $\tilde{p} = 0$. First, tiny sand-like jobs fill the greedy schedule (on the left) to a height of almost 1. Small jobs increase the height to $2 - 2/m$. Finally, a large job of size 1 causes a makespan of size $3 - 2/m$. The optimum schedule (on the right) places the sand-like jobs on a single machine. Since only Γ jobs fail the load is 1. The next $m - 1$ machines are filled with small jobs to a height of 1. The final machine captures the large job.

4 An improved deterministic algorithm

The shortcoming of the Greedy strategy is that it creates 'critically flat' schedules. Jobs with high additional time tend to be spread thin onto separate machines when it would be better to cluster them. Moreover, a single large job J assigned to a flat schedule easily causes a high makespan. OPT commonly can 'sink J down' profiting a lot. Our algorithm thus tries to avoid flat schedules. When presented with one, it prefers to use a medium machine to make it steeper. Of course, recklessly using a medium machine on a dangerous schedule is folly. Said machine is only used if the algorithm can guarantee c-competitiveness. We specify the competitive ratio $c = c_{\Gamma,m}$, depending on both Γ and m, later.

For any input sequence $\mathcal{J} = J_1, \ldots, J_n$ and any time t consider the schedule of the algorithm right after after job J_t is scheduled. For $t = 0$ consider the empty schedule. We order machines by their robust loads, breaking ties arbitrarily but consistently. We call the $d = \left\lfloor \frac{c-2}{c} m \right\rfloor \approx 0.3116m$ least loaded machines *small*; the following d machines are *medium* and the remaining $m - 2d$ most-loaded ones are *large*. Let $\mathcal{M}^t_{\text{med}}$ be the set of medium machines and let $L^t_{\text{med}} = \frac{1}{|\mathcal{M}^t_{\text{med}}|} \sum_{M \in \mathcal{M}^t_{\text{med}}} l^t_M$ be their average robust load. Let M^t_{med} be the machine in $\mathcal{M}^t_{\text{med}}$ of least robust load $l^t_{\text{med}} = \min_{M \in \mathcal{M}^t_{\text{med}}} l^t_M$. Note that $l^t_{\text{med}} \leq L^t_{\text{med}}$. We use similar notation for the small and large machines: $\mathcal{M}^t_{\text{med}}$, L^t_{med}, l^t_{med}, $\mathcal{M}^t_{\text{large}}$, etc. with the index chosen accordingly. In particular, M^t_{small} denotes the least-loaded machine. Finally, let L^t denote the average (robust) load at time t.

We call the schedule at time $t \geq 0$ *steep* if $L^{t-1}_{\text{small}} \leq \left(1 - \frac{1}{2(c-1)}\right) L^{t-1}_{\text{large}}$ and *flat* else. Steep schedules are highly desirable. Our algorithm can and will always pick the least loaded machine. If the schedule is flat, our goal should be to make it steep again. Thus, first the least loaded medium machine M^{t-1}_{med} is sampled. If scheduling a job on machine M^{t-1}_{med} will not cause its load to exceed $\frac{c}{2} L^{t-1}$, i.e. if $l^{t-1}_{\text{med}} + p_t \leq \frac{c}{2} L^{t-1}$, we use M^{t-1}_{med} . This guarantees c-competitiveness, see Lemma 4. Else, the least loaded machine M_{small} has to be used. Seeing that this does not break c-competitiveness is the main challenge for the analysis.

Algorithm 1 *How to schedule job J_t with processing time p_t.*

1: **if** $L^{t-1}_{\text{small}} > \left(1 - \frac{1}{2(c-1)}\right) L^{t-1}_{\text{large}}$ and $l^{t-1}_{\text{med}} + p_t \leq \frac{c}{2} L^{t-1}$ **then**
2: Schedule job J_t on the least loaded medium machine M^{t-1}_{med};
3: **else** schedule job J_t on the least loaded machine M^{t-1}_{small}.

The values of c. Recall that $d = \left\lfloor \frac{c-2}{c} m \right\rfloor$. Let $\Gamma \geq 2$. The competitive ratio c is chosen minimally such that $c \geq \frac{7+\sqrt{17}}{4} \approx 2.7808$ and the following holds:

$$\left(1 - \frac{d}{2(c-1)m} - 2\frac{\Gamma+1}{c\Gamma}\right)\left(1 + \frac{c}{2m}\right)^d + 2\frac{\Gamma+1}{c\Gamma} \geq \frac{2}{c-1} \cdot \frac{m-1}{m}. \quad (1)$$

Unless m is chosen extremely small c is determined by Inequality (1) and fulfills it with equality. We show in the full version that c is below 2.9052 for $m, \Gamma \to \infty$.

The following is the main result of this paper.

Theorem 2. *The algorithm is c-competitive with $c < 2.9052$ for Γ large.*

Using a suitable data-structure that maintains the values L^{t-1}_{small}, L^{t-1}_{large} and l^{t-1}_{med} the algorithm can schedule each job efficiently in time $O(\log(m+\Gamma))$.

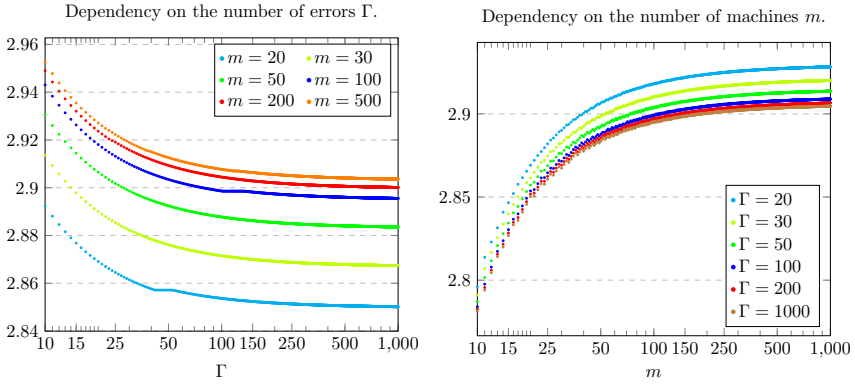


Fig. 2. The competitive ratios for different m and Γ. The ratio c is monotonously decreasing in Γ and tends to increase in m albeit not monotoneously due to the rounding involved in d. The x-axes are log-scaled. The graphs are colored.

Analysis of the algorithm Consider any input sequence $\mathcal{J} = J_1, \ldots, J_n$ and let $\text{OPT} = \text{OPT}(J_1, \ldots, J_n)$. By induction on n the makespan of the online algorithm did not exceed $c\text{OPT}$ before job J_n was scheduled. We need to show that J_n did not cause the makespan to exceed this value either. Let $L = L^n$ be the average robust load of the online schedule after all jobs in $\mathcal{J}$ have been scheduled.

Lemma 4. *We have $L \leq 2\text{OPT}$. In particular, if job J_n was scheduled on the least loaded medium machine M^{n-1}_{med} the makespan was at most $c\text{OPT}$.*

Proof. The first part follows from Lemma 1. Now, observe that if job J_n was assigned to machine M^{n-1}_{med} its load could not have exceeded $\frac{c}{2}L^{n-1} \leq c\text{OPT}$ afterwards per definition of the algorithm. □

We focus for simplicity on the case $m \to \infty$. The improvements required for small values of m are detailed at the end of the paragraph in Remark 1. Consider the case that J_n is scheduled on a least loaded machine. Let λ be its robust load. We need to bound $\lambda + p_n$, its load after receiving job J_n. Since $p_n \leq \text{OPT}$, we thus need to show that $\lambda \leq (c-1)\text{OPT}$. If $\lambda \leq \frac{c-1}{2}L$ this is a consequence of

Lemma 4. Hence, we are left to consider the case where all machines have load at least $\lambda > \frac{c-1}{2}L$. We call such a schedule *critically flat*.[1] Our algorithm cannot always prevent such schedules. The main part of our analysis shows that such schedules exhibit a highly specific structure.

Lemma 5. *If a schedule is critically flat, i.e. all machines have load* $\lambda > \frac{c-1}{2}L$, *then every machine contains a job of processing time at least* $\frac{\lambda}{2(c-1)}$.

We are going to prove this lemma in the next section. Let us first use it to conclude the analysis.

Proof of Theorem 2. By the previous analysis we only need to consider the case that job J_n is scheduled on a critically flat schedule where the least loaded machine has load λ. Since $\lambda \leq L \leq 2\mathrm{OPT}$ we are done if job J_n has processing time $p_n \leq \frac{\lambda}{2(c-1)} \leq \frac{\mathrm{OPT}}{c-1} \leq (c-2)\mathrm{OPT}$. The last inequality uses the condition $c \geq 2.7808$. If $p_n \leq \frac{\lambda}{2(c-1)}$ job J_n could only cause a makespan of $\lambda + p_n \leq 2\mathrm{OPT} + (c-2)\mathrm{OPT} = c\mathrm{OPT}$.

But else, we have shown that the sequence $J_1, \ldots, J_n$ contains $m+1$ jobs of processing time $\frac{\lambda}{2(c-1)}$. One of them is J_n while the remaining m exist due to Lemma 5. By the pigeonhole principle OPT needs to place two such jobs on a single machine, attaining a makespan of at least $2\frac{\lambda}{2(c-1)} = \frac{\lambda}{c-1}$. But this shows that $\lambda \leq (c-1)\mathrm{OPT}$ and thus $\lambda + p_n \leq (c-1)\mathrm{OPT} + \mathrm{OPT} = c\mathrm{OPT}$. □

Remark 1. Our previous definition of *critically flat* schedules given for $m \to \infty$ can be improved if m, the number of machines, is small. We then call the schedule *critically flat* if $\lambda > \frac{c-1}{2}\frac{m}{m-1}L^{n-1}$. Note that both definitions agree for $m \to \infty$. The previous arguments can be generalized to show that we are c-competitive if job J_n is not assigned to a schedule which is critically flat using this definition. The proof is left to the full version.

Understanding how critically flat schedules are formed. In our analysis we have to differentiate between *early* and *late* jobs. The latter will be scheduled on a full and flat schedule. For them to be assigned to a least loaded machine they will need to be fairly sizable. This requires the adversary to present quite large jobs constituting a lot of processing volume to achieve a critically flat schedule. Formally, we call a job J_t *late* if two conditions are met. We require job J_t to be scheduled on a flat schedule and we require machine M_{med}^{t-1} to have load at least λ before job J_t is scheduled. If a job J_t is not late or followed by a job which is not late, we call it *early*. In particular, a job can be both early and late. We next are going to consider late jobs.

Lemma 6. *If job* J_t *is late and caused a machine to first reach load* λ, *its (robust) processing time is at least* $p_t \geq \frac{\lambda}{2(c-1)}$.

[1] The term 'critically flat' is not a misnomer, by Remark 2 such a schedule is, in particular, flat.

Proof. Let $l = l^{t-1}_{\text{med}}$ be the load of M^{t-1}_{med}. By assumption of J_t being late there holds $l \geq \lambda$. Since J_t caused a machine to first reach load λ it was not scheduled on M^{t-1}_{med} but on the least loaded machine M^{t-1}_{small} instead. Since the schedule was flat, job J_t must have had processing time exceeding $\frac{c}{2}L^{t-1} - l$, i.e. $p_t > \frac{c}{2}L^{t-1} - l$. If the least loaded machine M^{t-1}_{small}, which J_t caused to reach load λ, had load less than $\lambda - \frac{\lambda}{2(c-1)}$, the statement of the lemma follows immediately. Else, all small machines had load at least $\lambda - \frac{\lambda}{2(c-1)}$ and all medium and large machines had load at least l. Thus $L^{t-1} \geq \frac{m-d}{m}l + \frac{d}{m}\left(\lambda - \frac{\lambda}{2(c-1)}\right)$. In particular

$$p_t > \frac{c}{2}L^{t-1} - l \geq \frac{c}{2}\left(\frac{m-d}{m}l + \frac{d}{m}\left(\lambda - \frac{\lambda}{2(c-1)}\right)\right) - l.$$

Note that d was chosen precisely maximal such that $d \leq \frac{c-2}{c}m$, or equivalently $\frac{c}{2}\frac{m-d}{m} - 1 \geq 0$. In fact, this inequality motivates the choice of d. The inequality implies that the previous term is non-decreasing in l and minimal for $l = \lambda$. Setting $l = \lambda$, we get that

$$p_t \geq \frac{c}{2}\left(1 - \frac{d}{2(c-1)m}\right)\lambda - \lambda > \left(\frac{c}{2} - \frac{c-2}{4(c-1)} - 1\right)\lambda \geq \frac{1}{2(c-1)}\lambda.$$

The first inequality uses that $d < \frac{c-2}{c}m$. The second inequality is simply an algebraic computation which uses that $c \geq \frac{7+\sqrt{17}}{4}$. □

The critical machines. We call a time t *early* or *late* if job J_t had this property. So far, we have treated late times. Now, we need to establish a corresponding result for early times. This requires us to understand a certain set of critical machines. Given any time t, let $\mathcal{M}^{t-1}_{\text{crit}}$ be the set of d most-loaded machines whose load has not yet reached λ before job J_t is scheduled. If less than d machines fulfill the latter condition, then all of them belong to $\mathcal{M}^{t-1}_{\text{crit}}$. Let $L^{t-1}_{\text{crit}} = \frac{1}{|\mathcal{M}^{t-1}_{\text{crit}}|}\sum_{M \in \mathcal{M}^{t-1}_{\text{crit}}} l^{t-1}_M$ be their average load. We use the convention $1/0 = \infty$, or in other words set $L^{t-1}_{\text{crit}} = \infty$ if $\mathcal{M}^{t-1}_{\text{crit}} = \emptyset$. The following lemma is fairly technical and will be proven in the full version.

