Online computation beyond standard models

Shendan Jin

DOCTORAL THESIS

Online Computation Beyond Standard Models

Author:
Shendan JIN

Supervisor:
Christoph DÜRR
Spyros ANGELOPOULOS

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Computer Science

in the

Operations Research Group
Laboratoire d'informatique de Paris 6

April 30, 2020

Declaration of Authorship

I, Shendan JIN, declare that this thesis titled, "Online Computation Beyond Standard Models" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

Acknowledgements

This PhD thesis would not be possible without advice and support from people around me.

First of all, I would like to express my deep gratitude to my supervisors Christoph Dürr and Spyros Angelopoulos for their guidance and support. I am very happy that I became your student three years ago. Your passion about research was a great source of motivation. It was a pleasure to collaborate with you and see the way you approach research problems. Thank you for teaching me how to write a research report and how to present my work to broad audience. Especially, I would like to thank Christoph not only for your effort made while supervising me but also for your help and support in my life in Paris. I would also like to thank Spyros for your patience, help and effort made while meeting the deadlines of conference. I would like to thank for the financial support for attending conferences and workshops around the world. I could not have asked for better conditions for pursuing a PhD.

I would like to thank the members of my thesis committee. Spyros Angelopoulos, Evripidis Bampis, Cristina Bazgan, Christoph Dürr, Claire Mathieu and Christophe Picouleau, thank you all for coming to my defense, evaluating my thesis and providing me feedback. Special thanks to Claire Mathieu and Christophe Picouleau for being a member of the mid-term evaluation committee.

I am grateful to all my teachers, colleagues and friends whom I met earlier in my life and who introduced me to the world of Mathematics and Computer Science. Especially, I would like to thank my high school competitive programming coach, Weikang Huang, for his inspiring lectures in algorithms, which served as a springboard for my journey in the area of algorithms.

I would also like to thank my parents and grandparents for their unconditional support and love. Thanks to my grandfather for making me love Mathematics since childhood. Thanks to my parents for all their patience, understanding and financial supports.

Last and most importantly, I would like to thank my wife, Hanbo, for standing by me during the PhD life. During the PhD studies it was sometimes hard to keep a balance between work and personal life. Hanbo, thank you for your unlimited support. Thank you for sharing your life with me. Especially, thank you for giving a birth of our daughter, Annie. I look forward to everything that is to come.

SORBONNE UNIVERSITÉ

Résumé

Faculté des Sciences et Ingénierie
Laboratoire d'informatique de Paris 6

Doctorat en Informatique

Calcul en Ligne au-delà des Modèles Standards

par Shendan JIN

Dans le cadre standard du calcul en ligne, l'entrée de l'algorithme n'est pas entièrement connue à l'avance, mais elle est révélée progressivement sous forme d'une séquence de requêtes. Chaque fois qu'une requête arrive, l'algorithme en ligne doit prendre des décisions irrévocables pour servir la requête, sans connaissance des requêtes futures. Dans le domaine des algorithmes en ligne, le cadre standard utilisé pour évaluer les performances des algorithmes en ligne est *l'analyse compétitive*. De manière informelle, le concept d'analyse compétitive consiste à comparer les performances d'un algorithme en ligne dans le pire des cas à une solution optimale hors ligne qui aurait pu être calculée si toutes les données étaient connues d'avance.

Dans cette thèse, nous étudierons de nouvelles façons d'approcher les problèmes en ligne. Dans un premier temps, nous étudions le calcul en ligne dans le modèle avec ré-optimisation, dans lequel l'irrévocabilité des décisions en ligne est relâchée. Autrement dit, l'algorithme en ligne est autorisé à revenir en arrière et changer les décisions précédemment prises. Plus précisément, nous montrons comment identifier le compromis entre le nombre de réoptimisation et les performances des algorithmes en ligne pour le problème de couplage maximale en ligne.

De plus, nous étudions des mesures autres que l'analyse compétitive pour évaluer les performances des algorithmes en ligne. Nous observons que parfois, l'analyse compétitive ne peut pas distinguer les performances de différents algorithmes en raison de la nature la plus défavorable du ratio de compétitivité. Nous démontrons qu'une situation similaire se pose dans le problème de la *recherche sur la ligne*. Plus précisément, nous revisitons le problème de la recherche sur la ligne et introduisons une mesure, qui peut être appliquée comme un raffinement du ratio de compétitivité.

Enfin, nous étudions le calcul en ligne dans le modèle avec des conseils, dans lequel l'algorithme reçoit en entrée non seulement une séquence de requêtes, mais aussi quelques conseils sur la séquence de requêtes. Plus précisément, nous étudions un modèle récent avec des conseils non fiables, dans lequel les conseils peuvent être fiables ou non. Supposons que dans ce dernier cas, les conseils peuvent être générés à partir d'une source malveillante. Nous montrons comment identifier une stratégie optimale de Pareto pour le problème *online bidding* dans le modèle de conseil non fiable.

Mots-clés : calcul en ligne ; ratio de compétitivité ; ré-optimisation ; mesures de performance ; algorithme en ligne avec conseil ; recherche sur la ligne.

SORBONNE UNIVERSITY

Abstract

Sciences and Engineering Faculty
Laboratoire d'informatique de Paris 6

Doctor of Computer Science

Online Computation Beyond Standard Models

by Shendan JIN

In the standard setting of online computation, the input is not entirely available from the beginning, but is revealed incrementally, piece by piece, as a sequence of requests. Whenever a request arrives, the online algorithm has to make immediately irrevocable decisions to serve the request, without knowledge on the future requests. Usually, the standard framework to evaluate the performance of online algorithms is *competitive analysis*, which compares the worst-case performance of an online algorithm to an offline optimal solution.

In this thesis, we will study some new ways of looking at online problems. First, we study the online computation in the recourse model, in which the irrevocability on online decisions is relaxed. In other words, the online algorithm is allowed to go back and change previously made decisions. More precisely, we show how to identify the trade-off between the number of re-optimization and the performance of online algorithms for the online maximum matching problem.

Moreover, we study measures other than competitive analysis for evaluating the performance of online algorithms. We observe that sometimes, competitive analysis cannot distinguish the performance of different algorithms due to the worst-case nature of the competitive ratio. We demonstrate that a similar situation arises in the linear search problem. More precisely, we revisit the linear search problem and introduce a measure, which can be applied as a refinement of the competitive ratio.

Last, we study the online computation in the advice model, in which the algorithm receives as input not only a sequence of requests, but also some advice on the request sequence. Specifically, we study a recent model with untrusted advice, in which the advice can be either trusted or untrusted. Assume that in the latter case, the advice can be generated from a malicious source. We show how to identify a Pareto optimal strategy for the online bidding problem in the untrusted advice model.

Keywords: online algorithms; competitive ratio; online algorithms with recourse; performance measures; online algorithms with advice; linear search.

Contents

Chapter 1

Introduction

1.1 Online Computation

In a typical optimization problem, one is given a specific input and has to output a solution, satisfying some constraints and optimizing some objective function, which can be either a cost minimization or a profit maximization. The aspect of this setting which we would like to emphasize is that the input is entirely given in advance. Let us illustrate by an example of such a problem.

At Sorbonne University, each student majoring in Computer Science has to complete a programming project by the end of the semester. The student can either work by himself or in a group of two. Each student has his own preference for programming languages and two students can collaborate if they have a common preferred programming language. To encourage collaboration, the instructor wants to maximize the number of student duos. This problem can be formulated as a *matching problem* in a graph. More precisely, the vertices are all students and an edge between two vertices means that the two corresponding students have a common preferred programming language. More formally, the above problem can be formalized as follows. We are given a graph with vertices and edges, and the objective is to find a set of vertex-disjoint edges of maximum cardinality in the graph. This maximization problem is also known as the *maximum matching problem*, which is a fundamental combinatorial optimization problem.

However, in many real-life applications, the entire input is not available from the beginning. In contrast, the input is only revealed incrementally, piece by piece, as a sequence of requests. For each such request, the algorithm has to immediately make decisions without any knowledge on future requests, and usually, these decisions are *irrevocable* (i.e. the algorithm is not allowed to go back to change the previously made decisions).

Let us see how it can be applied to the previous example in the context of matching. The instructor puts students in a queue. The students are queried by the instructor one by one for their preference in programming languages. As soon as the instructor learns the preferences of some student, the instructor has to assign the student to someone that the instructor has queried before, who has a common preferred programming language and is not yet paired, whenever it is possible. Again, this problem can be formulated as a matching problem in the vertex arrival model, in which the vertices are revealed one by one with its incident edges. (A more detailed discussion on different matching models is in Chapter 2.) More precisely, here, each request represents an arriving vertex in a graph. Whenever a vertex appears, the corresponding incident edges to the previous vertices are revealed together. For each such request, the algorithm has to either accept it, by choosing an incident edge that has no common endpoint with any selected edges and adding it in the current solution or reject it.

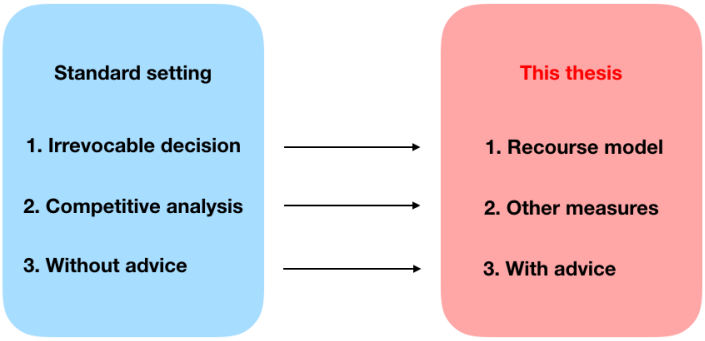


FIGURE 1.1 – The standard online computation model and some recent new approaches on online problems.

Problems of this form are known as *online optimization problems*, or simply *online problems*. The algorithm that we use to solve such an online problem is known as an *online algorithm*. An *offline optimal algorithm* is the one that optimally solves the online problem assuming that the whole input sequence is given in advance.

Online algorithms have been studied for decades in the context of matching, scheduling and other topics [34, 64]. A formal description of the standard online computation model is given as follows. In an online problem, the input is not entirely available from the beginning, but is only revealed incrementally, piece by piece, as a sequence of requests $\sigma_1, \sigma_2, \ldots, \sigma_n$, where n is the number of such requests. Whenever a request σ_i arrives, the online algorithm has to make immediately irrevocable decisions to serve the request, without knowledge on the future requests $\sigma_{i+1}, \ldots, \sigma_n$. The next request σ_{i+1} arrives only after that the item σ_i is completely processed. Usually, the standard framework to evaluate the performance of online algorithms is competitive analysis, which compares the worst-case performance of an online algorithm to an offline optimal solution. A more formal description of the competitive analysis is given in Section 1.1.1.

For the standard online computation model, we emphasize that:

1. The decisions made by the online algorithm are irrevocable;
2. Competitive analysis is the standard framework for the performance measure;
3. There is no additional knowledge on the request sequence for the online algorithm.

The above standard model captures many online problems, but it does not capture certain applications. For instance, sometimes, the online algorithm wants to go back and change some decisions that it made before. Typically, this action is not captured by the standard online computation model. Recently, there are more ways of looking at online problems. In this thesis, we will study some new ways of looking at online problems, which are listed as follows. (See also Figure 1.1.)

1. **Online computation with recourse:** In this model, the irrevocability on online decisions is relaxed. In other words, the online algorithm is allowed to go back and change previously made decisions. Intuitionally, the more re-optimization can be done, the better performance can be achieved. The objective is to study the trade-off between the number of re-optimization and the performance of online algorithms. We will show how to do this for the *online maximum matching* problem. A more detailed introduction on the model and on the online matching problem is given in Section 1.2.

2. **Other measures:** Sometimes, competitive analysis cannot separate the performance of different online algorithms because of the worst-case nature of the competitive ratio. More precisely, for a given problem, sometimes there are many algorithms with optimal competitive ratio, which show different behavior in practice. To remedy this situation, we will study other measures. We will introduce a measure for the *linear search* problem, which can be applied as a refinement of the competitive ratio. See Section 1.3 for a more detailed introduction on the measure and on the *linear search* problem.
3. **Online computation with advice:** In the advice setting, the algorithm receives as input not only a sequence of requests, but also some advice on the request sequence, which encodes some information on the input. For instance, it may encode information on optimal decisions that the online algorithm should make. We will work on a specific application of a recent advice model, namely *online bidding with untrusted advice*, in which the advice can be either *trusted* or *untrusted* (i.e. the advice can be generated from some malicious source). See Section 1.4 for a more detailed introduction.

1.1.1 Competitive Analysis

In the area of online algorithms, the standard framework used to evaluate the performance of online algorithms is *competitive analysis*. Informally, the concept of competitive analysis consists of comparing the worst-case performance of an online algorithm with an offline optimal solution. The competitive analysis framework was first introduced by Graham in 1966 [34] to analyze algorithms for the job shop scheduling problem. Eventually, competitive analysis became the standard measure in the evaluation of online algorithms following the work of Sleator and Tarjan in 1985 [60].

Formally, consider a cost minimization online problem (or a profit maximization online problem, respectively). For any request sequence σ , let $OPT(\sigma)$ denote the optimal cost (profit, respectively) on σ. For an online algorithm ALG, let $ALG(\sigma)$ denote the cost (profit, respectively) of ALG on instance σ. The *competitive ratio* is defined as follows.

Definition 1.1 (Competitive ratio)**.** For a cost minimization problem, an online algorithm ALG is said to be *asymptotically c-competitive* if there is a constant d such that $\mathrm{OPT}(\sigma) \geq \mathrm{ALG}(\sigma)/c + d$, for all finite request sequences σ.

For a profit maximization problem, an online algorithm ALG is said to be *asymptotically c-competitive* if there is a constant d such that $\mathrm{ALG}(\sigma) \geq \mathrm{OPT}(\sigma)/c - d$, for all finite request sequences σ.

Whenever $d = 0$ the algorithm is called *strictly* c-competitive. The smallest c for which an online algorithm ALG is c-competitive is called the *competitive ratio* of ALG. The strict competitive ratio is defined similarly. If it so happens that this minimum value does not exist, the competitive ratio is actually defined by the corresponding infimum.

Usually, there is no restriction on the computing time and space used by the online algorithm. However, in practice, we aim for efficient algorithms that achieve the optimal or a nearly optimal competitive ratio.

Note that there is also an alternative interpretation of competitive analysis [16, 64]. The online problem can be considered as a game played between an algorithm and a malicious *adversary*. The algorithm makes online decisions, while the adversary constructs the worst possible input sequence, based on the knowledge of the

player's algorithm. For instance, in a profit maximization problem, the goal for the player is to minimize the competitive ratio while the adversary wants to maximize it.

There are not only deterministic online algorithms but also randomized online algorithms. A randomized algorithm is an algorithm that employs some randomness for decision making. Typically a randomized algorithm uses uniform random bits as an auxiliary input to guide its behavior. Although this is an important part of the online computation, in this thesis, we only study deterministic algorithms, we will not discuss issues related to randomized algorithms. The book of Borodin and El-Yaniv [16] covers many problems analyzed in competitive analysis.

1.1.2 Techniques in Design and Analysis of Online Algorithms

There are many online algorithms and analysis techniques, some of them are tailormade for specific problems. However, there are some generic frameworks which apply to many different online problems. In this section, we list out some categories of online algorithms and analysis techniques that we use in the thesis. Note that this is not an exhaustive list in the design and analysis of online algorithms.

Greedy Algorithms

Informally, a greedy algorithm builds up a solution piece by piece, by making the locally optimal choice at each step in the hope that this local choice will eventually lead to a globally optimal or near-optimal solution. For an optimization problem, we have an objective function that needs to be optimized. A greedy algorithm makes locally greedy choices at each step to ensure that the objective function is optimized. Usually, the greedy algorithm does not produce an optimal solution, but it is one of the simplest and the most straightforward methods to provide an approximate solution. Very often, the greedy algorithm is computationally efficient, since the local optimization problem is often easy to solve.

In this thesis, we use certain variants of greedy algorithms for the *online matching* problem under the recourse setting. Here, the *amortized analysis* is used to analyze the performance. This technique of analysis was first introduced by Tarjan in 1985 [61]. Instead of looking at the cost of each operation individually, the principle of amortized analysis consists of considering both the costly and less costly operations together over the whole process of the algorithm. See a more detailed description and application on amortized analysis in Chapter 2.

Doubling Algorithms

Doubling is one of the useful methods for designing online and offline approximation algorithms. Informally, the idea is to produce a solution, piece by piece, with the help of a geometrically increasing estimate on the optimal solution. The term "doubling" comes from the fact that sometimes the geometric estimate sequence $(2^i)_{i\in\mathbb{N}}$ is used when this approach was first proposed for some specific problems [11, 23]. However, for many problems, we often need a geometric estimate sequence with a common ratio different from 2.

The survey of Marek Chrobak and Claire Mathieu [24] illustrates this method on different online and offline optimization problems.

In Chapter 2, we will see how this idea can be applied to the *online matching with recourse* problem. In addition, in Chapter 3, we will analyze a simple doubling

algorithm for the *linear search* problem. We provide different techniques of analysis in different contexts. More details of designing and analyzing doubling algorithms are given in the following chapters.

Design and Analysis of Online Algorithms via Linear Programming (LP)

Some problems can be elegantly solved through a linear programming formulation. The goal is to assign values to variables so to optimize some linear objective value and to satisfy some linear constraints. There are several equivalent formulations of a combinatorial problem, for example, consider the following general linear program for a cost minimization problem, where the coefficients $a_{i,j}, b_j$ and c_i are related to the parameters of the problem:

$$(P) : \min \sum_{i=1}^{n} c_i x_i \quad s.t.$$

$$\sum_{i=1}^{n} a_{ij} x_i \geq b_j, \quad \forall 1 \leq j \leq m,$$

$$x_i \geq 0 \quad \forall 1 \leq i \leq n.$$

Such a linear program is also known as a *Covering* problem if $a_{ij}, b_j, c_i \geq 0$ for all $i = 1 \ldots n$ and $j = 1 \ldots m$. Any vector $x = (x_1, x_2, \ldots, x_n)$ that satisfies all the constraints of (P) is referred to as a feasible solution to the linear program (P). Every linear program has a corresponding dual linear program, and the original linear program is called the primal linear program. The dual linear program of (P) is a profit maximization linear program: it has m dual variables that correspond to the primal constraints and it has n constraints that correspond to the primal variables. The dual program (D) corresponding to the linear program formulation (P) is

$$(D) : \max \sum_{j=1}^{m} b_j y_j \quad s.t.$$

$$\sum_{j=1}^{m} a_{ij} y_j \leq c_i, \quad \forall 1 \leq i \leq n,$$

$$y_j \geq 0 \quad \forall 1 \leq j \leq m.$$

Such a linear program is also known as *Packing* problem if $a_{ij}, b_j, c_i \geq 0$ for all $i = 1 \ldots n$ and $j = 1 \ldots m$.

Designing and analyzing online algorithms using linear programming has been well studied for decades, see [63]. In particular, the primal-dual method is used to design algorithms. The principle of this method consists of improving a feasible dual solution jointly with a potential primal solution, using the complementary slackness conditions as guidance and terminating when the primal becomes feasible. This technique has proved to be extremely useful for a wide variety of problems in the area of approximation algorithms. The survey of Buchbinder and Naor [19] demonstrates that this approach is still applicable in the area of online computation.

In this thesis, we are not going to solve linear programs using duality. In contrast, for the problems at hand, we will prove certain properties about the optimal solutions. More precisely, we will look into the structure of the optimal solutions to

linear programming. For instance, in some cases, we can argue that optimal solutions saturate some constraints. Sometimes, these equalities are useful for establishing some complex but still efficiently computable recurrence relations on variables. In some cases, it allows us to compute an optimal solution to the linear programming by solving these recurrence relations. We emphasize that, in this case, we do not need a solver to solve the linear programming, and that the linear programming is only used as a guide in the design and analysis of the algorithm. In Chapter 3, we will see how this idea can be applied to the *linear search* problem. In Chapter 4, we provide another application on a variant of the *contract scheduling* problem.

1.2 Online Computation with Recourse

In the standard setting of online computation, the input to the algorithm is not entirely known in advance, but it is revealed incrementally as a sequence of requests. For each such request, the algorithm has to make a decision, which is usually *irrevocable*. More precisely, the decision that the algorithm makes is associated with the request and the algorithm cannot change previously made decisions during later requests. Informally, this setting can be described as "the past cannot be undone". Indeed, there are many real-life applications in which it makes sense to forbid re-optimizing the online solution during the execution of the algorithm. For instance, consider the taxi customer assignment task. One cannot undo the assignment once the customer's journey is completed.

Recently, an online computation model with the possibility to undo previous decisions was introduced. The model in which the online algorithm is allowed to do some modifications on previously made decisions is known as the *recourse model*. The algorithm is expected to have a better performance by re-optimizing the current solution. However, there must be some limitation on the use of such modification, otherwise, we can obtain the optimal solution all the time. Hence a natural question here is to quantify the trade-off between the competitive ratio and a measure on the modifications allowed on the solution. Typically, each modification leads to an additional cost. There are two approaches to the trade-off between the guaranteed competitive ratio and the cost of re-optimizing the current solution. One such approach has studied the minimum total re-optimization cost required to maintain an optimal solution after each request, see also Bernstein *et al*. [12]. Another approach has focused on the best achievable competitive ratio when there is some budget on the allowed re-optimization, which has been studied by Avitabile *et al*. [7], and is the main model we consider in this thesis.

Several online combinatorial optimization problems have been studied under recourse settings. For instance, there is a fundamental work for *general online set packing problem* in the recourse model by Avitabile *et al*. [7]. The problem consists of a universe C and a collection of subsets of C with different sizes and weights. The objective is to produce a collection of disjoint sets of maximum weight. In their recourse setting, the algorithm is allowed to remove sets from the current solution. Note that this generalizes the *matching problem* when all sizes are equal to 2. In [7], the problem is formulated by a linear program and each decision variable is allowed to be changed a constant number of times.

Another work studies the *online Steiner tree problem*. In the *online Steiner tree problem*, the input is a complete graph together with edge lengths. A sequence of *terminal* vertices arrive one by one, and we are required to maintain a subgraph connecting the terminals. More precisely, whenever a new terminal arrives, we need to buy

edges to add to the current solution so that the newly arrival terminal is connected to the previous terminals. The objective is to minimize the total length of the bought edges. In this standard setting of the *online Steiner tree problem*, we know that the simple greedy algorithm is optimal and that it has competitive ratio $\Theta(\log k)$, where k is the number of terminals [39]. Usually, we consider the vertex arrival model, although Imase and Waxman in [39], studied the variant in which terminals may also leave. In the context of re-optimization, Gupta *et al*. studied the recourse model for online Steiner trees in the vertex arrival model and they showed that with a single edge swap (i.e. undo a bought edge and buy a new edge) per request we can bring the competitive ratio down to a constant [35].

Other problems have been studied in the recourse setting, such as the *minimum spanning tree problem*, the *traveling salesman problem* [55], the *knapsack problem* [37, 40] and the *assignment problems* in bipartite unweighted graphs [36].

1.2.1 Online Matching with Recourse

In this thesis, we will study the recourse model for the *online maximum matching problem*. The offline matching problem is one of the most fundamental algorithmic problems. It has played a central role in the development of the theory of algorithms [28, 29, 50]. Moreover, it is also central to resource allocation in the scheduling and Operations Research literature. One such typical example is allocating jobs to machines in the context of cloud computing. Moreover, the online version of the problem has generated a lot of interest in the algorithms community, with the introduction of a large number of new problems, models and algorithmic techniques [56].

We define the *online matching problem with k edge-recourse*, for a given $k \in \mathbb{N}^*$. Here, the request sequence is an order on the edge set E in a graph $G = (V, E)$. The online algorithm knows the set V and the parameter k, but not the set E. At each time point, one edge of E is revealed according to the ordering, and the online algorithm must maintain a matching $M \subseteq E'$, where $E' \subseteq E$ is the set of currently revealed edges. Specifically, for each revealed edge e in the sequence, the online algorithm either *accepts* e, by adding it in its matching, or *rejects* it. In addition, the algorithm must obey the edge recourse constraint: every edge has an integer type, which is set to 0 upon the arrival of an edge. Whenever the algorithm decides either to include an edge e to its matching or to remove it, the type of e is increased by 1. The algorithm can perform these operations at any time subject to the constraint that no edge type exceeds k.

The objective of the problem is to design an online algorithm of minimum competitive ratio, which is the worst-case ratio $\mathrm{OPT}(\sigma)/\mathrm{ALG}(\sigma)$, over all request sequences σ. Recall that $\mathrm{OPT}(\sigma)$ denotes the cardinality of the optimal matching for the request sequence σ, and $\mathrm{ALG}(\sigma)$ denotes the cardinality of the matching output by the online algorithm on the request sequence σ. We emphasize that in this thesis we consider the maximum cardinality matching problem. Some previous work has considered the generalized *weighted matching* problem, in which each edge has a weight and the objective is to maximize the weight of matched edges [42, 65].

1.3 Other Performance Measures

As mentioned before, competitive analysis is the standard framework used for evaluating the performance of online algorithms. Informally, the principle of competitive analysis consists of comparing the worst-case performance of an online algorithm with an offline optimal solution. However, for certain online problems, sometimes competitive analysis cannot distinguish the performance of different algorithms, which show different behavior in practice. This is due to the pessimistic, worst-case nature of the competitive ratio.

For instance, consider the *Paging problem* in a virtual memory system. There are k cache locations, each of them can hold one page. A sequence of requested pages arrives one by one. If a requested page is in the cache then this is called a *hit*, otherwise, it is called a *fault*. In case of a fault, a page already in the cache must be ejected, and we put the requested page in the cache, which costs 1. Otherwise, we do nothing and the cost is 0. The goal is to minimize the total cost. The paging problem has been well studied. The paging strategy *Least-Recently-Used*, which ejects the page whose past use was least recent in the case of a fault, is very efficient in practice. However, it has the same competitive ratio as many extremely naive strategies, such as *Flush-When-Full*, which evicts the entire cache whenever the cache is full [60].

The above example shows that the theoretical performance evaluation is not compatible with the empirical performance evaluation. It has motivated a lot of research on alternative measures to the competitive ratio (see [27] for a survey). In this thesis, we demonstrate that a similar situation arises in the context of online search. More specifically, we revisit one of the simplest and fundamental search problems, namely the *linear search*, also known as the *cow-path* problem [9]. A formal description of the problem is given below. To remedy the undesired situation where competitive analysis cannot separate the performance of different algorithms, we introduce the *discovery ratio*, which is a pairwise comparison of strategies (see Section 1.3.2). We emphasize that we apply the discovery ratio as a refinement of the competitive ratio, instead of using it as a measure that replaces the competitive ratio altogether.

1.3.1 The Linear Search Problem

In general, a search problem consists of an environment, a mobile *searcher* and an immobile *hider* (sometimes also called *target*). The searcher may have knowledge of the environment, and the target hides at some position that is not known to the searcher. The objective is to define a *search strategy*, which is a traversal of the environment, that optimizes a certain efficiency criterion.

In the *linear search problem*, the environment is an unbounded line, with a point O designated as its origin. A mobile searcher is initially placed at the origin and an immobile hider is at some position on the line that is unknown to the searcher. More specifically, the searcher does not know whether the hider is at the left branch (i.e. branch 1) or at the right branch (i.e. branch 0) of the line (see Figure 1.2 for illustration). The searcher's *strategy* S defines its exploration of the line, whereas the hider's strategy H is determined by its placement on the line. Given strategies S, H, the *cost* of locating the hider, denoted by $c(S, H)$, is the total distance traversed by the searcher at the first time it passes over H. Let $dist(H)$ denote the distance of the hider from the origin. The competitive ratio of S, denoted by $\mathrm{cr}(S)$, is the worst-case

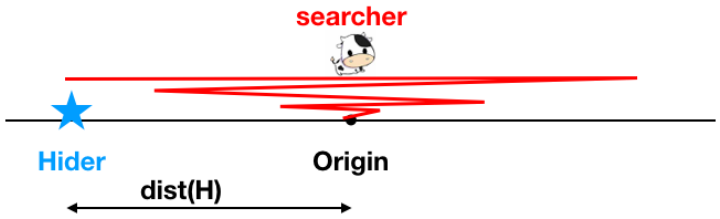


FIGURE 1.2 – Illustration of the linear search problem.

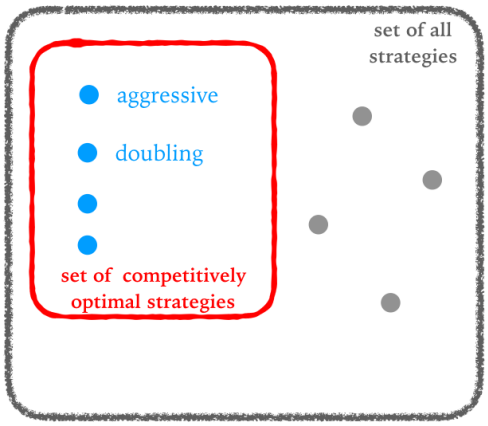


FIGURE 1.3 – Both `doubling` and `aggressive` are competitively optimal.

normalized cost of S, among all possible hider strategies. Formally,

$$\mathrm{cr}(S) = \sup_{dist(H)\geq 1} \frac{c(S,H)}{dist(H)}. \tag{1.1}$$

Note that there is a standard assumption that the hider must be at distance at least 1 from O, since no strategy can have bounded competitive ratio if this distance can be arbitrarily small.

It has been known since 1964 [9, 32] that the competitive ratio of linear search is 9. The following simple *doubling* strategy achieves the competitive ratio 9: in iteration i, the searcher starts from O, explores branch i mod 2 at a length equal to 2^i, and then returns to O. However, this strategy is not uniquely optimal. It is known that there is an infinite number of competitively optimal strategies for linear search (see also Lemma 3.3 in Section 3.3.1). In particular, consider an *aggressive* strategy, which searches a branch to the maximum possible extent in each iteration, while maintaining a competitive ratio equal to 9. This can be achieved by searching, in iteration i, branch i mod 2 to a length equal to $(i+2)2^{i+1}$ (see also Corollary 3.5 in Section 3.3.1). See Figure 1.3 for an illustration.

1.3.2 The Discovery Ratio

As mentioned above, both `doubling` and `aggressive` are optimal in terms of competitive ratio. However, the latter may be better than the former in some real-life applications. For instance, consider a search-and-rescue mission for a missing backpacker who has disappeared in one of two concurrent hiking paths. Assuming that we select our search strategy from the space of 9-competitive strategies, it makes

sense to choose one that is eager to discover new territory, rather than a conservative strategy that tends to often revisit already explored areas.

To capture the eagerness of a strategy to discover new territory, we introduce the *discovery ratio* as follows:

Definition 1.2. Let S_1, S_2 denote two search strategies, we define the *discovery ratio of S_1 against S_2*, denoted by $\mathrm{dr}(S_1, S_2)$, as

$$\mathrm{dr}(S_1, S_2) = \sup_{\ell \in \mathbb{R}^+} \frac{D(S_1, \ell)}{D(S_2, \ell)},$$

where $D(S, \ell)$ denotes the cost incurred by S the first time the searcher has explored an aggregate length equal to ℓ combined in both branches. Moreover, given a class $\mathcal{S}$ of search strategies, the *discovery ratio of S against the class $\mathcal{S}$* is defined as

$$\mathrm{dr}(S, \mathcal{S}) = \sup_{S' \in \mathcal{S}} \mathrm{dr}(S, S').$$

In the case $\mathcal{S}$ is the set Σ of all possible strategies, we simply call $\mathrm{dr}(S, \mathcal{S})$ the *discovery ratio of S*, and we denote it by $\mathrm{dr}(S)$.

Intuitively, an efficient strategy should be such that $D(S, \ell)$ is small, for all ℓ, meaning it has a small discovery ratio. However, it is insufficient to consider this criterion itself: consider a strategy that first searches one branch to a length equal to L, where L is very large. Then $D(S, \ell)$ is as small as possible for all $\ell < L$; however, this is hardly a good strategy, since it ignores one of the branches and its competitive ratio becomes unbounded as $L \to \infty$. To this end, we will apply the discovery ratio as supplementary to the competitive ratio. In other words, we will restrict our interest to the set of competitively optimal strategies and use the discovery ratio to separate the performance of those competitively optimal strategies.

In Chapter 3, we will show how to distinguish the performance of `doubling` and `aggressive` using discovery ratio.

1.3.3 An Application from Artificial Intelligence

As discussed in Section 1.1.2, for some problems, it is convenient to formulate them using linear programming. Sometimes, we can look into the structure of the optimal solutions to linear programming. For the linear search problem, we can argue that optimal solutions saturate some constraints, which are useful to derive some recurrence relations on variables of the linear programming. As mentioned before, we show how this idea can be applied to the linear search problem in Chapter 3. Moreover, in Chapter 4, we also provide an application from Artificial Intelligence of this idea in the design of the interruptible system using contract algorithms. The next section gives an introduction to this topic.

Design of Interruptible System using Contract Algorithms

An anytime algorithm is an algorithm that can return a valid solution to a problem even if it is interrupted before it ends. The algorithm is expected to find better solutions the longer it keeps running. For instance, consider the *local search* algorithm, which consists in moving from solution to solution in the space of candidate solutions by applying local changes, until an optimal solution is found or a time bound is elapsed. The local search is then an anytime algorithm, and it is often used

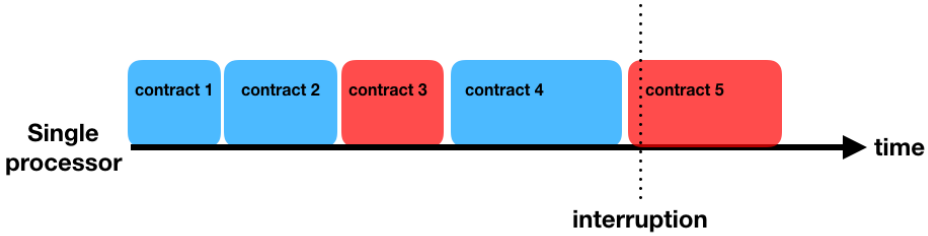


FIGURE 1.4 – Two contract algorithms (blue/red) are scheduled in a single processor. Each block represents a contract (i.e. a piece of execution).

to design interruptible systems. In contrast, the system produced by *Dynamic Programming* is not interruptible. One natural question here is to find a way to provide anytime capability using only algorithms that are not interruptible.

Anytime capability plays a central role in the design of intelligent systems. An interruptible system is required to output a solution if it is interrupted during its execution. There are many ways to create an interruptible system. In particular, we will consider an interruptible system produced by *contract algorithms*.

A contract algorithm receives an intended queried time as one of its input parameters. The contract algorithm may give a meaningless result if it is queried before this time, which implies that the contract algorithm is not interruptible and motivates research on using contract algorithm to produce an interruptible system. For this purpose, a general technique was first given in [58]. The idea is to run multiple times the contract algorithm by iteratively increasing the available execution time of the contract algorithm. Each piece of execution is then called a *contract*. Thus, there is a trade-off between the quality of output and the total available computation time, and usually we know the latter in advance. The performance is evaluated by means of *acceleration ratio*, which is defined as the worst-case ratio between the interruption time t and the contract length of the contract algorithm that has made the least progress by time t. See Figure 1.4 for an illustration.

Assuming that an interruption can occur arbitrarily in the future, previous work focuses on the strict performance guarantees for different variants of this problem. However, in some real-life application, the schedule reaches a point beyond which further progress will only be marginal. In other words, the schedule can be considered complete beyond a certain point during the execution.

In Chapter 4, we will show how to optimize the time at which the system reaches the desired performance objective, while maintaining interruptible guarantees throughout the execution. More precisely, we consider the setting in which there are n contract algorithms scheduled in a single processor. Each contract algorithm has to attain a predefined *end guarantee* upon completion, where the end guarantee is defined as the minimum required value to attain upon completion on the length of the largest contract completed. In other words, for a given end guarantee L, each contract algorithm has to complete a contract of length at least L upon completion.

1.4 Online Computation with Advice

As mentioned in Section 1.1, in the standard online computation model, for each arrival request, the algorithm has to immediately make irrevocable decisions without any knowledge on future requests. However, in practice, some additional information on the input sequence is sometimes given to online algorithms, which may encode partial information on optimal decisions that the online algorithm should take. Typically, this situation is not captured by the standard online computation model. A new model is then required to quantify the power of this additional information.

This type of information that is given along with the input sequence to the online algorithm, is called *advice*. There is a trade-off between the number of advice bits and the performance of online algorithms. Intuitionally, the more advice bits are available, the better performance can be achieved. A natural question is then to quantify the trade-off between the performance of online algorithms and advice size, which has been studied since 2009. In the last decade, many online optimization problems have been studied in the advice model, such as k-server, paging, makespan scheduling, etc. The survey of Boyar *et al.* [18] provides a further discussion on this topic.

However, all previous work assumed that the advice is always *correct*. This assumption is unrealistic in practice because the advice can be generated by some untrusted source, or in the worst case, a malicious adversary may take control of the advice oracle, which may have a catastrophic impact on the performance of the online algorithm. This motivates the topic on the untrusted advice model, in which the advice may not be treated as infallible (see Section 5.1.1 for a more detailed introduction on the advice model).

1.4.1 A New Model with Untrusted Advice

In the machine-learning community, a new advice model was proposed to capture the above observations by Lykouris and Vassilvitskii [52], and Purohit *et al.* [57]. In their work, they use predictors to design and analyze online algorithms. If the predictor is bad, then the online algorithm should perform close to the one without predictions; if the predictor is good, then the online algorithm should perform close to the optimal offline algorithm.

Motivated by the above work from machine learning, in this thesis, we consider the advice model in which the advice can be either *trusted* or *untrusted*. In this setting, the performance of an online algorithm can be characterized by a pair of competitive ratios, namely the competitive ratio when the advice is trusted and the competitive ratio when the advice is untrusted. We also assume that, in the latter case, the untrusted advice is generated by a malicious adversary to be in accordance with the worst-case nature of the incompetitive analysis.

Thus the competitiveness of an algorithm can be presented as a point in the 2-dimensional space with coordinates depicted by the competitive ratios when the advice is either trusted or untrusted. We are interested in finding a Pareto frontier of the set of all possible algorithms assuming that the advice has an infinite number of bits so that the advice can encode the optimal solution. A more formal description of the model and of the competitiveness will be given in Section 5.1.2.

In this thesis, we will work on a specific application of this model, namely the *online bidding with untrusted advice* problem, which is presented as follows.

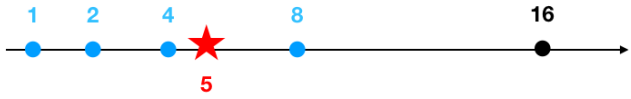


FIGURE 1.5 – In this example, the player submits the sequence $(2^i)_{i\geq 0}$ to guess the hidden unknown value $u = 5$. The player's cost is equal to $1 + 2 + 4 + 8 = 15$, while the offline optimal cost is 5.

1.4.2 Online Bidding with Untrusted Advice

In the standard *online bidding* problem, a player has to guess an unknown hidden value u by submitting a sequence $X = (x_i)$ of increasing *bids*. The strategy of the player is defined by this sequence of bids, and the cost of guessing the value u is equal to $\sum_{i=1}^{j} x_i$, where j is such that $x_{j-1} < u \leq x_j$. In other words, the cost is equal to the sum of all bids less than u plus the first bid that exceeds u as illustrated in Figure 1.5. The competitive ratio of the player's strategy X is

$$w_X = \sup_{u\geq 1} \frac{\sum_{i=1}^{j} x_i}{u}, \text{ where } j \text{ is such that } x_{j-1} < u \leq x_j.$$

Note that there is a standard assumption that u is at least 1, since no strategy can have bounded competitive ratio if u can be arbitrarily small.

In Chapter 5, we will study the *online bidding* problem in the untrusted advice setting described as in the previous section. More precisely, we will study the competitiveness of algorithms for both trusted and untrusted advice. Consider the whole set of algorithms, each algorithm can be represented by a point in a 2-dimensional space. The objective is to identify a Pareto frontier of this set (see Chapter 5 for more details).

1.5 Summary of the Thesis

The thesis consists of studying some recent approaches to online problems.

In Chapter 2, we study the setting in which the online algorithm is allowed to go back and change decisions that it made before. The objective is to study the power and limitation of such re-optimization. More specifically, we study the online matching problem with edge k-bounded recourse. First, we revisit the doubling algorithm of [7] that was originally analyzed in the general context of online packing problems. We give a better analysis using concepts and ideas related to the matching problem. Moreover, we propose and analyze a variant of the greedy algorithm which we call L-Greedy. While this algorithm is thus not superior to the doubling algorithm for large k (and more specifically, to its improved analysis in the context of the matching problem), for small k it does achieve an improved competitive ratio. In terms of techniques, we analyze both the doubling algorithm and L-Greedy using amortization arguments in which the profit of the algorithms is expressed in terms of weights appropriately distributed over nodes in the graph. We achieve these improvements by exploiting properties of augmenting paths in matching algorithms. Last, we study the problem in the *edge arrival/departure model*, which is the fully dynamic variant of the online matching problem. We obtain improved lower bounds by modeling the game between the algorithm and the adversary as a game played over *strings* of numbers 0 up to k.

In Chapter 3, instead of using competitive ratio as a standard framework of the performance measure, we introduce the discovery ratio and use it as a supplementary measure. The objective is to remedy a situation where the theoretical performance evaluation using competitive analysis is not compatible with the empirical performance evaluation. More specifically, we revisit the linear search problem. First, we obtain strategies of optimal discovery ratio against all possible strategies. Specifically, we show that there are strategies of discovery ratio $2 + \epsilon$, for arbitrarily small $\epsilon > 0$, which is tight. However, they have an unbounded competitive ratio. Moreover, we restrict our interest to the set of competitively optimal strategies, which we further analyze using the discovery ratio as a supplementary measure. We prove that the strategy `aggressive`, which explores the branches to the furthest possible extent while satisfying the competitiveness constraint, has discovery ratio $\frac{8}{5}$. We show that this is the optimal discovery ratio in this setting. In contrast, we show that the strategy `doubling` has discovery ratio $\frac{7}{3}$. We provide evidence that such "aggressiveness" is necessary: more precisely, we show that any competitively optimal strategy that is also optimal with respect to the discovery ratio must have the exact same behavior as the aggressive strategy in the first five iterations. In addition, we show the non uniqueness of such competitively optimal strategies with optimal discovery ratio.

In terms of techniques, we observe that for some problems, it is convenient to formulate them using linear programming. Sometimes, we can look into the structure of the optimal solutions to the linear programming. For the linear search problem, we can argue that optimal solutions saturate some constraints, which is useful to derive some recurrence relations on variables of the linear programming. In Chapter 4, we show how this idea can be applied to an application from Artificial Intelligence in the design of interruptible algorithms. The main contribution is an optimal schedule for the problem described above (see Section 1.3.3), namely for earliest completion scheduling of contract algorithms with end guarantees. We propose a schedule that is theoretically optimal and can be computed in polynomial time in the size of the end guarantee L. Assuming that the number of problem instances n is constant and independent of L, the time complexity is then polynomial in the size of the input. In addition, we present some computational results on its implementation, which demonstrate that it achieves a considerable improvement over the known schedule that optimizes the acceleration ratio, but is oblivious of L.

In Chapter 5, we study an online computation setting in which some additional information (i.e. advice) on request sequence is given to the online algorithm. More specifically, we work on a specific application of a very recent model, in which the advice can be either trusted or untrusted as described in Section 1.4. We work on the online bidding problem under this setting. The main contribution is to identify a Pareto frontier assuming that the advice encodes the hidden target.

Chapter 2

Online Maximum Matching with Recourse

This chapter contains material from the joint paper, *Online Maximum Matching with Recourse*, with Christoph Dürr and Spyros Angelopoulos. This work appeared in the *43rd International Symposium on Mathematical Foundations of Computer Science* conference (MFCS), 2018 [3].

2.1 Introduction

As discussed in Section 1.2, the recourse model has been studied for many online combinatorial optimization problems, in which the algorithm is allowed to do some modifications on previously made decisions. In this chapter, we will study the recourse model for the *online maximum matching problem*. Specifically, we will study the *online maximum cardinality matching with edge k-bounded recourse* problem, which is presented in Section 1.2.1.

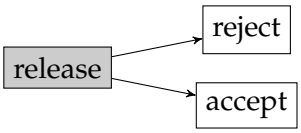
2.1.1 Related Work

Models

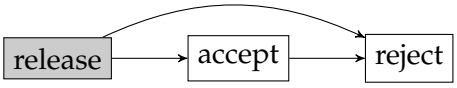
Concerning online matching, two different request models have been studied in the past. In the *vertex arrival model*, vertices arrive in online fashion, revealing, at the same time, the edges incident to previously arrived vertices. This model has mainly been considered for bipartite graphs, with left side vertices arriving online, and right side vertices being initially known (see the survey [56]). In the *edge arrival model*, the edges arrive online in arbitrary order, revealing at the same time incident vertices [1].

In the standard online model, every request (either vertex or edge, depending on the request model) is served in an irrevocable manner. In contrast, for online matching with recourse and edge arrivals, several models have been proposed that relax the irrevocable nature of a decision. In the *late reject* model [17], which is also called the *preemptive model* [21], an edge can be accepted only upon its arrival but can be later rejected. The problem we study in this work, namely online matching with k edge-recourse was introduced in [7]. Boyar *et al*. [17] refer to this model for $k = 1$ as the *late accept* model, and for $k = 2$ as the *late accept/reject* model. Clearly, the competitive ratio is monotone in k. Figure 2.1 provides an illustration of the algorithm's actions under the different models.

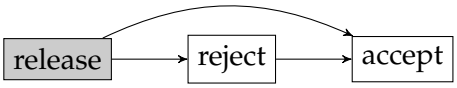
1. We emphasize that in our work we consider the maximum cardinality matching problem; some previous work (with or without recourse) has considered the generalized *weighted matching* problem, in which each edge has a weight and the objective is to maximize the weight of matched edges.



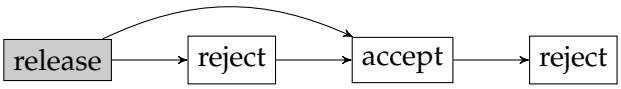
Standard model



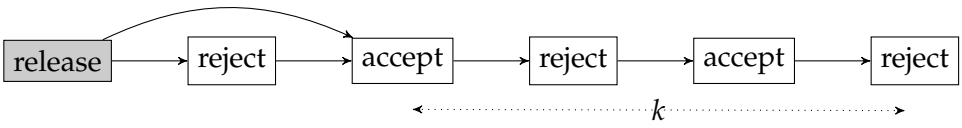
Late reject model, also called free disposal or preemptive model



Late accept model, also called 1 edge-recourse model



Late accept/reject model, also called edge 2 edge-recourse model



k edge-recourse model

FIGURE 2.1 – Illustration of the actions of an online matching algorithm under the different edge-arrival models with recourse.

Known results

For online maximum matching without recourse, and for the vertex arrival model, the seminal work of Karp *et al.* [45] gave a randomized online algorithm with competitive ratio $e/(e-1)$ in the vertex arrival model together with a matching lower bound on any online algorithm. In contrast, for the edge arrival model and the randomized competitive ratio, [20] showed a lower bound of $(3+1/\varphi^2)/2$ as well as an upper bound of 1.8 for the special case of forests, where φ is the golden ratio.

It is well known that any inclusion-wise maximal matching has cardinality at least half of the optimal maximum cardinality matching. From this, it follows that the greedy online algorithm, which accepts an edge as long as it can be added to the current matching, has competitive ratio at most 2, which in the standard model is optimal among all deterministic online algorithms.

Late reject In the vertex arrival model, the greedy algorithm achieves trivially the competitive ratio of 2, which is optimal for all deterministic online algorithms. The situation differs in the edge arrival model. Epstein *et al.* [31] showed that for online weighted matching, the deterministic competitive ratio is exactly $3+2\sqrt{2} \approx 5.828$, as the upper bound of [54] matches the lower bound of [62]. The same paper [31] shows that the randomized competitive ratio is between $1+\ln 2 \approx 1.693$ and 5.356. Chiplunkar *et al.* [21] presented a randomized 28/15-competitive algorithm based on a primal-dual analysis.

k edge-recourse This model was introduced and studied by Avitabile *et al.* [7] for the edge arrival setting, in the context of a much broader class of online packing problems. They gave an algorithm, which we call AMP, that combines doubling techniques with optimal solutions to offline instances of the problem, and which has competitive ratio $1 + O\left(\frac{\log k}{k}\right)$ (see Section 2.2.1 for an analysis of AMP). We note that this result is formulated in [7] in a "dual" setting. More precisely, [7] asks the question: how big should the edge budget k be such that there is a $1+\varepsilon$ competitive online algorithm that makes at most k changes per edge? They showed that $k = O(\ln(1/\varepsilon)/\varepsilon)$ suffices. On the negative side, they showed that no randomized algorithm can be better than $1 + 1/(9k-1)$-competitive; we note also that their construction implies a lower bound of $1 + 1/k$ for all deterministic algorithms.

Boyar *et al.* [17] showed that the deterministic competitive ratio is 2 for $k = 1$ and $3/2$ for $k = 2$, and these optimal ratios are achieved by the greedy algorithm. Moreover [17] studied several other problems for a value of the recourse parameter equal to 2, such as independent set, vertex cover and minimum spanning forest.

Minimizing recourse Bernstein *et al.* [12] studied a different recourse model in which the algorithm has to maintain an optimal matching, while minimizing a recourse measure, namely the total number of times edges enter or leave the matching maintained by the algorithm. They considered the setting of a bipartite graph and the vertex arrival model and showed that a simple greedy algorithm achieves optimality using $O(n \log^2 n)$ replacements, where n is the number of nodes in the arriving bipartition, whereas the corresponding lower bound for any replacement strategy is $\Omega(n \log n)$.

2.1.2 Contributions

In the first part of this work, we study the online matching problem with edge k-bounded recourse under the edge arrival model. For this problem, we provide improvements on both upper and lower bounds on the competitive ratio. First, we revisit the doubling algorithm of [7] that was originally analyzed in the general context of online packing problems. We give better analysis, specifically for the problem at hand, that uses concepts and ideas related to the matching problem; we also show that the AMP algorithm has competitive ratio $1 + O(\frac{\log k}{k})$. On the negative side, we show that no deterministic algorithm is better than $1 + 1/(k-1)$ competitive, improving upon the known bound of $1 + 1/k$ of [7].

At first sight, these improvements may seem marginal; however one should take into consideration that k is typically a small parameter, and thus the improvements are by no means negligible. In this spirit, we propose and analyze a variant of the greedy algorithm which we call L-Greedy. This algorithm applies, at any step, augmenting paths as long as their length is at most $2L + 1$. We show that for a suitable choice of L, this algorithm is $1 + O(1/\sqrt{k})$-competitive. While this algorithm is thus not superior to AMP for large k (and more specifically, to its improved analysis in the context of the matching problem), for small k (and in particular, for $k \leq 20$) it does achieve an improved competitive ratio, see Figure 2.2. Moreover, we extend a result of Boyar *et al.* [17] that showed that the greedy algorithm is $3/2$ competitive for $k = 2$ to all even k (for odd k, the competitive ratio is 2).

In terms of techniques, we analyze both AMP and L-Greedy using amortization arguments in which the profit of the algorithms is expressed in terms of weights

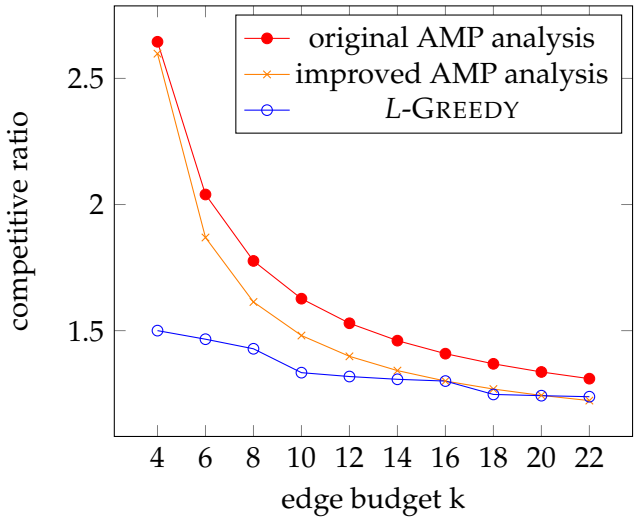


FIGURE 2.2 – Comparison of the competitive ratios of the algorithm AMP and the algorithm L-GREEDY

appropriately distributed over nodes in the graph. We achieve these improvements by exploiting the properties of augmenting paths in matching algorithms.

The second part of the work is devoted to the *edge arrival/departure model*, which is the fully dynamic variant of the online matching problem. First, we observe that the analysis of L-Greedy and AMP carries through in this model as well. On the negative side, we show a lower bound of $(k^2 - 3k + 6)/(k^2 - 4k + 7)$ for all even $k \geq 4$. For $k \in \{2, 3\}$, the competitive ratio is $3/2$. We obtain these lower bounds by modeling the game between the algorithm and the adversary as a game played over *strings* of numbers 0 up to k. These strings represent alternating paths, and each number represents how many times the algorithm has modified its decision on the corresponding edge. This provides a simpler combinatorial aspect to the game played between the adversary and the algorithm.

We note that, for the analysis of AMP and L-GREEDY, we assume that k is even. This assumption is borrowed from [7] and is required for the analysis. Of course for odd $k \geq 3$ these algorithms can be run with budget $k - 1$, providing a valid upper bound on the competitive ratio. Note that our lower bound in the arrival model holds for all values of k.

2.1.3 Preliminaries

A *matching* in a graph $G = (V, E)$ is a set of edges $M \subseteq E$ with disjoint endpoints. A vertex $v \in V$ is said to be *matched* by M if there is an edge $e \in M$ incident to v, and is *unmatched* otherwise. A key concept in maximum matching algorithms is the notion of an *augmenting alternating path*, or simply *augmenting path*. A path P in G is a sequence of vertices $v_1, v_2, \ldots, v_\ell$ for some length $\ell \geq 2$, such that $(v_i, v_{i+1}) \in E$ for all $i = 1, \ldots, \ell - 1$. It is said to be *alternating with respect to* M if every other edge of P belongs to M. Moreover, an alternating path is *augmenting* if the first and the last vertex in the path is unmatched by M. Applying P to M consists in removing from M the edges in $M \cap P$ and adding the edges in $P \setminus M$. The resulting matching has cardinality $M + 1$, and every previously matched vertex remains matched.

We define some concepts that will be useful in the analysis of algorithms throughout the chapter. We will associate each edge with a *type* which is an integer in $[0, k]$.

An edge is of type i if it has undergone i decision flips by the algorithm. Hence, for an edge of type k, where k is the recourse budget, its decision has been finalized, and cannot change further; we call such an edge *blocked*. The type of a path P is defined by the sequence of the types of its edges, and to make this concept unambiguous, we choose between the two orientations of the path the one that results in the lexicographically minimal such sequence. Note that when the algorithm applies some augmenting path P to its current matching M, then the type of every edge in P is increased by 1. Moreover, the two extreme edges of an augmenting path are of type 0, because the endpoints of P are unmatched. We will call a path *blocked* if it contains a blocked edge.

Given a request sequence σ, let $G(\sigma)$ denote the graph induced by σ. For an online algorithm ALG, and a connected component C of $G(\sigma)$, we define the *local ratio* in C as the ratio between the cardinality of the optimum matching and the cardinality of $ALG's$ matching, restricted to edges in C.

2.2 Online matching in the edge arrival model

2.2.1 The Algorithm AMP

We study the performance of an algorithm proposed by Avitabile *et al*. [7] for the more general *online set packing problem*. In this problem sets arrive online, and the objective is to maintain a collection of disjoint sets that has maximum cardinality. More specifically, [7] proposed a *doubling algorithm* which is defined for even k only. The algorithm has a parameter $r > 1$ and there is a decision variable for every set which can be changed at most k times. The algorithm works in phases, sequentially numbered by an integer p. Initially $p = 0$, and the algorithm's current solution is $\text{AMP}_0 = \emptyset$. Let ℓ be the largest integer such that the optimal solution has value at least r^ℓ, where ℓ is defined to be $-\infty$ if the optimal solution is empty. Whenever this value increases, the algorithm starts a new phase. We define $\ell(p)$ as the value of ℓ during phase p, and thus

$$\ell(p) + i \leq \ell(p+i) \tag{2.1}$$

for every positive integer i. At the beginning of a new phase, all decision variables that have been changed fewer than k times are set as in OPT, resulting in the current solution AMP_p (note that the algorithm crucially depends on k being even in order to produce a feasible solution). In between phases, AMP does not make any changes to its current solution.

Avitabile *et al*. show that the value $|AMP|$ of the solution returned by the algorithm is at least

$$\left(1 - \left(\frac{r-1}{r} + \frac{r}{r^k(r-1)}\right)\right)|OPT|,$$

which implies that

$$\frac{|OPT|}{|AMP|} \leq \frac{1}{1 - \left(\frac{r-1}{r} + \frac{r}{r^k(r-1)}\right)} = \frac{r^k(r-1)}{r^{k-1}(r-1) - r}.$$

Thus, for a given $r > 1$, the competitive ratio of AMP is at most $\frac{r^k(r-1)}{r^{k-1}(r-1)-r}$. Let r_0 denote the root of $r^{k-1}(r-1) - r = 0$. If $r < r_0$, then for all $k \geq 4$, we have $r^{k-1}(r-1) - r < 0$ since this function is increasing in r. We conclude that the competitive

ratio of AMP is upper bounded by

$$\min_{r>r_0} \frac{r^k(r-1)}{r^{k-1}(r-1)-r}. \tag{2.2}$$

We will show how to obtain an improved analysis of this algorithm in the context of the matching problem. Since we know optimal algorithms for $k = 1, 2$ [17] and, since we can show optimality of the greedy algorithm for $k = 3$ (see Section 2.2.4), we can assume, for the purposes of the analysis, that $k \geq 4$. We begin by a restatement of the update phase that will help us exploit the structure of solutions obtained via augmenting paths. More specifically, on every edge arrival, the algorithm updates a current optimal solution OPT. At the beginning of a new phase, the algorithm produces a matching AMP_p obtained from AMP_{p-1} as follows: every edge $e \in \text{AMP}_{p-1} \setminus \text{OPT}$ is removed from the current matching, and every edge $e \in \text{OPT} \setminus \text{AMP}_{p-1}$ which is of type strictly smaller than k is added to the current matching. Note that edges incident with e have been removed, hence AMP_p is indeed a matching. Also note that all edges added or removed by the algorithm have their type increased by one.

Since AMP_{p-1} and OPT are matchings, their symmetric difference, excluding type k edges, consists of alternating cycles and alternating paths which can be of even or odd length. This means that the algorithm simply applies at the beginning of every phase all those alternating paths and cycles.

Let OPT_p denote the matching produced by OPT as phase p is about to begin. From the statement of AMP, we obtain the following series of inequalities, for every phase p.

$$r^{\ell(p)} \leq |\text{OPT}_p| \leq |\text{OPT}| < r^{\ell(p)+1}. \tag{2.3}$$

For any given phase $p \geq 1$, we aim to bound the ratio $|\text{OPT}|/|\text{AMP}_p|$, since this will allow us to bound the competitive ratio of AMP, for a worst-case choice of p. Note that the type of an edge increases by at most 1 with each phase. Hence, in the beginning of the k first phases AMP "synchronizes" with OPT as there are no blocked edges yet, and as a result during these phases the ratio $|\text{OPT}|/|\text{AMP}_p|$ does not exceed r, from (2.3).

For the remaining phases we need the following argument.

Proposition 2.1. *For even k and any phase $p \geq k+1$, AMP maintains a matching AMP_p of cardinality at least $r^{\ell(p)} - r^{\ell(p-k+1)+1}$.*

Proof. We denote by the *type of a vertex* v the maximum type of the edges incident with v and by $n_{i,p}$ the number of vertices of type i in phase p. With every phase change the type of a vertex increases at most by 1. Hence every vertex of type k in phase p had positive type in phase $p-k+1$. Thus

$$n_{k,p} \leq \sum_{i=1}^{k} n_{i,p-k+1} \leq 2 \cdot |\text{OPT}_{p-k+1}|,$$

where the last inequality uses the fact that the left hand side counts the number of vertices matched by the algorithm. In phase p, the difference between the cardinality of the optimal matching and the cardinality of AMP's matching is at most the number of blocked augmenting paths, and each of them contains at least two type k

vertices. Hence,

$$\begin{aligned} |AMP_p| &\geq |\mathrm{OPT}_p| - \frac{1}{2} \cdot n_{k,p} \\ &\geq |\mathrm{OPT}_p| - |\mathrm{OPT}_{p-k+1}| \\ &> r^{\ell(p)} - r^{\ell(p-k+1)+1}, \end{aligned}$$

where the last inequality follows from (2.3).

□

Combining Proposition 2.1 with the bounds (2.3) we obtain the following.

Proposition 2.2. *The competitive ratio of AMP for $k \geq 4$ is upper bounded by the expression*

$$\min_{r>1} \frac{r^k}{r^{k-1} - r}. \tag{2.4}$$

Proof. Consider an arbitrary phase p and a fixed parameter $r > 1$. The expression $r^k/(r^{k-1} - r)$ is at least r. As observed earlier, at the end of the first k phases, the competitive ratio is at most r, hence the proof holds for $p \leq k$. For the remaining case $p \geq k + 1$, we have that

$$\begin{aligned} \frac{|\mathrm{OPT}|}{|\mathrm{AMP}_p|} &\leq \frac{r^{\ell(p)+1}}{r^{\ell(p)} - r^{\ell(p-k+1)+1}} \\ &\leq \frac{r^{\ell(p)+1}}{r^{\ell(p)} - r^{\ell(p)-k+2}} && \text{(From (2.1))} \\ &= \frac{r}{1 - r^{-k+2}} \\ &= \frac{r^k}{r^{k-1} - r}. \end{aligned}$$

□

Next, we show that Proposition 2.2 can yield an improved analysis of AMP over the original bound (2.2).

Proposition 2.3. *For all even $k \geq 4$, we have that the competitive ratio as expressed by (2.2) is at least the expression (2.4).*

Proof. For $k = 4$, we obtain numerically that (2.2) is 2.64526 whereas (2.4) is 2.59808.

For even $k \geq 6$, first we show that the minimizer (for $r > 1$) of

$$\frac{r^k(r-1)}{r^{k-1}(r-1) - r}$$

is between r_0 and 2. The derivative of the above expression is

$$\frac{(r-1)^2 r^{2k} + (k - 1 - kr) r^{k+2}}{\left(r^2 - (r-1)r^k\right)^2},$$

which we claim to be positive for any $r \geq 2$. This follows by the inequality $r^{k-3} \geq k$ which holds for any $r \geq 2$ and $k \geq 6$.

Similarly, for even $k \geq 6$, we show that the minimizer (for $r > 1$) of

$$\frac{r^k}{r^{k-1} - r}$$

is between 1 and 2. The derivative in r of the above expression is

$$\frac{r^k \left(r^k - (k-1)r^2\right)}{\left(r^2 - r^k\right)^2},$$

which we claim to be positive for any $r \geq 2$. Again this follows by the inequality $r^{k-2} \geq k-1$, which holds for any $r \geq 2$ and $k \geq 6$. The proof follows since for $1 < r \leq 2$, we have

$$\frac{r^k(r-1)}{r^{k-1}(r-1) - r} \geq \frac{r^k(r-1)}{r^{k-1}(r-1) - r(r-1)} = \frac{r^k}{r^{k-1} - r}.$$

□

The following theorem concludes the asymptotic analysis of the performance of AMP.

Theorem 2.4. *For all even k, AMP has competitive ratio $1 + O(\frac{\log k}{k})$.*

Proof. We first sketch a simple argument based on the Puiseux series expansion [59]: this is a type of power series that allows fractional powers, as opposed to only integer ones (e.g., Taylor series). Let r denote the optimal choice of the parameter, namely the one that minimizes (2.4). By analyzing the derivative, it follows that $r = (k-1)^{1/(k-2)}$, hence the competitive ratio is at most $\frac{(k-1)^{\frac{k-1}{k-2}}}{k-2}$, whose Puiseux series expansion at $k = \infty$ is $1 + \frac{\log k + 1}{k} + O(\frac{1}{k^2})$.

For completeness, we give a second proof that relies only on standard calculus. Let f be such that $r = 1 + f/k$, then since $r = (k-1)^{1/(k-2)}$, we have that the competitive ratio is at most

$$r \cdot \frac{k-1}{k-2} = \left(1 + \frac{f}{k}\right) \frac{k-1}{k-2} = 1 + O\left(\frac{f}{k} + \frac{1}{k}\right). \tag{2.5}$$

Suffices then to show that $f = O(\log k)$. Consider the function

$$g(x) = x^{k-2} - k + 1,$$

and note that r must be a root of g. We can rewrite $g(r)$ as

$$g(r) = \left(1 + \frac{f}{k}\right)^{k-2} - k + 1 = e^{(k-2)\ln(1+f/k)} - k + 1.$$

Hence, we have

$$g(r) \geq e^{(k-2)2f/(2k+f)} - k + 1.$$

Here we used the following logarithmic inequality [51] for $n \geq 0$:

$$\ln(1 + 1/n) \geq \frac{2}{2n+1}.$$

Suppose now that $f = 4 \ln k$. In this case, we claim that $(k-2)\frac{2f}{2k+f} \geq \frac{f}{2}$, or equivalently $2k \geq 8 + f$, which holds for sufficiently large k. Thus we have

$$g\left(1 + \frac{4 \ln k}{k}\right) \geq e^{f/2} - k + 1 = k^2 - k + 1 > 0.$$

On the other hand, $g(0) = -k + 1 < 0$. As a result, since g is continuous, there exists $f \in [0, 4 \ln k]$ such that $g(r) = 0$. Therefore, $f = O(\log k)$, which concludes the proof. □

2.2.2 The Algorithm GREEDY

We consider the algorithm GREEDY, which repeatedly applies an arbitrary augmenting path whenever possible. More precisely, let E denote the set of edges that have been revealed to the algorithm, and let e denote a newly arriving edge. Then as long as there is a non-blocked augmenting path in the graph $(V, E \cup \{e\})$, where V is the vertex set, GREEDY will apply such a path until, eventually, no such path any longer exists. Note, in particular, that GREEDY does nothing if no such path exists upon the arrival of e.