Lemma 7. *Let s be the last early time, i.e. if $t > s$, then t is late. Then $L^s_{\text{crit}} \leq \left(1 - \frac{1}{2(c-1)}\right)\lambda$.*

Remark 2. The lemma implies that $\mathcal{M}^s_{\text{crit}} \neq \emptyset$. Using that $\mathcal{M}^{n-1}_{\text{crit}} = \emptyset$ we get that $s < n-1$ or, equivalently, that critically flat schedules are always flat.

Recall that our algorithm considers certain machines to be small, large and medium. It turns out that if it was not for the online setting, i.e. if the algorithm had advance knowledge of the sequence, the critical machines are the true contestants for the label "medium". To be precise we will establish that the critical machines separate "large" machines of load at least λ from 'small' ones of load at most $\left(1 - \frac{1}{2(c-1)}\right)\lambda$. The following claim is the first step towards establishing this result in Lemma 8.

Claim. Assume that $L^t_{\text{crit}} \leq \left(1 - \frac{1}{2(c-1)}\right)\lambda$. If $M^{t-1}_{\text{med}} \in \mathcal{M}^{t-1}_{\text{crit}}$ then the schedule is steep. In particular, our algorithm never uses machine in $\mathcal{M}^{t-1}_{\text{crit}} \setminus \{M^{t-1}_{\text{small}}\}$.

Proof. Assume $M^{t-1}_{\text{med}} \in \mathcal{M}^{t-1}_{\text{crit}}$. Since $d-1$ machines lie strictly in between M^{t-1}_{med} and the machines in $\mathcal{M}^{t-1}_{\text{large}}$ the latter must be disjoint from $\mathcal{M}^{t-1}_{\text{crit}}$. Thus, they all had load λ. In particular, $L^{t-1}_{\text{large}} \geq \lambda$. Using this, we conclude that the schedule was steep and, consequently, that the algorithm used M^{t-1}_{small}:

$$L^{t-1}_{\text{small}} \leq L^t_{\text{small}} \leq L^t_{\text{crit}} \leq \left(1 - \frac{1}{2(c-1)}\right)\lambda \leq \left(1 - \frac{1}{2(c-1)}\right) L^{t-1}_{\text{large}}. \qquad \square$$

Lemma 8. *If t is early, $L^{t-1}_{\text{crit}} \leq \left(1 - \frac{1}{2(c-1)}\right)\lambda$. Moreover job J_t was either scheduled on a machine of load at least λ or on a machine of load at most $\left(1 - \frac{1}{2(c-1)}\right)\lambda$.*

Proof. Assume for contradiction sake that an early time t existed with $L^{t-1}_{\text{crit}} > \left(1 - \frac{1}{2(c-1)}\right)\lambda$. We may wlog. choose t maximal with that property. Then there holds $L^t_{\text{crit}} \leq \left(1 - \frac{1}{2(c-1)}\right)\lambda$ either by the maximality of t or by Lemma 7. Thus job J_t caused L_{crit} to decrease. This can only happen if job J_t was assigned to a machine in $\mathcal{M}^{t-1}_{\text{crit}}$ that was not M^{t-1}_{small}. But by the previous claim this does not happen, a contradiction.

For the second part observe that job J_t is either scheduled on a machine having load λ or on a machine of load at most $L^{t-1}_{\text{crit}} \leq \left(1 - \frac{1}{2(c-1)}\right)\lambda$, which is either the least loaded machine in $\mathcal{M}_{\text{crit}}$ or a machine of lesser load. $\square$

We now conclude our analysis by proving the structural Lemma 5.

Proof of Lemma 5. Given any machine M, we show that job J_t that caused M to first reach (robust) load λ had processing time at least $\frac{\lambda}{2(c-1)}$. If job J_t was late, this already follows from Lemma 6. Else, by Lemma 8, job J_t was scheduled on a machine which had (robust) load $\left(1 - \frac{1}{2(c-1)}\right)\lambda$ before and λ after receiving job J_t. Thus job J_t had (robust) processing time at least $\frac{\lambda}{2(c-1)}$. $\square$

4.1 Deterministic Lower Bounds

We can show that no general competitive ratio below 2 is possible unless very small numbers of machines are considered. We leave the proof to the full version.

Theorem 3. *No deterministic algorithm is better than 2-competitive for general m and $\Gamma = 2$.*

It may also be interesting to consider the debugging model, where all jobs have real processing time 0. In this case the classical lower bounds still apply if Γ is not chosen too small. For example, the lower bound of 1.852 for [1] holds for $\Gamma \geq 4$. On the other hand it is not clear that any algorithm performs better in tise case. Lemma 3 shows that Greedy does not.

References

1. Albers, S.: Better bounds for online scheduling. SIAM Journal on Computing **29**(2), 459–473 (1999), publisher: SIAM
2. Albers, S., Janke, M.: Scheduling in the Random-Order Model. In: 47th International Colloquium on Automata, Languages, and Programming (ICALP 2020). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2020)
3. Aloulou, M.A., Della Croce, F.: Complexity of single machine scheduling problems under scenario-based uncertainty. Operations Research Letters **36**(3), 338–342 (2008), publisher: Elsevier
4. Bartal, Y., Fiat, A., Karloff, H., Vohra, R.: New algorithms for an ancient scheduling problem. In: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing. pp. 51–58 (1992)
5. Bartal, Y., Karloff, H.J., Rabani, Y.: A better lower bound for on-line scheduling. Inf. Process. Lett. **50**(3), 113–116 (1994)
6. Bertsimas, D., Sim, M.: Robust discrete optimization and network flows. Mathematical programming **98**(1-3), 49–71 (2003), publisher: Springer
7. Bertsimas, D., Sim, M.: The price of robustness. Operations research **52**(1), 35–53 (2004), publisher: Informs
8. Bougeret, M., Jansen, K., Poss, M., Rohwedder, L.: Approximation results for makespan minimization with budgeted uncertainty. In: International Workshop on Approximation and Online Algorithms. pp. 60–71. Springer (2019)
9. Bougeret, M., Pessoa, A.A., Poss, M.: Robust scheduling with budgeted uncertainty. Discrete Applied Mathematics **261**, 93–107 (2019)
10. Chen, L., Ye, D., Zhang, G.: Approximating the optimal algorithm for online scheduling problems via dynamic programming. Asia-Pacific Journal of Operational Research **32**(01), 1540011 (2015), publisher: World Scientific
11. Cheng, T.E., Kellerer, H., Kotov, V.: Semi-on-line multiprocessor scheduling with given total processing time. Theoretical computer science **337**(1-3), 134–146 (2005), publisher: Elsevier
12. Cohen, I., Im, S., Panigrahi, D.: Online Two-Dimensional Load Balancing. In: 47th International Colloquium on Automata, Languages, and Programming (ICALP 2020). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2020)
13. Dantzig, G.B.: Linear programming under uncertainty. Management science **1**(3-4), 197–206 (1955), publisher: INFORMS
14. Dürr, C., Erlebach, T., Megow, N., Meißner, J.: Scheduling with explorable uncertainty. In: 9th Innovations in Theoretical Computer Science Conference (ITCS 2018). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2018)
15. Dürr, C., Erlebach, T., Megow, N., Meißner, J.: An Adversarial Model for Scheduling with Testing. Algorithmica pp. 1–46 (2020), publisher: Springer
16. Englert, M., Özmen, D., Westermann, M.: The power of reordering for online minimum makespan scheduling. In: 2008 49th Annual IEEE Symposium on Foundations of Computer Science. pp. 603–612. IEEE (2008)
17. Erlebach, T., Hoffmann, M., Kammer, F.: Query-competitive algorithms for cheapest set problems under uncertainty. Theoretical Computer Science **613**, 51–64 (2016), publisher: Elsevier
18. Faigle, U., Kern, W., Turán, G.: On the performance of on-line algorithms for partition problems. Acta cybernetica **9**(2), 107–119 (1989)
19. Feder, T., Motwani, R., Panigrahy, R., Olston, C., Widom, J.: Computing the median with uncertainty. In: Proceedings of the thirty-second annual ACM symposium on Theory of computing. pp. 602–607 (2000)

20. Fleischer, R., Wahl, M.: On-line scheduling revisited. Journal of Scheduling **3**(6), 343–353 (2000), publisher: Wiley Online Library
21. Galambos, G., Woeginger, G.J.: An on-line scheduling heuristic with better worst-case ratio than Grahams list scheduling. SIAM Journal on Computing **22**(2), 349–355 (1993), publisher: SIAM
22. Graham, R.L.: Bounds for certain multiprocessing anomalies. Bell System Technical Journal **45**(9), 1563–1581 (1966), publisher: Wiley Online Library
23. Gupta, M., Sabharwal, Y., Sen, S.: The update complexity of selection and related problems. arXiv preprint arXiv:1108.5525 (2011)
24. Hochbaum, D.S., Shmoys, D.B.: Using dual approximation algorithms for scheduling problems theoretical and practical results. Journal of the ACM (JACM) **34**(1), 144–162 (1987), publisher: ACM New York, NY, USA
25. Im, S., Kell, N., Kulkarni, J., Panigrahi, D.: Tight bounds for online vector scheduling. In: 2015 IEEE 56th Annual Symposium on Foundations of Computer Science. pp. 525–544. IEEE (2015)
26. Kahan, S.: A model for data in motion. In: Proceedings of the twenty-third annual ACM symposium on Theory of computing. pp. 265–277 (1991)
27. Karger, D.R., Phillips, S.J., Torng, E.: A better algorithm for an ancient scheduling problem. Journal of Algorithms **20**(2), 400–430 (1996), publisher: Elsevier
28. Kasperski, A., Kurpisz, A., Zieliński, P.: Approximating a two-machine flow shop scheduling under discrete scenario uncertainty. European Journal of Operational Research **217**(1), 36–43 (2012), publisher: Elsevier
29. Kasperski, A., Kurpisz, A., Zieliński, P.: Parallel machine scheduling under uncertainty. In: International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems. pp. 74–83. Springer (2012)
30. Khanna, S., Tan, W.C.: On computing functions with uncertainty. In: Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. pp. 171–182 (2001)
31. Lenstra, J.K., Shmoys, D.B., Tardos, E.: Approximation algorithms for scheduling unrelated parallel machines. Mathematical programming **46**(1-3), 259–271 (1990), publisher: Springer
32. Mastrolilli, M., Mutsanas, N., Svensson, O.: Approximating single machine scheduling with scenarios. In: Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques, pp. 153–164. Springer (2008)
33. Olston, C., Widom, J.: Offering a precision-performance tradeoff for aggregation queries over replicated data. Tech. rep., Stanford (2000)
34. Roy, A.B., Bougeret, M., Goldberg, N., Poss, M.: Approximating robust bin packing with budgeted uncertainty. In: Workshop on Algorithms and Data Structures. pp. 71–84. Springer (2019)
35. Rudin, J.F.: Improved bounds for the on-line scheduling problem (2001)
36. Sanders, P., Sivadasan, N., Skutella, M.: Online scheduling with bounded migration. Mathematics of Operations Research **34**(2), 481–498 (2009), publisher: INFORMS
37. Singla, S.: The price of information in combinatorial optimization. In: Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 2523–2532. SIAM (2018)
38. Tadayon, B., Smith, J.C.: Algorithms and complexity analysis for robust single-machine scheduling problems. Journal of Scheduling **18**(6), 575–592 (2015), publisher: Springer

A Proof of Lemma 7

The behavior of our algorithm and the statement of the lemma does not change if we scale all jobs by the same constant factor. We will thus, for ease of notation, scale all jobs such that $\lambda = 1$. Let us now assume for contradiction sake that the statement of the lemma was false, i.e. that we had

$$L^s_{\text{crit}} > 1 - \frac{1}{2(c-1)}. \tag{2}$$

Given a time $t \geq s$, let $m(t)$ denote the number of machines that have yet to reach load $\lambda = 1$ after job J_t was scheduled. Recall that $m(s) \leq d$ since by definition no job that caused the machine M_{med} to reach load $\lambda = 1$ arrived after time s. Moreover, $m(n) = m(n-1) = 0$, because all machines reached load $\lambda = 1$ before job J_n is scheduled. Given a time $t \geq s$, consider the set $\bar{\mathcal{M}}^t$ of the smallest $m(t)$ machines whose load reached at least λ. We break ties the same way we did before in the definition of the algorithm. In particular, none of the large machines in $\mathcal{M}^t_{\text{large}}$ lies in $\bar{\mathcal{M}}^t$ since by assumption the $d \geq |\bar{\mathcal{M}}^t|$ smaller machines in $\mathcal{M}^t_{\text{med}}$ have also reached load λ. In other words $\bar{\mathcal{M}}^t \subset \mathcal{M}_{\text{small}} \cup \mathcal{M}_{\text{med}}$. Let $\bar{L}^t$ be the average robust load of all machines if the load of the machines in $\bar{\mathcal{M}}_t$ is reduced to $\lambda = 1$, that is $\bar{L}^t = \frac{1}{m}\left(\sum_{M \in \mathcal{M} \setminus \bar{\mathcal{M}}^t} l^t_M + m(t)\right)$. Note that $\bar{L}^t \leq L^t$ with equality for $t \geq n-1$. We now are going to prove the following inequality via induction for all $t \geq s$.

$$\bar{L}^t > \left(1 - \frac{d}{2(c-1)m} - 2\frac{\Gamma+1}{c\Gamma}\right)\left(1 + \frac{c}{2m}\right)^{d-m(t)} + 2\frac{\Gamma+1}{c\Gamma}. \tag{3}$$

Note that this inequality is the motivation for our choice of c. Namely, using Inequality (1), we see that $L^{n-1} = \bar{L}^{n-1} > \frac{2}{c-1} \cdot \frac{m-1}{m}$.

Claim (Base case). Inequality (3) holds for $t = s$.