This algorithm achieves an upper bound of $3/2$ for $k = 2$, as shown in [17]. We show that the same guarantee holds for all even k. In what concerns the lower bound, the idea is to force the algorithm to augment an arbitrarily long path in order to create a configuration with an arbitrarily large number of blocked augmenting paths of lengths 5, which have local ratio $3/2$.

Proposition 2.5. *The competitive ratio of* GREEDY *is* $3/2$ *if* k *is even, and* 2, *if* k *is odd.*

Proof. First note that GREEDY has the property that every edge in the optimal matching has at least one endpoint matched by GREEDY. As a result, the competitive ratio is at most 2 for any k, and in particular for any odd k.

For even positive k we give a stronger upper bound of $3/2$. Consider the symmetric difference of the matching produced by GREEDY and an optimal matching which consists of alternating cycles and paths. In each of these components the local ratio is 1 for the alternating cycles and alternating paths of even length. We claim that alternating paths of odd length have length ℓ at least 5, and therefore the local ratio is at most $3/2$. To see this, observe that the case $\ell = 1$ corresponds to an edge with both endpoints unmatched, and GREEDY would have included it in its matching. In the case $\ell = 3$, the center edge has an odd budget, meaning that GREEDY would have applied this augmenting path.

The proof of the lower bound for even positive k consists of an instance on which GREEDY achieves the competitive ratio $3/2 - \epsilon$ for any small constant $\epsilon > 0$; see Figure 2.3. Let n be a sufficiently large integer. First the adversary releases $2n + 1$ vertex disjoint edges, which the algorithm includes in its matching. Then these edges are connected with $2n + 2$ new edges to form an augmenting path of length $4n + 3$. From now on, each time the algorithm applies this augmenting path in its matching, the path is extended with one additional edge on each end, until there is at least one edge of type k on the path. At this point the edge types on the path form a sequence which starts with types from 1 to $k - 1$, then alternates between types k and $k - 1$ and finally ends with types from $k - 1$ to 1. To complete the instance, the adversary attaches a new edge on each endpoint of every second type k edge. As a result there are n augmenting paths of length 5, which are all blocked by a type k edge, together with 2 alternating paths of length k. The size of the matching produced by GREEDY

is $2n + k$, whereas the optimal matching has size $3n + k$, showing a lower bound of $3/2$.

For odd k the construction can be further strengthened. In the final step, the adversary attaches an edge on each endpoint of every type k edge, creating an arbitrary large number of blocked augmenting paths of length 3.

Note that the lower bounds hold even for the asymptotic competitive ratio, by repeating these constructions as in the proof of Lemma 2.8.

□

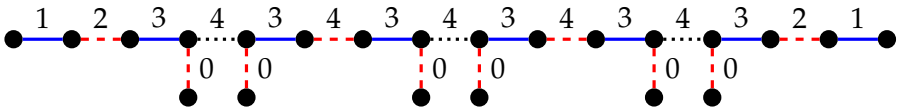


FIGURE 2.3 – Lower bound construction on the competitive ratio of GREEDY for $k = 4$ and $n = 2$. Edges are labeled by their type. Solid/blue edges depict the algorithm's matching, dashed/red edges depict the optimal matching, dotted/black edges belong to none of the matchings.

2.2.3 The algorithm L-GREEDY

How can the greedy algorithm be improved? As illustrated in the proof of Proposition 2.5, the greedy algorithm has inferior performance because it augments along arbitrarily long augmenting paths, therefore sometimes sacrificing edge budget for only a marginal increase in the matching size. A natural idea towards an improvement would be to apply only short augmenting paths, as they are more budget efficient. For technical reasons, we restrict the choice of augmenting paths even further.

We define the algorithm L-GREEDY for some given parameter L, which applies any non-blocked augmenting path of length at most $2L + 1$ that is in the symmetric difference between the current matching and some particular optimal matching OPT. The latter is updated after each edge arrival by applying an augmenting path to OPT.

To make the above more precise, let E denote the set of edges that have been revealed to the algorithm, and let e denote a newly arriving edge. First, we explain how the optimal matching is updated: If $\mathrm{OPT}(E)$ denotes the optimal matching after all edges in E have been revealed, then $\mathrm{OPT}(E \cup \{e\})$ is obtained by applying an augmenting path to $\mathrm{OPT}(E)$. Then, L-GREEDY serves request e by consecutively applying non-blocked augmenting path of length at most $2L + 1$ that is in the symmetric difference between its current matching (prior to the arrival of e) and $\mathrm{OPT}(E)$.

Note that L-GREEDY may not change its solution even if there is a short augmenting path in the current graph if it contains edges which are not in this particular optimal matching OPT. We will later optimize the parameter L as a function of k.

Analysis of L-GREEDY

We begin by observing that for $L = 0$ the algorithm collects greedily vertex disjoint edges without any recourse, which is precisely the behavior of GREEDY for $k = 1$ and has competitive ratio 2. For $L = 1$ the algorithm L-GREEDY applies only augmenting paths of length at most 3. In this case, the same argument as in the proof of Proposition 2.5 shows that the competitive ratio of L-GREEDY is $3/2$.

In what follows we analyze the general case $L \geq 2$. To this end, we assign weights to vertices in a way that the total vertex weight equals the size of the current matching. Therefore, whenever the size of the matching is increased by 1, a total weight of 1 is distributed on the vertices along the augmenting path as follows. First, vertices in this path that were already matched receive a weight α, where $\alpha \geq 0$ is some constant that we specify later. Second, the two vertices on the endpoints of the augmenting path receive the remaining weight, that is $1/2 - \ell\alpha$, where $2\ell + 1$ is the length of the path. It follows, from this weight assignment, that every unmatched vertex has weight 0, that every matched vertex has weight at least $1/2 - L\alpha$, and that every endpoint of a type k edge has weight at least $1/2 - L\alpha + (k-1)\alpha$.

Suppose that L-GREEDY reaches a configuration in which it cannot apply any augmenting path, as specified in its statement. We consider the symmetric difference between the matching produced by the algorithm and the optimal matching maintained internally by the algorithm. This symmetric difference consists of alternating paths and/or alternating cycles, and we will upper bound for each such component its local ratio. In particular, a component in the symmetric difference falls in one of the following cases: Either it is an augmenting path of length $2\ell + 1 \leq 2L + 1$, or an augmenting path of length $2\ell + 1 > 2L + 1$, or an alternating cycle or alternating path of even length.

Case 1: Augmenting path of length $2\ell + 1 \leq 2L + 1$. Note that such a path contains at least one edge of type k, otherwise the algorithm would augment it. It follows that $\ell \geq 2$, since an augmenting path of length 1 is a single type 0 edge, and an augmenting path of length 3 has edge types respectively $0, t, 0$ for some odd t (and k is assumed to be even). The path contains 2ℓ matched vertices, and at least 2 of them are incident with a type k edge. Hence the total vertex weight is at least $2\ell\left(\frac{1}{2} - L\alpha\right) + 2(k-1)\alpha$, and the local ratio of this component is at most

$$\frac{\ell + 1}{\ell - 2\ell L\alpha + 2(k-1)\alpha}. \tag{2.6}$$

Case 2: Augmenting path of length $2\ell + 1 > 2L + 1$. Such a path contains 2ℓ matched vertices and therefore the local ratio is at most

$$\frac{\ell + 1}{\ell - 2\ell L\alpha}. \tag{2.7}$$

Case 3: Alternating cycle or path of even length. Such a component contains 2ℓ matched vertices and therefore the local ratio is at most

$$\frac{\ell}{\ell - 2\ell L\alpha},$$

which is dominated by (2.7). We obtain the following performance guarantee.

Theorem 2.6. *The competitive ratio of* L-GREEDY *with* $L = \lfloor\sqrt{k-1}\rfloor$ *is at most*

$$\frac{k(L+2) - 2}{(L+1)(k-1)} = 1 + O\left(\frac{1}{\sqrt{k}}\right),$$

for even $k \geq 6$ *and at most* $3/2$ *for* $k = 4$.

Proof. We choose α so as to minimize the maximum of the local ratios, as defined by (2.6) and (2.7). Then for this choice of α we optimize L as stated in the theorem. Note that for $k = 4$, this leads to the choice $L = 1$, which we analyzed in the beginning of the section.

For $k \geq 6$, we have to minimize over α and L the maximum over ℓ of the two ratios given by (2.6) and (2.7). First we upper bound (2.7) as

$$\frac{\ell+1}{\ell-2\ell L\alpha} \leq \frac{L+2}{L+1-2L^2\alpha-2L\alpha}, \tag{2.8}$$

where the inequality follows from $\ell \geq L+1$ and the fact that the left hand side is decreasing in ℓ as can be seen by dividing both the numerator and denominator by ℓ.

In order to upper bound (2.6), we find its derivative in ℓ which is equal to

$$\frac{2\alpha(L+k-1)-1}{(\ell+2\alpha(k-L\ell-1))^2}.$$

This means that (2.6) is increasing or decreasing in ℓ depending on the sign of $2\alpha(L+k-1)-1$. Hence we distinguish two cases.

Case 1: $\alpha < 1/(2(L+k-1))$ In this case (2.6) is decreasing in ℓ, and by $\ell \geq 2$ is at most

$$\frac{3}{2\alpha(k-2L-1)+2}. \tag{2.9}$$

Subcase 1a: $k-2L-1 < 0$ In this case (2.9) and (2.8) are increasing in α and hence minimized at $\alpha = 0$. For this choice of α (2.9) is $3/2$, while (2.8) is $(L+2)/(L+1)$ which by $L \geq 2$ is less than $3/2$. In conclusion, the competitive ratio in Subcase 1a is at most $3/2$.

Subcase 1b: $k-2L-1 \geq 0$ In this case (2.9) is non-increasing in α while (2.8) is increasing in α. Hence the maximum of (2.9) and (2.8) is minimized at the equality of the expressions, which happens for

$$\alpha = \frac{L-1}{2L^2+4k+2kL-4L-4}.$$

It can readily be verified that this choice of α indeed satisfies the assumption of Case 1 for all even $k \geq 2$ and $L \geq 2$. The corresponding ratio is

$$\frac{L^2+2k+kL-2L-2}{(k-1)(L+1)}. \tag{R1b}$$

Case 2: $\alpha \geq 1/(2(L+k-1))$ In this case (2.6) is increasing in ℓ, and at $\ell = L$ becomes

$$\frac{L+1}{L+2\alpha(k-L^2-1)}. \tag{2.10}$$

Subcase 2a: $k - L^2 - 1 < 0$ In this case, (2.10) and (2.8) are increasing in α. At $\alpha = 1/(2(L+k-1))$, the former becomes

$$\begin{aligned}\frac{L+1}{L+\frac{k-L^2-1}{L+k-1}} &= \frac{(L+1)(L+k-1)}{L(L+k-1)+k-L^2-1} \\ &= \frac{(L+1)(L+k-1)}{(L+1)(k-1)} \\ &= \frac{L+k-1}{k-1}. \end{aligned} \tag{2.11}$$

Similarly, (2.8) becomes

$$\begin{aligned}\frac{L+2}{L+1-\frac{L(L+1)}{L+k-1}} &= \frac{(L+2)(L+k-1)}{(L+1)(L+k-1)-L(L+1)} \\ &= \frac{(L+2)(L+k-1)}{(L+1)(k-1)}, \end{aligned} \tag{R2a}$$

which dominates (2.11) and therefore upper bounds the competitive ratio in Subcase 2a.

Subcase 2b: $k - L^2 - 1 \geq 0$ In this case, (2.10) is non-increasing in α while (2.8) is increasing in α. We have equality of the expressions for

$$\alpha = \frac{1}{2k(L+2)-4},$$

which implies that the competitive ratio in Subcase 2b is at most

$$\frac{k(L+2)-2}{(L+1)(k-1)}. \tag{R2b}$$

This concludes the case analysis. Thus it remains to optimize L.

For $k = 4$, the assumptions of the Subcases 1b and 2b are not valid. Hence, we have to choose L so to minimize the minimum of $3/2$ and (R2a), which is larger than $3/2$. This means that any choice $L \geq 2$ leads to a competitive ratio at most $3/2$. Note that the same guarantee is obtained for $L = 1$.

Finally we consider $k \geq 6$ and choose L so to minimize the minimum of $3/2$ and the ratios (R1b), (R2a), (R2b). We claim that (R1b) is dominated by $3/2$. Indeed its derivative in k is

$$-\frac{L(L-1)}{(k-1)^2(L+1)} < 0.$$

Hence (R1b) is maximized for k, L such that $k - 2L - 1 = 0$, and this maximum value is precisely $3/2$.

By comparing the numerators of the ratios (R1b), (R2a) and (R2b) the minimum ratio is (R2b). Its derivative in L is

$$-\frac{k-2}{(L+1)^2(k-1)} \leq 0.$$

Hence the minimum of (R2b) is attained at the upper bound for L given by the Subcase 2b assumption, namely

$$L = \lfloor \sqrt{k-1} \rfloor.$$

It follows that the competitive ratio is at most

$$\frac{k(\lfloor\sqrt{k-1}\rfloor+2)-2}{(\lfloor\sqrt{k-1}\rfloor+1)(k-1)} \leq 1+\frac{2k+\sqrt{k-1}-1}{(\sqrt{k-1}+1)(k-1)} = 1+O\left(\frac{1}{\sqrt{k}}\right).$$

□

We complete the analysis of L-GREEDY by showing that the upper bound is essentially tight.

Lemma 2.7. *For even $k \geq 4$, the strict competitive ratio of L-GREEDY with $L = \lfloor\sqrt{k-1}\rfloor$ is at least*

$$1+\Omega\left(\frac{1}{\sqrt{k}}\right).$$

Proof. For the proof we show that for any positive parameter $L \geq 3$ the (strict) competitive ratio of L-GREEDY is at least

$$\frac{3\lfloor\frac{L-1}{2}\rfloor+k-2}{L+k-3}. \tag{2.12}$$

The statement follows then by lower bounding this expression for $L = \lfloor\sqrt{k-1}\rfloor$.

We will show how to create an instance in which L-GREEDY has the competitive ratio expressed by (2.12). The adversarial construction is depicted in Figure 2.4, and consists of the following steps. First the adversary releases $L-2$ vertex disjoint edges, denoted by $e_1,\ldots,e_{L-2}$, which the algorithm includes in its matching. Then these edges are connected with $L-1$ new edges (which are denoted by $f_1, f_2, \ldots, f_{L-1}$ in Figure 2.4) so as to form an augmenting path P of length $2L-3$, which is short enough to ensure that the algorithm applies it. Note that the precise order in which the edges arrive is not important. At this stage the edges of P have alternating types 1 and 2, and by the choice of $L \geq 3$ there is at least one edge of type 2 in P.

In the next phase, the adversary releases edges from the sets A and B, in an interleaved manner; specifically, P is extended with two additional edges on each end, so as to form augmenting paths of length $2L-1$ and $2L+1$, until there are edges of type k on the path. At this point, the edge types on the medium part of length $2L-3$ form an alternating sequence of $k-1$ and k, and note that there are $k/2-1$ edges of type 2 and $k/2-1$ edges of type 1 in each set A and B. To complete the instance, the adversary attaches a new edge on each of the type $k-1$ edges in P, alternating between the left endpoint and on the right endpoint of P. As a result, all augmenting paths are blocked with a type k edge. The size of the matching produced by L-GREEDY is $L+k-3$, whereas the optimal matching has size at least $3\lfloor\frac{L-1}{2}\rfloor+k-2$, concluding the proof.

□

The previous lemma can be extended to the asymptotic competitive ratio, using a standard technique based on multiple copies of the adversarial instance.

Lemma 2.8. *For even k and $L = \lfloor\sqrt{k-1}\rfloor \geq 3$, the asymptotic competitive ratio of L-GREEDY is at least the expression (2.12) and hence is $1+\Omega(1/\sqrt{k})$.*

Proof. Assume that the asymptotic competitive ratio of L-GREEDY is R for R being strictly smaller than expression (2.12). Then by definition there exists a constant d such that

$$|L\text{-GREEDY}(\sigma)| \geq |\text{OPT}(\sigma)|/R - d$$

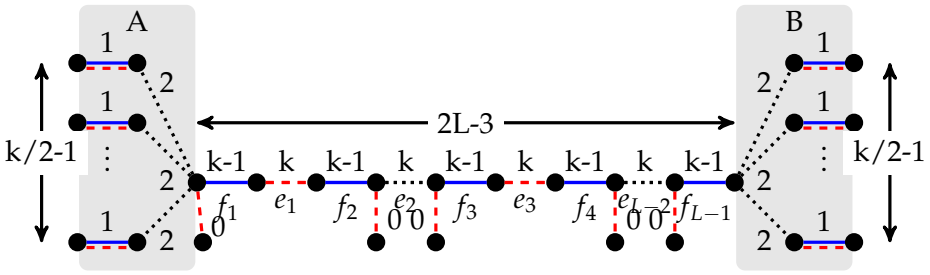


FIGURE 2.4 – Lower bound construction for even k and $L = 6$. Numbers on edges describe the edge types at the end of the algorithm's execution. Solid/blue edges depict the algorithm's matching, dashed/red edges depict the optimal matching and dotted/black edges belong to none of the matchings.

for all request sequences σ. Let σ' be the adversarial construction defined in the proof of Lemma 2.7. For an arbitrary positive integer p let σ be the result of repeating each edge of σ' p times, such that the resulting graph consists of p disjoint copies of the graph produced by σ'. By the above assumption we have

$$\begin{aligned} |L\text{-GREEDY}(\sigma)| &\geq |\text{OPT}(\sigma)|/R - d && \equiv \\ p \cdot |L\text{-GREEDY}(\sigma')| &\geq p \cdot |\text{OPT}(\sigma')|/R - d && \equiv \\ |L\text{-GREEDY}(\sigma')| &\geq |\text{OPT}(\sigma')|/R - d/p. \end{aligned}$$

Since d/p can be arbitrarily close to 0, this would mean that L-GREEDY is R-competitive, a contradiction.

□

2.2.4 Lower Bound on the Competitive Ratio of Deterministic Algorithms

Boyar *et al.* [17] show that the deterministic competitive ratio of the problem is 2 for $k = 1$ and $3/2$ for $k = 2$. We complete this picture by showing a lower bound of $1 + \frac{1}{k-1}$ for all $k \geq 3$. Note that the lower bound is tight for $k = 3$, as the algorithm GREEDY, which works by assuming that k is only 2, has competitive ratio $3/2$.

Theorem 2.9. *The deterministic competitive ratio of the online matching problem with k edge-recourse is at least $1 + \frac{1}{k-1}$ for all $k \geq 3$.*

Proof. We consider three cases, namely the cases $k = 3$, k is even and at least 4, and finally k is odd and at least 5. For each case we present an appropriate adversarial argument.

Case $k = 3$. Suppose, by way of contradiction, some algorithm claims a competitive ratio strictly smaller than $(3n + 2)/(2n + 2)$ for some arbitrary $n \geq 1$. The adversary releases a single edge, creating an augmenting path of length 1. Then the algorithm applies the augmenting path, which the adversary extends by appending one edge on each side, creating an augmenting path of type 0,1,0, as shown in Figure 2.5(a). Since the current competitive ratio is 2, the algorithm needs to apply this path, which the adversary again extends by appending an edge on each side, creating an augmenting path of type 0,1,2,1,0, as shown in Figure 2.5(b). Since the current competitive ratio is $3/2$, the algorithm applies this path. In response the adversary appends an edge at each endpoint of the type 3 edge, and at each endpoint of one

of the type 1 edges, as shown in Figure 2.5(c). The resulting graph has a blocked augmenting path of type 0,3,0, and an augmenting path of type 0,1,0, as shown in Figure 2.5(c). The algorithm needs to apply the latter one as the competitive ratio is currently $5/3 > 3/2$.

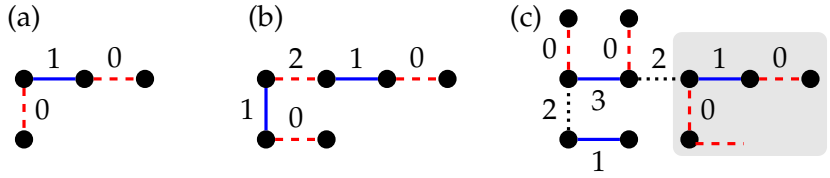


FIGURE 2.5 – Lower bound construction on the competitive ratio for the case $k = 3$.

At this point, the adversary repeats this construction $n-1$ times, by identifying the shaded part of Figure 2.5(c) as the graph of Figure 2.5(a), and reapplying the above construction. The final graph consists of n blocked augmenting paths of type 0,3,0 and $n+2$ edges of type 1 that belong both to the optimal and the algorithm's matchings. Hence, the competitive ratio is $(3n+2)/(2n+2)$, which contradicts the claimed ratio and shows a lower bound on the competitive ratio of $3/2$.

Case k is even and at least 4. Fix an algorithm that claims a competitive ratio strictly smaller than $k/(k-1)$. The adversary releases a single edge, creating an augmenting path of length 1. Whenever the algorithm applies the augmenting path[2], the adversary extends it by appending one edge on each end, eventually creating an alternating path of type $1, 2, \ldots, k-1, k, k-1, \ldots, 2, 1$. Then the adversary appends an edge to each endpoint of the type $k-1$ edges. The resulting graph has two augmenting paths of type $0, k-1, 0$, see Figure 2.6(a). The algorithm needs to apply them as the competitive ratio is currently $(k+2)/k$, which is strictly greater than $k/(k-1)$ if $k \geq 4$. Each augmentation is responded, by the adversary, with an extension of the path resulting in the configuration depicted in Figure 2.6(b) of ratio $(k+4)/(k+2) \geq k/(k-1)$ where all augmenting paths are blocked by type k edges. Hence the competitive ratio of the algorithm is not strictly smaller than $k/(k-1)$.

Case k is odd and at least 5. Fix an algorithm that claims a competitive ratio strictly smaller than $k/(k-1)$. The adversary proceeds as in the previous case, until the graph consists of a path of type $1, 2, \ldots, k-1, k, k-1, \ldots, 2, 1$. This time, the adversary appends one edge to each endpoint of the type $k-1$ edges, but also appends one edge at each endpoint of the path. As a result, there are two augmenting paths of type $0, 1, 2, \ldots, k-2, 0$ and a single blocked augmenting path of type $0, k, 0$, see Figure 2.6(c).

The algorithm needs to apply an augmenting path as the competitive ratio is currently $(k+3)/k$, which is strictly greater than $k/(k-1)$ for $k \geq 5$. The adversary responds each augmentation of a path by appending an edge on both ends of

2. We can assume, without loss of generality, that this is the only viable choice for the online algorithm. This is because the only way an algorithm can transform a given matching M_1 to a matching M_2 is via a sequence which can only consist of the following: i) augmenting paths; ii) alternating cycles; and iii) alternating, or even "decreasing" paths (i.e., paths such that if the algorithm applies them, then the cardinality of the matching remains the same, or decreases, respectively). This is a well-known result from matching theory. In principle an online algorithm, say A, could apply paths and cycles in cases (ii) and (iii), but such an algorithm can be converted to another algorithm A' which is at least as good as A in terms of matching size, and which maintains edges of smaller types than A.

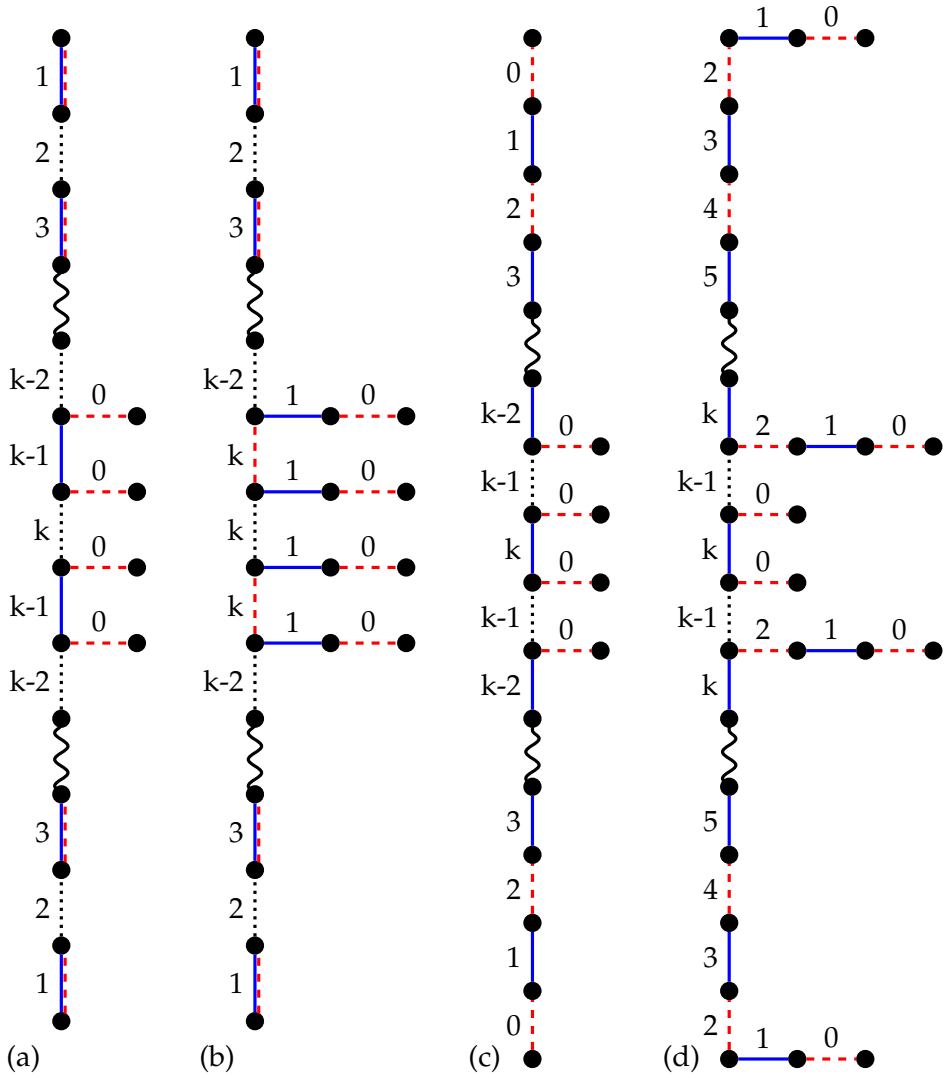


FIGURE 2.6 – Lower bound constructions for the deterministic competitive ratio. Solid/blue edges depict the algorithm's matching, dashed/red edges depict the optimal matching, dotted/black edges belong to none of the matchings and wiggled lines represent parts of the graph that are contracted for readability.

this path. At this moment, the ratio decreased slightly, but still exceeds the claimed ratio, forcing the algorithm to continue augmenting. Eventually this leads to a configuration formed by two blocked augmenting paths of type $0, 1, 2, \ldots, k, 2, 1, 0$ and a blocked augmenting path of type $0, k, 0$, see Figure 2.6(d). The competitive ratio of this configuration is $(k+7)/(k+4)$ which exceeds $k/(k-1)$ for $k \geq 5$.

□

2.2.5 Comparing the Algorithms L-GREEDY and AMP

We have analyzed two deterministic online algorithms: the algorithm AMP, which has competitive ratio $1 + O(\log k/k)$, and the algorithm L-GREEDY, which has competitive ratio $1 + \Theta(1/\sqrt{k})$. Since the analysis of L-GREEDY is tight, it follows that AMP is asymptotically (i.e., for large k) superior to L-GREEDY. However, for small

values of k, namely $k \leq 20$, we observe that L-GREEDY performs better, in comparison to the performance bound we have shown for AMP. These findings are summarized in Table 2.1 and Figure 2.2 (Section 2.1.2).

k	LB (arr.)	LB (arr./dep.)	L-GREEDY	AMP
4	1.333333	1.428571	1.5	2.598076
6	1.2	1.263158	1.466667	1.869186
8	1.142857	1.179487	1.428571	1.613602
10	1.111111	1.134328	1.333333	1.480583
12	1.090909	1.106796	1.318182	1.398080
14	1.076923	1.088435	1.307692	1.341500
16	1.066666	1.075377	1.300000	1.300080
18	1.058823	1.065637	1.247059	1.268330
20	1.052631	1.058104	1.242105	1.243150
22	1.047619	1.052109	1.238095	1.222640

TABLE 2.1 – Summary of lower bounds (LB) and upper bounds on the competitive ratio for the problem, for all even k with $4 \leq k \leq 22$. The lower bounds for the (limited) arrival/departure model are discussed in Section 2.3. The analysis of L-GREEDY and AMP carry through to the (limited) arrival/departure model. For $k \geq 22$, the upper bound of AMP is superior to the upper bound of L-GREEDY.

2.3 Online Matching in the Edge Arrival/Departure Model

In this section we consider the online matching problem with k edge-recourse in the setting in which edges may arrive but also *depart* online. More precisely, a request sequence for this problem is of the form $(p_i, e_i)_{i \geq 1}$, namely the i-th request consists of an edge e_i and its *mode* $p_i \in \{\texttt{arrive}, \texttt{depart}\}$. If $p_i = \texttt{arrive}$ then the edge e_i becomes available; this corresponds to the arrival setting studied in Section 2.2. If $p_i = \texttt{depart}$, then e_i is removed from the graph, and can be used by neither the online algorithm or the optimal offline algorithm. We emphasize that in this model, an edge of the form (u, v) may arrive and depart several times in the course of serving a request sequence, but every time it arrives it is considered as a "fresh" edge. As a consequence, upon each arrival, such an edge is assigned type 0. Moreover, a departing edge ceases to exist in any matchings.

As explained in the Introduction, we will further distinguish between two models. In the *limited departure model*, an edge cannot depart while it is being used in the matching of the online algorithm, whereas in the stronger *full departure model* any edge can depart.

It turns out that the full departure model is quite restrictive. This is because it is possible for the adversary to force an online algorithm to augment some augmenting path and then to remove one of the edges in its matching. Eventually the algorithm can end up with blocked edges (type k), without having the chance to augment its matching. This intuition is formalized in the following lemma.

Lemma 2.10. *The competitive ratio of online matching with k edge-recourse in the full departure model is* 2.

Proof. To show an upper bound of 2, consider an algorithm which adds to its matching any edge whose endpoints are unmatched. The edge types are either 0 or 1, and

therefore the algorithm is not sensitive to the given edge budget. The matching produced by the algorithm is (inclusion wise) a maximal matching, and it is well known that its size is at least $1/2$ the size of the maximum matching.

To show a lower bound of 2, consider a graph consisting of vertices 1, 2, 3, 4, with the edge $(2,3)$ arriving at the beginning. Then the edges $(1,2)$, $(3,4)$ arrive and depart repeatedly, alternating between two configurations. When the graph consists of the single edge $(2,3)$, the algorithm needs to include it in its matching. When the graph consists of the path $(1,2,3,4)$, the algorithm needs to apply this augmenting path if it claims to have a competitive ratio strictly lower than 2. As a result, the type of the edge $(2,3)$ keeps increasing, and when it reaches k, the algorithm cannot augment the matching anymore. Thus, for even k, the algorithm has a matching of size 0, while the optimal matching consists of the edge $(2,3)$. Similarly, for odd k, the algorithm has a matching of size 1, while the optimal matching has size 2. Hence, no algorithm can achieve a competitive ratio strictly lower than 2. □

Since the full departure model is very restrictive for the algorithm, as shown in Lemma 2.10, we will concentrate on the limited departure model, as defined in the introduction. For this model, we observe that the algorithms L-GREEDY and AMP have the same performance guarantee as in the edge arrival model. This is because the analysis of L-GREEDY uses weights on vertices which are not affected by edge departures, and the analysis of AMP is based on an upper bound over the number of type k edges, which still holds under edge departures. We thus focus on obtaining stronger lower bounds in this model (also included in Table 2.1). We begin by observing that the bound of $3/2$ of the competitive ratio in the edge arrival model for $k \in \{2,3\}$ still holds for the limited departure model, in which the adversary is stronger. Hence, the smallest interesting value for k in this model is $k = 4$, for which we provide the following specific lower bound. The proof will also provide some intuition about the adversarial argument for general k.