Let us treat the case $m(s) = d$. One observes that the $m(s) = d$ least loaded machines have average load at least $L_{\text{crit}} > 1 - \frac{1}{2(c-1)}$ while all other machines have load at least $\lambda = 1$. Thus

$$\begin{aligned}\bar{L}^t &> \frac{1}{m}\left((m-d) + d\left(1 - \frac{1}{2(c-1)}\right)\right) \\ &= \left(1 - \frac{d}{2(c-1)m} - 2\frac{\Gamma+1}{c\Gamma}\right)\left(1 + \frac{c}{2m}\right)^0 + 2\frac{\Gamma+1}{c\Gamma}.\end{aligned}$$

For smaller values $m(s)$, one uses the fact that the schedule is steep right before job J_s is scheduled. Using the properties of steepness one then can compute a lower bound that exceeds the right side of Inequality (3). Since this is merely computational we leave it to Appendix C.

Claim (Inductive step). Let $s < t$. If Inequality (3) holds for $t-1$, then it holds for t.

Proof. Indeed, consider job J_t. If J_t does not cause the load of a machine to exceed $\lambda = 1$, then this may only increase the left side of Inequality (3) and the statement of the lemma is obvious.

Else, $m(t) = m(t-1) - 1$. Let $l \geq 1$ be the load of M_{med}^{t-1} and p_t the (robust) processing time of J_t. Then $\bar{L}^t$ will increase by at least $\frac{l-1+p^t-\Gamma^{-1}}{m}$. Indeed, since the size of $\bar{\mathcal{M}}_t$ decreases by 1 a machine whose load before only counted as $\lambda = 1$ toward $\bar{L}$ will now count fully. Since its load will be at least l this causes an increase of $\frac{l-1}{m}$. Let us provisionally assume that job J_t fails machine M_{small} and that all jobs which do so, continue doing so. Then the load M_{small} will 'provisionally' increase by p_t. Of course this provisional increase may assume that $\Gamma+1$ jobs fail machine M_{small}. If this is the case, we have to subtract Δp the smallest additional processing time of these $\Gamma+1$ failing jobs. At least Γ jobs of additional size Δp already failed M_{small}^{t-1}. Thus $\Gamma \cdot \Delta p \leq l_{\text{small}}^{t-1} < 1$. We therefore have to subtract at most $\Delta p < \Gamma^{-1}$ for a job that ceases failing machine M_{small}^{t-1} or for job J_t not doing so. This justifies an increase in $\bar{L}$ by $\frac{p_t-\Gamma^{-1}}{m}$. Note that it does not matter if machine M_{small}^{t-1} enters $\bar{\mathcal{M}}_t$ after receiving job J_t since this will cause another machine of at least its load to leave $\bar{\mathcal{M}}_t$ in its stead. In total, we have shown that

$$\bar{L}^t \geq \bar{L}^{t-1} + \frac{1}{m}\left(l + p^t - 1 - \Gamma^{-1}\right). \tag{4}$$

Now let us consider the job J_t. Recall that since $t > s$ this job was scheduled flatly and M_{med}^{t-1} already had load $\lambda = 1$. Since it caused a machine to first reach load $\lambda = 1$ it was thus scheduled on a least loaded machine although the current schedule was flat. By definition this implies that $l + p^t > \frac{c}{2}L^{t-1} \geq \frac{c}{2}\bar{L}^{t-1}$. Combining this with Inequality (4) leads to

$$\bar{L}^t > \bar{L}^{t-1} + \frac{1}{m}\left(\frac{c}{2}\bar{L}^{t-1} - 1 - \Gamma^{-1}\right) = \left(1 + \frac{c}{2m}\right)\bar{L}^{t-1} - \frac{\Gamma+1}{m\Gamma}.$$

The claim follows by plugging the induction hypothesis for $\bar{L}^{t-1}$ into the previous term and using that by assumption $m(t) + 1 = m(t-1)$. □

The previous two claims show, using induction, that Inequality (3) holds for all $t > s$. In particular it holds for $t = n - 1$. Now combining Inequality (1) and Inequality (3) we get that

$$\bar{L}^{n-1} > \left(1 - \frac{d}{2(c-1)m} - 2\frac{\Gamma+1}{c\Gamma}\right)\left(1 + \frac{c}{2m}\right)^{d-0} + 2\frac{\Gamma+1}{c\Gamma} \geq \frac{2}{c-1}\frac{m-1}{m}.$$

But the definition of critically flat schedules in Remark 1 implies the opposite inequality:

$$\bar{L}^{n-1} = L^{n-1} < \frac{2}{c-1}\frac{m-1}{m}\lambda = \frac{2}{c-1}\frac{m-1}{m}.$$

This is a contradiction to the assumption that Lemma 7 was not true.

B Lower Bounds

Proof of Lemma 3. Let $\Gamma = m - 1 + N$ for some $N \gg 0$. Consider the following sequence in the debugging model where $p = 0$. First, $N \cdot m$ tiny jobs with processing vector $(p, \Delta p) = (0, \Gamma^{-1})$ arrive. Then, $m(m-2)$ small jobs with processing vector $(p, \Delta p) = (0, \frac{1}{m})$ follow. Finally, one large job with processing vector $(p, \Delta p) = (0, 1)$ has to be scheduled. The greedy strategy will first assign precisely N tiny jobs to each machine, then $m-2$ small jobs per machine and finally the large job will arrive. The machine containing the large job then contains precisely $N + m - 2 + 1 = \Gamma$ jobs. All additional processing times count! The robust load is $N \cdot \Gamma^{-1} + (m-2) \cdot \frac{1}{m} + 1 = 3 - \frac{2}{m} - \frac{m-1}{N} \xrightarrow{N \to \infty} 3 - \frac{2}{m}$.

We are left to show that OPT can schedule these jobs having makespan 1. Indeed, the first machine receives all tiny jobs. Its robust load is $\Gamma \cdot \Gamma^{-1} = 1$ since at most Γ jobs count towards the load. The next $m-2$ machines receive m small jobs of processing time $\frac{1}{m}$ each; their load is $m \cdot \frac{1}{m} = 1$. The last remaining machine receives the single large job and thus also has load 1. □

Proof of Theorem 3. Let $m \geq 9$ and $\Gamma = 2$. Consider a sequence of m *debugging jobs* $(p, \Delta p) = (0, 1)$ followed by $2(m-1)$ *real jobs* with processing vector $(p, \Delta p) = (1, 0)$ and 3 *final jobs* of processing vector $(p, \Delta p) = (3, 0)$. Consider the schedule of any (deterministic) online algorithm A after the debugging jobs are scheduled. Since so far OPT $= 1$ all debugging jobs need to be on a separate machine, else A would already cease to be 2-competitive on the prefix consisting only of debugging jobs. Now, consider the prefix consisting of all the debugging jobs and the $2(m-1)$ *real jobs*. Here, OPT $= 2$ (schedule all debugging jobs on one machine, every other machine gets two real jobs). If A is better than 2-competitive, its makespan must stay below 4 and thus it can only place 2 real jobs (in addition to the one debugging job) on any machine. By a simple counting argument at most 2 machines can have load strictly below 3 afterwards. In particular, one of the 3 final jobs needs to be placed on a machine of load 3 and thus A has a makespan of 6.

We are left to show that OPT can schedule the whole sequence having a makespan of 3. It first places all debugging jobs on one machine. Then it places all the final jobs on 3 different machines. Now, it can still schedule $3m - 11$ real jobs while maintaining a makespan of 3. In particular it can schedule $2(m-1)$ real jobs for $m \geq 9$. □

C Missing Proofs and Computations

Proof of Remark 1. Let A be our algorithm then $L^{n-1} = L[A, J_1, \dots J_{n-1}]$ and set $L^n = L[A, \mathcal{J}]$. Let $\hat{p}_n$ be the processing time by which job J_n changes the total (robust) load of the machine $M = M^{n-1}_{\text{small}}$ it is scheduled on. This depends on whether job J_n does fail this machine, and if it does, whether it causes another job not to fail M anymore. In general $\hat{p}_n$ could take any values in between $\tilde{p}_n$ and p_n. Using this, we see that $L^{n-1} = L^n - \hat{p}_n/m$ and the load of machine M after receiving job J_n is $\lambda + \hat{p}_n$.

Assume that $\lambda \leq \frac{c-1}{2}\frac{m}{m-1}L^{n-1}$. We have to show that $\lambda + \hat{p}_n \leq c\mathrm{OPT}$.

Using Lemma 1, we see that for our online algorithm A we have $L^n = L[A, \mathcal{J}] \leq L[\mathrm{OPT}, \mathcal{J}] + \left(1 - \frac{1}{m}\right)\mathrm{OPT}[\mathcal{J}] \leq \left(2 - \frac{1}{m}\right)\mathrm{OPT}$. In particular $L^{n-1} \leq \frac{2m-1}{m}\mathrm{OPT} - \frac{\hat{p}_n}{m}$. Using this, wee see that

$$\lambda + \hat{p}_n \leq \frac{c-1}{2}\frac{m}{m-1}L^{n-1} + \hat{p}_n \leq \frac{c-1}{2}\frac{m}{m-1}\left(\frac{2m-1}{m}\mathrm{OPT} - \frac{\hat{p}_n}{m}\right) + \hat{p}_n.$$

Recall that $\hat{p}_n \leq p_n \leq \mathrm{OPT}$. Using that $\frac{c-1}{2}\frac{m}{m-1}\frac{1}{m} < 1$ for $m \geq 2$ the previous term is increasing in $\hat{p}_n$ and thus maximized if we set $\hat{p}_n = \mathrm{OPT}$. Thus

$$\lambda + \hat{p}_n \leq \frac{c-1}{2}\frac{m}{m-1}\frac{2m-2}{m}\mathrm{OPT} + \mathrm{OPT} = c\mathrm{OPT}. \qquad \square$$

Lemma 9. *For $m, \Gamma \to \infty$ the competitive ratio c approaches a value slightly below* 2.9052*.*

Proof. The value c will be the solution of the following equation, assuming that such a solution with $c \geq \frac{7+\sqrt{17}}{4}$ exists.

$$\left(1 - \frac{d/m}{2(c-1)} - 2\frac{\Gamma+1}{c\Gamma}\right)\left(1 + \frac{c}{2m}\right)^d + 2\frac{\Gamma+1}{c\Gamma} = \frac{2}{c-1} \cdot \frac{m-1}{m}.$$

For $m \to \infty$, we can replace $\frac{d}{m}$ by $\frac{c-2}{c}$. We replace $\left(1 + \frac{c}{2m}\right)^d$ by $e^{\frac{cd}{2m}}$ and then by $e^{\frac{c-2}{2}}$. The term $\frac{m-1}{m}$ simply approaches 1. For $\Gamma \to \infty$, $2\frac{\Gamma+1}{c\Gamma}$ becomes $\frac{2}{c}$. Using this the previous equation simplifies for $m, \Gamma \to \infty$ to

$$\left(1 - \frac{c-2}{2c(c-1)} - \frac{2}{c}\right)e^{\frac{c-2}{2}} + \frac{2}{c} = \frac{2}{c-1}.$$

This term does not have a closed-form solution, but we can numerically approach its solution to $c \approx 2.905186$ where the last digit is rounded up. $\square$

Missing part of the proof of the first claim in Appendix A. Observe that at time s the $m(s)$ smallest machines had average load L^s_{crit} while the next smallest $d - m(s)$ loads are at least 1. Thus

$$L_{\mathrm{small}} \geq \frac{m(s)L^s_{\mathrm{crit}} + (d - m(s))}{d} \geq 1 - \frac{m(s)}{2(c-1)d}.$$

The second inequality follows from Equation (2). If the schedule was steep after job J_s was scheduled we had

$$L_{\mathrm{large}} \geq \left(1 - \frac{1}{2(c-1)}\right)^{-1} L_{\mathrm{small}} \geq \frac{2(c-1)}{2c-3}\left(1 - \frac{m(s)}{2(c-1)d}\right).$$

If the schedule was not steep job J_s caused M_{med} to become full. Thus $m(s) = d$ and the same inequality follows trivially via

$$L_{\mathrm{large}} \geq \lambda = 1 = \left(1 - \frac{1}{2(c-1)}\right)^{-1}\left(1 - \frac{1}{2(c-1)}\right) = \frac{2(c-1)}{2c-3}\left(1 - \frac{m(s)}{2(c-1)d}\right).$$

We already in Appendix A that $\bar{\mathcal{M}}^t \cap \mathcal{M}^t_{\text{large}} = \emptyset$, thus we obtain:

$$\begin{aligned}\bar{L}^s &> \frac{(m-2d)L^s_{\text{large}} + (2d - m(s)) + m(s)L^s_{\text{crit}}}{m} \\ &\geq \frac{m-2d}{m} \cdot \frac{2(c-1)}{2c-3}\left(1 - \frac{m(s)}{2(c-1)d}\right) + \frac{2d-m(s)}{m} + \frac{m(s)}{m} \cdot \left(1 - \frac{1}{2(c-1)}\right).\end{aligned}$$

Let $f(m(s))$ denote the term on the right side of the previous inequality and let $g(m(s)) = \left(1 - \frac{d}{2(c-1)m} - \frac{\Gamma+1}{c\Gamma}\right)\left(1 + \frac{c}{2m}\right)^{d-m(s)} + \frac{\Gamma+1}{c\Gamma}$ be the right side of Inequality 3. Then we are left to show the following analytic fact: $f(x) \geq g(x)$ for all possible choices $x = m(s) = 0, \ldots, d$. In fact, the definition of the functions f and g canonically extends to the real numbers, so we will prove $f(x) \geq g(x)$ for all $0 \leq x \leq d$. When defined on the reals the function $f(\cdot)$ is convex, while $g(\cdot)$ is linear. Using convexity it therefore suffices to prove $f(0) \geq g(0)$ and $f(d) \geq g(d)$. One can readily compute that in the latter case $f(d) = g(d) = 1 - \frac{d}{2(c-1)m}$. By the definition of c, as mentioned right before the statement of the lemma $g(0) = \frac{1}{c-1} \cdot \frac{m+1}{m-1} < \frac{1}{c-1}$ while direct computation yields that $f(0) > \frac{1}{c-1}$ for $c > 2.52$. □

Appendix E

Machine Covering in the Secretary Model

Bibliographic Information *Machine Covering in the Random-Order Model.* S. Albers, W. Gálvez, M. Janke. To appear in: The 32nd International Symposium on Algorithms and Computation (**ISAAC**) 2021

Summary of Contributions We study Online Machine Covering in the secretary model. Parallel and identical machines process jobs. Each job, defined by its processing time, runs on precisely one machine. In particular, preemption is not allowed. The goal is to maximize the minimum load of a machine or, equivalently, the time all machines are busy.