Theorem 2.11. *The competitive ratio of online matching with k edge-recourse in the* limited departure model *is at least* $10/7$ *for* $k = 4$.

Proof. We will prove the theorem by applying a game between the online algorithm and the adversary. In particular, the adversary will enforce arrivals and deletions of edges in such a way that, at every moment in time, the symmetric difference between the matching produced by the algorithm and the optimal matching consists only of augmenting paths. In particular, this symmetric difference will have no alternating cycles or alternating paths of even length.

Any such augmenting path can be represented as a *string* of integers in $\{0,1,\ldots,k\}$, which is precisely the type of this path. Note that for an augmenting path, this string begins and ends with 0. We can thus think of the above-defined symmetric difference as a collection of strings, which in turn allows us to define the game between the algorithm and the adversary over this collection of strings as opposed to defining it over the actual graph.

Whenever the algorithm applies an augmenting path, this translates into the increment of all edge types of the corresponding string, for example the string 01210 becomes 12321. The adversary will respond to this augmentation by a combination of the following three types of *operations*.

— The adversary may *append* 0's to both ends of a string, for example $101 \rightarrow 01010$. To do so, the adversary releases two new edges (of type 0). Each edge is incident with only one endpoint of the path described by the string, and is not incident with any other edge in the current graph.

— The adversary may *split* the string into smaller strings, for example $12321 \to \{123, 1\}$ or $12321 \to \{1, 3, 1\}$. This can be done via the departure of certain edges which are not in the algorithm's matching (of even type). For instance, the operation $12321 \to \{123, 1\}$ can be done by having one edge of type 2 depart, and the operation $12321 \to \{1, 3, 1\}$ can be done by having both edges of type 2 depart.
— The adversary may *merge* certain strings, for example $\{1, 1\} \to 101$. This can be done via the arrival of a new edge (of type 0).

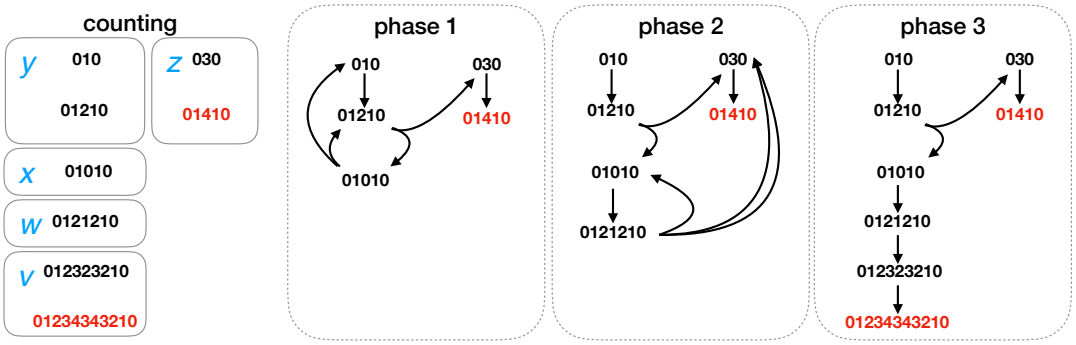


FIGURE 2.7 – An illustration of the three phases in the game between the algorithm and the adversary, for the proof of Theorem 2.11. Blocked strings are depicted in bold face. The arcs illustrate the actions of the adversary, after an augmentation by the algorithm. For example, if the algorithm augments the string 01010 in phase 1, then the adversary replaces the resulting string by the strings 010 and 01210, whereas in phases 2 and 3 it replaces it with the string 0121210. The numbers x, y, z count the number of strings which belong to the corresponding shown boxes.

The game between the algorithm and the adversary The main idea behind the adversarial construction is as follows. We suppose that the online algorithm has competitive ratio at most $(10 - \epsilon)/7$ for arbitrarily small $\epsilon > 0$. We will then show that the adversary can eventually force the algorithm to a competitive ratio at least $10/7$, thus leading to a contradiction.

The game begins with the adversary presenting the string 0, which the algorithm has to augment, resulting in a single string 010. From this point onwards, the game proceeds in three *phases*, which are depicted in Figure 2.7. In each phase, a sequence of algorithm/adversary *actions* takes place. Each action is of the following form: The algorithm chooses some string s to augment, which results in a string s'. Then the adversary will perform a sequence of the above defined operations on s', which will result in either a single new string (say $\bar{s}$), or to several new strings, say $\bar{s}_1, \bar{s}_2, \ldots \bar{s}_k$ (in our construction, it will be that $k \in [1, 3]$). This adversarial action is depicted by means of an arrow from s to each of the $\bar{s}_1, \bar{s}_2, \ldots \bar{s}_k$ in Figure 2.7.

As an example, in phase 1, if the algorithm augments string $s = 010$ (thus obtaining string $s' = 121$), the adversary appends two zeros at both ends of s', which results in the string $\bar{s}_1 = 01210$. If the algorithm augments string $s = 01210$, thus obtaining string $s' = 12321$, then the adversary first splits s' to two strings $1, 3$ and 1, which he then transforms into the strings $\bar{s}_1 = 01010$ and $\bar{s}_2 = 030$, by merging them and appending zero's. If the algorithm augments string $s = 030$, thus obtaining string $s' = 141$, then the adversary appends two zeros at both ends of s', which results in the string $\bar{s}_1 = 01410$. Last, if the algorithm augments string $s = 01010$

(thus obtaining $s' = 12121$) then the adversary first splits s' to two strings 1 and 121, then appends two zero's to the end of each string. This results in two strings $\bar{s}_1 = 010$ and $\bar{s}_2 = 01210$. The above are all possible actions that can occur in phase 1. Actions for phases 2 and 3 are similar and defined by the graphs in Figure 2.7.

We also need to explain how the game *transitions* between phases; i.e., under which conditions the game moves from phase i to phase $i+1$. To this end, we define some variables which count the numbers of some specific strings. More precisely:

— x denotes the number of strings 01010, 0121210, 012323210 or 01234343210; such strings have local ratio $3/2$, $4/3$, $5/4$ and $6/5$, respectively.
— y denotes the number of strings 0, 010 or 01210; such strings have local ratio ∞, 2 and $3/2$, respectively.
— z denotes the number of strings 030 or 01410; such strings have local ratio 2 and $3/2$, respectively.

In Figure 2.7 we use "boxes" to illustrate this grouping of strings.

In particular, we will call strings 030 or 01410 *bad strings*. This is motivated by the observation that 01410 is *blocked* and has large local ratio equal to $3/2$; note that the algorithm cannot augment such a string. Moreover, the string 030 has local ratio 2, and immediately after an augmentation it becomes 01410.

The goal of the adversary is to reach a configuration with only blocked strings 01410 and 01234343210 with a maximum proportion of bad strings 01410, since this maximizes the competitive ratio. It is relatively easy for the adversary to generate bad strings, but this comes at the expense of generating strings 01010. His goal is to minimize the proportion of these strings, and this is done through three different phases. At a high level, the objective of phase 1 is to create a large number of bad strings. This is is also the objective of phase 2, but with a more *efficient* generation of bad strings, in the sense that less 01010 strings are generated per bad string. The objective of phase 3 is simply to bring the game in a situation with only blocked strings.

The game starts with the adversary entering phase 1 and generating the single string 0. In phase 1, immediately after each action of the adversary, the algorithm has competitive ratio at least $3/2$, meaning that he is forced to augment strings, since $3/2 > (10-\epsilon)/7$, which is the competitive ratio claimed by the algorithm. This is because in this phase all strings have local ratio at least $3/2$. Throughout phase 1 we have the invariant

$$2x + y = z + 1, \tag{Inv 1}$$

which can be verified by inspecting each possible action of phase 1. For example the augmentation of 01010 decrements x and increments y by 2. Eventually the inequality $7z + 3 > 2/\epsilon$ will start to hold, simply because after at most $x+1$ augmentations, the counter z increases strictly. At that moment, the adversary moves to phase 2.

Throughout phase 2 we have the invariants

$$7z + 3 > 2/\epsilon \tag{Inv 2.1}$$

and

$$2x + y \leq z + 1. \tag{Inv 2.2}$$

(Inv 2.1) holds because the left-hand side will not decrease throughout the phase. To show (Inv 2.2), we first observe that by invariant (Inv 1) phase 2 starts with equality, and the actions of phase 2 preserve the inequality (Inv 2.2), which can be easily shown by inspecting each possible action during phase 2.

Phase 2 continues for as long as $z < 8(x+y)$, and phase 3 begins at the point in which $z \geq 8(x+y)$. This condition will eventually be reached, because the quantity $x+y$ is invariant during phase 2, whereas any sequence of at least $x+1$ actions increases z by at least 1.

We will now argue that in phase 2, right after each action of the adversary, the competitive ratio of the algorithm is strictly greater than $(10-\epsilon)/7$, which implies that the algorithm must, in turn, respond with an augmentation to every action of the adversary in phase 2, since we assumed that the algorithm is $(10-\epsilon)/7$-competitive. To this end, we observe that at each point in phase 2, the algorithm maintains certain types of strings whose local ratio we lower bounded above. In particular, there are $y+z$ strings of local ratio at least $3/2$ and x strings of local ratio at least $4/3$. Therefore, a lower bound to the competitive ratio during phase 1, can be stated as follows, where we use the property (P1): $f(x,y) := \frac{ax+cy}{bx+dy}$ is decreasing on x and increasing on y, if $a,b,c,d > 0$ and $\frac{a}{b} \leq \frac{c}{d}$.

$$\begin{aligned}
\frac{4x+3y+3z}{3x+2y+2z} &= \frac{8x+6y+6z}{6x+4y+4z} \\
&= \frac{4(2x+y)+2y+6z}{3(2x+y)+y+4z} \\
&\geq \frac{4(z+1)+2y+6z}{3(z+1)+y+4z} && \text{(from P1 and Inv 2.2)} \\
&= \frac{2y+10z+4}{y+7z+3} \\
&\geq \frac{10z+4}{7z+3} && \text{(from P1)} \\
&= \frac{10}{7} - \frac{\frac{2}{7}}{7z+3} \\
&> \frac{10}{7} - \frac{\frac{2}{7}}{\frac{2}{\epsilon}} && \text{(from Inv 1)} \\
&= \frac{10-\epsilon}{7}.
\end{aligned}$$

Recall that when the condition $z \geq 8(x+y)$ becomes satisfied, the game moves to the final phase, namely phase 3. Moreover, the condition $z \geq 8(x+y)$ holds throughout phase 3, since $x+y$ is invariant and z can only increase in this phase. We also obtain that

$$y+z \geq y+8(x+y) \geq 8x,$$

which will be useful. Similar to the previous argument, we can lower bound the competitive ratio after each action of the adversary by

$$\begin{aligned}
\frac{3(y+z)+6x}{2(y+z)+5x} &\geq \frac{3\cdot(8x)+6x}{2\cdot(8x)+5x} && \text{(From } y+z \geq 8x \text{ and P1)} \\
&= \frac{10}{7}.
\end{aligned}$$

Therefore, the algorithm must augment after each action of the adversary, and eventually must find itself in a configuration which consists only of blocked strings (either 01410 or 01234343210). At this configuration, the algorithm cannot do any further augmentations, hence its competitive ratio is at least $10/7$, a contradiction.

□

We can generalize the ideas in the proof of Theorem 2.11 so as to obtain a non-trivial lower bound for general even $k \geq 4$.

Theorem 2.12. *The competitive ratio online matching with k edge-recourse in the limited departure model is at least* $\frac{k^2-3k+6}{k^2-4k+7}$*, for all even* $k \geq 4$*.*

Since $\frac{k^2-3k+6}{k^2-4k+7} > 1 + \frac{1}{k-1}$ for all $k \geq 4$, Theorem 2.12 shows a stronger lower bound for even k than Theorem 2.9 under the limited departure model.

Proof. First, we observe that for $k = 4$, the expression $\frac{k^2-3k+6}{k^2-4k+7}$ is equal to $10/7$, which is precisely the value obtained in Theorem 2.11. Therefore, it suffices to prove the result for even $k \geq 6$.

The proof generalizes the ideas behind the proof of Theorem 2.11, and in particular the concept of a game between the online algorithm and the adversary. Again, we suppose that the online algorithm has competitive ratio at most $\frac{k^2-3k+6-\epsilon}{k^2-4k+7}$ for arbitrarily small $\epsilon > 0$. We will then show that the adversary can eventually force the algorithm to a competitive ratio at least $\frac{k^2-3k+6}{k^2-4k+7}$, thus leading to a contradiction.

The game begins with the adversary presenting the string 0, which the algorithm has to augment, resulting in a single string 010. From this point onwards the game proceeds in two phases, which are depicted in Figure 2.8. In each phase, a sequence of algorithm/adversary actions takes place. Actions are defined to be consistent with Figure 2.8 for the two phases of the game. It is worth pointing out that the game for $k \geq 6$ consists of two phases, while in contrast, the game for $k = 4$ (as described in the proof of Theorem 2.11) consists of three phases. The second phase for $k = 4$ is necessary for the adversary to force the algorithm to keep augmenting strings.

We also need to explain under which conditions the game transitions from phase 1 to phase 2. To this end, we define again some variables which count the numbers of certain specific strings. More precisely:

— x denotes the number of strings $0123 \ldots i(i-1)i \ldots 3210$ for all $1 \leq i \leq k-2$; such strings have local ratio $\frac{i+2}{i+1}$.

— a_i, where i is even in $[2, k]$, denotes the number of strings $0(i-1)0$ or $01i10$; such strings have local ratio 2 and $3/2$, respectively.

— v denotes the number of strings $0123 \ldots (k-3)0, 0123 \ldots (k-2)10, 0123 \ldots (k-1)210$ or $0123 \ldots k3210$; such strings have local ratio $\frac{k/2}{k/2-1}$, $\frac{k/2+1}{k/2}$, $\frac{k/2+2}{k/2+1}$ and $\frac{k/2+3}{k/2+2}$, respectively.

In phase 1, immediately after each action of the adversary, the algorithm has competitive ratio at least $3/2$. This is because in this phase all strings have local ratio at least $3/2$. However, after each augmentation by the algorithm the competitive ratio can be much better, and possibly smaller than $\frac{k^2-3k+6}{k^2-4k+7}$. For this reason, the adversary will move eventually the game to phase 2. In particular, phase 2 begins once the following condition is satisfied.

$$\sum_{j=1}^{k/2} (2j(k-1) - 3k + 7) \cdot a_{2j} > \frac{2(k-3)}{\epsilon} - (k-1).$$

Note that this condition will be satisfied because the coefficient $(2j(k-1) - 3k + 7)$ is non-negative for $j \geq 2$ and increasing in j for $j \geq 1$, and whenever a_{2j} decreases by one, $a_{2(j+1)}$ increases by one during the execution of phase 1. Intuitively, the

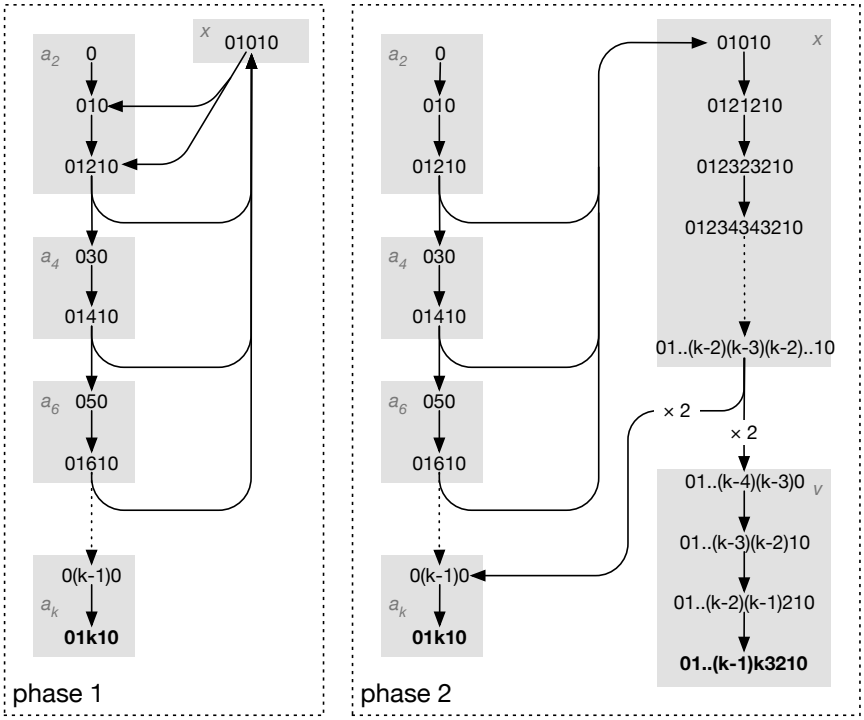


FIGURE 2.8 – The lower bound construction in the arrival/departure model. Blocked strings are depicted in bold face. The arcs illustrate the adversarial strategy. For example if the algorithm augments the string $01\ldots(k-2)(k-3)(k-2)\ldots10$ the adversary replaces the resulting string by two strings $0(k-1)0$ and two strings $01\ldots(k-4)(k-3)0$. The numbers $a_2, a_4, a_6, \ldots, a_k, x, v$ count the number of strings which belong to the corresponding shown boxes.

objective in phase 1 is to create a large number of strings counted by $a_4, a_6, \ldots, a_k$, whereas in phase 2 the objective is to force the algorithm in a configuration with only blocked strings.

Throughout phase 2, we have the invariants

$$\sum_{j=1}^{k/2} (2j(k-1) - 3k + 7) \cdot a_{2j} > \frac{2(k-3)}{\epsilon} - (k-1), \tag{Inv 3.1}$$

$$2x \leq 1 + \sum_{j=1}^{k/2} (2j-3)a_{2j} - (k-2)v. \tag{Inv 3.2}$$

Invariant (Inv 3.1) holds because the left-hand side will not decrease throughout the phase. To show Invariant (Inv 3.2), we first observe that at the end of phase 1, it holds that

$$2x \leq 1 + \sum_{j=1}^{k/2} (2j-3)a_{2j};$$

this can be easily shown by induction on the number of actions during phase 1. Thus, at the beginning of phase 2, Invariant (Inv 3.2) holds, since $v = 0$ at that point. Again, a simple inductive argument on the number of actions throughout phase 2 can show that the invariant is maintained.

We will argue that Invariants (Inv 3.1) and (Inv 3.2) imply that throughout phase 2, the competitive ratio is strictly larger than $\frac{k^2-3k+6-\epsilon}{k^2-4k+7}$, and hence throughout phase 2 the algorithm is forced to augment strings, until all strings are blocked. This is because the competitive ratio in phase 2 can be lower bounded by

$$\begin{aligned}
&\frac{(\frac{k}{2}+3)v + 3\sum_{j=1}^{k/2} a_{2j} + kx}{(\frac{k}{2}+2)v + 2\sum_{j=1}^{k/2} a_{2j} + (k-1)x} \\
&= \frac{(k+6)v + 6\sum_{j=1}^{k/2} a_{2j} + 2kx}{(k+4)v + 4\sum_{j=1}^{k/2} a_{2j} + 2(k-1)x} \\
&\geq \frac{(k+6)v + 6\sum_{j=1}^{k/2} a_{2j} + k\left(1 + \sum_{j=1}^{k/2} (2j-3)a_{2j} - (k-2)v\right)}{(k+4)v + 4\sum_{j=1}^{k/2} a_{2j} + (k-1)\left(1 + \sum_{j=1}^{k/2} (2j-3)a_{2j} - (k-2)v\right)} \\
&\qquad \text{(From P1 and Inv 3.2)} \\
&= \frac{(-k^2+3k+6)v + \sum_{j=1}^{k/2} (k(2j-3)+6)\, a_{2j} + k}{(-k^2+4k+2)v + \sum_{j=1}^{k/2} ((k-1)(2j-3)+4)\, a_{2j} + k - 1}
\end{aligned}$$

To complete the proof, it remains to show that

$$\frac{(-k^2+3k+6)v + \sum_{j=1}^{k/2} (k(2j-3)+6)\, a_{2j} + k}{(-k^2+4k+2)v + \sum_{j=1}^{k/2} ((k-1)(2j-3)+4)\, a_{2j} + k - 1} > \frac{k^2-3k+6-\epsilon}{k^2-4k+7}. \tag{2.13}$$

By a simple mathematical manipulation, for (2.13) to hold, it suffices that

$$v \cdot C_v + \sum_{j=1}^{k/2} a_{2j} \cdot C_{2j} > 2(k-3) - (k-1)\epsilon, \tag{2.14}$$

where C_v and C_{2j} are defined as

$$C_v = 3(k^2 - 7k + 10) - (k^2 - 4k - 2)\epsilon$$
$$C_{2j} = 2(k-3)(k-2j) + (2j(k-1) - 3k + 7)\epsilon.$$

We observe that C_v and the first additive term in the expression of C_{2j} (i.e. $2(k-3)(k-2j)$) are non-negative for sufficiently small ϵ, $k \geq 6$ and $1 \leq j \leq k/2$. Removing these terms from the left hand side of (2.14), it suffices to show that

$$\sum_{j=1}^{k/2} (2j(k-1) - 3k + 7)\epsilon \cdot a_{2j} > 2(k-3) - (k-1)\epsilon.$$

This inequality is precisely Invariant (Inv 3.1), which concludes the proof.

□

2.4 Conclusion

In this chapter we provided improved upper and lower bounds for online maximum matching with k edge-recourse. More specifically, we analyzed two online algorithms for the edge arrival model, namely AMP and L-GREEDY which seem to be incomparable: the former is asymptotically superior, in terms of k, but the latter has a better performance analysis for small k. It would be interesting to analyze an algorithm that combines the ingredients of these two algorithms, namely, an algorithm that combines the doubling techniques with augmenting only along short paths. The difficulty in the analysis of such an algorithm lies in that reasonable charging schemes tend to have "local" properties, wheres the doubling algorithm applies a "global" criterion which does not easily translate into some structural property that can be useful in analysis.

The problems we consider remain challenging even for k as small as 4, and some gap between the upper and lower bounds remains. Bringing this gap will probably require new ideas and techniques. To this end, the amortization arguments we used in our analysis may have connections to LP-based algorithms (since the dual of the maximum matching problem is a weighted vertex minimization problem). Thus, it would be very interesting to use a duality-based approach, such as dual fitting, towards the design and analysis of improved algorithms.

Chapter 3

Searching on the Line Using Discovery Ratio

This chapter contains material from the joint paper, *Best-of-two-worlds Analysis of Online Search*, with Spyros Angelopoulos and Christoph Dürr. This work appeared in the *36th International Symposium on Theoretical Aspects of Computer Science* (STACS), 2019 [4].

3.1 Introduction

Competitive analysis is the standard framework used for evaluating the performance of online algorithms. However, as discussed in Section 1.3, sometimes competitive analysis cannot distinguish the performance of different algorithms due to the worst-case nature of the competitive ratio. In this chapter, we demonstrate that a similar situation arises in the context of online search. More specifically, we study the *linear search* problem. The linear search problem and the corresponding competitive analysis is presented in Section 1.3.1.

In Section 1.3.1, we discussed that both `doubling` and `aggressive` are optimal in terms of competitive ratio, while the latter is better than the former in some situations. In other words, the competitive analysis, on itself, sometimes is not enough to separate performance of different algorithms. To remedy this situation, we introduced the *discovery ratio* as defined in Section 1.3.2 to quantify the eagerness of discovering new territory for a search strategy. We emphasize that we apply the discovery ratio as supplementary to the competitive ratio, instead of using it as a measure that replaces the competitive ratio altogether.

3.1.1 Related Work

The *linear search* problem, also known as *cow path* problem, was first introduced by Bellman [11] and Beck [9]. The optimal deterministic competitive ratio for the linear search problem is equal to 9 and the optimal strategies were first given by Beck and Newman [10]. Moreover, an optimal randomized algorithm was also given by Kao *et al*. [44]

The *Star search* problem is a natural generalization of the linear search problem, in which the search environment consists of m concurrent branches. The linear search problem can be considered as a special case for $m = 2$. The optimal strategies under competitive analysis of the star search problem was given by Gal [33], which was later rediscovered by Baeza-Yates *et al*. [8].

Some variants of the linear search problem have been studied before. For instance, instead of considering only one searcher, a multi-searcher model was studied by López-Ortiz and Schuierer in [49]; The model with multi-target searching was also studied in [46, 53]; Recently, the model in which there is an additional cost for each turn was studied in [25, 2]. In addition, the variant in which the algorithm has some probabilistic information on target placement was explored in [41, 43]. u The linear search problem and its generalization are also related to resource allocation under uncertainty. There are some applications in the field of Artificial Intelligence. For instance, there are some works in the design of *interruptible algorithms* [13, 1]. See also Chapter 4 for another application from Artificial Intelligence on contract scheduling.

Concerning the search strategy `aggressive`, it has been studied in [38, 41]. In these work, `aggressive` is used so as to maximize the *reach* of a strategy, which is defined as the maximum possible extent to which the branches can be searched without violating competitiveness constraints. In contrast, we use the *discovery ratio* to compare strategy `aggressive` with other competitively optimal strategies. To our knowledge, this is the first work that quantifies the intuition that `aggressive` strategy is indeed good.

3.1.2 Contribution

In Section 3.2, we obtain strategies of optimal discovery ratio against all possible strategies. Specifically, we show that there are strategies of discovery ratio $2 + \epsilon$, for arbitrarily small $\epsilon > 0$. However, they have an unbounded competitive ratio. Moreover, we show that the strategy `doubling`, which is optimal under competitive analysis, has discovery ratio equal to 3. This demonstrates that the discovery ratio, on itself, does not lead to a useful classification of strategies, when we consider the entire space of strategies. This motivates us to study the discovery ratio against competitively optimal strategies.

In Section 3.3.1, we restrict our interest to the set of competitively optimal strategies, which we analyze using the discovery ratio as a supplementary measure. As a result, we prove that the strategy `aggressive`, has discovery ratio $\frac{8}{5}$; moreover, we show that this is the optimal discovery ratio in this setting. In contrast, we show that the strategy `doubling` has discovery ratio $\frac{7}{3}$. In addition, we provide evidence that such "aggressiveness" is necessary: more precisely, we show that any competitively optimal strategy that is also optimal with respect to the discovery ratio must have the exact same behavior as the aggressive strategy in the first five iterations. Moreover, in Section 3.3.3 we show the non uniqueness of such competitively optimal strategies with optimal discovery ratio.

In Section 3.4, we provide some computational results to separate the performance of `doubling` and `aggressive`. Based on worst case analysis, we show that `doubling` has a worse discovery ratio because of the worse performance at the very first steps. Computationally, we observe a strong separation on the performance of these two algorithms on average case analysis.

3.1.3 Preliminaries

For the *linear search* problem, the search environment consists of two branches, which are numbered by 0, 1, respectively. The search strategy can be described as an infinite sequence of *search segments* $\{x_0, x_1, \ldots\}$ such that $x_i > 0$ and $x_{i+2} > x_i$ for all $i \in \mathbb{N}$. More precisely, in iteration i, the searcher starts from the origin O and

searches branch i mod 2 to distance x_i from the origin. The searcher completes the iteration by returning back to O. Note that the searcher always prefers to explore a new portion of the line in each iteration, which is characterized by the constraint $x_{i+2} > x_i$. Because any other strategy X that does conform to this property, namely the one in which $x_{i+2} \leq x_i$, or iterations i and $i+1$ search the same branch, can be transformed to a strategy X' such that for any hider H, we have $c(X', S) \leq c(X, H)$.

We say that the searcher *turns* in iteration i when it switches directions during iteration i. More precisely, it is the moment when it completes the exploration of length x_i and returns back to the origin. Eventually, both branches of the line have to be completely explored. Formally speaking, for every $d \in \mathbb{R}^+$, there exist $i, j \in \mathbb{N}$ such that $x_{2i} \geq d$, and $x_{2j+1} \geq d$.

We denote by Σ the set of all search strategies, and by Σ_c the set of strategies with competitive ratio c. Σ_c is then a subset of Σ. Since it is well known that the optimal competitive ratio is 9 for the linear search problem, we denote by Σ_9 the set of competitively optimal strategies. Recall that there is also a standard assumption that the hider must be at distance at least 1 from O, since no strategy can have bounded competitive ratio if this distance can be arbitrarily small.

For convenience of notation, we define $x_i := 0$, for all $i < 0$. Given a strategy X, we define $T_n(X)$ (or simply T_n, when X is clear from context) to be equal to $\sum_{i=0}^{n} x_i$. For $n < 0$, we define $T_n := 0$.

3.2 Strategies of Optimal Discovery Ratio in Σ

In this section, we setup some properties of the measure and obtain strategies of optimal discovery ratio in Σ.

Let X, Y, denote two strategies in Σ, with $X = (x_0, x_1, \ldots)$. From the definition of the discovery ratio we have that

$$\mathrm{dr}(X, Y) = \sup_{i \in \mathbb{N}} \sup_{\delta \in (0, x_i - x_{i-2}]} \frac{D(X, x_{i-1} + x_{i-2} + \delta)}{D(Y, x_{i-1} + x_{i-2} + \delta)}.$$

Note that for $i = 0$, we have

$$\frac{D(X, x_{i-1} + x_{i-2} + \delta)}{D(Y, x_{i-1} + x_{i-2} + \delta)} = \frac{D(X, \delta)}{D(Y, \delta)} \leq \frac{\delta}{\delta} = 1.$$

This is because for all $\delta \leq x_0$, $D(X, \delta) = \delta$, and for all $\delta > 0$, $D(Y, \delta) \geq \delta$. Therefore,

$$\mathrm{dr}(X, Y) = \sup_{i \in \mathbb{N}^*} \sup_{\delta \in (0, x_i - x_{i-2}]} \frac{D(X, x_{i-1} + x_{i-2} + \delta)}{D(Y, x_{i-1} + x_{i-2} + \delta)}. \tag{3.1}$$

The following theorem provides an expression of the discovery ratio in terms of the search segments of the strategy.

Theorem 3.1. *Let $X = (x_0, x_1, \ldots)$. Then*

$$dr(X, \Sigma) = \sup_{i \in \mathbb{N}^*} \frac{2 \sum_{j=0}^{i-1} x_j + x_{i-2}}{x_{i-1} + x_{i-2}}.$$

Proof. Fix $Y \in \Sigma$. From the definition of search segments in X, we have that

$$D(X, x_{i-1} + x_{i-2} + \delta) = 2\sum_{j=0}^{i-1} x_j + x_{i-2} + \delta, \quad \text{for } \delta \in (0, x_i - x_{i-2}]. \tag{3.2}$$

Moreover, for every Y, we have

$$D(Y, x_{i-1} + x_{i-2} + \delta) \geq x_{i-1} + x_{i-2} + \delta. \tag{3.3}$$

Substituting (3.2) and (3.3) in (3.1) we obtain

$$\begin{aligned} \mathrm{dr}(X, Y) &\leq \sup_{i \in \mathbb{N}^*} \sup_{\delta \in (0, x_i - x_{i-2}]} \frac{2\sum_{j=0}^{i-1} x_j + x_{i-2} + \delta}{x_{i-1} + x_{i-2} + \delta} \\ &\leq \sup_{i \in \mathbb{N}^*} \frac{2\sum_{j=0}^{i-1} x_j + x_{i-2}}{x_{i-1} + x_{i-2}}. \end{aligned}$$

For the lower bound, consider a strategy $Y_i = (y_0^i, y_1^i, \ldots)$, for which $y_0^i = x_{i-1} + x_{i-2} + \delta$ (the values of y_j^i for $j \neq 0$ are not significant, as long as Y_i is a valid strategy). Clearly, we have

$$D(Y_i, x_{i-1} + x_{i-2} + \delta) = x_{i-1} + x_{i-2} + \delta.$$

Therefore, (3.1) implies

$$\begin{aligned} \mathrm{dr}(X, Y_i) &\geq \sup_{\delta \in (0, x_i - x_{i-2}]} \frac{2\sum_{j=0}^{i-1} x_j + x_{i-2} + \delta}{x_{i-1} + x_{i-2} + \delta} \\ &= \frac{2\sum_{j=0}^{i-1} x_j + x_{i-2}}{x_{i-1} + x_{i-2}}. \end{aligned}$$

The lower bound on $\mathrm{dr}(X, \Sigma)$ follows from $\mathrm{dr}(X, \Sigma) \geq \sup_{i \in \mathbb{N}^*} \mathrm{dr}(X, Y_i)$. □

In particular, note that for $i = 2$, Theorem 3.1 shows that for any strategy X,

$$\mathrm{dr}(X, \Sigma) \geq \frac{3x_0 + 2x_1}{x_0 + x_1} \geq 2.$$

We will show that there exist strategies with discovery ratio arbitrarily close to 2, thus optimal for Σ. To this end, we will consider the geometric search strategy defined as $G_\alpha = (1, \alpha, \alpha^2, \ldots)$, with $\alpha > 1$.