In the secretary model, jobs are presented to the online algorithm in a uniformly random order instead of a worst-case order. Moreover, input size n is known in advance. We first analyze the Greedy strategy whose competitive ratio improves to $\Theta(\frac{m}{\log(m)})$ under random-order arrival. This bound is tight up to a factor of $2 + o(1)$. We then present a more sophisticated algorithm, which is $\tilde{O}(\sqrt[4]{m})$-competitive and outperforms all lower bounds on worst-case orders. These results are complemented by a first lower bound of $\tilde{\Omega}(\log(m))$. This lower bound follows from a novel variant of the Secretary Problem, called the Talent Contest Problem, which might be of independent research interest.

Individual Contributions

- Development of the bound for the Greedy strategy.
- Development and analysis of the improved algorithm using sampling to enhance the scheme of Azar and Epstein [33].
- Development of the lower bound and analysis of the Talent Contest Problem.
- Composition of some parts of the manuscript, including graphics, graphs, technical, non-technical parts.

Machine Covering in the Random-Order Model

Susanne Albers
Department of Computer Science, Technical University of Munich, Germany

Waldo Gálvez ORCID
Institute of Engineering Sciences, Universidad de O'Higgins, Chile

Maximilian Janke
Department of Computer Science, Technical University of Munich, Germany

Abstract

In the Online Machine Covering problem jobs, defined by their sizes, arrive one by one and have to be assigned to m parallel and identical machines, with the goal of maximizing the load of the least-loaded machine. Unfortunately, the classical model allows only fairly pessimistic performance guarantees: The best possible deterministic ratio of m is achieved by the Greedy-strategy, and the best known randomized algorithm has competitive ratio $\tilde{O}(\sqrt{m})$ which cannot be improved by more than a logarithmic factor.

Modern results try to mitigate this by studying semi-online models, where additional information about the job sequence is revealed in advance or extra resources are provided to the online algorithm. In this work we study the Machine Covering problem in the recently popular *random-order* model. Here no extra resources are present, but instead the adversary is weakened in that it can only decide upon the input set while jobs are revealed uniformly at random. It is particularly relevant to Machine Covering where lower bounds are usually associated to highly structured input sequences.

We first analyze Graham's Greedy-strategy in this context and establish that its competitive ratio decreases slightly to $\Theta\left(\frac{m}{\log(m)}\right)$ which is asymptotically tight. Then, as our main result, we present an improved $\tilde{O}(\sqrt[4]{m})$-competitive algorithm for the problem. This result is achieved by exploiting the extra information coming from the random order of the jobs, using sampling techniques to devise an improved mechanism to distinguish jobs that are relatively large from small ones. We complement this result with a first lower bound showing that no algorithm can have a competitive ratio of $O\left(\frac{\log(m)}{\log\log(m)}\right)$ in the random-order model. This lower bound is achieved by studying a novel variant of the Secretary problem, which could be of independent interest.

2012 ACM Subject Classification Theory of computation → Online algorithms; Theory of computation → Scheduling algorithms

Keywords and phrases Machine Covering, Online Algorithm, Random-Order, Competitive Analysis, Scheduling

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2021.57

Funding Work supported by the European Research Council, Grant Agreement No. 691672, project APEG.

Waldo Gálvez: This project was carried out when the author was a postdoctoral researcher at the Department of Computer Science of the Technical University of Munich in Germany.

1 Introduction

We study the *Machine Covering* problem, a fundamental load balancing problem where n jobs have to be assigned (or scheduled) onto m identical parallel machines. Each job is characterized by a non-negative size, and the goal is to maximize the smallest machine load. This setting is motivated by applications where machines consume resources in order to work, and the goal is to keep the whole system active for as long as possible. Machine Covering has found additional applications in the sequencing of maintenance actions for

aircraft engines [20] and in the design of robust Storage Area Networks [41]. The offline problem, also known as Santa-Claus or Max-Min Allocation Problem, received quite some research interest, see [44, 10, 6] and references therein. In particular, the problem is known to be strongly NP-hard but to allow for a Polynomial-Time Approximation Scheme (PTAS) [44].

This paper focuses on the online version of the problem, where jobs arrive one by one and must be assigned to some machine upon arrival. Lack of knowledge about future jobs can enforce very bad decisions in terms of the quality of the constructed solutions: In a classical lower bound sequence, m jobs of size 1 arrive first to the system and they must be assigned to m different machines by a competitive deterministic online algorithm. Then subsequent $m-1$ jobs of size m arrive, which make the online algorithm perform poorly, see Figure 1. Indeed, the best possible deterministic algorithm achieves a competitive ratio of m [44], and if randomization is allowed, the best known competitive ratio is $\tilde{O}(\sqrt{m})$, which is best possible up to logarithmic factors [7]. The corresponding lower bound also uses that the online algorithm cannot schedule the first m jobs correctly, at least not with probability exceeding $\frac{1}{\sqrt{m}}$.

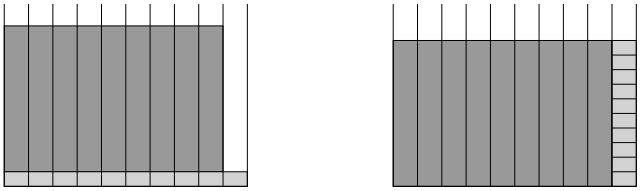


Figure 1 The instance showing that no deterministic algorithm is better than m-competitive for Machine Covering. To the left, the best possible solution that an online algorithm can construct achieves minimum load 1. To the right, the optimal minimum load is m.

Such restrictive facts have motivated the study of different semi-online models that provide extra information [7, 14, 34, 37] or extra features [41, 42, 22, 16] to the online algorithm.

This work studies the Online Machine Covering problem in the increasingly popular *random-order* model. In this model, jobs are still chosen worst possible by the adversary but they are presented to the online algorithm in a uniformly random order. The random-order model derives from the Secretary Problem [13, 35] and has been applied to a wide variety of problems such as generalized Secretary problems [32, 8, 33, 18, 9], Scheduling problems [39, 38, 2, 3], Packing problems [30, 31, 5, 4], Facility Location problems [36] and Convex Optimization problems [25] among others. See also [26] for a survey chapter. It is particularly relevant to Machine Covering, where hard instances force online algorithms to make an irredeemable mistake right on the first m jobs due to some hidden large job class at the end.

Indeed, we show that the competitive ratio of Graham's Greedy-strategy improves from m to $O\left(\frac{m}{\log(m)}\right)$, and that this is asymptotically tight. We also develop an $\tilde{O}(\sqrt[4]{m})$-competitive algorithm, providing evidence that known hardness results rely on "pathological" inputs, and complement it by proving that no algorithm can be $O\left(\frac{\log(m)}{\log\log(m)}\right)$-competitive in this model.

1.1 Related Results

The most classical Scheduling problem is *Makespan Minimization* on parallel and identical machines. Here, the goal is dual to Machine Covering; one wants to minimize the maximum load among the machines. This problem is strongly NP-hard and there exists a PTAS [27]. The online setting received considerable research attention, and already in 1966 Graham

showed that his famous Greedy-strategy is $(2-1/m)$-competitive. A long line of research [21, 11, 29, 1, 19] starting in the 1990s lead to the currently best competitive ratio of 1.9201 due to Fleischer and Wahl [19]. Regarding lower bounds, again after a sequence of results [17, 12, 24] the current best one is 1.88 [40].

The landscape for Online Machine Covering differs considerably from Online Makespan Minimization as discussed before, which has motivated the study of semi-online models to deal with the implied hard restrictions. If the value of the optimal minimum load of the instance is known in advance, Azar et al. have shown that a simple greedy algorithm already is $(2-1/m)$-competitive and that no algorithm can attain a competitive ratio better than 7/4 [7]. These bounds were improved by Ebenlendr et al. to 11/6 and 1.791 respectively [14]. In the *bounded migration* model, whenever a job of size p arrives, older jobs of total size at most $\beta \cdot p$ can be reassigned to different machines. Sanders et al. [41] provide a 2-competitive algorithm for $\beta = 1$. Later results [42, 22] study the interplay between improved competitive ratios and larger values of β. Another semi-online model provides the online algorithm with a *reordering buffer*, which is used to rearrange the input sequence "on the fly". Epstein et al. [16] provide a $(H_{m-1}+1)$-competitive algorithm using a buffer of size $m-1$, and show that this ratio cannot be improved for any sensible buffer size. These and many more semi-online models have also been studied for Makespan Minimization, see the survey in [15] and references therein.

The first Scheduling result in the random-order model is due to Osborn and Torng [39]. They establish that the Greedy-strategy for Makespan Minimization does not achieve a competitive ratio better than 2 for general m. Recently, [2, 3] show that for Makespan Minimization the random-order model allows for better performance guarantees than the classical model. Molinaro [38] has studied the Online Load Balancing problem with the objective to minimize general l_p-norms of the machine loads, providing an algorithm that returns solutions of expected norm $(1+\varepsilon)\mathrm{OPT} + O\left(\frac{p}{\varepsilon}(m^{1/p}-1)\right)$ in the random-order model, where OPT denotes the optimal norm. Göbel et al. [23] have studied Average Weighted Completion Time Minimization on one machine in the random-order model. Their competitive ratio is logarithmic in the input length n for general job sizes and constant if all jobs have size 1. To the best of our knowledge, no previous result is known for Online Machine Covering in the random-order model.

1.2 Our Contribution

We first establish that the Greedy-strategy is $\Theta\left(\frac{m}{\log(m)}\right)$-competitive in the random-order model. This is only a tiny, albeit significant improvement compared to worst-case orders. Since the bound is tight, more refined strategies that make particular use of the characteristics of the random-order model are required. The analysis also gives first intuitions about what these characteristics are and also about the techniques used to analyze the main algorithm.

The following theorem summarizes the central result of this paper.

▶ **Theorem 1.** *There exists a $\tilde{O}(\sqrt[4]{m})$-competitive algorithm for the online Machine Covering problem in the random-order model.*

In the classical online Machine Covering problem, difficult instances are usually related to the inability of distinguishing "small" and "large" jobs induced by a lack of knowledge about large job classes hidden at the end of the sequence. Figure 1 depicts the easiest example on which deterministic schedulers cannot perform well as they cannot know that the first m jobs are tiny. Azar and Epstein [7] ameliorate this by maintaining a randomized threshold, which is used to distinguish small and large job sizes. They have to correctly classify up to m large

jobs with constant probability while controlling the total size of incorrectly classified small jobs, which leads to their randomized competitive ratio of $\tilde{O}(\sqrt{m})$. However, their lower bound shows that general randomized algorithms are still unable to schedule the first m jobs correctly with probability exceeding $\frac{1}{\sqrt{m}}$, again due to relevant job classes being hidden at the end of the input.

Random-order arrival makes such hiding impossible. This already helps the Greedy-strategy as now large jobs in the input are evenly distributed instead of being clustered at the end. Our $\tilde{O}(\sqrt[4]{m})$-competitive algorithm enhances the path described previously by making explicit use of the no-hiding-feature; it determines those large jobs the adversary would have liked to hide. Information about large job sizes is, as is common in Secretary problems, estimated in a sampling phase, which returns a threshold distinguishing all except for $\sqrt{m}$ of the large jobs. This reduction by a square root carries over to the competitive ratio: We now can allow to misclassify these remaining $\sqrt{m}$ jobs with a higher probability, which in turn leads to a better classification of small jobs and a better competitive ratio of $\tilde{O}(\sqrt[4]{m})$.

We complement the upper bound of $\tilde{O}(\sqrt[4]{m})$ with a lower bound of $\omega\left(\frac{\log(m)}{\log\log(m)}\right)$ for the competitive ratio in the random-order model. Lower bounds in the random-order model are usually considered hard to devise since one cannot hide larger pieces of input. Instead of hiding large job classes, we figuratively make them hard to distinguish by adding noise. To this end, we study a novel variant of the Secretary problem, the *Talent Contest* problem, where the goal is to find a good but not too good candidate (or secretary).

More in detail, we want to pick the K-th best among a randomly permuted input set of candidates. Unlike classical Secretary problems (or the more general Postdoc problem [43]), we may pick several candidates as long as they are not better than the K-th candidate. Furthermore, we interview candidates t times and make a decision at each arrival. In this setting, information gained by earlier interviews helps the decisions required in later ones. It can be proven that the expected number of times the desired candidate can be correctly identified relates to the ability of distinguishing exactly the $m-1$ largest jobs from a Machine Covering instance in the random-order model, and hence bounding the aforementioned expected value allows us to obtain the desired hardness result.

1.3 Organization of the paper

In Section 2 we provide the required definitions and tools, and then in Section 3 analyze the Greedy-algorithm in the context of random-order arrival. In Section 4 we present our main algorithm and its analysis. Section 5 then introduces the Talent Contest Problem and concludes with a lower bound for the best competitive ratio in the random-order model. Due to space constraints, some proofs from Section 5 are deferred to the full version of the paper.