Lemma 3.2. *For G_α defined as above, we have $dr(G_\alpha, \Sigma) = \frac{2\alpha^2 + \alpha - 1}{\alpha^2 - 1}$.*

Proof. From Theorem 3.1 we have

$$\begin{aligned} \mathrm{dr}(G_\alpha, \Sigma) &= \sup_{i \in \mathbb{N}^*} \frac{2\sum_{j=0}^{i-1} \alpha^j + \alpha^{i-2}}{\alpha^{i-1} + \alpha^{i-2}} \\ &= \sup_{i \in \mathbb{N}^*} \frac{2(\frac{\alpha^i - 1}{\alpha - 1}) + \alpha^{i-2}}{\alpha^{i-1} + \alpha^{i-2}} \\ &= \sup_{i \in \mathbb{N}^*} \frac{2(\alpha^i - 1) + \alpha^{i-1} - \alpha^{i-2}}{\alpha^i - \alpha^{i-2}}. \end{aligned}$$

The derivative of the function $f(i) := \frac{2(\alpha^i-1)+\alpha^{i-1}-\alpha^{i-2}}{\alpha^i-\alpha^{i-2}}$ in i is

$$f'(i) = \frac{2\alpha^{2-i}\log\alpha}{\alpha^2-1},$$

which is positive. Thus, $\sup_{i\in\mathbb{N}^*} f(i) = \lim_{i\to\infty} f(i)$, which gives

$$\begin{aligned}\mathrm{dr}(G_\alpha,\Sigma) &= \lim_{i\to+\infty} f(i)\\ &= \lim_{i\to+\infty} \frac{2(\alpha^i-1)+\alpha^{i-1}-\alpha^{i-2}}{\alpha^i-\alpha^{i-2}}\\ &= \frac{2\alpha^2+\alpha-1}{\alpha^2-1}.\end{aligned}$$

□

Figure 3.1 depicts the tradeoff between the competitive and discovery ratios as attained by the search strategy G_α.

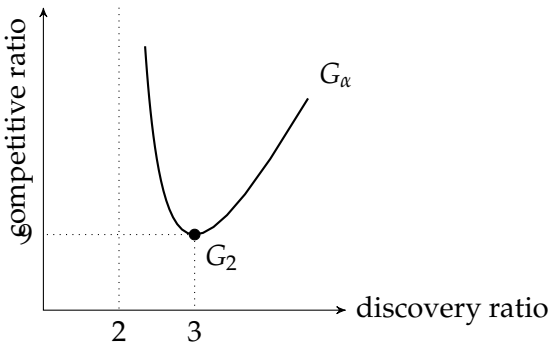


FIGURE 3.1 – Illustration of the tradeoff between the competitive and discovery ratios in G_α. Here, each point corresponds to a value of $\alpha > 1$.

In particular, Lemma 3.2 shows that the discovery ratio of G_α tends to 2, as $\alpha \to \infty$, hence G_α has asymptotically optimal discovery ratio. However its competitive ratio is unbounded. Furthermore, strategy `doubling` (i.e. G_2) has optimal competitive ratio equal to 9, whereas its discovery ratio is equal to 3. This motivates the topic of the next section.

3.3 The Discovery Ratio of Competitively Optimal Strategies

3.3.1 Properties of Competitively Optimal Strategies

In this section we focus on strategies in Σ_9, namely the set of competitively optimal strategies. For any strategy $X \in \Sigma_9$, it is known that there is an infinite set of linear inequalities that relate its search segments, as shown in the following lemma (see, e.g, [41]).

Lemma 3.3. *The strategy $X = (x_0, x_1, x_2, \ldots)$ is in Σ_9 if and only if its segments satisfy the following inequalities*

$$1 \leq x_0 \leq 4, \quad x_1 \geq 1 \quad \text{and} \quad x_n \leq 3x_{n-1} - \sum_{i=0}^{n-2} x_i, \quad \text{for all } n \geq 1.$$

Proof. It is well-known that the competitive ratio of X is determined by seeking, for all $n \geq 0$, a target that is placed at distances $x_n + \epsilon$, where $\epsilon \to 0$, and in the same branch that is searched by X in iteration n, namely branch n mod 2; call this target the *n-th target*. The cost incurred by X for locating the $(n-1)$-th target, where $n \geq 1$ is equal to $2(\sum_{i=0}^{n} x_i) + x_{n-1} + \epsilon$, whereas the optimal cost is $x_{n-1} + \epsilon$. From the definition of the competitive ratio, and since $\epsilon \to 0$, we obtain that

$$2\sum_{i=0}^{n} x_i + x_{n-1} \leq 9 \cdot x_{n-1} \Rightarrow x_n \leq 3x_{n-1} - \sum_{i=0}^{n-2} x_i.$$

Moreover, we can obtain one more inequality that involves x_0, by assuming a target placed at distance 1 from O in branch 1. Thus, we obtain that $2x_0 + 1 \leq 9$, or, equivalently, $x_0 \leq 4$.

Last, note that $x_0, x_1 \geq 1$ from the assumption that the target is at distance at least 1 from the origin. □

We now define a class of strategies in Σ_9 as follows. For given $t \in [1, 4]$, let R_t denote the strategy whose search segments are determined by the linear recurrence

$$x_0 = t, \quad \text{and} \quad x_n = 3x_{n-1} - \sum_{i=0}^{n-2} x_i, \quad \text{for all } n \geq 1.$$

In words, R_t is such that for every $n > 1$, the inequality relating $x_0, \ldots, x_n$ is tight. The following lemma determines the search lengths of R_t as a function of t. The lemma also implies that R_t is indeed a valid search strategy, for all $t \in [1, 4]$, in that $x_n > x_{n-2}$, for all n, and $x_n \to \infty$, as $n \to \infty$.

Lemma 3.4. *The strategy R_t is defined by the sequence $x_n = t(1 + \frac{n}{2})2^n$, for $n \geq 0$. Moreover, $T_n(R_t) = t(n+1)2^n$.*

Proof. The lemma is clearly true for $n \in \{0, 1\}$. For $n \geq 2$, the equality $x_n = 3x_{n-1} - \sum_{i=0}^{n-2} x_i$ implies that $T_n = \sum_{i=0}^{n} x_i = 4x_{n-1}$. Therefore,

$$T_n - T_{n-1} = 4x_{n-1} - 4x_{n-2} \Rightarrow x_n = 4(x_{n-1} - x_{n-2}).$$

The characteristic polynomial of the above linear recurrence is $\xi^2 - 4\xi + 4$, with the unique root $\xi = 2$. Thus, x_n is of the form $x_n = (a + bn)2^n$, for $n \geq 0$, where a and b are determined by the initial conditions $x_0 = t$ and $x_1 = 3t$. Summarizing, we obtain that for $n \geq 0$ we have that $x_n = t(1 + \frac{n}{2})2^n$, and $T_n = 4x_{n-1} = t(n+1)2^n$. □

Among all strategies in R_t we are interested, in particular, in the strategy R_4. This strategy has some intuitively appealing properties: It maximizes the search segments in each iteration (see Lemma 3.6) and minimizes the number of turns required to discover a certain length (as will be shown in Corollary 3.7). Using the notation of the introduction, we can say that $R_4 \equiv$ `aggressive`. In this section, we will show that `aggressive` has optimal discovery ratio among all competitively optimal strategies. Let us denote by $\bar{x}_i$ the search segment in the i-th iteration in `aggressive`.

Corollary 3.5. *The strategy* `aggressive` *can be described by the sequence* $\bar{x}_n = (n+2)2^{n+1}$*, for* $n \geq 0$*. Moreover,* $T_n(\texttt{aggressive}) = (n+1)2^{n+2}$*, for* $n \geq 0$*.*

The following lemma shows that, for any given n, the total length discovered by any competitively optimal strategy X at the turning point of the n-th iteration cannot exceed the corresponding length of `aggressive`. Its proof can also be found in [41], but we give a different proof using ideas that we will apply later (Lemma 3.8).

Lemma 3.6. *For every strategy* $X = (x_0, x_1, \ldots)$ *with* $X \in \Sigma_9$*, it holds that* $x_n \leq \bar{x}_n$*, for all* $n \in \mathbb{N}$*, where* $\bar{x}_n$ *is the search segment in the* n*-th iteration of* `aggressive`*. Hence, in particular, we have* $x_n + x_{n-1} \leq \bar{x}_n + \bar{x}_{n-1}$*, for all* $n \in \mathbb{N}$*.*

Proof. For a given $n \geq 0$, let P_n denote the following linear program.

$$\begin{aligned} \max \quad & x_n \\ \text{subject to} \quad & 1 \leq x_0 \leq 4, \\ & x_1 \geq 1, \\ & x_i \leq 3x_{i-1} - \sum_{j=0}^{i-2} x_j, \quad 1 \leq i \leq n. \end{aligned}$$

We will show, by induction on i, that for all $i \leq n$,

$$x_n \leq (i+2)2^{i-1}x_{n-i} - i2^{i-1}T_{n-i-1}(X).$$

The lemma will then follow, since for $i = n$ we have

$$x_n \leq (n+2)2^{n-1}x_0 \leq (n+2)2^{n-1} \cdot 4 = (n+2)2^{n+1} = \bar{x}_n,$$

where the last equality is due to Corollary 3.5. We will now prove the claim. Note that, the base case, namely $i = 1$, follows directly from the LP constraint. For the induction hypothesis, suppose that for $i \geq 1$, it holds that

$$x_n \leq (i+2)2^{i-1}x_{n-i} - i2^{i-1}T_{n-i-1}(X). \tag{3.4}$$

We will show that the claim holds for $i+1$. Since

$$x_{n-i} \leq 3x_{n-i-1} - T_{n-i-2}(X), \tag{3.5}$$

then

$$\begin{aligned} x_n &\leq (i+2)2^{i-1}(3x_{n-i-1} - T_{n-i-2}(X)) - i2^{i-1}T_{n-i-1}(X) && \text{(subst. (3.5) in (3.4))} \\ &= (i+2)2^{i-1}(3x_{n-i-1} - T_{n-i-2}(X)) - i2^{i-1}(T_{n-i-2}(X) + x_{n-i-1}) && \text{(def. } T_{n-i-1}\text{)} \\ &= (i+3)2^i x_{n-i-1} - (i+1)2^i T_{n-i-2}(X), && \text{(arranging terms)} \end{aligned}$$

which completes the proof of the claim. □

Given strategy X and $l \in \mathbb{R}^+$, define $m(X, l)$ as the number of turns that X has performed by the time it discovers a total length equal to l. Also define

$$m^*(l) = \inf_{X \in \Sigma_9} m(X, l),$$

that is, $m^*(l)$ is the minimum number of turns that a competitively optimal strategy is required to perform in order to discover length equal to l. From the constraint

$x_0 \leq 4$, it follows that clearly $m^*(l) = 0$, for $l \leq 4$. The following corollary to Lemma 3.6 gives an expression for $m^*(l)$, for general values of l.

Corollary 3.7. *For given $l > 4$, $m^*(l) = m(\texttt{aggressive}, l) = \min\{n \in \mathbb{N}_{\geq 1} : (3n+5)2^n \geq l\}$.*

Proof. From Lemma 3.6, the total length discovered by any $X \in \Sigma_9$ at the turning point of the n-th iteration cannot exceed $\bar{x}_n + \bar{x}_{n-1}$ for $n \geq 1$, which implies that $m^*(l) = n$, if $l \in (\bar{x}_{n-1} + \bar{x}_{n-2}, \bar{x}_n + \bar{x}_{n-1}]$ for $n \geq 1$. In other words,

$$m^*(l) = \min\{n \in \mathbb{N}_{\geq 1} : \bar{x}_n + \bar{x}_{n-1} \geq l\}.$$

From Corollary 3.5, we have $\bar{x}_n = (n+2)2^{n+1}$, for $n \geq 0$. Hence,

$$m^*(l) = \min\{n \in \mathbb{N}_{\geq 1} : (3n+5)2^n \geq l\}.$$

□

The following lemma is a central technical result that is instrumental in establishing the bounds on the discovery ratio. For a given $l \in \mathbb{R}^+$, define

$$d^*(l) = \inf_{X \in \Sigma_9} D(X, l).$$

In words, $d^*(l)$ is the minimum cost at which a competitively optimal strategy can discover a length equal to l. Trivially, $d^*(l) = l$ if $l \leq 4$. Lemma 3.8 gives an expression of $d^*(l)$ for $l > 4$ in terms of $m^*(l)$; it also shows that there exists a $t \in (1, 4]$ such that the strategy R_t attains this minimum cost.

We first give some motivation behind the purpose of the lemma. When considering general strategies in Σ, we used a lower bound on the cost for discovering a length l as given by (3.3), and which corresponds to a strategy that never turns. However, this lower bound is very weak when one considers strategies in Σ_9. This is because a competitive strategy needs to turn sufficiently often, which affects considerably the discovery costs.

We also give some intuition about the proof. We show how to model the question by means of a linear program. Using the constraints of the LP, we first obtain a lower bound on its objective in terms of the parameters l and $m^*(l)$. In this process, we also obtain a lower bound on the first segment of the strategy (x_0); this is denoted by t in the proof. In the next step, we show that the strategy R_t has discovery cost that matches the lower bound on the objective, which suffices to prove the result.

Lemma 3.8. *For $l > 4$, it holds*

$$d^*(l) = D(R_t, l) = l \cdot \frac{6m^*(l)+4}{3m^*(l)+5}, \quad \text{where } t = l \cdot \frac{2^{2-m^*(l)}}{3m^*(l)+5} \in (1, 4].$$

Proof. Let $X = (x_0, x_1, \ldots) \in \Sigma_9$ denote the strategy which minimizes the quantity $D(X, l)$. Then there must exist a smallest $n \geq m^*(l)$ such that the searcher discovers a total length l during the n-th iteration. More precisely, suppose that this happens when the searcher is at branch n mod 2, and at some position p (i.e., distance from O), with $p \in (x_{n-2}, x_n]$. Then we have $x_{n-1} + p = l$, and

$$d^*(l) = D(X, l) = 2\sum_{i=0}^{n-1} x_i + p = 2\sum_{i=0}^{n-1} x_i + (l - x_{n-1}) = 2\sum_{i=0}^{n-2} x_i + x_{n-1} + l.$$

Therefore, $d^*(l)$ is the objective of the following linear program.

$$\begin{aligned} \min \quad & 2\sum_{i=0}^{n-2} x_i + x_{n-1} + l \\ \text{subject to} \quad & x_n + x_{n-1} \geq l, \\ & 1 \leq x_0 \leq 4, \\ & x_{i-2} \leq x_i, && i \in [2, n] \\ & 1 \leq x_i \leq 3x_{i-1} - \sum_{j=0}^{i-2} x_j, && i \in [1, n]. \end{aligned}$$

Recall that $n \geq m^*(l)$ is a fixed integer. Let Obj denote the objective value of the linear program. We claim that, for $1 \leq i \leq n$,

$$x_{n-i} \geq \frac{2^{2-i}}{3i+5} l + \frac{3i-1}{3i+5} T_{n-i-1} \quad \text{and} \quad \text{Obj} \geq \frac{6i+4}{3i+5} l + \frac{9 \cdot 2^i}{3i+5} T_{n-i-1}.$$

The claim provides a lower bound of the objective, since for $i = n$ it implies that

$$x_0 \geq \frac{2^{2-n}}{3n+5} l \quad \text{and} \quad \text{Obj} \geq \frac{6n+4}{3n+5} l \geq \frac{6m^*(l)+4}{3m^*(l)+5} l,$$

where the last inequality follows from the fact $n \geq m^*(l)$. We will argue later that this lower bound is tight.

First, we prove the claim, by induction on i, for all $i \leq n$. We first show the base case, namely $i = 1$. Since $x_n \leq 3x_{n-1} - T_{n-2}$ and $x_n + x_{n-1} \geq l$, it follows that

$$x_{n-1} \geq l - x_n \geq l - (3x_{n-1} - T_{n-2}) \Rightarrow x_{n-1} \geq \frac{l}{4} + \frac{T_{n-2}}{4},$$

hence,

$$\text{Obj} = l + 2T_{n-2} + x_{n-1} \geq l + 2T_{n-2} + \frac{l}{4} + \frac{T_{n-2}}{4} = \frac{5}{4} l + \frac{9}{4} T_{n-2},$$

thus the base case holds. For the induction step, suppose that

$$x_{n-i} \geq \frac{2^{2-i}}{3i+5} l + \frac{3i-1}{3i+5} T_{n-i-1} \quad \text{and} \quad \text{Obj} \geq \frac{6i+4}{3i+5} l + \frac{9 \cdot 2^i}{3i+5} T_{n-i-1}.$$

Then,

$$\begin{aligned} 3x_{n-i-1} - T_{n-i-2} &\geq x_{n-i} && \text{(by LP constraint)} \\ &\geq \frac{2^{2-i}}{3i+5} l + \frac{3i-1}{3i+5} T_{n-i-1} && \text{(ind. hyp.)} \\ &= \frac{2^{2-i}}{3i+5} l + \frac{3i-1}{3i+5} (T_{n-i-2} + x_{n-i-1}) && \text{(def. } T_{n-i-1}\text{)} \end{aligned}$$

By rearranging terms in the above inequality we obtain

$$\begin{aligned}(3-\frac{3i-1}{3i+5})x_{n-i-1} &\geq \frac{2^{2-i}}{3i+5}l+(1+\frac{3i-1}{3i+5})T_{n-i-2} \quad \Rightarrow\\ \frac{6i+16}{3i+5}x_{n-i-1} &\geq \frac{2^{2-i}}{3i+5}l+\frac{6i+4}{3i+5}T_{n-i-2} \quad \Rightarrow\\ x_{n-i-1} &\geq \frac{2^{1-i}}{3i+8}l+\frac{3i+2}{3i+8}T_{n-i-2},\end{aligned}$$

and

$$\begin{aligned}\text{Obj} &\geq \frac{6i+4}{3i+5}l+\frac{9\cdot 2^i}{3i+5}T_{n-i-1} && \text{(ind. hyp.)}\\ &= \frac{6i+4}{3i+5}l+\frac{9\cdot 2^i}{3i+5}(T_{n-i-2}+x_{n-i-1}) && \text{(def. } T_{n-i-1})\\ &\geq \frac{6i+4}{3i+5}l+\frac{9\cdot 2^i}{3i+5}T_{n-i-2}+\frac{9\cdot 2^i}{3i+5}(\frac{2^{1-i}}{3i+8}l+\frac{3i+2}{3i+8}T_{n-i-2}) && \text{(ind. hyp.)}\\ &= \frac{6i+10}{3i+8}l+\frac{9\cdot 2^{i+1}}{3i+8}T_{n-i-2}.\end{aligned}$$

This concludes the proof of the claim, which settles the lower bound on $d^*(l)$. It remains to show that this bound is tight. Consider the strategy R_t, with $t=\frac{2^{2-m^*(l)}}{3m^*(l)+5}l$. In what follows we will show that R_t is a feasible solution of the LP, and that $D(R_t,l)=\frac{6m^*(l)+4}{3m^*(l)+5}l$.

First, we show that $t\in(1,4]$. For the upper bound, from Corollary 3.7, we have $(3m^*(l)+5)2^{m^*(l)}\geq l$, which implies that

$$\begin{aligned}1 &\geq l\cdot\frac{2^{-m^*(l)}}{3m^*(l)+5} \quad \Rightarrow\\ 4 &\geq l\cdot\frac{2^{2-m^*(l)}}{3m^*(l)+5} \quad \Rightarrow\\ 4 &\geq t.\end{aligned}$$

In order to show that $t>1$, consider first the case $l\in(4,5]$. Then $m^*(l)=1$, which implies that

$$\begin{aligned}t &= \frac{2^{2-m^*(l)}}{3m^*(l)+5}l\\ &= \frac{l}{4}\geq 1.\end{aligned}$$

Moreover, if $l>5$, by Corollary 3.7, $m^*(l)$ is the smallest integer solution of the inequality $(3n+5)2^n\geq l$, then

$$(3m^*(l)+2)2^{m^*(l)-1}<l, \tag{3.6}$$

hence

$$\begin{aligned}
t &= \frac{2^{2-m^*(l)}}{3m^*(l)+5} \cdot l && \text{(by definition of } t\text{)} \\
&= \frac{4l}{(3m^*(l)+5)2^{m^*(l)}} && \text{(arranging terms)} \\
&= \frac{2l}{(3m^*(l)+2)2^{m^*(l)-1} \cdot \frac{3m^*(l)+5}{3m^*(l)+2}} && \text{(arranging terms)} \\
&> \frac{2l}{l \cdot \frac{3m^*(l)+5}{3m^*(l)+2}} && \text{(substituting Inequation 3.6)} \\
&= \frac{6m^*(l)+4}{3m^*(l)+5} && \text{(arranging terms)} \\
&> 1.
\end{aligned}$$

The last inequality holds since we have $m^*(l) \geq 1$, for $l > 5$. This concludes that $t \in (1, 4]$, and R_t is a feasible solution of the LP since R_t satisfies all other constraints by its definition.

It remains thus to show that $D(R_t, l) = \frac{6m^*(l)+4}{3m^*(l)+5} \cdot l$. By Lemma 3.4, we have

$$\begin{aligned}
x_{m^*(l)} + x_{m^*(l)-1} &= t\left(1 + \frac{m^*(l)}{2}\right) 2^{m^*(l)} + t\left(1 + \frac{m^*(l)-1}{2}\right) 2^{m^*(l)-1} \\
&= t \cdot 2^{m^*(l)} \cdot \frac{3m^*(l)+5}{4} \\
&= \left(\frac{2^{2-m^*(l)}}{3m^*(l)+5} l\right) \cdot 2^{m^*(l)} \cdot \frac{3m^*(l)+5}{4} \\
&= l.
\end{aligned}$$

Then R_t has exactly discovered a total length l right before the $m^*(l)$-th turn. Hence,

$$\begin{aligned}
D(R_t, l) &= 2T_{m^*(l)-2} + x_{m^*(l)-1} + l \\
&= t \cdot (m^*(l)-1)\, 2^{m^*(l)-1} + t \cdot \left(1 + \frac{m^*(l)-1}{2}\right) 2^{m^*(l)-1} + l \\
& && \text{(by Lemma 3.4)} \\
&= t \cdot \frac{(3m^*(l)-1)2^{m^*(l)}}{4} + l && \text{(arranging terms)} \\
&= \frac{2^{2-m^*(l)}}{3m^*(l)+5} l \cdot \frac{(3m^*(l)-1)2^{m^*(l)}}{4} + l && \text{(substituting } t\text{)} \\
&= \left(\frac{3m^*(l)-1}{3m^*(l)+5} + 1\right) \cdot l = \frac{6m^*(l)+4}{3m^*(l)+5} \cdot l. && \text{(arranging terms)}
\end{aligned}$$

This concludes the proof of the lemma. □

We are now ready to prove the main results of this section. Recall that for any two strategies X, Y, $\mathrm{dr}(X, Y)$ is given by (3.1). Combining with (3.2), as well as with the fact that for $Y \in \Sigma_9$, we have that $D(Y, l) \geq d^*(l)$, (from the definition of d^*), we

obtain that

$$\mathrm{dr}(X,\Sigma_9) = \sup_{i\in\mathbb{N}^*} \sup_{\delta\in(0,x_i-x_{i-2}]} F_i(X,\delta), \quad \text{where } F_i(X,\delta) = \frac{2\sum_{j=0}^{i-1} x_j + x_{i-2} + \delta}{d^*(x_{i-1}+x_{i-2}+\delta)}. \tag{3.7}$$

Recall that for the strategy `aggressive` $\equiv R_4 = (\bar{x}_0, \bar{x}_1, \ldots)$, its segments $\bar{x}_i$ are given in Corollary 3.5.

3.3.2 Discovery Ratio of Strategies in Σ_9

We are now able to compute the discovery ratio of the strategy `aggressive` against Σ_9.

Theorem 3.9. *For the strategy* `aggressive` *it holds that* $dr(\texttt{aggressive}, \Sigma_9) = 8/5$.

Proof. We will express the discovery ratio using (3.7). For $i = 1$, and $\delta \in (0, \bar{x}_1]$, we have that

$$F_1(\texttt{aggressive},\delta) = \frac{2\bar{x}_0+\delta}{d^*(\bar{x}_0+\delta)} = \frac{8+\delta}{d^*(4+\delta)}.$$

From Lemma 3.8,

$$d^*(4+\delta) = (4+\delta)\cdot\frac{6\cdot 1+4}{3\cdot 1+5} = \frac{5(4+\delta)}{4};$$

this is because $1 \le m^*(4+\delta) \le m^*(16) = 1$. Then,

$$F_1(\texttt{aggressive},\delta) = \frac{8+\delta}{\frac{5(4+\delta)}{4}} = \frac{32+4\delta}{20+5\delta}.$$

Hence,

$$\sup_{\delta\in(0,\bar{x}_1]} F_1(\texttt{aggressive},\delta) = \frac{8}{5}. \tag{3.8}$$

For given $i \ge 2$, and $\delta \in (0, \bar{x}_i - \bar{x}_{i-2}]$, we have

$$F_i(\texttt{aggressive},\delta) = \frac{2T_{i-1}+\bar{x}_{i-2}+\delta}{d^*(\bar{x}_{i-1}+\bar{x}_{i-2}+\delta)},$$

where T_{i-1} is given by Corollary 3.5. Moreover, from Lemma 3.8 we have that

$$\begin{aligned} d^*(\bar{x}_{i-1}+\bar{x}_{i-2}+\delta) &= (\bar{x}_{i-1}+\bar{x}_{i-2}+\delta)\cdot\frac{6m^*(\bar{x}_{i-1}+\bar{x}_{i-2}+\delta)+4}{3m^*(\bar{x}_{i-1}+\bar{x}_{i-2}+\delta)+5} \\ &= (\bar{x}_{i-1}+\bar{x}_{i-2}+\delta)\cdot\frac{6i+4}{3i+5}, \end{aligned}$$

where the last equality follows from the fact that $m^*(\bar{x}_{i-1}+\bar{x}_{i-2}+\delta) = i$. This is because

$$\begin{aligned} i &\le m^*(\bar{x}_{i-1}+\bar{x}_{i-2}+\delta) \\ &\le m^*(\bar{x}_{i-1}+\bar{x}_{i-2}+\bar{x}_i-\bar{x}_{i-2}) \\ &= m^*(\bar{x}_i+\bar{x}_{i-1}) \\ &= i. \end{aligned}$$

Substituting with the values of the search segments as well as T_{i-1}, we obtain that

$$F_i(\texttt{aggressive},\delta) = \frac{i\cdot 2^{i+2}+i\cdot 2^{i-1}+\delta}{((i+1)2^i+i\cdot 2^{i-1}+\delta)\cdot\frac{6i+4}{3i+5}} = \frac{9i\cdot 2^{i-1}+\delta}{((3i+2)2^{i-1}+\delta)\cdot\frac{6i+4}{3i+5}}.$$

Since

$$\frac{\partial F_i(\texttt{aggressive},\delta)}{\partial\delta} = -\frac{2^{i+1}(3i-1)(3i+5)}{(3i+2)(2^n(3i+2)+2\delta)^2} \le 0,$$

then $F_i(\texttt{aggressive},\delta)$ is monotone decreasing in δ. Thus

$$\sup_{\delta\in(0,\bar{x}_i-\bar{x}_{i-2}]} F_i(\texttt{aggressive},\delta) = \frac{9i\cdot 2^{i-1}}{((3i+2)2^{i-1})\cdot\frac{6i+4}{3i+5}} = \frac{9i(3i+5)}{(3i+2)(6i+4)},$$

and then

$$\sup_{i\in\mathbb{N}_{i\ge 2}}\sup_{\delta\in(0,\bar{x}_i-\bar{x}_{i-2}]} F_i(\texttt{aggressive},\delta) = \frac{(9\cdot 2)(3\cdot 2+5)}{(3\cdot 2+2)(6\cdot 2+4)} = \frac{99}{64} < \frac{8}{5}. \tag{3.9}$$

Combining (3.7), (3.8) and (3.9) yields the proof of the theorem. □

The following theorem shows that aggressive has optimal discovery ratio among all competitively optimal strategies.

Theorem 3.10. *For every strategy $X \in \Sigma_9$, we have $dr(X,\Sigma_9) \ge \frac{8}{5}$.*

Proof. Let $X = (x_0,\ldots)$. We will consider two cases, depending on whether $x_0 < 4$ or $x_0 = 4$. Suppose, first, that $x_0 < 4$. In this case, for sufficiently small ϵ, we have $m^*(x_0+\epsilon) = 0$, which implies that $d^*(x_0+\epsilon) = x_0+\epsilon$, and therefore.

$$F_1(X,\epsilon) = \frac{2x_0+\epsilon}{d^*(x_0+\epsilon)} = \frac{2x_0+\epsilon}{x_0+\epsilon},$$

from which we obtain that

$$\sup_{\delta\in(0,x_1]} F_1(X,\delta) \ge F_1(X,\epsilon) \ge \frac{2x_0+\epsilon}{x_0+\epsilon} \to 2, \text{ as } \epsilon\to 0^+.$$

Next, suppose that $x_0 = 4$. In this case, for $\delta \in (0,x_1]$, it readily follows that $F_1(X,\delta) = F_1(\texttt{aggressive},\delta)$. Therefore, from (3.8), we have that

$$\sup_{\delta\in(0,x_1]} F_1(X,\delta) = \sup_{\delta\in(0,x_1]} \frac{32+4\delta}{20+5\delta} = \frac{8}{5}.$$

The lower bound follows directly from (3.7). □

Recall that doubling $\equiv G_2 = (2^0,2^1,2^2,\ldots)$. The following theorem shows that within Σ_9, doubling has worse discovery ratio than aggressive. The proof follows along the lines of the proof of Theorem 3.9, where instead of using the search segments $\bar{x}_i$ of aggressive, we use the search segment $x_i = 2^i$ of doubling.