2 Preliminaries

In this section we introduce the main definitions and tools that are used along this work.

In the Machine Covering problem, we are given n jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$, specified by their non-negative sizes p_i, which are to be assigned onto m parallel and identical machines. The *load* l_M of a machine M is the sum of the sizes of all the jobs assigned to it. The goal is to maximize the minimum load among the machines, i.e. to maximize $\min_M l_M$.

To an online algorithm, jobs are revealed one-by-one and each has to be assigned permanently and irrevocably before the next one is revealed. Formally, given the symmetric

group S_n on n elements, each permutation $\sigma \in S_n$ defines the order in which the elements of $\mathcal{J}$ are revealed, namely $\mathcal{J}^\sigma = (J_{\sigma(1)}, \ldots, J_{\sigma(n)})$. Classically, the performance of an online algorithm A is measured in terms of competitive analysis. That is, if we denote the minimum machine load of A[1] on $\mathcal{J}^\sigma$ by $A(\mathcal{J}^\sigma)$ and the minimum machine load an optimal offline algorithm may achieve by $\mathrm{OPT}(\mathcal{J})$ (which is independent on the order σ), one is interested in finding a small *(adversarial) competitive ratio* $c = \sup_{\mathcal{J}} \sup_{\sigma \in S_n} \frac{\mathrm{OPT}(\mathcal{J})}{A(\mathcal{J}^\sigma)}$[2].

In the random-order model, the job order is chosen uniformly at random. We consider the permutation group S_n as a probability space under the uniform distribution. Then, given an input set $\mathcal{J}$ of size n, we charge A *random-order cost* $A^{\mathrm{rom}}(\mathcal{J}) = \mathbf{E}_{\sigma \sim S_n}[A(\mathcal{J}^\sigma)] = \frac{1}{n!} \sum_{\sigma \in S_n} A(\mathcal{J}^\sigma)$. The *competitive ratio in the random-order model* of A is $c = \sup_{\mathcal{J}} \frac{\mathrm{OPT}(\mathcal{J})}{A^{\mathrm{rom}}(\mathcal{J})}$. Throughout this work we will assume that n is known to the algorithm; this assumption is common in the literature and it can be proven that it does not help in the adversarial setting, see the full version of the article for details. When clear from the context, we will omit the dependency on $\mathcal{J}$.

Given $0 \leq i \leq n$, let $P_i = P_i[\mathcal{J}]$ refer to the size of the i-th largest job in $\mathcal{J}$. Given $i \geq 1$, we define $L_i := \sum_{j \geq i} P_j$ to be the total size of jobs smaller than P_i. Note that the terminology 'smaller' uses an implicit tie breaker since there may be jobs of equal sizes.

3 Properties and Analysis of the Greedy-strategy

We now proceed with some useful properties of Graham's Greedy-strategy. Recall that this algorithm always schedules an incoming job on some least loaded machine breaking ties arbitrarily. The following two lemmas recall useful standard properties of the algorithm, which will help us later to restrict ourselves to simpler special instances.

▶ **Lemma 2.** *The minimum load achieved by the Greedy-strategy is at least P_m.*

Proof. If the m largest jobs get assigned to different machines, the bound holds directly. On the other hand, if one of the m largest jobs J_j gets assigned to a machine that already had one of the m largest jobs $J_{j'}$, of size $P_{j'}$, then at the arrival of J_j the minimum load was at least $P_{j'} \geq P_m$, concluding the claim as the minimum load does not decrease through the iterations. ◀

▶ **Lemma 3.** *The minimum load achieved by the Greedy-strategy is, for each $i \in \{1, \ldots, m\}$, bounded below by $\frac{L_i}{m} - P_i$.*

Proof. For each machine M, let $J_{\mathrm{last}}(M)$ be the last job assigned to machine M. Let $\mathcal{J}_{\mathrm{last}}$ be the set of all these last jobs. If ALG denotes the minimum load achieved by the Greedy-strategy, then the load of each machine M in the schedule is at most $ALG + J_{\mathrm{last}}(M)$ as jobs are iteratively assigned to a least loaded machine. Now remove the $i-1$ largest jobs in $\mathcal{J}_{\mathrm{last}}$ from the solution and let $\tilde{L}$ denote the total size of the remaining jobs in $\mathcal{J}_{\mathrm{last}}$. Then the total size of all remaining jobs is at most $m \cdot ALG + \tilde{L}$. On the other hand, since we only removed $i-1$ jobs, the total size of the remaining jobs is at least L_i. Since $\tilde{L} \leq (m-i+1) \cdot P_i$, we have that $L_i \leq m \cdot ALG + (m-i+1) \cdot P_i$. Hence, the minimum load achieved by the Greedy-strategy is at least $\frac{L_i}{m} - \frac{m-i+1}{m} P_i \geq \frac{L_i}{m} - P_i$. ◀

[1] In case A is randomized, we then refer to the expected minimum load.
[2] Using the convention that $0/0 = 1$ and $a/0 = \infty$ for $a > 0$.

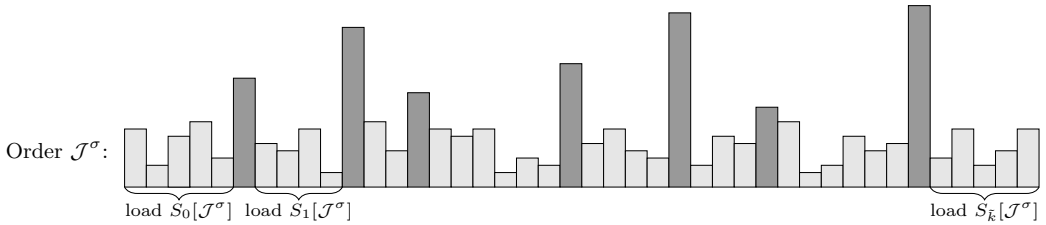


Figure 2 A possible order $\mathcal{J}^\sigma$ for an instance with $\tilde{k} < m$ large (dark) jobs. They partition the sequence into sets of small (light) jobs, each one having total size $S_i(\mathcal{J}^\sigma)$, $i = 0, \ldots, \tilde{k}$.

With these tools we can now prove that the Greedy-strategy has a competitive ratio of at most $O\left(\frac{m}{H_m}\right)$ in the random-order model, where $H_m = \sum_{i=1}^{m} \frac{1}{i}$ denotes the m-th harmonic number. We also describe a family of instances showing that this analysis is asymptotically tight.

▶ **Theorem 4.** *The Greedy-strategy is* $(2+o_m(1))\frac{m}{H_m}$*-competitive in the random-order model.*

Proof. Thanks to Lemma 2 we can assume that $P_m \leq \frac{H_m}{2m}\mathrm{OPT}$. We say that a job is *large* if its size is larger than $\frac{H_m}{2m}\mathrm{OPT}$, otherwise it is small. Let $\tilde{k} < m$ be the number of large jobs in the instance. Note that $L_{\tilde{k}+1} \geq (m-\tilde{k})\mathrm{OPT}$ since in the optimal solution at most $\tilde{k}$ machines receive large jobs. Using Lemma 3 we may assume $\tilde{k} \geq m - H_m$, as otherwise we are done.

For a given order $\mathcal{J}^\sigma$ and $0 \leq i \leq \tilde{k}$, let $S_i(\mathcal{J}^\sigma)$ be the total size of small jobs preceded by precisely i large jobs with respect to $\mathcal{J}^\sigma$ (see Figure 2). We will prove that the minimum load achieved by the Greedy-strategy is at least $\min\left\{\sum_{i=0}^{\tilde{k}} \frac{S_i(\mathcal{J}^\sigma)}{m-i} - \frac{H_m}{2m}\mathrm{OPT}, \frac{H_m}{2m}\right\}\mathrm{OPT}$.

A machine is said to be *full* once it receives a large job, and notice that we can assume that no full machine gets assigned further jobs as otherwise the minimum load would be already at least $\frac{H_m}{2m}\mathrm{OPT}$. Consider the set of small jobs that arrive to the system before the first large job in the sequence. At this point the average load of the machines is exactly $\frac{S_0(\mathcal{J}^\sigma)}{m}$ and, since the upcoming large job gets assigned to a least loaded machine, the average load of the remaining machines is still at least $\frac{S_0(\mathcal{J}^\sigma)}{m}$. Now the upcoming small jobs that arrive before the second large job in the sequence get assigned only to these non-full machines, whose average load is now at least $\frac{S_0(\mathcal{J}^\sigma)}{m} + \frac{S_1(\mathcal{J}^\sigma)}{m-1}$. Since the following large job gets assigned to the least loaded of these machines, the average load of the remaining ones is still lower-bounded by this quantity. By iterating this argument, it can be seen that the average load of the machines containing only small jobs is at least $\frac{1}{m-\tilde{k}}\sum_{i=0}^{\tilde{k}} \frac{S_i(\mathcal{J}^\sigma)}{m-i}$. Since in a Greedy-strategy the load of two machines cannot differ by more than the size of the largest job assigned to them, we conclude that the minimum load achieved by the algorithm is at least $\sum_{i=0}^{\tilde{k}} \frac{S_i(\mathcal{J}^\sigma)}{m-i} - \frac{H_m}{2m}\mathrm{OPT}$.

Now let us understand the random variable $S(\mathcal{J}^\sigma) = \sum_{i=0}^{\tilde{k}} \frac{S_i(\mathcal{J}^\sigma)}{m-i}$. If we pick $\sigma \sim S_n$ uniformly at random and consider the number $s(J)$ of large jobs that precede any fixed small job J, this number is uniformly distributed between 0 and $\tilde{k}$. Then we can rewrite $S(\mathcal{J}^\sigma) = \sum_{i=0}^{\tilde{k}} \frac{S_i(\mathcal{J}^\sigma)}{m-i} = \sum_{J \text{ small}} \frac{p_J}{m-s(J)}$. Since $\mathbf{E}[\frac{1}{m-s(J)}] = \sum_{i=0}^{\tilde{k}} \frac{1}{\tilde{k}+1}\frac{1}{m-i} = \frac{H_m - H_{m-\tilde{k}-1}}{\tilde{k}+1}$

we get by linearity of expectation that

$$\begin{aligned}\mathbf{E}[S(\mathcal{J}^\sigma)] &= \sum_{J \text{ small}} p_J \frac{H_m - H_{m-\tilde{k}-1}}{\tilde{k}+1} \\ &= \frac{H_m - H_{m-\tilde{k}-1}}{\tilde{k}+1} L_{\tilde{k}+1} \\ &\geq \frac{(m-\tilde{k})(H_m - H_{m-\tilde{k}-1})}{\tilde{k}+1} \frac{L_{\tilde{k}+1}}{m-\tilde{k}}.\end{aligned}$$

Notice that $\frac{L_{\tilde{k}+1}}{m-\tilde{k}}$ is a lower bound for OPT, which we will use later. The first factor is decreasing as a function of $\tilde{k} \leq m-1$, and consequently the expression is at least $\frac{H_m}{m}$. Thus $\mathbf{E}[S(\mathcal{J}^\sigma)] \geq \frac{H_m}{m} \cdot \frac{L_{\tilde{k}+1}}{m-\tilde{k}}$.

We also get $\operatorname{Var}\left[\frac{1}{m-s(J)}\right] \leq \mathbf{E}\left[\left(\frac{1}{m-s(J)}\right)^2\right] = \sum_{i=0}^{\tilde{k}} \frac{1}{\tilde{k}+1} \frac{1}{(m-i)^2} \leq \frac{1}{\tilde{k}+1} \sum_i \frac{1}{i^2} \leq \frac{1}{\tilde{k}+1} \frac{\pi^2}{6}$. Using that $\tilde{k} \geq m - H_m$, we broadly bound $\tilde{k}$ via $\tilde{k}+1 \geq \frac{m}{2} \geq \frac{m(m-\tilde{k})^2}{2H_m^2}$. Substituting this in the previous bound yields $\operatorname{Var}\left[\frac{1}{m-s(J)}\right] \leq \frac{H_m^2}{m} \frac{1}{(m-\tilde{k})^2} \frac{\pi^2}{3}$. Moreover, for two small jobs J_i and J_j we have $\operatorname{Cov}\left[\frac{1}{m-s(J_i)}, \frac{1}{m-s(J_j)}\right] \leq \left(\operatorname{Var}\left[\frac{1}{m-s(J_i)}\right] \operatorname{Var}\left[\frac{1}{m-s(J_j)}\right]\right)^{1/2} \leq \frac{H_m^2}{m} \frac{1}{(m-\tilde{k})^2} \frac{\pi^2}{3}$. For $i \neq j$ this bound is pessimistic. The correlation between $s(J_i)$ and $s(J_j)$ is positive but tiny. We use this covariance to bound $\operatorname{Var}[S(\mathcal{J}^\sigma)] = \sum_{J \text{ small}} \frac{p_J}{m-s(J)}$.

$$\begin{aligned}\operatorname{Var}[S(\mathcal{J}^\sigma)] &= \sum_{J_i, J_j \text{ small}} p_i p_j \operatorname{Cov}\left[\frac{1}{m-s(J_i)}, \frac{1}{m-s(J_j)}\right] \\ &\leq \sum_{J_i \text{ small}} p_i \cdot \sum_{J_j \text{ small}} p_j \cdot \frac{H_m^2}{m} \frac{1}{(m-\tilde{k})^2} \frac{\pi^2}{3} \\ &\leq \frac{\pi^2}{3} \frac{H_m^2}{m} \left(\frac{L_{\tilde{k}+1}}{m-\tilde{k}}\right)^2.\end{aligned}$$

The last inequality uses again that $L_{\tilde{k}+1} \geq (m-\tilde{k})\mathrm{OPT}$. Hence, the standard deviation $\mathrm{SD}[S(\mathcal{J}^\sigma)]$ is at most $C^{3/2} \frac{L_{\tilde{k}+1}}{m-\tilde{k}}$ for $C = \sqrt[3]{\frac{\pi^2}{3} \frac{H_m^2}{m}} = \Theta\left(\sqrt[3]{\frac{(\log(m))^2}{m}}\right)$. Chebyshev's inequality, which allows to bound the probability of deviating from the mean in terms of the standard deviation, yields

$$\begin{aligned}\mathbf{P}\left[S(\mathcal{J}^\sigma) \leq \left(\frac{H_m}{m} - C\right) \mathrm{OPT}\right] &\leq \mathbf{P}\left[S(\mathcal{J}^\sigma) \leq \left(\frac{H_m}{m} - C\right) \frac{L_{\tilde{k}+1}}{m-\tilde{k}}\right] \\ &\leq \mathbf{P}\left[|\mathbf{E}[S(\mathcal{J}^\sigma)] - S(\mathcal{J}^\sigma)| \geq C^{-1/2} \cdot \sigma[S(\mathcal{J}^\sigma)]\right] \\ &\leq C.\end{aligned}$$

We conclude that with probability $1-C$ the minimum load achieved by the Greedy-strategy exceeds $\min\left\{S(\mathcal{J}^\sigma) - \frac{H_m}{2m}, \frac{H_m}{2m}\right\} \mathrm{OPT} \geq \frac{H_m}{m}\left(\frac{1}{2} - C\right)\mathrm{OPT}$. Thus its competitive ratio in the random-order model is at most $(1-C)\frac{m}{H_m} \frac{1}{\left(\frac{1}{2}-C\right)} = (2 + o_m(1))\frac{m}{H_m}$. ◀

▶ **Theorem 5.** *The Greedy-strategy is not better than $\frac{m}{H_m}$-competitive in the random-order model.*

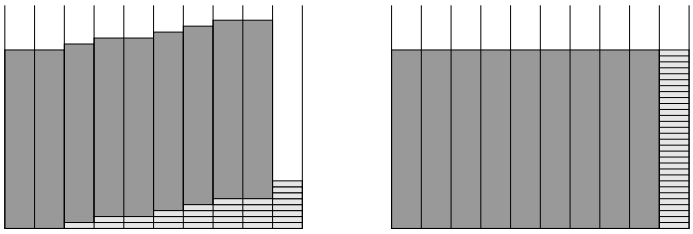


Figure 3 A comparison of the solution returned by the Greedy-strategy (left) and the optimal solution (right) for some order σ of the instance defined in Theorem 4. Dark jobs have size 1 and the remaining jobs have size $\varepsilon > 0$.

Proof. Consider the following instance for $\varepsilon > 0$: Job set $\mathcal{J}$ consists of $m-1$ jobs of size 1 and $\frac{1}{\varepsilon}$ jobs of size ε. It is not difficult to see that $\mathrm{OPT} = 1$ by assigning the jobs of size 1 to different machines and the jobs of size ε together on the remaining machine. Following a similar approach as the one above, interpreting the jobs of size 1 as large and the rest as small, we can prove that for any given order $\mathcal{J}^\sigma$, the expected minimum load achieved by the Greedy-strategy is at most $\sum_{i=0}^{m-1} \frac{S_i(\mathcal{J}^\sigma)}{m-i} + \varepsilon m$. If now $\sigma \sim S_n$ is chosen uniformly at random, then the minimum load achieved by the Greedy-strategy is at most $\sum_{i=0}^{m-1} \frac{1}{m(m-i)} + \varepsilon m = \frac{H_m}{m} + \varepsilon m$. In particular, the competitive ratio in the random-order model of the Greedy-strategy is at most $1/(\frac{H_m}{m} + \varepsilon m)$ which approaches $\frac{m}{H_m}$ as $\varepsilon \to 0$. ◀

4 An $\tilde{O}(\sqrt[4]{m})$-competitive algorithm

We now describe an improved competitive algorithm for the problem further exploiting the extra features of the random-order model. More in detail, we devise a sampling-based method to classify relatively large jobs (with respect to OPT). After this, we run a slight adaptation of the algorithm *Partition* due to Azar and Epstein [7] in order to distinguish large jobs that our initial procedure could not classify, leading to strictly better approximation guarantees (see Algorithm 1 for a description). We will assume w.l.o.g. that job sizes are rounded down to powers of 2, which induces an extra multiplicative factor of at most 2 in the competitive ratio.

4.1 Simple and Proper Inputs

Given an input sequence $\mathcal{J}$, we call any job J *large* if its size is larger than $\frac{\mathrm{OPT}[\mathcal{J}]}{100\sqrt[4]{m}}$, and we call it *small* otherwise. Let $k = k[\mathcal{J}]$ be the number of large jobs in $\mathcal{J}$. Let $L_{\mathrm{small}} = L_{k+1}$ be the total size of small jobs. The following definition allows to recognize instances where the Greedy-strategy performs well, see Proposition 7.

▶ **Definition 6.** *We call the input set $\mathcal{J}$* ***simple*** *if either*

- *The set $\mathcal{J}$ has size $n < m$,*
- *There are at least m large jobs, i.e. $k \geq m$, or*
- *There are at most $m - \frac{\sqrt[4]{m^3}}{50}$ large jobs, i.e. $k \leq m - \frac{\sqrt[4]{m^3}}{50}$.*

Note that the first condition is mostly included for ease of notation; the third condition implies that sequences with $n < m - \frac{\sqrt[4]{m^3}}{50}$ are simple, which is good enough for our purposes but somewhat clumsy to refer to.

Algorithm 1 The online Algorithm for Random-Order Machine Covering

Input: Job sequence $\mathcal{J}^\sigma$, m identical parallel machines.

Guess $t \in \{-1, 0, 1, \ldots, \lceil \frac{3}{4}\log(m) \rceil\}$ uniformly at random.
if $t = -1$, **then** Run the Greedy-strategy and **return** the computed solution.
else Partition the machines into 2^t *small* machines and $m - 2^t$ *large* machines.

Phase 1 – Sampling.

Schedule the first $n/8$ jobs iteratively into a least loaded large machine.
Let $P^\uparrow$ be the $\left(\frac{m-2^t}{8} - \frac{\sqrt{m}}{2}\right)$-th largest job size among these first $n/8$ jobs.

Phase 2 – Partition.

$\tau \leftarrow 0$
for $j = \frac{n}{8} + 1, \ldots, n$ **do**
    if $p_{\sigma(j)} \geq P^\uparrow$, **then** Schedule job $J_{\sigma(j)}$ onto a least loaded large machine.
    if $p_{\sigma(j)} > \tau$, **then** Update τ to $p_{\sigma(j)}$ with probability $\frac{1}{9 \cdot 2^t \sqrt{m}}$.
    if $p_{\sigma(j)} \leq \tau$, **then** Schedule job $J_{\sigma(j)}$ onto a least loaded small machine.
    if $p_{\sigma(j)} > \tau$, **then** Schedule job $J_{\sigma(j)}$ onto a least loaded large machine.
end for
return the computed solution.

▶ **Proposition 7.** *The Greedy-strategy achieves minimum load at least* $\frac{\mathrm{OPT}}{100\sqrt[4]{m}}$ *on simple inputs.*

Proof. We consider each case from Definition 6 separately:

- If the instance has less than m jobs, $\mathrm{OPT} = 0$ and every algorithm is optimal.
- If there are at least m large jobs in the instance, then the minimum load achieved by the Greedy algorithm is at least $P_m \geq \frac{\mathrm{OPT}}{100\sqrt[4]{m}}$ thanks to Lemma 2.
- If there are at most $m - \frac{\sqrt[4]{m^3}}{50}$ large jobs, then $L_{k+1} \geq \frac{\sqrt[4]{m^3}}{50}\mathrm{OPT}$ as there are at least $\frac{\sqrt[4]{m^3}}{50}$ machines without large jobs in the optimal solution. If we apply Lemma 3 with $i = k + 1$, the minimum load achieved by the Greedy algorithm is at least $\frac{\mathrm{OPT}}{100\sqrt[4]{m}}$. ◀

For an input set $\mathcal{J}$ which is not simple, let $d := \lceil \log_2(m - k) \rceil$. Note that $0 \leq d \leq \lceil \frac{3}{4}\log(m) \rceil$. We say that such an instance $\mathcal{J}$ is *proper (of degree d)*.

Algorithm 1 guesses a value t with probability $\Omega(1/\log(m))$ to address in a different way simple instances (case $t = -1$) and proper instances of degree t for each $0 \leq t \leq \lceil \frac{3}{4}\log(m) \rceil$, at the expense of an extra logarithmic factor in the competitive ratio. By Proposition 7, simple instances are sufficiently handled by the Greedy-strategy. From now on we will focus on proper instances of degree d and show that the corresponding case when $t = d$ in Algorithm 1 returns a $O(\sqrt[4]{m})$-approximate solution.

4.2 Algorithm for Proper Inputs of Degree d

Let $\mathcal{J}$ be proper of degree d and assume without loss of generality that its size n is divisible by 8 (an online algorithm can always simulate up to 7 extra jobs of size 0 to reduce to this case). The algorithm, assuming d is guessed correctly, chooses 2^d *small* machines $\mathcal{M}_{\text{small}}$, while the other machines $\mathcal{M}_{\text{large}}$ are called *large machines*. The algorithm will always assign the incoming job either to a least loaded small or to a least loaded large machine, the only choice it has to make is to which set of machines the job will go. Theoretically, the goal would be to assign all large jobs to large machines and all small ones to small machines

according to our definition. Large and small jobs, unfortunately, cannot be distinguished by an online algorithm with certainty. Instead, we have to use randomization and expect a small error. We aim for a small one-sided error, meaning that we want to avoid misclassifying large jobs at all cost while incorrectly labeling very few small jobs as large.

The algorithm starts with a *sampling phase*: the first $\frac{n}{8}$ jobs will be used for sampling purposes, and since we yet lack good knowledge about what should be considered large, these jobs will all be assigned to large machines. Let $P^{\uparrow}$ be the element of rank $\frac{m-2^d}{8} - \frac{\sqrt{m}}{2}$ among these elements. The following lemma shows that $P^{\uparrow}$ can be used as a threshold to distinguish most of the large jobs from small ones.

▶ **Lemma 8.** *For proper sequences, it holds that* $\mathbf{P}[P_{k-8\sqrt{m}-2^d} \geq P^{\uparrow} \geq P_k] \geq \frac{1}{3}$.

Proof. Let n_{large} be the number of large jobs among the first $\frac{n}{8}$ jobs in the sequence. Notice that n_{large} obeys an hypergeometric distribution with parameters $N = n$ (size of the total population), $K = k$ (number of elements with the desired property) and $r = \frac{n}{8}$ (size of the sample). This implies that $\mathbf{E}_{\sigma\sim S_n}[n_{\text{large}}] = \frac{k}{8} \geq \frac{m-2^d}{8}$. Moreover

$$\text{Var}[n_{\text{large}}] = \frac{n}{8} \cdot \frac{k}{n} \cdot \frac{n-k}{n} \cdot \frac{7n}{8(n-1)} = \frac{7}{64} \cdot \frac{k(n-k)}{n-1} < \frac{m}{8}.$$

If we use Cantelli's inequality, a one-sided version of Chebyshev's inequality, then

$$\mathbf{P}[P^{\uparrow} < P_k] = \mathbf{P}\left[n_{\text{large}} < \frac{m-2^d}{8} - \frac{\sqrt{m}}{2}\right] \leq \mathbf{P}\left[n_{\text{large}} < \mathbf{E}[n_{\text{large}}] - \frac{\sqrt{m}}{2}\right] \leq \frac{m/8}{\frac{m}{8}+\frac{m}{4}} = \frac{1}{3}.$$

Consider now n'_{large} to be the number of the $k - 8\sqrt{m} - 2^d$ largest jobs among the $\frac{n}{8}$ first jobs in the sequence. Similarly as before, n'_{large} obeys an hypergeometric distribution with parameters $N = n$, $K = k - 8\sqrt{m} - 2^d$ and $r = \frac{n}{8}$, which implies that $\mathbf{E}_{\sigma\sim S_n}[n'_{\text{large}}] = \frac{k-2^d-8\sqrt{m}}{8} < \frac{m-2^d}{8} - \sqrt{m}$. Here we use that $k \leq m$. Furthermore $\text{Var}[n'_{\text{large}}] = \frac{n}{8} \cdot \frac{k-8\sqrt{m}-2^d}{n} \cdot \frac{n-k+8\sqrt{m}+2^d}{n} \cdot \frac{7n}{8(n-1)} \leq \frac{m}{8}$. Then, using again Cantelli's inequality, we obtain that

$$\begin{aligned}\mathbf{P}\left[P_{k-8\sqrt{m}-2^d} < P^{\uparrow}\right] &= \mathbf{P}\left[n'_{\text{large}} \geq \frac{m-2^d}{8} - \frac{\sqrt{m}}{2}\right] \\ &\leq \mathbf{P}\left[n'_{\text{large}} \geq \mathbf{E}[n'_{\text{large}}] + \frac{\sqrt{m}}{2}\right] \leq \frac{m/8}{\frac{m}{8}+\frac{m}{4}} = \frac{1}{3}.\end{aligned}$$

We conclude that $\mathbf{P}[P_{k-8\sqrt{m}-2^d} \geq P^{\uparrow} \geq P_k] = 1 - \mathbf{P}\left[P_{k-8\sqrt{m}-2^d} < P^{\uparrow}\right] - \mathbf{P}[P^{\uparrow} < P_k] \geq 1/3$. ◀

After the aforementioned sampling phase is finished and $P^{\uparrow}$ is known, the algorithm will enter a *partition phase*. If a job now has size at least $P^{\uparrow}$, it is assigned to the large machines as it will be large with high probability. The large jobs below this value are the most difficult to assign. To this end, we define a *threshold value* τ, which we initialize to 0. If the incoming job has size at most τ we simply assign it to a least loaded small machine, but whenever we encounter a job J of size $p > \tau$, we set $\tau = p$ with probability $\frac{1}{9\cdot 2^d\sqrt{m}}$. If now $p \leq \tau$, which could happen if we just increased τ, we schedule J on the least loaded small machine. Else, J is assigned to the least loaded machine in $\mathcal{M}_{\text{large}}$ (see Figure 4 for a depiction of the procedure). As the following lemma shows, this procedure distinguishes all the large jobs with constant probability.