Theorem 3.11. *We have $dr(\texttt{doubling},\Sigma_9) = \frac{7}{3}$.*

Proof. We will express the discovery ratio using (3.7). For $i = 1$, and $\delta \in (0, x_1]$, we have that

$$F_1(G_2, \delta) = \frac{2x_0 + \delta}{d^*(x_0 + \delta)} = \frac{2 + \delta}{d^*(1 + \delta)}.$$

From the definition, $d^*(1 + \delta) = 1 + \delta$; this is because $0 \leq m^*(1 + \delta) \leq m^*(3) = 0$. Then,

$$F_1(G_2, \delta) = \frac{2 + \delta}{1 + \delta}, \text{ hence } \sup_{\delta \in (0, \bar{x}_1]} F_1(G_2, \delta) = 2. \tag{3.10}$$

For $i = 2$ and $\delta \in (0, x_2 - x_0]$, we have that

$$F_2(G_2, \delta) = \frac{3x_0 + 2x_1 + \delta}{d^*(x_0 + x_1 + \delta)} = \frac{7 + \delta}{d^*(3 + \delta)}.$$

From Lemma 3.8, $d^*(3 + \delta)$ either equals to $3 + \delta$ if $\delta \in (0, 1]$, or equals to $(3 + \delta) \cdot \frac{6 \cdot 1 + 4}{3 \cdot 1 + 5} = \frac{5(3+\delta)}{4}$ if $\delta \in (1, x_2 - x_0]$. This is because $0 \leq m^*(3 + \delta) \leq m^*(6) = 1$. Then, for $\delta \in (0, 1]$,

$$F_2(G_2, \delta) = \frac{7 + \delta}{3 + \delta}, \text{ hence } \sup_{\delta \in (0, 1]} F_2(G_2, \delta) = \frac{7}{3}. \tag{3.11}$$

For $\delta \in (1, x_2 - x_0]$,

$$F_2(G_2, \delta) = \frac{7 + \delta}{\frac{5(3+\delta)}{4}} = \frac{28 + 4\delta}{15 + 5\delta}, \text{ hence } \sup_{\delta \in (1, x_2 - x_0]} F_2(G_2, \delta) = \frac{28}{15}. \tag{3.12}$$

Combining (3.11) and (3.12) yields

$$\sup_{\delta \in (0, x_2 - x_0]} F_2(G_2, \delta) = \frac{7}{3}. \tag{3.13}$$

For given $i \geq 3$, and $\delta \in (0, x_i - x_{i-2}]$, we have

$$\begin{aligned} F_i(G_2, \delta) &= \frac{2T_{i-1} + x_{i-2} + \delta}{d^*(x_{i-1} + x_{i-2} + \delta)} \\ &= \frac{2^{i+1} - 2 + 2^{i-2} + \delta}{d^*(2^{i-1} + 2^{i-2} + \delta)} \\ &= \frac{9 \cdot 2^{i-2} - 2 + \delta}{d^*(2^{i-1} + 2^{i-2} + \delta)}. \end{aligned}$$

Moreover, from Lemma 3.8 we have that

$$d^*(2^{i-1} + 2^{i-2} + \delta) = (2^{i-1} + 2^{i-2} + \delta) \cdot \frac{6m^*(2^{i-1} + 2^{i-2} + \delta) + 4}{3m^*(2^{i-1} + 2^{i-2} + \delta) + 5}.$$

and

$$m^*(x_{i-1} + x_{i-2} + \delta) \geq m^*(x_{i-1} + x_{i-2}) = m^*(3 \cdot 2^{i-2}).$$

For $i \in \{3, 4\}$ and $\delta \in (0, x_i - x_{i-2}]$, we have

$$m^*(x_{i-1} + x_{i-2} + \delta) \geq m^*(3 \cdot 2^{i-2}) \geq 1,$$

then

$$d^*(2^{i-1} + 2^{i-2} + \delta) \geq (2^{i-1} + 2^{i-2} + \delta) \cdot \frac{6 \cdot 1 + 4}{3 \cdot 1 + 5} = \frac{15 \cdot 2^{i-2} + 5\delta}{4}.$$

Hence, for $i = 3$,

$$F_3(G_2, \delta) = \frac{9 \cdot 2^{i-2} - 2 + \delta}{d^*(2^{i-1} + 2^{i-2} + \delta)} \leq \frac{16 + \delta}{\frac{30+5\delta}{4}} = \frac{64 + 4\delta}{30 + 5\delta}.$$

We obtain that

$$\sup_{\delta \in (0, x_3 - x_1]} F_3(G_2, \delta) \leq \frac{64}{30}. \tag{3.14}$$

For $i = 4$,

$$F_4(G_2, \delta) = \frac{9 \cdot 2^{i-2} - 2 + \delta}{d^*(2^{i-1} + 2^{i-2} + \delta)} \leq \frac{32 + \delta}{\frac{}{60+5\delta}4} = \frac{128 + 4\delta}{60 + 5\delta}.$$

We obtain that

$$\sup_{\delta \in (0, x_4 - x_2]} F_4(G_2, \delta) \leq \frac{128}{60}. \tag{3.15}$$

For $i \geq 5$ and $\delta \in (0, x_i - x_{i-2}]$, we have

$$m^*(x_{i-1} + x_{i-2} + \delta) \geq m^*(3 \cdot 2^{i-2}) \geq 2,$$

then

$$d^*(2^{i-1} + 2^{i-2} + \delta) \geq (2^{i-1} + 2^{i-2} + \delta) \cdot \frac{6 \cdot 2 + 4}{3 \cdot 2 + 5} = \frac{48 \cdot 2^{i-2} + 16\delta}{11},$$

and

$$F_i(G_2, \delta) = \frac{9 \cdot 2^{i-2} - 2 + \delta}{d^*(2^{i-1} + 2^{i-2} + \delta)} \leq \frac{9 \cdot 2^{i-2} - 2 + \delta}{\frac{48 \cdot 2^{i-2} + 16\delta}{11}} = \frac{99 \cdot 2^{i-2} - 22}{48 \cdot 2^{i-2}} \leq \frac{99}{48},$$

hence, for $i \geq 5$,

$$\sup_{\delta \in (0, x_i - x_{i-2}]} F_i(G_2, \delta) \leq \frac{99}{48}. \tag{3.16}$$

Combining (3.10), (3.13), (3.14), (3.15) and (3.16) yields the proof of the theorem. □

3.3.3 On the Uniqueness of Strategies with Optimal Discovery Ratio

A natural question arises: Is `aggressive` the unique strategy of optimal discovery ratio in Σ_9? The following theorem provides evidence that optimal strategies cannot be radically different than `aggressive`, in that they must mimic it in the first few iterations.

Theorem 3.12. *Strategy $X = (x_0, x_1, \ldots) \in \Sigma_9$, has optimal discovery ratio in Σ_9 only if $x_i = \bar{x}_i$, for $0 \leq i \leq 4$.*

Proof. Consider a strategy $X(x_0, x_1, \ldots) \in \Sigma_9$. Recall that the discovery ratio of X is given by Equation (3.7). We will prove the theorem by induction on i.

We first show the base case, namely $i = 0$. The base case holds by the argument used in the proof of Theorem 3.10 which shows that if $x_0 < 4$, then $\mathrm{dr}(X, \Sigma_9) \geq 2$. For the induction step, suppose that, if X has optimal discovery ratio then for $j \in [0, i]$, $x_j = \bar{x}_j$, with $i < 4$. We will show $x_{i+1} = \bar{x}_{i+1}$ by contradiction, hence assume

$x_{i+1} < \bar{x}_{i+1}$. For sufficiently small $\epsilon > 0$, we have

$$\begin{aligned} m^*(x_{i+1} + x_i + \epsilon) &= m^*(x_{i+1} + \bar{x}_i + \epsilon) && \text{(by induction hypothesis)} \\ &\leq m^*(\bar{x}_{i+1} + \bar{x}_i) && \text{(by monotonicity of } m^* \text{ and Lemma 3.6)} \\ &= i + 1, && \text{(by definition of } m^*) \end{aligned}$$

which implies that, by Lemma 3.8,

$$d^*(x_i + x_{i-1} + \epsilon) = (x_i + x_{i-1} + \epsilon) \cdot \frac{6 \cdot m^*(x_{i+1} + x_i + \epsilon) + 4}{3 \cdot m^*(x_{i+1} + x_i + \epsilon) + 5} \leq (x_i + x_{i-1} + \epsilon) \cdot \frac{6 \cdot (i+1) + 4}{3 \cdot (i+1) + 5}. \tag{3.17}$$

Therefore

$$\begin{aligned} F_{i+2}(X, \epsilon) &= \frac{2 \cdot \sum_{j=0}^{i+1} x_j + x_i + \epsilon}{d^*(x_{i+1} + x_i + \epsilon)} \\ &= \frac{2T_i(\texttt{aggressive}) + 2x_{i+1} + \bar{x}_i + \epsilon}{d^*(x_{i+1} + \bar{x}_i + \epsilon)} && \text{(by ind. hyp.)} \\ &\geq \frac{2T_i(\texttt{aggressive}) + 2x_{i+1} + \bar{x}_i + \epsilon}{(x_{i+1} + \bar{x}_i + \epsilon) \cdot \frac{6 \cdot (i+1) + 4}{3 \cdot (i+1) + 5}} && \text{(Inequation (3.17))} \\ &= \frac{(i+1)2^{i+3} + (i+2)2^{i+1} + 2x_{i+1} + \epsilon}{(x_{i+1} + (i+2)2^{i+1} + \epsilon) \cdot \frac{6 \cdot (i+1) + 4}{3 \cdot (i+1) + 5}} && \text{(Corollary 3.5)} \\ &\geq \frac{(i+1)2^{i+3} + (i+2)2^{i+1} + (i+3)2^{i+3} + \epsilon}{(i+3)2^{i+2} + (i+2)2^{i+1} + \epsilon} \cdot \frac{3i+8}{6i+10}. \\ & && \text{(monoton. on } x_{i+1}) \end{aligned}$$

Hence

$$\begin{aligned} \sup_{\delta \in (0, x_{i+2} - x_i]} F_{i+2}(X, \delta) &\geq \frac{(i+1)2^{i+3} + (i+2)2^{i+1} + (i+3)2^{i+3}}{(i+3)2^{i+2} + (i+2)2^{i+1}} \cdot \frac{3i+8}{6i+10} \\ &= \frac{9i+18}{6i+10}, \end{aligned}$$

which is greater than $\frac{8}{5}$ if $i \leq 3$. We conclude, from (3.7) that $\mathrm{dr}(X, \Sigma_9) > 8/5$, which is a contradiction. □

By Theorem 3.12, we observe that it could be possible that there exists some strategy X with optimal discovery ratio in Σ_9 such that x_5 differs from $\bar{x}_5$. What if we only slightly decreases the value of x_5 and remain aggressive for x_i with $i \in [6, +\infty)$, is this strategy in Σ_9 and does it have optimal discovery ratio? The answer is positive. Formally, consider the strategy $X_\epsilon = (x'_0, x'_1, \ldots)$, defined by $x_i = \bar{x}_i$, for $0 \leq i \leq 4$, $x_5 = \bar{x}_5 - \epsilon$ and $x_i = 3x_{i-1} - T_{i-2}(X_\epsilon)$, for $i \geq 6$, where ϵ is some sufficiently small positive parameter.

Lemma 3.13. *The strategy X_ϵ can be described by the sequence $x'_n = (n+2)2^{n+1}$, for $n \in [1, 4]$, and $x'_n = (n+2)2^{n+1} - (n-3)2^{n-6}\epsilon$, for $n \geq 5$. Moreover, $T_n(X_\epsilon) = (n+1)2^{n+2}$, for $n \in [1, 4]$, and $T_n(X_\epsilon) = (n+1)2^{n+2} - (n-4)2^{n-5}\epsilon$, for $n \geq 5$.*

Proof. Since X_ϵ mimics `aggressive` in the first five iterations, then $x'_n = \bar{x}_n = (n+2)2^{n+1}$ and $T_n(X_\epsilon) = T_n(\texttt{aggressive}) = (n+1)2^{n+2}$, for $n \in [1, 4]$.

We will show, by induction on n, that for all $n \geq 5$, $x'_n = (n+2)2^{n+1} - (n-3)2^{n-6}\epsilon$ and $T_n(X_\epsilon) = (n+1)2^{n+2} - (n-4)2^{n-5}\epsilon$. Note that, the base case, namely $n = 5$, follows directly from the definition of X_ϵ. For the induction hypothesis, suppose that for $i \in [5, n]$, it holds that

$$x'_i = (i+2)2^{i+1} - (i-3)2^{i-6}\epsilon$$

and

$$T_n(X_\epsilon) = (n+1)2^{n+2} - (n-4)2^{n-5}\epsilon.$$

We will show that the claim holds also for $n+1$. Since

$$\begin{aligned} x'_{n+1} &= 3x'_n - T_{n-2}(X_\epsilon) && \text{(by definition of } x'_{n+1}) \\ &= 3(n+2)2^{n+1} - 3(n-3)2^{n-6}\epsilon - T_{n-2}(X_\epsilon). && \text{(by induction hypothesis)} \end{aligned}$$

We distinguish two cases. For $n = 5$,

$$\begin{aligned} x'_{n+1} &= 3(n+2)2^{n+1} - 3(n-3)2^{n-6}\epsilon - T_{n-1}(X_\epsilon) \\ &= 3(n+2)2^{n+1} - 3(n-3)2^{n-6}\epsilon - T_{n-1}(\texttt{aggressive}) && \text{(by definition of } X_\epsilon) \\ &= 3(n+2)2^{n+1} - 3(n-3)2^{n-6}\epsilon - n2^{n+1} && \text{(Subst. of } T_{n-1}(\texttt{aggressive})) \\ &= (n+3)2^{n+2} - 3(n-3)2^{n-6}\epsilon && \text{(arranging terms)} \\ &= (5+3)2^{5+2} - 3\epsilon. && \text{(Subst. of } n = 5 \text{ in } 3(n-3)2^{n-6}) \end{aligned}$$

Then

$$\begin{aligned} T_6(X_\epsilon) &= T_5(X_\epsilon) + x'_6 && \text{(by definition of } T_6(X_\epsilon)) \\ &= (5+1)2^{5+2} - (5-4)2^{5-5}\epsilon + (5+3)2^{5+2} - 3\epsilon && \text{(by ind. hyp.)} \\ &= 7 \cdot 2^8 - 4\epsilon. && \text{(arranging terms)} \end{aligned}$$

For $n \geq 6$,

$$\begin{aligned} x'_{n+1} &= 3(n+2)2^{n+1} - 3(n-3)2^{n-6}\epsilon - T_{n-1}(X_\epsilon) \\ &= 3(n+2)2^{n+1} - 3(n-3)2^{n-6}\epsilon - (n2^{n+1} - (n-5)2^{n-6}\epsilon) \\ & && \text{(Subst. of } T_{n-1}(X_\epsilon)) \\ &= (n+3)2^{n+2} - (n-2)2^{n-5}\epsilon. && \text{(arranging terms)} \end{aligned}$$

Then

$$\begin{aligned} T_{n+1}(X_\epsilon) &= T_n(X_\epsilon) + x'_{n+1} && \text{(by definition of } T_{n+1}(X_\epsilon)) \\ &= (n+1)2^{n+2} - (n-4)2^{n-5}\epsilon + (n+3)2^{n+2} - (n-2)2^{n-5}\epsilon \\ & && \text{(by ind. hyp.)} \\ &= (n+2)2^{n+3} - (n-3)2^{n-4}\epsilon, && \text{(arranging terms)} \end{aligned}$$

which completes the proof of the lemma. □

Corollary 3.14. *The strategy $X_\epsilon = (x'_0, x'_1, \ldots)$ with $\epsilon = 1$, is in Σ_9. Moreover, we have $x'_n = (n+2)2^{n+1}$, for $n \in [1, 4]$, and $x'_n = (127n + 259)2^{n-6}$, for $n \geq 5$. In addition, $T_n(X_1) = (n+1)2^{n+2}$, for $n \in [1, 4]$, and $T_n(X_1) = (127n + 132)2^{n-5}$, for $n \geq 5$.*

Proof. We choose the parameter $\epsilon = 1$, then by Lemma 3.13, we have $x'_n = (n+2)2^{n+1}$, for $n \in [1, 4]$ and $x'_n = (n+2)2^{n+1} - (n-3)2^{n-6} = (127n + 259)2^{n-6} \geq 0$,

for $n \geq 5$. Then the strategy is well defined. Moreover, $T_n(X_1) = (n+1)2^{n+2} - (n-4)2^{n-5} = (127n+132)2^{n-5}$, for $n \geq 5$. □

The following theorem provides evidence that the strategy of optimal discovery ratio in Σ_9 is not unique.

Theorem 3.15. *We have* $dr(X_1, \Sigma_9) = \frac{8}{5}$.

Proof. We will express the discovery ratio using (3.7). Since X_1 mimics `aggressive` in the first five iterations, then for $n \in [1,4]$, we have

$$\sup_{n\in[1,4]} \sup_{\delta\in(0,x'_n-x'_{n-2}]} F_n(X_1,\delta) = \frac{8}{5}.$$

For $n = 5$ and $\delta \in (0, x'_5 - x'_3]$, we have $x'_4 + x'_3 + \delta \in (x'_4 + x'_3, x'_4 + x'_5]$, which implies that $m^*(x'_4 + x'_3 + \delta) = 5$. Then

$$\begin{aligned}
F_5(X_1,\delta) &= \frac{2T_4(X_1) + x'_3 + \delta}{d^*(x'_4 + x'_3 + \delta)} && \text{(by Equation (3.7))}\\
&= \frac{2T_4(X_1) + x'_3 + \delta}{(x'_4 + x'_3 + \delta)\frac{6m^*(x'_4+x'_3+\delta)+4}{3m^*(x'_4+x'_3+\delta)+5}} && \text{(by Lemma 3.8)}\\
&= \frac{2T_4(X_1) + x'_3 + \delta}{(x'_4 + x'_3 + \delta)} \cdot \frac{20}{34} && \text{(Subst. of } m^*(x'_4 + x'_3 + \delta))\\
&= \frac{2\cdot(4+1)2^{4+2} + (3+2)2^{3+1} + \delta}{(4+2)2^{4+1} + (3+2)2^{3+1} + \delta} \cdot \frac{20}{34} && \text{(by Lemma 3.13)}\\
&= \frac{720+\delta}{272+\delta} \cdot \frac{20}{34} && \text{(arranging terms)}\\
&\leq \frac{720\cdot 20}{272\cdot 34} && \text{(monotonicity on } \delta)\\
&= \frac{450}{289} < \frac{8}{5}.
\end{aligned}$$

For $n = 6$ and $\delta \in (0, x'_6 - x'_4]$, we have $x'_5 + x'_4 + \delta \in (x'_5 + x'_4, x'_6 + x'_5]$, which implies that $m^*(x'_5 + x'_4 + \delta) \in [5,6]$. Then

$$\begin{aligned}
F_6(X_1,\delta) &= \frac{2T_5(X_1) + x'_4 + \delta}{d^*(x'_5 + x'_4 + \delta)} && \text{(by Equation (3.7))}\\
&= \frac{2T_5(X_1) + x'_4 + \delta}{(x'_5 + x'_4 + \delta)\frac{6m^*(x'_5+x'_4+\delta)+4}{3m^*(x'_5+x'_4+\delta)+5}} && \text{(by Lemma 3.8)}\\
&\leq \frac{2T_5(X_1) + x'_4 + \delta}{(x'_5 + x'_4 + \delta)} \cdot \frac{3\cdot 5+5}{6\cdot 5+4} && \text{(Subst. of } m^*(x'_{n-1} + x'_{n-2} + \delta))\\
&= \frac{2\cdot(5+1)2^{5+2} - 2 + (4+2)2^{4+1} + \delta}{(5+2)2^{5+1} - 1 + (4+2)2^{4+1} + \delta} \cdot \frac{20}{34} && \text{(by Lemma 3.13)}\\
&= \frac{1726+\delta}{639+\delta} \cdot \frac{20}{34} && \text{(arranging terms)}\\
&\leq \frac{1726\cdot 20}{639\cdot 34} && \text{(monotonicity on } \delta)\\
&= \frac{17260}{10863} < \frac{8}{5}.
\end{aligned}$$

For $n \geq 7$ and $\delta \in (0, x'_n - x'_{n-2}]$, we have

$$\begin{aligned} x'_{n-1} + x'_{n-2} + \delta &> x'_{n-1} + x'_{n-2} && \text{(by } \delta > 0\text{)} \\ &= (n+1)2^n - (n-4)2^{n-7} + n2^{n-1} - (n-5)2^{n-8} \\ & && \text{(by Lemma 3.13)} \\ &= (381n + 269)2^{n-8}. && \text{(arranging terms)} \end{aligned}$$

Since $(3(n-2)+5)2^{n-2} < (381n+269)2^{n-8} < (3(n-1)+5)2^{n-1}$, it implies that $m^*(x'_{n-1} + x'_{n-2} + \delta) \geq n-1$. Then

$$\begin{aligned} F_n(X_1, \delta) &= \frac{2T_{n-1}(X_1) + x'_{n-2} + \delta}{d^*(x'_{n-1} + x'_{n-2} + \delta)} && \text{(by Equation (3.7))} \\ &= \frac{2T_{n-1}(X_1) + x'_{n-2} + \delta}{(x'_{n-1} + x'_{n-2} + \delta)\frac{6m^*(x'_{n-1}+x'_{n-2}+\delta)+4}{3m^*(x'_{n-1}+x'_{n-2}+\delta)+5}} && \text{(by Lemma 3.8)} \\ &\leq \frac{2T_{n-1}(X_1) + x'_{n-2} + \delta}{(x'_{n-1} + x'_{n-2} + \delta)} \cdot \frac{3(n-1)+5}{6(n-1)+4} && \text{(Subst. of } m^*(x'_{n-1} + x'_{n-2} + \delta)\text{)} \\ &= \frac{2n2^{n+1} - (n-5)2^{n-5} + n2^{n-1} - (n-5)2^{n-8} + \delta}{(n+1)2^n - (n-4)2^{n-7} + n2^{n-1} - (n-5)2^{n-8} + \delta} \cdot \frac{3(n-1)+5}{6(n-1)+4} \\ & && \text{(by Lemma 3.13)} \\ &= \frac{9(127n+5)2^{n-8} + \delta}{(381n+269)2^{n-8} + \delta} \cdot \frac{3(n-1)+5}{6(n-1)+4} && \text{(arranging terms)} \\ &\leq \frac{9(127n+5)2^{n-8}}{(381n+269)2^{n-8}} \cdot \frac{3(n-1)+5}{6(n-1)+4} && \text{(monotonicity on } \delta\text{)} \\ &< \frac{8}{5}. && \text{(monotonicity on } n\text{)} \end{aligned}$$

□

3.4 Computational Evaluation

In this section we present computational results on the implementation of strategies `aggressive`, `doubling` and the function $d^*(L)$. Recall that $d^*(L)$ is the minimum cost at which a competitively optimal strategy can discover a length equal to L. In particular, we compare the discovery cost and the discovery ratio of `aggressive` and `doubling` for some given L. We choose L to be integral in the range $[1, 10^7]$.

Figure 3.2 illustrates the discovery cost of two strategies `aggressive`, `doubling` and also the optimal cost d^* for $L \in [1, 10^7]$. We observe that the optimal cost d^* is almost linear, unlike $D(\texttt{aggressive}, L)$ or $D(\texttt{doubling}, L)$ which are piecewise linear functions with slopes increasing in L. In addition, we observe that there is no clear dominance between $D(\texttt{aggressive}, L)$ and $D(\texttt{doubling}, L)$ for some fixed L.

Figure 3.3 illustrates the ratio $D(\texttt{aggressive}, L)/d^*(L)$ and $D(\texttt{doubling}, L)/d^*(L)$ for $L \in [1, 10^7]$. We observe that the fluctuations of the ratio, as a function of L, tend to decrease in L. In addition, the worst case ratio occurs at the very first steps for both `aggressive` and `doubling`. Strategy `doubling` has a worse discovery ratio because of the worse performance at the very first steps.

The above results are based on the worst-case analysis. However, for a given L, we cannot really distinguish the performance of `aggressive` and `doubling`. This

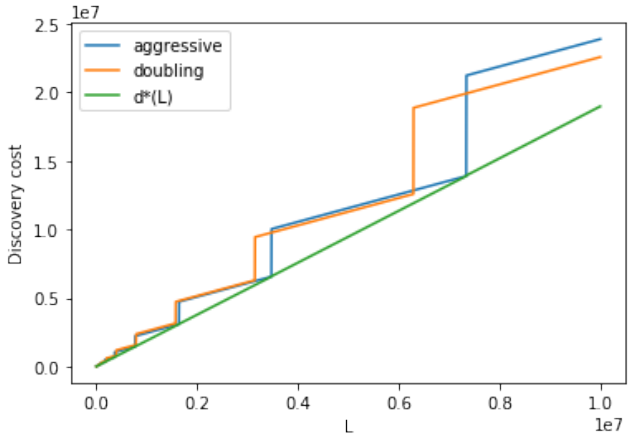


FIGURE 3.2 – $D(\texttt{aggressive}, L)$, $D(\texttt{doubling}, L)$ and $d^*(L)$ as a function of L.

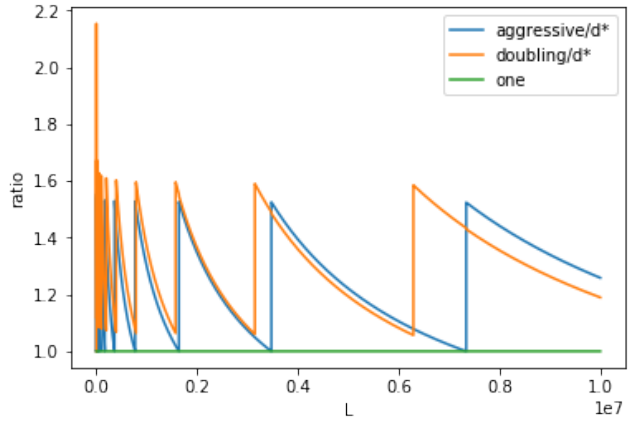


FIGURE 3.3 – Ratios $D(\texttt{aggressive}, L)/d^*(L)$ and $D(\texttt{doubling}, L)/d^*(L)$ as a function of L.

motivates a further study on average discovery cost to distinguish the performance of these two algorithms. Let $avg(X, L) := \int_1^L D(X, l) \cdot dl/(L-1)$ denote the average discovery cost of a strategy X on the segment $[1, L]$.

Figure 3.4 illustrates the average discovery cost $avg(\texttt{aggressive}, L)$ and $avg(\texttt{doubling}, L)$ for $L \in [1, 10^7]$. We observe that $avg(\texttt{aggressive}, L)$ dominates $avg(\texttt{doubling}, L)$ for $L \in [1, 10^7]$. This may imply a strong separation on the performance of these two algorithms on average-case analysis.

3.5 Conclusion

In this chapter, we revisited the linear search problem and provided a separation on the performance of `doubling` and `aggressive` by applying the discovery ratio as supplementary to the competitive ratio. More specifically, both `doubling` and `aggressive` have an optimal competitive ratio 9. However, we showed that the strategy `aggressive` has an optimal discovery ratio $\frac{8}{5}$ against the set of 9-competitive strategies. In contrast, we showed that the strategy `doubling` has discovery ratio $\frac{7}{3}$ against the set of 9-competitive strategies. In addition, we proved that any competitively optimal strategy that is also optimal with respect to the discovery ratio must

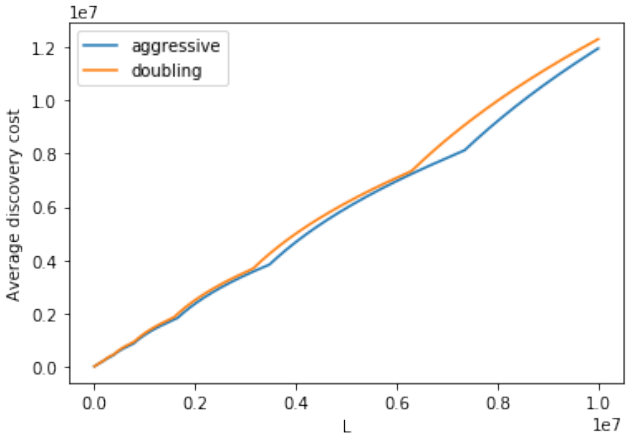


FIGURE 3.4 – Average discovery cost of `aggressive` and `doubling` as a function of L.

have the exact same behavior as the aggressive strategy in the first five iterations. Moreover, we showed that such competitively optimal strategies with optimal discovery ratio is not unique.

In terms of techniques, the main technical result in establishing the discovery ratios consisted of answering the following question: given a length $l \in \mathbb{R}^+$, what is the strategy S that minimizes, $D(S, l)$, the cost for discovering a total length equal to l in S, and how can one express this minimum discovery cost? This is a type of *inverse* problem that can be of independent interest in the context of search problems, which is also known as the *reach* of a strategy [38], or *extent* in [41]. This inverse problem is very useful in the competitive analysis of search strategies. We modeled this problem as a linear program for whose objective value we first gave a lower bound; then we showed this bound is tight by providing an explicit 9-competitive strategy which minimizes $D(S, l)$.

Furthermore, we discuss a variant of the linear search problem related to the stochastic dominance. Stochastic dominance defines a partial order on random variables. A random variable X is stochastically dominated by a random variable Y if, for all $c \in \mathbb{R}$, we have $Pr[X \leq c] \geq Pr[Y \leq c]$. Suppose that instead of the infinite line, the search environment consists of a bounded line of length L, which may or may not be known to the searcher, and that the origin is located in the middle of the line. Besides, we assume that there is a uniform distribution over the target position. We believe that our approach is still applicable to obtain a stronger conclusion on stochastic dominance using the discovery cost. The main technical obstacle is that we have to consider the edge case very carefully, where the target is located close to the endpoint of the line.

Chapter 4

Contract Scheduling with End Guarantees

This chapter contains material from the joint paper, *Earliest Completion Scheduling of Contract Algorithms with End Guarantees*, with Spyros Angelopoulos. This work appeared in the *28th International Joint Conference on Artificial Intelligence* (IJCAI), 2019 [6].

As discussed in Section 1.1.2, for some problems, it is convenient to formulate them using linear programming. Sometimes, we can look into the structure of the optimal solutions to linear programming to obtain an optimal solution. We have shown in Chapter 3 how this idea can be applied to the linear search problem. In this Chapter 4, we show another application from Artificial Intelligence in the design of the interruptible system using contract algorithms, optimizing a certain efficiency criterion.

4.1 Introduction

As discussed in Section 1.3.3, algorithms with anytime capabilities are important in the design of intelligent systems. Recall that such an algorithm can return a valid solution even if it is interrupted during its execution. Usually, the algorithm is expected to find better solutions the longer it keeps running. For instance, consider the *local search* algorithm, which consists of moving from solution to solution in the space of candidate solutions by applying local changes, until an optimal solution is found or a time-bound is elapsed. However, local search can not guarantee the optimality of the returned solution.

One natural goal is then to build an interruptible system using algorithms that may not be interruptible. This motivated research in the design of interruptible systems with *contract algorithms* [58]. Specifically, a contract algorithm has an intended queried time as one of its input parameters. If a contract algorithm is queried before this time, then it may output a meaningless result. We can see that the contract algorithm itself is not interruptible.

A general technique for obtaining interruptible systems by contract algorithms was first given in [58]. The idea is to run multiple times the contract algorithm by iteratively increasing the available execution time of the contract algorithm. Each piece of such execution is called a *contract*. More precisely, a natural approach is to iteratively double the available execution time of the contract algorithm. In other words, there is a *schedule* of executions of the same algorithm in which the i-th execution is run for a time equal to 2^i (see Figure 4.1 for illustration). More specifically, if an interruption occurs at time t, then it is still guaranteed that a contract of length

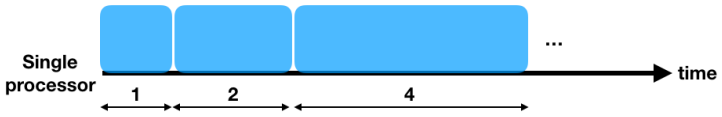


FIGURE 4.1 – A schedule of executions of the same algorithm in which the i-th execution time is 2^i.

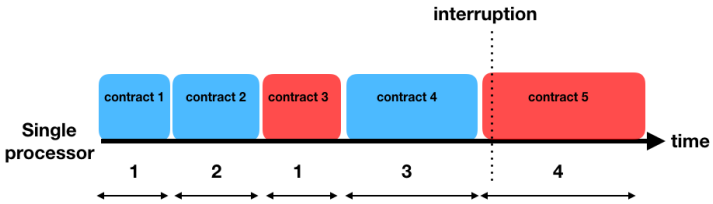


FIGURE 4.2 – A schedule of interleaved executions of 2 contract algorithms represented by a red (respectively, blue) blocks. In this example, the largest contract completed before interruption by red (respectively, blue) algorithm is of length 1 (respectively, 3).

at least $t/4$ has completed its execution. The performance of such interruptible algorithms is often measured using *acceleration ratio*, which was first introduced in [58]. The acceleration ratio is defined as the multiplicative gap between the interruption t and the largest contract length completed by time t in the worst case. For instance, the above described algorithm has an acceleration ratio equal to 4, which is optimal under this measure [58].

The above setting can be generalized to an optimization problem with n different instances, where each instance is associated with a contract algorithm [66]. The objective is to design a schedule of interleaved executions of these n contract algorithms, and the corresponding acceleration ratio is defined as the worst-case ratio between the interruption time t and the contract length of the problem instance that has made the least progress by time t (see Section 4.1.2 for a formal definition), as illustrated in Figure 4.2.

Contract scheduling has been studied under different variants. For instance, [66] gave an optimal schedule for multiple problem instances in a single processor, and [14] showed optimal schedules for a single problem instance in several parallel processors, which was later generalized to the multi-processor and multi-instance model in [13, 47]. Moreover, contract scheduling can also be considered as an application of resource allocation under uncertainty. For instance, minimizing the acceleration ratio for a single problem instance is equivalent to the *online bidding problem* (presented in Section 1.4.2), which is used in [22] as a framework of efficient algorithms based on iterative doubling for some combinatorial optimization problems.

As described earlier, the interruption may occur arbitrarily far in the future. All previous work on contract scheduling assumed that the schedule is unbounded. However, in practice, it is often the case where the execution of a schedule of contract algorithms may reach some point, beyond which any progress will be only marginal. In other words, the execution can be considered completed beyond this point. For instance, consider the class of Polynomial Time Approximation Schemes (PTAS). Many of them are based on Dynamic Programming and thus are not interruptible [47]. These algorithms take as input a parameter $\epsilon > 0$ and output a solution within a factor of $(1+\epsilon)$ of the optimal. Thus, for a given ϵ, we are able to

compute the required time (i.e., contract length) that is required to achieve the desired approximation. This implies that we can know exactly what is the minimum execution time to achieve a given target approximation.

Motivated by the above observation, our objective is to obtain a *finite* schedule with the following properties:

1. It should attain the optimal acceleration ratio;
2. It should minimize the time required to satisfy the *end guarantees*, among all schedules that satisfy the first property.

The *end guarantee* is defined as the minimum required value be attained upon completion on the length of the largest contract completed. In other words, for a given end guarantee L, each contract algorithm has to complete a contract of length at least L upon completion.

4.1.1 Contribution

The main contribution of this chapter is an optimal schedule for the problem described above, namely for the earliest completion scheduling of contract algorithms with end guarantees. We propose a schedule that is theoretically optimal and can be computed in polynomial time in the size of the end guarantee L, for $L > 0$. Assuming that the number of problem instances n is constant and independent of L, we will show that the time complexity is then polynomial in the size of the input. In addition, we present some computational results on its implementation, which demonstrate that it achieves a considerable improvement over the known schedule that optimizes the acceleration ratio, but is oblivious of L.