▶ **Lemma 9.** *All large jobs are scheduled onto large machines with constant probability.*

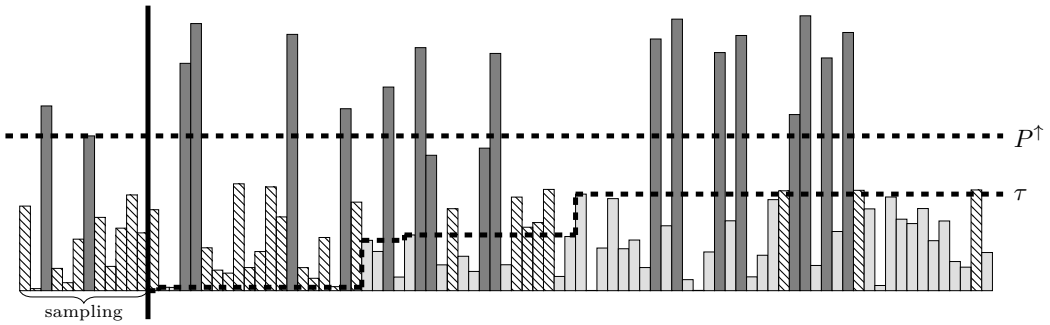


Figure 4 The classification of large (dark) and small (light and dashed) jobs. During sampling, all small jobs are misclassified (dashed ones). Threshold $P^{\uparrow}$ classifies large jobs, while threshold τ classifies small jobs. Jobs in between are conservatively classified as large, since misclassifying large jobs is fatal. Increasing τ due to a small job is a helpful event as less small jobs will be misclassified. On the other hand, increasing τ due to a large job below $P^{\uparrow}$ is a fatal event as large jobs will be misclassified. The choice of $P^{\uparrow}$ ensures that fatal events are unlikely to happen.

Proof. Let us assume that $P_{k-8\sqrt{m}-2^d} \geq P^{\uparrow} \geq P_k$. Now, the statement of the lemma can only be wrong if we decided to increase τ when encountering some large job of size less than $P^{\uparrow}$. By assumption there are less than $8\sqrt{m}+2^d$ such jobs. The desired probability is thus at least

$$\left(1-\frac{1}{9\cdot 2^d\sqrt{m}}\right)^{8\sqrt{m}+2^d} \geq 1-\frac{8\sqrt{m}+2^d}{9\cdot 2^d\sqrt{m}} \geq 1-\frac{8}{9\cdot 2}-\frac{1}{9\sqrt{m}} > \frac{4}{9}.$$

The first inequality is Bernoulli's inequality, the second one uses that $d \geq 1$. The lemma follows by multiplying with the probability from Lemma 8. ◀

We call an input sequence *orderly* if it satisfies the properties of Lemmas 8 and 9. The following lemma shows that the total size of misclassified small jobs can be, in expectation, bounded from above. This proof is an adaptation of one of the results from Azar and Epstein [7], which we present for the sake of completeness.

▶ **Lemma 10.** *The expected total size of small jobs scheduled onto small machines is at least* $\frac{31}{800\sqrt[4]{m}}L_{\text{small}}$*, even when conditioned on the input sequence being orderly.*

Proof. Let L'_{small} be the random variable corresponding to the size of small jobs assigned to large machines in the sampling phase. Since jobs appear in random order, we have that

$$\mathbf{E}_{\sigma\sim S_n}[L'_{\text{small}}] = \frac{1}{n!}\sum_{J_i \text{ small}}\frac{n}{8}\cdot(n-1)!\cdot p_i = \frac{1}{8}L_{\text{small}}.$$

Let us now bound the total size of misclassified small jobs in the partition phase. We will define, for a given set of $18\cdot 2^d\sqrt[4]{m}$ small jobs of the same size p_i (recall that jobs are rounded down to powers of 2), the following event: after $9\cdot 2^d\sqrt[4]{m}$ jobs from the set arrived, τ is at least p_i. If this were not the case, τ was never updated at any of these $9\cdot 2^d\sqrt[4]{m}$ first jobs albeit being smaller than p_i. The probability for this is at most

$$\left(1-\frac{1}{9\cdot 2^d\sqrt{m}}\right)^{9\cdot 2^d\sqrt[4]{m}} \leq e^{-\frac{1}{\sqrt[4]{m}}} \leq 1-\frac{1}{2\sqrt[4]{m}},$$

where we used the fact that $e^{-x} \leq 1-\frac{x}{2}$ if $0 \leq x \leq 1$. This implies that the probability of the previous event occurring is at least $\frac{1}{2\sqrt[4]{m}}$.

Let $\mathcal{S}$ be the set of small jobs remaining after the sampling phase. We will partition $\mathcal{S}$ into batches of $18\sqrt[4]{m}$ jobs of the same size. There will be jobs that are not assigned to any batch because there are not enough jobs of the same size to complete it, but the total size of these jobs is at most

$$18 \cdot 2^d \sqrt[4]{m} \sum_{i \geq 1} \frac{\mathrm{OPT}}{100\sqrt[4]{m} \cdot 2^i} \leq \frac{18 \cdot 2^{d-1} \cdot \mathrm{OPT}}{25} \leq \frac{18 L_{\mathrm{small}}}{25},$$

where the last inequality holds as there are at least 2^{d-1} machines in the optimal solution without large jobs. For each of the batches, the probability of assigning at least half of its size to small machines is bounded below by the probability of the previously described event occurring, which is at least $\frac{1}{2\sqrt[4]{m}}$. Hence, the expected total size of small jobs assigned to small machines is at least

$$\frac{1}{2\sqrt[4]{m}} \cdot \frac{1}{2} \sum_{J_i \in \mathcal{S}} p_i \geq \frac{1}{4\sqrt[4]{m}} \cdot \left(1 - \frac{1}{8} - \frac{18}{25}\right) L_{\mathrm{small}} = \frac{31}{800\sqrt[4]{m}} L_{\mathrm{small}}.$$

To observe that we can condition on the sequence being orderly, it suffices to note that the arguments work for every way to fix $P^{\uparrow}$ and that they do not make any assumptions on τ being increased at large jobs. ◀

Putting all the previous ingredients together we can conclude the following proposition.

▶ **Proposition 11.** *The previously described algorithm is $O(\sqrt[4]{m})$-competitive in the random-order model for the case of proper inputs of degree d.*

Proof. By assumption, all $k \geq m - 2^d$ large jobs are scheduled onto large machines. A lower bound of $P_k = \frac{\mathrm{OPT}}{100\sqrt[4]{m}}$ for the minimum load achieved in the large machines follows then from Lemma 2. By Lemma 3, the minimum load among small machines is at least $\frac{31}{800\sqrt[4]{m}} \frac{L_{\mathrm{small}}}{2^d} - \frac{\mathrm{OPT}}{100\sqrt[4]{m}}$. The proposition follows from observing that $L_{\mathrm{small}} \geq 2^{d-1}\mathrm{OPT}$ since the optimal solution contains at least $m - k \geq 2^{d-1}$ machines with only small jobs. ◀

4.3 The Final Algorithm

As discussed before, our final algorithm first guesses whether the instance is simple or proper of degree d. Then we apply the appropriate algorithm, the Greedy-strategy or the previously described algorithm for the right degree. Since there are $O(\log(m))$ many possibilities, this guessing induces an extra logarithmic factor on the final competitive ratio, which concludes the proof of Theorem 1 restated below.

▶ **Theorem 1.** *There exists a $\tilde{O}(\sqrt[4]{m})$-competitive algorithm for the online Machine Covering problem in the random-order model.*

5 A Lower Bound for the Random-Order Model

The main difficulty for Online Machine Covering algorithms, including our main result, is to tell large jobs apart from the largest small jobs. In this section we prove that doing so is, to a certain extent, inherently hard. The main difference to adversarial models is that hardness is not obtained through withholding information but rather through obscuring it. This relates to some studied variants of the classical Secretary Problem such as the Postdoc Problem [43] but requires additional features particularly catered to our needs.

5.1 The Talent Contest Problem

Consider the following selection problem: To a yearly talent show contest n candidates apply. To appeal to a general audience, we try to exclude the best candidates because an imperfect performance is more entertaining, but we also want to have at least an appropriate candidate who can be presented as the winner. To do so, each candidate will participate in t trials (each trial is considered as an arrival) and we must decide for every arrival if we mark the candidate or not, meaning that we consider her to be the K-th best candidate or worse. The global order in which candidates arrive for trials is uniformly distributed, thus at later trials we have much more information to go by. Our final goal is to maximize the number of trials for which we successfully marked the K-th best candidate without marking any better candidate.

Formally, the *Talent Contest* problem is specified by three parameters K, n and t, where $K \leq n$. Candidates have pairwise different non-negative valuations $v_1, v_2, \ldots, v_n$, and each candidate arrives t times; the arrival order is chosen uniformly at random. The valuation of each candidate is revealed when the candidate arrives for the first time. We may decide to mark each arrival or not, though once the next candidate arrives such marking decision is permanent. For each value $1 \leq h \leq t$ we get one point if we marked the h-th arrival of the K-th best candidate, but not the h-th arrival of any better candidate. In particular, we can get up to t points in total, one for each value of h. Let $P(K, t, n)$ be the expected number of points the optimal online algorithm scores given the three values K, t and n. Similar to the classical Secretary problem, we are mostly interested in the limit case $P(K, t) = \lim_{n\to\infty} P(K, t, n)$.

We require for our desired results one extra technical definition. Given $\lambda \geq 1$, we call the valuations of candidates λ-steep if all candidate valuations are guaranteed to be at least by a factor λ apart, i.e. $\min_{v_i > v_j} \frac{v_i}{v_j} \geq \lambda$. It is possible to prove the following bound on the expected value $P(K, t)$, whose proof we defer to the full version of the paper.

▶ **Lemma 12.** *It holds that* $P(K, t) \leq \frac{\zeta(t/2)(t+1)^{t/2}}{2\pi\sqrt{K}}$, *where* ζ *is the Riemann Zeta Function. This bound still holds if we restrict ourselves to* λ*-steep valuations for some* $\lambda \geq 1$.

Roughly speaking, the proof relies on the fact that if an algorithm manages to perform relatively well in the Talent Contest problem, then it could be used to guess the value of a binomially distributed random variable. For the latter problem, difficulty can be directly established. However, the proof involves more technical aspects as some few irregular orders do not allow this reduction and have to be excluded beforehand. The property of λ-steepness is ensured by choosing μ sufficiently large and applying the transformation $v \mapsto \mu^v$ to each valuation.

5.2 Reduction to Machine Covering

It is possible to show that the Talent Contest problem and the Online Machine Covering problem in the random-order model are related as the following lemma states.

▶ **Lemma 13.** *Given* K *and* t, *let* $m = (K-1) \cdot t + 1$. *No (possibly randomized) algorithm for Machine Covering in the random-order model on* m *machines can be better than* $\frac{t}{P(K,t)+1}$*-competitive.*

Proof. Let $\lambda > t$. Consider any instance of the Talent Contest problem with λ-steep valuations. We will treat the arrival sequence of candidates as a job sequence, where each arrival corresponds to a job of size given by the valuation of the corresponding candidate.

We call the $m-1=t(K-1)$ jobs corresponding to the arrivals of the K largest candidates *large*. The t jobs corresponding to the next candidate are called *medium*. Notice that the size of a medium job is at most $\frac{OPT}{t}$ as evidenced by the schedule that assigns each large job on a separate machine and the t medium jobs onto the single remaining machine. Jobs which are neither large nor medium have total size at most $t\sum_{i=1}^{\infty}\lambda^{-i}\frac{\mathrm{OPT}}{t}=\frac{\mathrm{OPT}}{\lambda-1}$ and are thus called *small*. They will become negligible for $\lambda\to\infty$.

Consider an online algorithm $\mathcal{A}_{MC}$ for Machine Covering in the random-order model. We will derive an algorithm $\mathcal{A}_{TC}$ for the Talent Contest problem as follows: $\mathcal{A}_{TC}$ marks each job that gets assigned to a machine that already contains a job of the same size. Let P be the number of points this strategy gets. We will first show that the schedule of $\mathcal{A}_{MC}$ contains a machine which has at most $P+1$ medium jobs and no big job. For this we consider any fixed input order, and if $\mathcal{A}_{MC}$ is randomized, we consider any fixed outcome of its coin tosses.

For $2\le i\le t$, let w_i be an indicator variable that is 1 if $\mathcal{A}_{TC}$ gains a point for the i-th arrival, i.e. if it marks the i-th arrival of the K-th best candidate but not the i-th arrival of a better candidate; $w_i=0$ otherwise. Let also r_i be an indicator variable that is 1 if $\mathcal{A}_{TC}$ marked the i-th arrival of the K-th best candidate but still loses due to also marking the i-th arrival of a better candidate. Finally, let $\mathcal{M}_{\mathrm{small}}$ be the machines which do not receive a large job in the schedule of $\mathcal{A}_{MC}$, and let Z_{med} be the average number of medium jobs on machines in $\mathcal{M}_{\mathrm{small}}$. Our intermediate goal is to show that $Z_{\mathrm{med}}<2+\sum_{i=2}^{t}w_i$.

Since there are only $m-1$ large jobs, of which at least $\sum_{i=2}^{t}r_i$ are scheduled on a machine already containing a large job, we have that $|\mathcal{M}_{\mathrm{small}}|\ge 1+\sum_{i=2}^{t}r_i$. Let $d\ge 0$ such that $|\mathcal{M}_{\mathrm{small}}|=1+d+\sum_{i=2}^{t}r_i$. Now, observe that $\sum_{i=2}^{t}(w_i+r_i)$ counts the number of medium jobs that are placed on a machine already containing a medium job. Thus, the number of medium jobs on machines in $\mathcal{M}_{\mathrm{small}}$ is at most $|\mathcal{M}_{\mathrm{small}}|+\sum_{i=2}^{t}(w_i+r_i)=1+d+\sum_{i=2}^{t}(w_i+2r_i)$. Now we can bound the average number of medium jobs on $\mathcal{M}_{\mathrm{small}}$, namely $Z_{\mathrm{med}}\le\frac{1+d+\sum_{i=2}^{t}(w_i+2r_i)}{1+d+\sum_{i=2}^{t}r_i}$. Let us assume that $Z_{\mathrm{med}}\ge 2$; then the term on the right hand side increases if we set d and all r_i to zero, and we obtain that $Z_{\mathrm{med}}\le 1+\sum_{i=2}^{t}w_i$ and thus $Z_{\mathrm{med}}<2+\sum_{i=2}^{t}w_i$. To derive this inequality we assumed that $Z_{\mathrm{med}}\ge 2$ but it is trivially true if $Z_{\mathrm{med}}<2$.

Now, let z_{med} be the least number of medium jobs on a machine in $\mathcal{M}_{\mathrm{small}}$. Then $z_{\mathrm{med}}\le Z_{\mathrm{med}}<2+\sum_{i=2}^{t}w_i$. Since z_{med} and the right hand side are both integers, it holds that $z_{\mathrm{med}}\le 1+\sum_{i=2}^{t}w_i$. Since $\sum_{i=2}^{t}w_i=P$, the number of points obtained by algorithm $\mathcal{A}_{TC}$ for the Talent Contest problem, we have shown that the schedule of $\mathcal{A}_{MC}$ contains a machine M with at most $P+1$ medium jobs and no large one as desired.

As argued before, each medium job has size at most $\frac{\mathrm{OPT}}{t}$ and the small jobs have total size at most $\frac{\mathrm{OPT}}{\lambda-1}$ in total. Thus, machine M has load at most $\frac{P+1}{t}\mathrm{OPT}+\frac{\mathrm{OPT}}{\lambda-1}$. In conclusion, the expected load of the least loaded machine in the schedule of $\mathcal{A}_{MC}$ is at most $\frac{P(K,t)+1}{K}\mathrm{OPT}+\frac{\mathrm{OPT}}{\lambda-1}$, given a worst-case input for the Talent Contest Problem. This concludes the proof by taking $\lambda\to\infty$. ◀

By setting $K=(t+1)^t$ and combining this with the lower bound in Lemma 12, we obtain the following general lower bound.

▶ **Theorem 14.** *The competitive ratio of no online algorithm for Machine Covering in the random-order model, deterministic or randomized, is better than* $\frac{\lfloor e^{W(\ln(m))}\rfloor-1}{1.16+o(1)}$. *Here,* $W(x)$ *is the Lambert W-function, i.e. the inverse to* $x\mapsto xe^x$. *In particular, no algorithm can be* $O\left(\frac{\log(m)}{\log\log(m)}\right)$*-competitive for Online Machine Covering in the random-order model.*

Proof. Let $K = (t+1)^t$. Then Lemma 12 yields $P(K,t) \leq \frac{\zeta(t/2)}{2\pi} \leq 1.16+o(1)$. By Lemma 13 no algorithm can be better than $\left(\frac{t}{1.16+o(1)}\right)$-competitive for $m = (K-1)\cdot t+1 < (t+1)^{t+1}$. We can always add a few jobs of large enough size so that the lower bound extends to larger numbers of machines. The theorem follows since the inverse function of $x \mapsto (t+1)^{t+1}$ is $t \mapsto e^{W(\ln(m))} - 1$; the second part uses the identity $W(x) \geq \log(x) - \log\log(x) + \omega(1)$, see [28]. ◀

References

1. Susanne Albers. Better bounds for online scheduling. *SIAM J. Comput.*, 29(2):459–473, 1999. doi:10.1137/S0097539797324874.
2. Susanne Albers and Maximilian Janke. Scheduling in the random-order model. In *47th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 168, pages 68:1–68:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.68.
3. Susanne Albers and Maximilian Janke. Scheduling in the secretary model. In *41st Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2021. To appear.
4. Susanne Albers, Arindam Khan, and Leon Ladewig. Best fit bin packing with random order revisited. In *45th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 170, pages 7:1–7:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.MFCS.2020.7.
5. Susanne Albers, Arindam Khan, and Leon Ladewig. Improved online algorithms for knapsack and GAP in the random order model. *Algorithmica*, 83(6):1750–1785, 2021. doi:10.1007/s00453-021-00801-2.
6. Arash Asadpour and Amin Saberi. An approximation algorithm for max-min fair allocation of indivisible goods. *SIAM J. Comput.*, 39(7):2970–2989, 2010. doi:10.1137/080723491.
7. Yossi Azar and Leah Epstein. On-line machine covering. *Journal of Scheduling*, 1(2):67–77, 1998. doi:https://doi.org/10.1002/(SICI)1099-1425(199808)1:2<67::AID-JOS6>3.0.CO;2-Y.
8. Moshe Babaioff, Nicole Immorlica, David Kempe, and Robert Kleinberg. A knapsack secretary problem with applications. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 10th International Workshop (APPROX/RANDOM)*, volume 4627, pages 16–28. Springer, 2007. doi:10.1007/978-3-540-74208-1_2.
9. Moshe Babaioff, Nicole Immorlica, David Kempe, and Robert Kleinberg. Matroid secretary problems. *J. ACM*, 65(6):35:1–35:26, 2018. doi:10.1145/3212512.
10. Nikhil Bansal and Maxim Sviridenko. The santa claus problem. In *38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 31–40. ACM, 2006. doi:10.1145/1132516.1132522.
11. Yair Bartal, Amos Fiat, Howard J. Karloff, and Rakesh Vohra. New algorithms for an ancient scheduling problem. *J. Comput. Syst. Sci.*, 51(3):359–366, 1995. doi:10.1006/jcss.1995.1074.
12. Yair Bartal, Howard J. Karloff, and Yuval Rabani. A better lower bound for on-line scheduling. *Inf. Process. Lett.*, 50(3):113–116, 1994. doi:10.1016/0020-0190(94)00026-3.
13. Eugene B. Dynkin. The optimum choice of the instant for stopping a Markov process. *Soviet Math. Dokl*, 4, 1962.
14. Tomás Ebenlendr, John Noga, Jirí Sgall, and Gerhard J. Woeginger. A note on semi-online machine covering. In *Approximation and Online Algorithms, Third International Workshop (WAOA)*, volume 3879, pages 110–118. Springer, 2005. doi:10.1007/11671411_9.
15. Leah Epstein. A survey on makespan minimization in semi-online environments. *J. Sched.*, 21(3):269–284, 2018. doi:10.1007/s10951-018-0567-z.
16. Leah Epstein, Asaf Levin, and Rob van Stee. Max-min online allocations with a reordering buffer. *SIAM J. Discret. Math.*, 25(3):1230–1250, 2011. doi:10.1137/100794006.

564 **17** Ulrich Faigle, Walter Kern, and György Turán. On the performance of on-line algorithms for
565 partition problems. *Acta Cybern.*, 9(2):107–119, 1989.
566 **18** Moran Feldman, Ola Svensson, and Rico Zenklusen. A simple $O(\log \log(\text{rank}))$-competitive
567 algorithm for the matroid secretary problem. *Math. Oper. Res.*, 43(2):638–650, 2018. doi:
568 10.1287/moor.2017.0876.
569 **19** Rudolf Fleischer and Michaela Wahl. Online scheduling revisited. In *Algorithms, 8th Annual*
570 *European Symposium (ESA)*, volume 1879, pages 202–210. Springer, 2000. doi:10.1007/
571 3-540-45253-2_19.
572 **20** Donald K. Friesen and Bryan L. Deuermeyer. Analysis of greedy solutions for a replacement
573 part sequencing problem. *Math. Oper. Res.*, 6(1):74–87, 1981. doi:10.1287/moor.6.1.74.
574 **21** Gábor Galambos and Gerhard J. Woeginger. An on-line scheduling heuristic with better
575 worst case ratio than graham's list scheduling. *SIAM J. Comput.*, 22(2):349–355, 1993.
576 doi:10.1137/0222026.
577 **22** Waldo Gálvez, José A. Soto, and José Verschae. Symmetry exploitation for online machine
578 covering with bounded migration. *ACM Trans. Algorithms*, 16(4):43:1–43:22, 2020. doi:
579 10.1145/3397535.
580 **23** Oliver Göbel, Thomas Kesselheim, and Andreas Tönnis. Online appointment scheduling in
581 the random order model. In *Algorithms, 23rd Annual European Symposium (ESA)*, volume
582 9294, pages 680–692. Springer, 2015. doi:10.1007/978-3-662-48350-3_57.
583 **24** Todd Gormley, Nick Reingold, Eric Torng, and Jeffery R. Westbrook. Generating adversaries
584 for request-answer games. In *11th Annual ACM-SIAM Symposium on Discrete Algorithms*
585 *(SODA)*, pages 564–565. ACM/SIAM, 2000.
586 **25** Anupam Gupta, Ruta Mehta, and Marco Molinaro. Maximizing profit with convex costs in
587 the random-order model. In *45th International Colloquium on Automata, Languages, and*
588 *Programming (ICALP)*, volume 107, pages 71:1–71:14. Schloss Dagstuhl - Leibniz-Zentrum für
589 Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.71.
590 **26** Anupam Gupta and Sahil Singla. Random-order models. In *Beyond the Worst-Case Analysis of*
591 *Algorithms*, pages 234–258. Cambridge University Press, 2020. doi:10.1017/9781108637435.
592 015.
593 **27** Dorit S. Hochbaum and David B. Shmoys. Using dual approximation algorithms for scheduling
594 problems theoretical and practical results. *J. ACM*, 34(1):144–162, 1987. doi:10.1145/7531.
595 7535.
596 **28** Abdolhossein Hoorfar and Mehdi Hassani. Inequalities on the lambert w function and
597 hyperpower function. *J. Inequal. Pure and Appl. Math*, 9(2):5–9, 2008.
598 **29** David R. Karger, Steven J. Phillips, and Eric Torng. A better algorithm for an ancient
599 scheduling problem. *J. Algorithms*, 20(2):400–430, 1996. doi:10.1006/jagm.1996.0019.
600 **30** Claire Kenyon. Best-fit bin-packing with random order. In *7th Annual ACM-SIAM Symposium*
601 *on Discrete Algorithms (SODA)*, pages 359–364. ACM/SIAM, 1996.
602 **31** Thomas Kesselheim, Klaus Radke, Andreas Tönnis, and Berthold Vöcking. Primal beats dual
603 on online packing lps in the random-order model. *SIAM J. Comput.*, 47(5):1939–1964, 2018.
604 doi:10.1137/15M1033708.
605 **32** Robert D. Kleinberg. A multiple-choice secretary algorithm with applications to online
606 auctions. In *16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages
607 630–631. SIAM, 2005.
608 **33** Oded Lachish. O(log log rank) competitive ratio for the matroid secretary problem. In *55th*
609 *IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 326–335. IEEE
610 Computer Society, 2014. doi:10.1109/FOCS.2014.42.
611 **34** Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online
612 scheduling via learned weights. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete*
613 *Algorithms (SODA)*, pages 1859–1877. SIAM, 2020. doi:10.1137/1.9781611975994.114.
614 **35** D. V. Lindley. Dynamic programming and decision theory. *Journal of the Royal Statistical*
615 *Society*, 10(1):39–51, March 1961.

36 Adam Meyerson. Online facility location. In *42nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 426–431. IEEE Computer Society, 2001. doi:10.1109/SFCS.2001.959917.

37 Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. In *Beyond the Worst-Case Analysis of Algorithms*, pages 646–662. Cambridge University Press, 2020. doi:10.1017/9781108637435.037.

38 Marco Molinaro. Online and random-order load balancing simultaneously. In *28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1638–1650. SIAM, 2017. doi:10.1137/1.9781611974782.108.

39 Christopher J. Osborn and Eric Torng. List's worst-average-case or WAC ratio. *J. Sched.*, 11(3):213–215, 2008. doi:10.1007/s10951-007-0019-7.

40 John F. Rudin III. *Improved bounds for the on-lijne scheduling problem.* PhD thesis, The University of Texas at Dallas, 2001.

41 Peter Sanders, Naveen Sivadasan, and Martin Skutella. Online scheduling with bounded migration. *Math. Oper. Res.*, 34(2):481–498, 2009. doi:10.1287/moor.1090.0381.

42 Martin Skutella and José Verschae. Robust polynomial-time approximation schemes for parallel machine scheduling with job arrivals and departures. *Math. Oper. Res.*, 41(3):991–1021, 2016. doi:10.1287/moor.2015.0765.

43 Robert J. Vanderbei. The postdoc variant of the secretary problem. http://www.princeton.edu/~rvdb/tex/PostdocProblem/PostdocProb.pdf, Unpublished.

44 Gerhard J. Woeginger. A polynomial-time approximation scheme for maximizing the minimum machine completion time. *Oper. Res. Lett.*, 20(4):149–154, 1997. doi:10.1016/S0167-6377(96)00055-7.