This chapter is structured as follows: We begin in Section 4.2 by showing the existence of an optimal *cyclic* schedule. Note that a strategy is called cyclic if contracts are assigned to problems in round-robin fashion. This allows us to formulate the problem by a linear program (LP). In Section 4.3 we show that an optimal cyclic strategy saturates the constraints of the LP. This allows us to define an appropriate recurrence relation over the contract lengths, which yields the optimal schedule. In Section 4.4, we provide a computational evaluation of the obtained schedule.

4.1.2 Preliminaries

We assume a single processor and n *problem instances* or simply *problems*, numbered from 0 to $n-1$. A schedule X of m contracts can be described as a sequence of the form $((x_i, p_i))_{i\in[1,m]}$, meaning that the i-th scheduled contract in X has length $x_i > 0$ and is assigned to problem $p_i \in \{0, \dots, n-1\}$. We also define $N_i(X)$ as the index of the next contract that is assigned to the problem p_i after the i-th contract (x_i, p_i) in the schedule X. In other words, $N_i(X) = \min\{j > i | p_j = p_i\}$. If there is no $j > i$ with $p_j = p_i$, then we define $N_i = i$. For interruption time t, let $\ell_{p,t}(X)$ denote the length of the longest execution of a contract algorithm for problem p that has completed by time t in X. Then the *acceleration ratio of* X [58] is defined as

$$\rho(X) = \sup_{t,p\in[0,n-1]} \frac{t}{\ell_{p,t}(X)}. \tag{4.1}$$

We denote by $T_j(X)$ the completion time of the j-th contract in X and by $T(X)$ the completion time of X. In other words, $T(X)$ is also the completion time of its last contract. It is not hard to see that the worst-case interruptions occur infinitesimally prior to the completion of a contract, hence we obtain the following useful formula

for a schedule of m contracts:

$$\rho(X) = \sup_{j\in[n+1,m],p\in[0,n-1]} \frac{T_j(X)}{\ell_{p,T_j^-(X)}(X)}, \tag{4.2}$$

where $T_j^-(X)$ denotes a time right before $T_j(X)$ but arbitrarily close. We also make the standard assumption that no interruption occurs unless each problem instance has completed a contract in the schedule. Without loss of generality, we can assume that X will never schedule a contract of length l for problem p if it has already finished a contract for p of length at least l by that time.

A schedule is called *cyclic* if its i-th contract is assigned to the problem i mod n, and *monotone* if $x_{i+1} \geq x_i$, for all i. The optimal acceleration ratio, denoted by ρ_n^*, can be attained by a cyclic (and monotone) strategy such that $x_i = b^i$, with $b = \frac{n+1}{n}$ [66], from which it follows that $\rho_n^* = n\left(\frac{n+1}{n}\right)^{n+1}$. We will denote by S_n^* the set of all schedules of optimal acceleration ratio ρ_n^*.

Given a schedule X, and an end guarantee $L \in \mathbb{R}^+$, we say that X is *feasible for* L if for each problem p there is a contract for p in X that has length at least L. A schedule X that is feasible for L is called *earliest for* L if for any other schedule X' feasible for L, $T(X) \leq T(X')$. Using this notation, we can state our problem as follows: Find schedule X^* (feasible for L) such that $X^* \in S_n^*$, and for every $X' \in S_n^*$, $T(X) \leq T(X')$. We call such a schedule *optimal*.

Example 4.1. Consider the simple case $n = 1$, and $L = 30$. An example of a schedule feasible for L is a schedule X with contract lengths $1, 2, 4, 8, 16, 30$, which has completion time $T(X) = 61$. Note also that X is in S_1^*, since it has an optimal acceleration ratio equal to 4.

4.2 Cyclic Schedules and the LP Formulation

In this section we show that for a given end guarantee L, there is a cyclic schedule that is optimal for L. This will allow us to focus exclusively on this class of schedules, which we will later analyze by our technique on LP. To this end, we first define a property that will be useful in the proof.

Definition 4.2. A schedule $X = ((x_i, p_i))_{i\in[1,m]}$ is called *normalized* if for each $i \in [1, m]$, $\ell_{p_i,T_{i-1}(X)}(X) \leq \ell_{q,T_{i-1}(X)}(X)$, for all $q \neq p_i$.

Informally, a normalized schedule X assigns, at each time, a contract to the problem that has been worked the least among all problems up to that time. The following lemma shows that for every L there exists an optimal normalized schedule. Its proof expands the property that is already known, namely that there exists a normalized schedule that has optimal acceleration ratio [47].

Lemma 4.3. *For every schedule X feasible for L, there exists a normalized schedule X' feasible for L such that $\rho(X') \leq \rho(X)$ and $T(X') \leq T(X)$.*

Proof. Suppose that X is not normalized for the i-th contract. Then, there exists a problem $q \neq p_i$ such that $\ell_{q,T_{i-1}(X)}(X) < \ell_{p_i,T_{i-1}(X)}(X)$. Consider X' that is derived from X as follows: X' is identical to X for the first $i-1$ contracts and its i contract is (x_i, q). Furthermore, for all $j > i$, the j-th contract in X is (x_j, r), where $r = p_j$, if $p_j \notin \{p_i, q\}$, $r = q$, if $p_j = p_i$, and $r = p_i$, if $p_j = q$. In words, X' swaps the problem assignments of contracts for p_i and q for all such contracts scheduled after

time $T_{i-1}(X)$ in X. Note that $T(X') = T(X)$. We first argue that if X is feasible for L, then so is X'. We consider two cases: if a contract for q is scheduled after time $T_i(X)$ in X, then both p_i, q have completed a contract of length at least L in both X and X', hence X' is feasible for L. Otherwise, since X is feasible for L, X' clearly completes a contract of length at least L for p_i, and remains to argue the same about q. This is indeed the case, since for X to be feasible, it must be that there is a contract of length at least L for q that was scheduled prior to time $T_{i-1}(X)$ in X, hence one such contract for q is in X' as well.

It remains to show that $\rho(X') \leq \rho(X)$. Again we consider two cases, depending on whether a contract for q is scheduled after time $T_i(X)$ in X. Suppose first that this is not the case, then we observe that an interruption $t \geq T_i$ that occurs right before a contract for p_i in X corresponds to an interruption t that occurs right before a contract for q in X'. Since $\max\{\ell_{p_i,t}(X), \ell_{q,t}(X)\} \leq \max\{\ell_{p_i,t}(X'), \ell_{q,t}(X')\}$, it follows that in this case $\rho(X') \leq \rho(X)$. It remains to consider the case that there is a contract for q that is scheduled after time $T_i(X)$ in X; let T'_q denote the end time of this contract. From the previous argument, interruptions that occur prior to T'_q contribute to the acceleration ratio of X at least as much as the corresponding interruptions in X'. Moreover, for all interruptions t with $t \geq T'_q$ have the same contribution in X as in X', since $\max_{i\in[0,\ldots,n-1]} \ell_{i,t}(X) = \max_{i\in[0,\ldots,n-1]} \ell_{i,t}(X')$. Thus, it follows that in this case as well $\rho(X') \leq \rho(X)$ and the proof is completed. □

Corollary 4.4. *The acceleration ratio of a normalized schedule $X = ((x_i, p_i))_{i\in[1,m]}$ of m contracts can be computed as follows:*

$$\rho(X) = \sup_{j\in[n+1,m]} F_j(X),$$

where $F_j(X) = \frac{T_j(X)}{x_{N_j^{-1}(X)}}$ and $N_j^{-1}(X)$ is the index of the last scheduled contract for the problem p_j before j in the schedule X.

Proof. By definition of a normalized schedule, for $j \in [n+1, m]$, we have

$$\inf_{p\in[0,n-1]} \ell_{p,T_j^-(X)}(X) = \ell_{p_j,T_j^-(X)}(X) = x_{N_j^-(X)}.$$

Combining with (4.2), we get

$$\begin{aligned}\rho(X) &= \sup_{j\in[n+1,m]} \sup_{p\in[0,n-1]} \frac{T_j(X)}{\ell_{p,T_j^-(X)}(X)} \\ &= \sup_{j\in[n+1,m]} \frac{T_j(X)}{\ell_{p_j,T_j^-(X)}(X)} \\ &= \sup_{j\in[n+1,m]} \frac{T_j(X)}{x_{N_j^{-1}(X)}},\end{aligned}$$

which concludes the proof. □

The following property follows easily from the definition of a normalized schedule.

Lemma 4.5. *Any optimal normalized schedule X is such that for every problem p, X schedules at most one contract for p of length at least L, where L is the end guarantee.*

Proof. By way of contradiction, suppose that there is a problem p for which X schedules two contracts of length at least L. Let C_1, C_2 denote these contracts, in the order they appear in the schedule. Since X is normalized, at the moment C_2 is about to start, every other problem in X has completed a contract of length at least the length of C_1, and thus at least L. Thus, at that moment, X has met the end guarantee. One could then obtain a schedule X' that is identical to X up to, but not including C_2. X' is also feasible for L; moreover, $\rho(X') \leq \rho(X)$. However, $T(X') < T(X)$, a contradiction. □

Lemma 4.6. *For every normalized schedule X optimal for L, there exists a normalized and monotone schedule X^* optimal for L such that $\rho(X^*) \leq \rho(X)$ and $T(X^*) \leq T(X)$.*

Proof. The proof is very similar to the proof by López-Ortiz *et al.* [48] for searching on bounded rays. Let $X = (x_i, p_i)$ be a normalized schedule optimal for L. If X is monotone, then there is nothing to show. Otherwise, we assume that there is a contract k, $1 \leq k \leq m$ such that $x_{k+1} < x_k$. Let X^* be a schedule which is equal to X except that, for all contracts $i \geq k$ (i.e. scheduled after contract k), the role of p_k and p_{k+1} is exchanged as are x_k and x_{k+1}. More precisely, this can be achieved by setting, $(x^*_k, N_k(X^*)) = (x_{k+1}, N_{k+1}(X))$; $(x^*_{k+1}, N_{k+1}(X^*)) = (x_k, N_k(X))$; For all $i \in [1, m]$, $p^*_i = p_i$. For $i \notin \{k, k+1\}$, $(x^*_i, N_i(X^*)) = (x_i, N_i(X))$, unless $x^*_{k+1} \geq L$ ($x^*_k \geq L$, respectively), in which case we set $N_{k+1}(X^*) = k+1$ ($N_k(X^*) = k$, respectively).

By definition of X^*, we have $T(X^*) = T(X)$. Moreover, X is feasible for L, which implies that X^* is also feasible for L by construction. It remains to show that $\rho(X^*) \leq \rho(X)$. By Corollary 4.4, it suffices to show that

$$\sup_{j \in [n+1,m]} F_j(X^*) \leq \sup_{j \in [n+1,m]} F_j(X),$$

with $F_j(X) = \frac{T_j(X)}{x_{N_j^{-1}(X)}}$.

By observing that $F_j(X)$ and $F_j(X^*)$ differ only for the indices k, $k+1$, N_k and N_{k+1}. We are going to show that

$$F_k(X^*) < F_k(X);$$

$$F_{k+1}(X^*) = F_{k+1}(X);$$

$$F_{N_k(X^*)}(X^*) \leq F_{N_{k+1}(X)}(X);$$

$$F_{N_{k+1}(X^*)}(X^*) = F_{N_k(X)}(X),$$

which conclude the proof. To this end, for $i \in \{k, k+1\}$, we distinguish two cases, namely for $N_i(X) \neq i$ and $N_i(X) = i$. We have

$$\begin{aligned}
F_k(X^*) &= \frac{T_k(X^*)}{x^*_{N_k^{-1}(X^*)}} && \text{(By definition of } F_k(X^*)) \\
&= \frac{T_k(X)}{x_{N_k^{-1}(X)}} && \text{(By definition of } X^*) \\
&< \frac{T_{k-1}(X) + x_k}{x_{N_k^{-1}(X)}} && \text{(From } x_{k+1} < x_k) \\
&= F_k(X); && \text{(By definition of } F_k(X))
\end{aligned}$$

$$\begin{aligned} F_{k+1}(X^*) &= \frac{T_{k+1}(X^*)}{x^*_{N_{k+1}^{-1}(X^*)}} && \text{(By definition of } F_{k+1}(X^*)) \\ &= \frac{T_{k-1}(X) + x_{k+1}}{x_{N_k^{-1}(X)}} && \text{(By definition of } X^*) \\ &= F_k(X); && \text{(By definition of } F_k(X)) \end{aligned}$$

For the case $N_k \neq k$, we have

$$\begin{aligned} F_{N_k(X^*)}(X^*) &= \frac{T_{N_k(X^*)}(X^*)}{x^*_k} && \text{(By definition of } F_{N_k(X^*)}(X^*)) \\ &= \frac{T_{N_{k+1}(X)}(X)}{x_{k+1}} && \text{(By definition of } X^*) \\ &= F_{N_{k+1}(X)}(X); && \text{(By definition of } F_{N_{k+1}(X)}(X)) \end{aligned}$$

For the case $N_k = k$, we have

$$\begin{aligned} F_{N_k(X^*)}(X^*) &= \frac{T_k(X^*)}{N_k^{-1}(X^*)} && \text{(By definition of } F_{N_k(X^*)}(X^*)) \\ &= \frac{T_k(X^*)}{N_k^{-1}(X)} && \text{(By definition of } X^*) \\ &< \frac{T_k(X)}{N_k^{-1}(X)} && \text{(From } x_{k+1} < x_k) \\ &= F_{N_k(X)}(X); && \text{(By definition of } F_{N_k(X)}(X)) \end{aligned}$$

For the case $N_{k+1} \neq k+1$, we have

$$\begin{aligned} F_{N_{k+1}(X^*)}(X^*) &= \frac{T_{N_{k+1}(X^*)}(X^*)}{x^*_{k+1}} && \text{(By definition of } F_{N_{k+1}(X^*)}(X^*)) \\ &= \frac{T_{N_k(X)}(X)}{x_k} && \text{(By definition of } X^*) \\ &= F_{N_k(X)}(X). && \text{(By definition of } F_{N_k(X)}(X)) \end{aligned}$$

For the case $N_{k+1}(X) = k+1$, we have

$$\begin{aligned} F_{N_{k+1}(X^*)}(X^*) &= F_{k+1}(X^*) \\ &= \frac{T_{k+1}(X^*)}{x^*_{N_{k+1}^{-1}(X^*)}} && \text{(By definition of } F_{k+1}(X^*)) \\ &= \frac{T_{k+1}(X)}{x_{N_{k+1}^{-1}(X)}} && \text{(By definition of } X^*) \\ &= F_{N_k(X)}(X); && \text{(By definition of } F_{N_k(X)}(X)) \end{aligned}$$

We conclude the proof by an exchanging argument as what we do for a bubble sort.

□

The next theorem is central in that it allows us to obtain an LP formulation from

the problem. The theorem implies that optimal schedules can be found in the space of cyclic and monotone schedules.

Theorem 4.7. *Given the end guarantee L, there is a cyclic and monotone schedule that is optimal.*

Proof. Let $X = ((x_i, p_i))_{i\in[1,m]}$ denote an optimal schedule for the given L. From Lemma 4.3, 4.5 and 4.6, we can assume that X is monotone, normalized, and that for every problem p, it schedules at most one contract of length at least L. Again, we follow the proof by López-Ortiz *et al.* [48]. If X is cyclic, then there is nothing to show. Otherwise, let $X^* = (x_i^*, p_i^*)_{i\in[1,m]}$ be a schedule which is equal to X except that it is considered cyclic. More precisely, for all $i \in [1, m]$, $x_i^* = x_i$, and for $i \in [n+1, m]$, $p_i^* = p_{i-n}^*$. By observing that $T(X^*) = T(X)$ and that X^* is feasible for L, it remains to show that $\rho(X^*) \leq \rho(X)$. It suffices to show that, for every $k \in [n+1, m]$, there is a $j \in [n+1, m]$ with $F_k(X^*) \leq F_j(X)$, by Corollary 4.4.

For each problem $p \in [0, n-1]$, let j_p be the last contract scheduled on problem p before contract k. Thus, there exists one problem p such that $j_p \leq k - n$. By monotonicity of X, we have $x_{j_p} \leq x_{k-n}$. Let $k_p \geq k$ be the index of the first contract scheduled on problem p after contract k. Note that k_p must exist, otherwise we have $x_{j_p} \geq L$ and for all $i \geq j_p$, $x_i \geq L$ by monotonicity of X. In particular, there are at least $n+1$ contracts of length at least L, namely for the indices $i \in [j_r, k]$. By Lemma 4.5, there are at most n contracts of length at least L if X is a normalized schedule and optimal for L. Thus, we obtain a contradiction. Hence,

$$\begin{aligned}
F_k(X^*) &= \frac{T_k(X^*)}{x^*_{N_k^{-1}(X^*)}} && \text{(By definition of } F_k(X^*)) \\
&= \frac{T_k(X)}{x_{k-n}} && \text{(By definition of } X^*) \\
&\leq \frac{T_k(X)}{x_{j_p}} && \text{(From } x_{j_p} \leq x_{k-n}) \\
&\leq \frac{T_{k_p}(X)}{x_{j_p}} && \text{(From } k \leq k_p \text{ and monotonicity of } X) \\
&= F_{k_p}(X), && \text{(By definition of } F_{k_p}(X^*))
\end{aligned}$$

which completes the proof. □

Theorem 4.7 allows us to formulate our problem using an LP. More precisely, it suffices to show that there exists a cyclic schedule X^* of m^* contracts of the form $X^* = (x_1^*, \ldots, x_{m^*}^*)$ whose contracts lengths are the optimal solution to the following LP, which we denote by P_m.

$$\begin{aligned}
\min \quad & \sum_{i=1}^{m} x_i \\
\text{subject to} \quad & x_i \geq L, \quad i \in [m-n+1, m] && (F_i) \\
& \sum_{j=1}^{i} x_j \leq \rho_n^* \cdot x_{i-n}, \quad i \in [n+1, m] && (C_i) \\
& x_i \leq x_{i+1}, \quad i \in [1, m-1] && (M_i) \\
& x_1 \leq \tau. && (I)
\end{aligned}$$

Here, constraints (F_i) model the feasibility of X for L; constraints (C_i) imply that X has optimal acceleration ratio, using (4.2); and constraints (M_i) model the monotonicity of X. Last, we need an initialization constraint for x_1, namely (I), which states that x_1 has length at most a fixed number τ, which is a constant that does not depend on other parameters. This is a reasonable assumption but is also required to exclude some unacceptable solutions. Otherwise, a cyclic schedule that starts with n contracts of length L would be feasible and optimal for the LP, but this is by no means an interruptible schedule.

4.3 Obtaining an Optimal Schedule

We will show how to obtain an optimal schedule given L. We will denote by $T^*(L)$ the completion time of an optimal schedule. Let $\mathcal{C}_m^*$ denote the class of all cyclic schedules with m contracts that have optimal acceleration ratio ρ_n^*. For simplicity we denote $T_i(X)$ by T_i when X is clear from context, with $T_0 = 0$.

We first give a road map of our approach. We begin by showing a lower bound on the lengths x_i of any schedule of optimal acceleration ratio (Lemma 4.9). This lower bound is expressed inductively in terms of T_i and two sequences a, b, which are defined appropriately in order to satisfy the inductive arguments. Next, we need to find the best value of m. To this end, we first show that if P_m is feasible, then the lower bounds on the x_i's hold with equality. This allows us to express the objective in terms of the parameters n, L and the sequences a, b (Lemma 4.10). Finally, to find the best value of m, we argue that it suffices to identify the smallest m for which P_m is feasible (Lemma 4.11).

We define the sequence a and b recursively as follows:

$$a_i = \begin{cases} 1, & i \in [0, n-1] \\ \frac{\sum_{j=0}^{n-1} a_{i-n+j} \prod_{k=0}^{j-1} (b_{i-n+k}+1)}{\rho_n^* - \prod_{k=0}^{n-1}(b_{i-n+k}+1)}, & i \geq n \end{cases} \tag{4.3}$$

and

$$b_i = \begin{cases} 0, & i \in [0, n-1] \\ \frac{\prod_{k=0}^{n-1}(b_{i-n+k}+1)}{\rho_n^* - \prod_{k=0}^{n-1}(b_{i-n+k}+1)}, & i \geq n. \end{cases} \tag{4.4}$$

Lemma 4.8. *For every $i \geq 0$, it holds that $a_i > 0$ and $b_i \in [0, \frac{1}{n}]$. In addition, $(b_i)_{i \geq 0}$ is monotone increasing with $\lim_{i \to +\infty} b_i = \frac{1}{n}$.*

The following lemma lower bounds the contract lengths of schedules in $\mathcal{C}_m^*$.

Lemma 4.9. *For any positive integers m, n with $m > n$ and for every schedule $X = (x_1, \ldots, x_m)$, with $X \in \mathcal{C}_m^*$, it holds that $x_i \geq a_{m-i} \cdot x_{m-n+1} + b_{m-i} \cdot T_{i-1}$ for $i \in [1, m]$. In addition, $x_i = a_{m-i} \cdot x_{m-n+1} + b_{m-i} \cdot T_{i-1}$ if constraints (C_j) for $j \in [i, m]$ and (M_i) for $i \in [m-n+1, m-1]$ are tight.*

Proof. The proof is by induction on i, for $i \in [1, m]$. The base cases can be readily verified. For the inductive step, suppose that for $i \leq m - n$ it holds that $x_j \geq a_{m-j} x_{m-n+1} + b_{m-j} T_{j-1}$ with $j \in [i+1, m]$. We will show that $x_i \geq a_{m-i} x_{m-n+1} +$

$b_{m-i}T_{i-1}$.

$$\begin{aligned}\rho_n^* x_i &\geq T_{i+n} = x_{i+n} + T_{i+n-1} \\ &\geq a_{m-i-n} \cdot x_{m-n+1} + (b_{m-i-n}+1)T_{i+n-1} \\ &= a_{m-i-n} \cdot x_{m-n+1} \\ &\quad + (b_{m-i-n}+1)(x_{i+n-1} + T_{i+n-2}).\end{aligned}$$

Inductively, it follows that

$$\begin{aligned}\rho_n^* x_i \geq &\left(\sum_{j=0}^{n-1} a_{m-n-i+j} \prod_{k=0}^{j-1} (b_{m-n-i+k}+1)\right) x_{m-n+1} \\ &+ \prod_{k=0}^{n-1} (b_{m-n-i+k}+1) T_i,\end{aligned}$$

which is equivalent to $x_i \geq a_{m-i} \cdot x_{m-n+1} + b_{m-i}T_{i-1}$. □

Lemma 4.9 allows us to find the optimal objective value of P_m, for a given m, assuming P_m has a feasible solution. This is shown in the next lemma.

Lemma 4.10. *Given $m \geq n$, assuming that P_m has a feasible solution, then the objective value of P_m is minimized if constraints (F_i) for $i \in [m-n+1, m]$, (M_i) for $i \in [m-n+1, m-1]$ and (C_i) for $i \in [n+1, m]$ are tight. Moreover, the minimum objective value is*

$$\left(n + \sum_{j=n}^{m-1} a_j \prod_{k=n}^{j-1} (b_k+1)\right) L,$$

Proof. Given $m \geq n$, for any feasible solution $X = (x_1, \ldots, x_m)$ of P_m, by Lemma 4.9, we have

$$\begin{aligned}T_m &= \sum_{i=m-n+1}^{m} x_i + T_{m-n} \\ &\geq nL + T_{m-n} \\ &= nL + (x_{m-n} + T_{m-n-1}) \\ &\geq (n+a_n)L + (b_n+1)T_{m-n-1} \\ &= (n+a_n)L + (b_n+1)(x_{m-n-1} + T_{m-n-2}).\end{aligned}$$

It follows inductively that

$$T_m \geq \left(n + \sum_{j=n}^{m-1} a_j \prod_{k=n}^{j-1} (b_k+1)\right) L.$$

We note that this lower bound is reached if constraints (F_i) for $i \in [m-n+1, m]$, (M_i) for $i \in [m-n+1, m-1]$ and (C_i) for $i \in [n+1, m]$ are tight. □

From Lemma 4.9 and 4.10, it follows that for a given m, under the assumption that P_m has a feasible solution, the optimal solution is derived by means of the recurrence relation

$$x_i = a_{m-i}x_{m-n+1} + b_{m-i}T_{i-1}, \text{ with } x_1 = a_{m-1}L.$$

The initial condition on x_1 comes from the constraint (C_{n+1}) in the LP P_m. Note that if $a_{m-1}L > \tau$, then P_m is not feasible, from Lemma 4.9.

Moreover, from the statement of x_i, we have that $T_i = \rho_n^* \cdot x_{i-n}$ for all $i \in [n+1, m]$, which implies that $x_i = \rho_n^*(x_{i-n} - x_{i-n-1})$, for all $i \in [n+2, m]$. Moreover, from Lemma 4.9, $x_i > 0$. Therefore, $x_{i-n} > x_{i-n-1}$ for $i \in [n+2, m]$. This in turn means that the solution defined above satisfies constraints (M_i) of P_m (since $x_i = L$ for $i \in [m-n+1, m]$).

Lemma 4.11. *The optimal objective value of P_m is monotone increasing in m. Thus, $T^*(L)$ is attained by the optimal objective value of P_M, where M is the smallest integer m such that P_m has a feasible solution.*

Proof. For $m \geq n$, by Lemma 4.10, the optimal objective value of P_m is $\alpha(m) \cdot L$, with $\alpha(m) = n + \sum_{j=n}^{m-1} a_j \prod_{k=n}^{j-1}(b_k + 1)$. By Lemma 4.8, it follows that $\alpha(m)$ is monotone increasing in m. This concludes the proof. □

It remains to find the smallest integer m such that P_m has feasible solutions; denote such m by m^*. To this end, we give an upper bound to m^*, as follows: we observe that the exponential cyclic strategy in which the i-th contract has length b^{i-1}, where $b = \frac{n+1}{n}$ has optimal acceleration ratio ρ_n^* (see the discussion in Section 4.1.2). Thus, there are at most $\lceil \frac{\log L}{\log b} + n \rceil = O(n \log L)$ candidate values for m^*, and the overall complexity of the algorithm is $O(n^2 \log^2 L)$. Algorithm 1 summarizes the steps needed to obtain the optimal schedule.

Algorithme 1 : Earliest-completion scheduling of contract algorithms with end guarantee L

```
Input: n ≥ 1, L > 0 and τ > 0
U ← ⌈log L / log b + n⌉
Compute (a_i)_{i∈[1,U]}, (b_i)_{i∈[1,U]} using the recurrence relations (4.3) and (4.4)
for m = n + 1 ... U do
    if a_{m-1} L ≤ τ then
        Output x_i, using the recurrence relation
          x_i = a_{m-i} x_{m-n+1} + b_{m-i} T_{i-1}(X), and stop.
    end
end
```

4.4 Computational Evaluation

In this section we present computational results on the implementation of our schedule. Recall that we denote completion time by $T^*(L)$. In particular, we compare $T^*(L)$ to the completion time of the exponential cyclic schedule with base $b = \frac{n+1}{n}$, whose completion time we denote by $T_{\texttt{exp}}(L)$. Recall that the latter is the known strategy with optimal acceleration ratio, which, however is oblivious of L. We choose τ to be equal to 1, and L to be integral in the range $[1, 10^6]$.

Figure 4.3 illustrates the completion times of the two schedules for $n = 5$. We observe that $T^*(L)$ is almost linear, unlike $T_{\texttt{exp}}(L)$ which is a step function. Similar almost-linear shapes were observed for $T^*(L)$ for the values of n for which we evaluated our schedule, with slopes increasing in n.

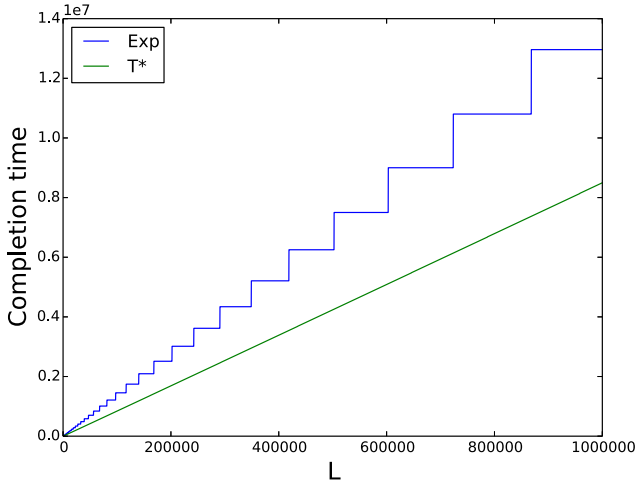


FIGURE 4.3 – Completion times $T^*(L)$ and $T_{\texttt{exp}}(L)$ as a function of L for $n = 5$.

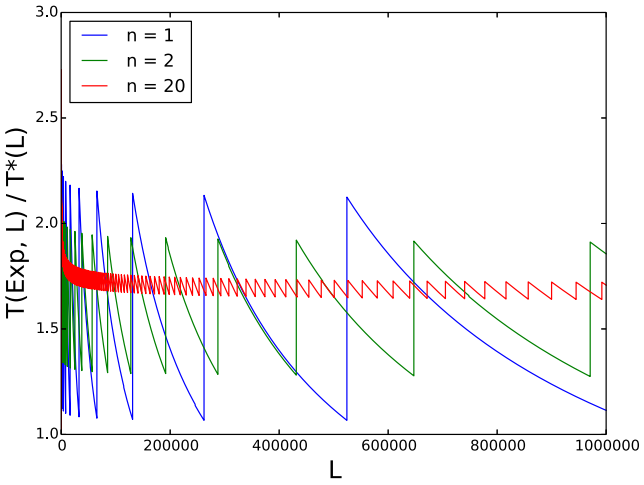


FIGURE 4.4 – Ratio $T_{\texttt{exp}}(L)/T^*(L)$ as a function of L.

Figure 4.4 illustrates the ratio $T_{\texttt{exp}}(L)/T^*(L)$ for $n \in \{1, 2, 20\}$. We observe that the fluctuations of the ratio, as a function of L, tend to decrease in n.

Figure 4.4 motivates the evaluation of the ratio $T_{\texttt{exp}}(L)/T^*(L)$, for large L. This is shown in Figure 4.5. The experiments suggest that a constant multiplicative gain is achieved by the optimal schedule, and for $L = 10^6$ the ratio tends to approximately 1.65, for large n.

We conclude this section with an observation on the running time of the implementation. In our computational evaluation, we observed that the values (a_i) described by (4.3) appear to be monotone decreasing in i, although this is hard to prove analytically. If this indeed holds, then one can show the following fact concerning the LP: If there exists m such that P_m has a feasible solution, then so does P_{m+1}. This implies a heuristic in which instead of $O(n \log L)$ candidate values for m, one needs to check only $O(\log n + \log\log L)$ values, using binary search, which improves the complexity to $O(n \log(\log n + \log\log L))$.

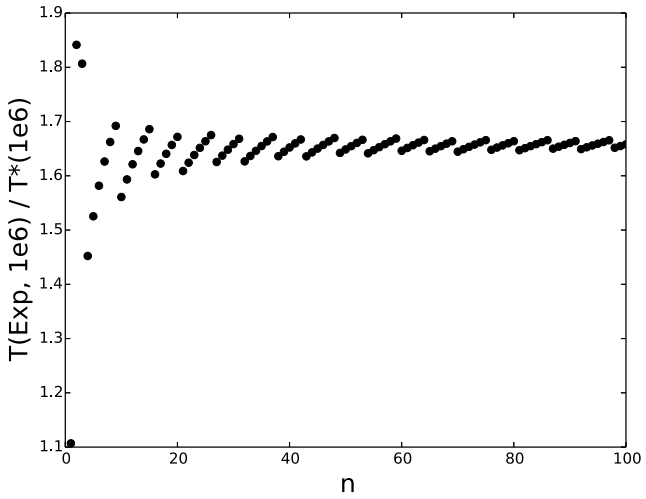


FIGURE 4.5 – Ratio $T_{\texttt{exp}}(L)/T^*(L)$, for $L = 10^6$.

4.5 Conclusion

In this chapter, we studied contract scheduling in a model in which the interruptible system is considered to be complete once certain performance guarantee has been reached on all problem instances. In particular, we demonstrated that the idea of using LP formulation only as guidance from Chapter 3 works so as to design a schedule that has best-of-both worlds guarantees (i.e. it completes the earliest possible, and guarantees optimal interruptible performance according to the acceleration ratio).

Furthermore, we can solve optimally a "dual" problem in which the interruptible system is given a *deadline*, and the objective is to maximize the worst-case performance among all problem instances while maintaining optimality according to the acceleration ratio. More precisely, given a *deadline* D, we would like to obtain a schedule X of contract algorithms for n problems with the following properties:

1. $T(X) \leq D$ (i.e. the schedule respects the deadline);
2. $\rho(X) = \rho_n^*$; (optimal for acceleration ratio);
3. X maximizes the parameter $\min_p \ell(p, D)$.

In other words, the objective is to obtain an interruptible system which has optimal interruptible behavior up to the deadline, and which maximizes the progress that has been made by the deadline among all problem instances. There are some real-life applications of this setting. For instance, consider the medical diagnostic systems, in which the deadline models the absolute time by which a diagnosis needs to be obtained. To solve this dual problem, we can apply Algorithm 1 in combination with the binary search over the space of end guarantees, namely with $O(\log D)$ applications of our algorithm.

Chapter 5

Online Bidding with Untrusted Advice

This chapter contains material from the joint paper, *Online Computation with Untrusted Advice*, with Spyros Angelopoulos, Christoph Dürr, Shahin Kamali and Marc Renault. This work appeared in the *11th Innovations in Theoretical Computer Science* conference (ITCS), 2020 [5]. The untrusted advice model was proposed by Spyros Angelopoulos, Christoph Dürr and Shahin Kamali, and is presented in Section 1.4. My contribution is to study the *online bidding* problem under the untrusted advice model and to identify a Pareto optimal strategy assuming that the advice encodes the hidden target. This part of the contribution is presented in Section 5.1.3.

5.1 Introduction

5.1.1 Online Computation with Advice

We first revisit the standard setting of online computation. As mentioned in Section 1.1, in the standard online computation model, the entire input is not available at the beginning. In contrast, the input is only revealed incrementally, piece by piece, as a sequence of requests. For each such request, the algorithm has to immediately make decisions without any knowledge on future requests, and these decisions are irrevocable. Usually, the performance of an online algorithm is evaluated by the competitive ratio as discussed in Section 1.1.1.

The above model captures many online problems, but it does not capture certain situations. In practice, some additional information on the input sequence is sometimes given to online algorithms, which may encode partial information on optimal decisions that the online algorithm should take. For instance, the online algorithm may know the size of the input sequence at the beginning; or the online algorithm may have some look ahead on the input sequence instead of looking at only one request item at each time. Thus, the online algorithm is expected to use this kind of additional information to improve its performance. Typically, this action is not captured by the standard online computation model. A new model is then required to quantify the power of this additional information.

This type of information that is given along with the input sequence to the online algorithm, is called *advice*. In general, the advice is an arbitrary function of the input, and is designed together with the algorithm. There is a trade-off between the number of advice bits and the performance of online algorithms. For instance, consider the following two extreme cases:

— If the advice is empty, then the advice model reduces to the standard online computation setting;

— if the number of advice bits is infinite, then the advice may encode the offline optimal decisions that the online algorithm should make, in which the algorithm should have an optimal performance by following the advice.

A natural question here is to quantify the trade-off between the performance of online algorithms and advice size, which has been studied since 2009. The term *advice complexity* was introduced by Dobrev *et al.* [26] and the formal models were later given by Böckenhauer *et al.* [15] and Emek *et al.* [30]. More precisely, an online problem P is c-competitive with advice of size $f(n)$ if there is a c-competitive online algorithm for P with advice of size at most $f(n)$, where n is the length of the input sequence. In the last decade, many online optimization problems have been studied in the advice model, such as k-server, paging, makespan scheduling, etc. The survey of Boyar *et al.* [18] provides a further discussion on this topic.

All previous work assumed that the advice is always *correct*. In other words, the advice is assumed to be generated by some *trusted* oracle. Thus, there is no reason for any online algorithm with advice to ignore the given advice. This assumption is unrealistic in practice: the advice can be generated by some untrusted source, or in the worst case, a malicious adversary may take control of the advice oracle, which may have a catastrophic impact on the performance of the online algorithm.

Let us illustrate this situation by considering the *ski rental* problem, which is a fundamental resource allocation problem. You are going skiing for D days, where D is an unknown integer. Suppose that renting skis costs 1 per day and buying skis costs $B > 1$. At the beginning of each day, you have to decide whether to continue renting skis for one more day or to buy a pair of skis unless you have already bought them. The objective is to minimize the ratio between what you pay using the algorithm and what you would pay optimally if you knew D in advance in the worst-case instance. In the traditional advice model, one advice bit suffices to be optimal: 0 for renting throughout the D days, 1 for buying right away. However, if this advice bit is wrong, the algorithm can perform extremely badly. In other words, it can have unbounded competitive ratio. In contrast, an online algorithm without advice can have competitive ratio $2 - \frac{1}{B}$, which can be achieved by renting skis until day $B - 1$ and buying skis on day B.

5.1.2 A New Model with Untrusted Advice

Recently, a new advice model was proposed to capture the above observations in the context of online algorithm with machine-learned predictors by Lykouris and Vassilvitskii [52], and Purohit *et al.* [57]. In their work, they use predictors to design and analyze online algorithms. More precisely, if the predictor is bad, then the online algorithm should perform close to the one without predictions; if the predictor is good, then the online algorithm should perform close to the optimal offline algorithm.

Motivated by the above work from machine learning, in this chapter, we consider the advice model in which the advice can be either *trusted* or *untrusted*. Thus, it is natural to characterize the performance of an online algorithm A by a pair of competitive ratios, denoted by (r_A, w_A), respectively. Here, r_A is the competitive ratio of A when the advice is trusted. In contrast, w_A is the competitive ratio of A when the advice is untrusted. More precisely, we assume that the untrusted advice is generated by a malicious adversary to be in accordance with the worst-case nature of the competitive analysis.

More formally, let σ denote the input sequence and $\phi(\sigma)$ denote the advice. Let $A(\sigma, \phi(\sigma))$ denote the cost incurred by A on input sequence σ, using an advice $\phi(\sigma)$.

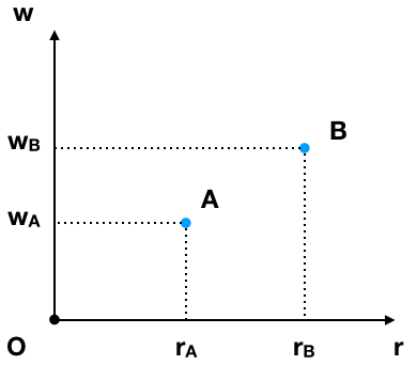


FIGURE 5.1 – Each algorithm is represented by a point in the 2-dimensional space. In this example, algorithm A Pareto dominates algorithm B.

Denote by r_A, w_A as

$$r_A = \sup_{\sigma} \inf_{\phi} \frac{A(\sigma, \phi(\sigma))}{\mathrm{OPT}(\sigma)}, \quad \text{and} \quad w_A = \sup_{\sigma} \sup_{\phi} \frac{A(\sigma, \phi(\sigma))}{\mathrm{OPT}(\sigma)}. \tag{5.1}$$

We say that algorithm A is (r, w)-competitive for every $r \geq r_A$ and $w \geq w_A$.

Let us illustrate the above definition on ski rental problem. The previously described 1-bit advice algorithm is $(1, \infty)$-competitive. In contrast, the standard competitively optimal algorithm without advice is $(2, 2)$-competitive. In general, we observe that, by definition, every online algorithm A without advice or ignoring its advice is (w, w)-competitive, where w is the competitive ratio of A in the standard online setting.

Hence, we can associate every algorithm A to a point in the 2-dimensional space with coordinates (r_A, w_A). We say that algorithm A dominates algorithm B if $r_A \leq r_B$ and $w_A \leq w_B$. We illustrate this graphical representation in Figure 5.1. A natural goal is to describe the Pareto frontier in this 2-dimensional space. More precisely, we would like to characterize it by a single *family* $\mathcal{A}$ of algorithms, with similar statements (e.g., algorithms in $\mathcal{A}$ are obtained by appropriately selecting a parameter). We say that $\mathcal{A}$ is *Pareto-optimal* if it consists of pairwise incomparable algorithms, and for every algorithm B, there exists $A \in \mathcal{A}$ such that A dominates B. We illustrate this definition in Figure 5.2. For a given $\mathcal{A}$, we will describe its competitiveness by a function $f : \mathbb{R}_{\geq 1} \to \mathbb{R}_{\geq 1}$ such that for every ratio w there is an $(f(w), w)$-competitive algorithm in $\mathcal{A}$.

5.1.3 Online Bidding with Untrusted Advice

The standard *online bidding* problem is presented in Section 1.4.2, which was introduced in [22] as a typical example for formalizing efficient doubling in the context of online and offline optimization problems. In this chapter, we are going to study the online bidding problem in the untrusted advice setting. As discussed above, the competitiveness of each algorithm is represented by a point in a 2-dimensional space. Our objective is to identify the Pareto frontier of the set of all possible algorithms in this space assuming that there is an infinite number of advice bits encoding the hidden target.

It is well known that, in the standard online setting, the best competitive ratio is 4 [22], which can be achieved using the doubling strategy $x_i = 2^i$. In contrast,

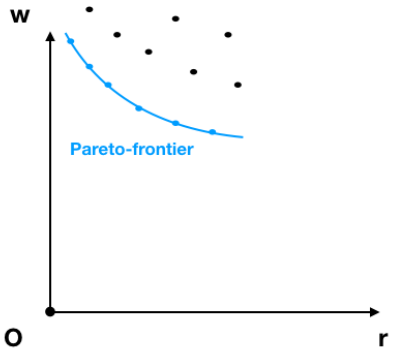


FIGURE 5.2 – Each point depicts the competitiveness of an algorithm. Blue points are the Pareto-frontier of the set of all possible algorithms. Note that these points are above the diagonal since $w \geq r$.

in the advice model, if the advice encodes the value u, we have an optimal strategy with $x_1 = u$, assuming that the advice is trusted. Hence, in the untrusted advice setting, there are simple strategies that are $(4, 4)$-competitive and $(1, \infty)$-competitive, respectively.

The objective of this chapter is to show a Pareto-optimal bidding strategy X^*_u for a given fixed parameter $w \geq 4$, assuming that the advice encodes the hidden value u, which is $(\frac{w-\sqrt{w^2-4w}}{2}, w)$-competitive (see Theorem 5.8).

5.1.4 Preliminaries

For convenience, we will say that an algorithm A is w-competitive if it is (r, w)-competitive, for some r.

Suppose that $w \geq 4$ is a fixed given parameter. For $m \in \mathbb{N}^+$, let $S_{m,u}$ denote the set of bidding strategies with advice u which are w-competitive, and which, if the advice is trusted, succeed in finding the value with precisely the m-th bid. Let p denote the value $\frac{w-2-\sqrt{w^2-4w}}{2}$, which will be useful in the technical proof.

We say that a strategy $X \in S_{m,u}$ that is (r, w)-competitive dominates $S_{m,u}$ if for every $X' \in S_{m,u}$, such that X' is (r', w)-competitive, $r \leq r'$.

5.2 Identifying a Pareto-optimal Bidding Strategy

5.2.1 Algorithm Overview

We first show a high-level idea of the algorithm. The algorithm consists of two phases. In the first phase of the algorithm, we identify a dominant strategy, denoted by $X^*_{m,u}$ in $S_{m,u}$ (if a such dominant strategy exists) for any given m. In the second phase, we identify the best choice of m, denoted by m^*, which minimizes the total cost to find the hidden value u by the strategy $X^*_{m,u}$. Then $X^*_{m^*,u}$ is a Pareto-optimal bidding strategy.

5.2.2 Phase 1: Identifying a dominant strategy $X^*_{m,u}$ in $S_{m,u}$

For a given m, in order to identify a dominant strategy $X_{m,u}$ in $S_{m,u}$, we will follow the same idea as what we did in Chapter 3 and Chapter 4. More precisely, the road-map is as follows:

1. Use linear programming to formulate the problem.
2. Argue that optimal solutions saturate certain constraints.
3. Establish recurrence relations on LP variables using equations obtained in the previous step.
4. Solve the recurrence relations to obtain an optimal solution.

Let us first formulate the problem using linear programming. Let $X^*_{m,u}$ denote such a strategy, and denote by $(r^*_{m,u}, w)$ its competitiveness. Then $X^*_{m,u}$ and $r^*_{m,u}$ are the solutions to an infinite linear program which we denote by $(P_{m,u})$, and which is shown below. For convenience, for any strategy X, we define x_0 to be equal to 1.

$$
\begin{aligned}
\min \quad & r_{m,u} && (P_{m,u})\\
\text{s.t. } & x_i < x_{i+1}, \quad i \in \mathbb{N}^+ \\
& x_{m-1} < u \leq x_m \\
& \sum_{j=1}^{m} x_j \leq r_{m,u} \cdot u \\
& \sum_{j=1}^{i} x_j \leq w \cdot x_{i-1}, \quad i \in \mathbb{N}^+ \\
& x_i \geq 0, \quad i \in \mathbb{N}^+.
\end{aligned}
$$

Note that in $(P_{m,u})$ the constraints $\sum_{j=1}^{i} x_j \leq w \cdot x_{i-1}$ guarantee that the competitive ratio of X is at most w, if the advice is untrusted, whereas the constraints $\sum_{j=1}^{m} x_j \leq r_{m,u} \cdot u$ and $x_{m-1} < u \leq x_m$ guarantee that if the advice is trusted, then X succeeds in finding u precisely with its m-th bid, and in this case the competitive ratio is $r_{m,u}$.

We also observe that an optimal solution $X^*_{m,u} = (x^*_i)_{i \geq 1}$ for $(P_{m,u})$ must be such that

$$x^*_m = u,$$

otherwise one could define a strategy $X'_{m,u}$ in which $x'_i = x^*_i / \alpha$, for all $i \geq 1$, with $\alpha = u / x^*_m$. Then the strategy $X'_{m,u}$ with $x'_m = u$ is still feasible for $(P_{m,u})$ and has better objective value than $X^*_{m,u}$, a contradiction.

Furthermore, in an optimal solution, the constraint $\sum_{i=1}^{m} x_i \leq r_{m,u} \cdot u$ must hold with equality, which implies that

$$r_{m,u} = \frac{\sum_{i=1}^{m} x_i}{u}.$$

Therefore, $X^*_{m,u}$ and $r^*_{m,u}$ are solutions to the linear program, $(L_{m,u})$, as follows.

$$
\begin{aligned}
\min \quad & \frac{1}{u} \cdot \sum_{i=1}^{m} x_i && (L_{m,u})\\
\text{s.t. } & x_m = u && (F)\\
& x_i < x_{i+1}, \quad i \in \mathbb{N}^+ && (M_i)\\
& \sum_{j=1}^{i} x_i \leq w \cdot x_{i-1}, \quad i \in \mathbb{N}^+ && (C_i)\\
& x_i \geq 0, \quad i \in \mathbb{N}^+.
\end{aligned}
$$

For convenience, we define

$$r_u^* = \inf_m r_{m,u}^* \quad \text{and} \quad r^* = \sup_u r_u^*.$$

Informally, r_u^*, r^* are the optimal competitive ratios, assuming trusted advice. More precisely, the dominant strategy in the space of all w-competitive strategies is (r_u^*, w)-competitive, and r^* is an upper bound on r_u^*, assuming the worst-case choice of u.

Our objective is then equivalent to compute the value of r^* for any given $u \geq 1$. As discussed above, we can compute r^* by solving the linear program $(L_{m,u})$ with a specific value of m that minimizing the objective value of the LP.

Given $u, m \geq 1$, assuming that $(L_{m,u})$ is feasible, we are going to show how to compute the optimal objective value of $(L_{m,u})$. The high-level idea is to first establish recurrence relations on LP variables using equations from constraints, then we obtain the optimal solution by solving the obtained recurrence relations.

For convenience, let T_i denote $\sum_{j=1}^{i} x_j$, with $T_0 = T_{-1} = 0$.

We define the sequences $(a_i)_{i\geq 0}$, $(b_i)_{i\geq 0}$, $(c_i)_{i\geq 0}$ and $(d_i)_{i\geq 0}$ as follows, which will be useful to establish recurrence relations on variables of $(L_{m,u})$ as shown in Lemma 5.2.

$$a_i = \frac{a_{i-1}}{w - 1 - b_{i-1}}, \text{ with } a_0 = 1, \tag{5.2}$$

$$b_i = \frac{1 + b_{i-1}}{w - 1 - b_{i-1}}, \text{ with } b_0 = 0, \tag{5.3}$$

$$c_i = c_{i-1} + d_{i-1} \cdot a_{i-1}, \text{ with } c_0 = 0, \tag{5.4}$$

$$d_i = d_{i-1} \cdot (1 + b_{i-1}), \text{ with } d_0 = 1. \tag{5.5}$$

The above sequences satisfy the following technical properties.

Lemma 5.1. *For $i \geq 0$, we have*

$$a_i = \begin{cases} \frac{2}{i+2} \cdot \frac{1}{2^i}, & w = 4 \\ \frac{p^2-1}{p^{i+2}-1} \cdot (\frac{p}{w})^{\frac{i}{2}}, & w > 4 \end{cases}, \quad b_i = \begin{cases} \frac{i}{i+2}, & w = 4 \\ p \cdot \frac{p^i-1}{p^{i+2}-1}, & w > 4 \end{cases},$$

$$c_i = \begin{cases} 2 - \frac{2}{i+1}, & w = 4 \\ 1 + p - \frac{p^i(p^2-1)}{p^{i+1}-1}, & w > 4 \end{cases} \quad \text{and} \quad d_i = \begin{cases} \frac{2^i}{i+1}, & w = 4 \\ \frac{p-1}{p^{i+1}-1} \cdot (pw)^{\frac{i}{2}}, & w > 4 \end{cases},$$

with $p = \frac{w-2-\sqrt{w^2-4w}}{2}$.

Proof. It is easy to verify that $p = \frac{w-2-\sqrt{w^2-4w}}{2}$ satisfies the following equation

$$p = \frac{1+p}{w-1-p}. \tag{5.6}$$

From (5.3), we have

$$b_i - p = \frac{(p+1)(b_{i-1} - p)}{w - 1 - p - (b_{i-1} - p)},$$

which implies that

$$\frac{1}{b_i - p} = \frac{w-1-p}{p+1} \cdot \frac{1}{b_{i-1}-p} - \frac{1}{p+1}.$$

Define the sequence $(u_i)_{i\geq 0}$ as $u_i = \frac{1}{b_i - p}$ for $i \geq 0$, then

$$u_i = \frac{1}{p} \cdot u_{i-1} - \frac{1}{p+1}, \quad \text{with} \quad u_0 = \frac{-1}{p}.$$

Thus,

$$u_i = \begin{cases} -\frac{i+2}{2}, & w = 4 \\ -\frac{p^{i+2}-1}{(p^2-1)p^{i+1}}, & w > 4 \end{cases},$$

which implies that

$$b_i = \begin{cases} \frac{i}{i+2}, & w = 4 \\ p \cdot \frac{p^i - 1}{p^{i+2}-1}, & w > 4 \end{cases}.$$

Then

$$\prod_{j=1}^{i} b_j = \begin{cases} \frac{2}{(i+1)(i+2)}, & w = 4 \\ p^i \cdot \frac{(p-1)(p^2-1)}{(p^{i+1}-1)(p^{i+2}-1)}, & w > 4 \end{cases}, \tag{5.7}$$

In addition, from (5.2) and (5.5), for $i \geq 1$, we have

$$a_i = \prod_{j=1}^{i} \frac{1}{w-1-b_{j-1}} \quad \text{and} \quad d_i = \prod_{j=1}^{i}(1+b_{j-1}).$$

Then, for $i \geq 2$,

$$a_i d_i = \prod_{j=1}^{i} \frac{(1+b_{j-1})}{w-1-b_{j-1}} = \prod_{j=1}^{i} b_j. \tag{5.8}$$

Moreover, from (5.3), we have

$$1 + b_i = \frac{w}{w-1-b_{i-1}},$$

then

$$\prod_{j=1}^{i}(1+b_j) = w^i \cdot \prod_{j=1}^{i} \frac{1}{w-1-b_{j-1}},$$

which implies that

$$d_{i+1} = w^i \cdot a_i. \tag{5.9}$$

Combining (5.5), (5.8) and (5.9), we have

$$a_i = \sqrt{\frac{(1+b_i)\cdot \prod_{j=1}^{i} b_j}{w^i}} \quad \text{and} \quad d_i = \sqrt{\frac{w^i \cdot \prod_{j=1}^{i} b_j}{1+b_i}}.$$

Thus, if $w > 4$, then we have

$$a_i = \sqrt{\frac{(1+p\cdot\frac{p^i-1}{p^{i+2}-1})\cdot p^i \cdot \frac{(p-1)(p^2-1)}{(p^{i+1}-1)(p^{i+2}-1)}}{w^i}} = \frac{p^2-1}{p^{i+2}-1} \cdot \left(\frac{p}{w}\right)^{\frac{i}{2}}$$

and

$$d_i = \sqrt{\frac{w^i \cdot p^i \cdot \frac{(p-1)(p^2-1)}{(p^{i+1}-1)(p^{i+2}-1)}}{(1+p\cdot\frac{p^i-1}{p^{i+2}-1})}} = \frac{p-1}{p^{i+1}-1}\cdot(pw)^{\frac{i}{2}}.$$

If $w = 4$, then we have

$$a_i = \sqrt{\frac{(1+\frac{i}{i+2})\cdot\frac{2}{(i+1)(i+2)}}{w^i}} = \frac{2}{i+2}\cdot\frac{1}{w^{\frac{i}{2}}}$$

and

$$d_i = \sqrt{\frac{w^i\cdot\frac{2}{(i+1)(i+2)}}{1+\frac{i}{i+2}}} = \frac{1}{i+1}w^{\frac{i}{2}}$$

From (5.4), for $i \geq 1$, we have

$$c_i = \sum_{j=1}^{i} a_{j-1}d_{j-1} = 1 + \sum_{j=2}^{i}\prod_{k=1}^{j-1} b_k = 1 + \sum_{j=1}^{i-1}\prod_{k=1}^{j} b_k.$$

Then, by combining with (5.7), we have

$$\begin{aligned} c_i &= \begin{cases} 1+\sum_{j=1}^{i-1}\frac{2}{(j+1)(j+2)}, & w=4 \\ 1+\sum_{j=1}^{i-1} p^j\cdot\frac{(p-1)(p^2-1)}{(p^{j+1}-1)(p^{j+2}-1)}, & w>4 \end{cases} \\ &= \begin{cases} 1+\sum_{j=1}^{i-1}\frac{2}{j+1}-\frac{2}{j+2}, & w=4 \\ 1+\sum_{j=1}^{i-1}(p^2-1)\left(\frac{p^j}{p^{j+1}-1}-\frac{p^{j+1}}{p^{j+2}-1}\right), & w>4 \end{cases} \\ &= \begin{cases} 2-\frac{2}{i+1}, & w=4 \\ 1+p-\frac{p^i(p^2-1)}{p^{i+1}-1}, & w>4 \end{cases}. \end{aligned}$$

This concludes the proof. □

Assuming that $(L_{m,u})$ is feasible, the following lemma shows that for every feasible solution X of $(L_{m,u})$, there is a lower bound on x_i for all $i \in [1,m]$, as well as a lower bound on T_m. In addition, we argue that these lower bounds can be attained by saturating all constraints (C_i) with equality for all i.

Lemma 5.2. *For every feasible solution $X = (x_1, x_2, \ldots)$ of $(L_{m,u})$, it holds that,*

$$x_i \geq a_{m-i}\cdot u + b_{m-i}\cdot T_{i-1}, \quad \text{for } i \in [1,m],$$

and

$$T_m \geq c_{m-i}\cdot u + d_{m-i}\cdot T_i, \quad \text{for } i \in [0,m].$$

In addition, for $i \in [1,m]$,

$$x_i = a_{m-i}\cdot u + b_{m-i}\cdot T_{i-1}, \quad \text{and} \quad T_m = c_{m-i}\cdot u + d_{m-i}\cdot T_i,$$

if and only if constraints (C_j) for $j \in [i+1,m]$ are all satisfied with equality.

Proof. The proof is by induction on i, for $i \in [0,m]$. The base case, namely for $i = m$, is verified, since

$$x_m = u = 1\cdot u + 0\cdot T_{m-1} \quad \text{and} \quad T_m = 0\cdot u + 1\cdot T_m.$$

For the inductive step, suppose that for $i \in [1, m-1]$ it holds that $x_j \geq a_{m-j} \cdot u + b_{m-j} \cdot T_{j-1}$ and $T_m \geq c_{m-j} \cdot u + d_{m-j} \cdot T_j$ with $j \in [i+1, m]$. We will show that $x_i \geq a_{m-i} \cdot u + b_{m-i} \cdot T_{i-1}$ and $T_m \geq c_{m-i} \cdot u + d_{m-i} \cdot T_i$. By the constraint (C_{i+1}), we have

$$\begin{aligned} w \cdot x_i &\geq T_{i+1} \\ &= x_{i+1} + T_i && \text{(By definition of } T_{i+1}) \\ &\geq a_{m-i-1} \cdot u + b_{m-i-1} \cdot T_i + T_i && \text{(Induction hypothesis on } x_{i+1}) \\ &= a_{m-i-1} \cdot u + (1 + b_{m-i-1}) \cdot T_i \\ &= a_{m-i-1} \cdot u + (1 + b_{m-i-1}) \cdot (x_i + T_{i-1}) && \text{(By definition of } T_i) \end{aligned}$$

It implies that

$$x_i \geq \frac{a_{m-i-1}}{w - 1 - b_{m-i-i}} \cdot u + \frac{1 + b_{m-i-1}}{w - 1 - b_{m-i-i}} \cdot T_{i-1},$$

which is equivalent to

$$x_i \geq a_{m-i} \cdot u + b_{m-i} \cdot T_{i-1}.$$

It is straightforward to see that the previous inequality holds with equality if and only if constraints (C_j) are tight for $j \in [i+1, m]$. Moreover, from induction hypothesis, we have

$$\begin{aligned} T_m &\geq c_{m-i-1} \cdot u + d_{m-i-1} \cdot T_{i+1} \\ &= c_{m-i-1} \cdot u + d_{m-i-1} \cdot (x_{i+1} + T_i) && \text{(By definition of } T_{i+1}) \\ &\geq c_{m-i-1} \cdot u + d_{m-i-1} \cdot (a_{m-i-1} \cdot u + b_{m-i-1} \cdot T_i + T_i) && \text{(By ind. hyp. on } x_{i+1}) \\ &= (c_{m-i-1} + d_{m-i-1} \cdot a_{m-i-1}) \cdot u + d_{m-i-1} \cdot (1 + b_{m-i-1}) T_i, \end{aligned}$$

which is equivalent to

$$T_m \geq c_{m-i} \cdot u + d_{m-i} \cdot T_i.$$

The inequality holds with equality if and only if constraints (C_j) are tight for $j \in [i+1, m]$. This concludes the proof. □

We obtain the following result immediately from the proof of Lemma 5.2, as shown in Corollary 5.3.

Corollary 5.3. *If $X = (x_i)_{i \geq 1}$ satisfies constraints (C_j) with equality, for $j \in [2, m]$, then $x_1 = a_{m-1} \cdot u$ if and only if $x_m = u$.*

For $u \geq 1$ and $m \geq 1$, consider the strategy $X^*_{m,u} = (x^*_i)_{i \geq 0}$ defined as follows:

$$x^*_i = w(x^*_{i-1} - x^*_{i-2}) \text{ for } i \geq 3, \text{ with } x^*_2 = (w-1) \cdot x^*_1, \text{ and } x^*_1 = a_{m-1} \cdot u.$$

We are going to show that the strategy $X^*_{m,u}$ satisfies all constraints of $(L_{m,u})$ except for (C_1).

Lemma 5.4. *The strategy $X^*_{m,u}$ satisfies constraints (F), (M_i) for $i \in [1, m-1]$. It also satisfies constraints (C_j) with equality for $j \in [2, m]$.*

Proof. For $i \geq 2$, we have

$$\begin{aligned}\sum_{j\in[1,i]} x_j^* &= x_1^* + x_2^* + \sum_{j=3}^{i} x_j^* \\ &= x_1^* + x_2^* + \sum_{j=3}^{i} w(x_{j-1}^* - x_{j-2}^*) \\ &= x_1^* + (w-1)\cdot x_1^* + w\cdot x_{i-1}^* - w\cdot x_1^* \\ &= w\cdot x_{i-1}^*.\end{aligned}$$

Thus $X_{m,u}^*$ satisfies constraints (C_j) with equality for $j \geq [2,m]$. By Corollary 5.3, we have $x_m^* = u$. It remains to show that $X_{m,u}^*$ satisfies $x_i^* < x_{i+1}^*$ for $i \in [1, m-1]$. In fact, it suffices to show that $x_i^* > 0$ for $i \geq 3$, since we have $x_i^* = w(x_{i-1}^* - x_{i-2}^*)$ for $i \geq 3$. To this end, we will show that for $i \geq 1$, $x_i^* \geq x_1^* \cdot \left(\frac{w-\sqrt{w^2-4w}}{2}\right)^{i-1}$, which implies that $x_i^* > 0$, for $i \geq 3$.

For this purpose, we argue that $\frac{x_i^*}{x_{i-1}^*} \geq \frac{w-\sqrt{w^2-4w}}{2}$, for $i \geq 2$. The proof is by induction on i. For the base case, namely for $i = 2$, we have $x_2^* = (w-1)x_1^*$, then for $w \geq 4$, it holds that

$$\frac{x_2^*}{x_1^*} = w - 1 \geq \frac{w-\sqrt{w^2-4w}}{2}.$$

For the inductive step, suppose that for all $j \in [1, i-1]$, we have $\frac{x_{j+1}^*}{x_j^*} \geq \frac{w-\sqrt{w^2-4w}}{2}$. Then,

$$\begin{aligned}\frac{x_{i+1}^*}{x_i^*} &= \frac{w(x_i^* - x_{i-1}^*)}{x_i^*} \\ &= w\left(1 - \frac{x_{i-1}^*}{x_i^*}\right) \\ &\geq w\left(1 - \frac{2}{w-\sqrt{w^2-4w}}\right) \\ &= \frac{w-\sqrt{w^2-4w}}{2},\end{aligned}$$

which concludes the proof. □

We are now able to identify an optimal solution to $(L_{m,u})$, which is the main result in this section.

Lemma 5.5. *Given $u \geq 1$ and $m \geq 1$, $(L_{m,u})$ is feasible if and only if it holds that $a_{m-1}\cdot u \leq w$. In addition, if $(L_{m,u})$ is feasible, then $X_{m,u}^*$ is an optimal solution to $(L_{m,u})$, and the optimal objective value is c_m.*

Proof. Suppose that $(L_{m,u})$ is feasible, then from Lemma 5.2, it holds that $x_1 \geq a_{m-1}\cdot u$. Moreover, by constraint (C_1), we have $x_1 \leq w$, which implies that $a_{m-1}\cdot u \leq w$.

Suppose that $a_{m-1}\cdot u \leq w$ holds, then by Lemma 5.4, $X_{m,u}^*$ is a feasible solution to $(L_{m,u})$. In addition, by Lemma 5.2, $X_{m,u}^*$ is optimal, and the optimal objective value is c_m. □

5.2.3 Phase 2: Identifying an optimal strategy X_u^*

The objective of phase 2 is to identify an optimal strategy X_u^* for a given $u \geq 1$. To this end, it suffices to identify m for which $(L_{m,u})$ is feasible and its objective value is minimized. The following lemma shows that it suffices to choose a smallest m^* for which $(L_{m^*,u})$ is feasible.

Lemma 5.6. *For a given $u \geq 1$, the optimal objective value of $(L_{m,u})$ is monotone increasing in m, assuming that $(L_{m,u})$ is feasible.*

Proof. From Lemma 5.2, we have $T_m \geq c_m \cdot u$, and the corresponding lower bound can be attained by the strategy $X_{m,u}^*$ by Lemma 5.5. Thus for $m \geq 1$, assuming that $(L_{m,u})$ is feasible, the optimal objective value of $(L_{m,u})$ is c_m, which is monotone increasing in m by Lemma 5.1. □

We can now give the statement of the optimal strategy X_u^*. By Lemma 5.6, it suffices to find the objective value of the smallest m^* for which $L_{m^*,u}$ is feasible. This can be accomplished with a binary search in the interval $[1, \lceil \log u \rceil]$, since we know that the doubling strategy in which the i-th bid equals 2^i is w-competitive for all $w \geq 4$; hence $m^* \leq \lceil \log u \rceil$. Then $X_u^* := X_{m^*,u}^*$ where $m*$ is the smallest value such that $(L_{m,u})$ is feasible.

The advice complexity of the algorithm is $O(\log \log u)$, since we can describe each a_i, b_i, and hence a_{m-1} in closed form, avoiding the recurrence which would add an $O(\log u)$ factor.

Last, the following lemma allows us to obtain the exact value of r^*.

Lemma 5.7. *It holds that $r^* = \frac{w - \sqrt{w^2 - 4w}}{2}$.*

Proof. By Lemma 5.2 and Lemma 5.5, we have $r_{m,u}^* = c_m$. Combining with Lemma 5.1, we have

$$r_{m,u}^* = \begin{cases} 2 - \frac{2}{m+1}, & w = 4 \\ 1 + p - \frac{p^m(p^2-1)}{p^{m+1}-1}, & w > 4 \end{cases},$$

which implies that,

$$r_u^* = \inf_m r_{m,u}^* = r_{m^*,u}^*.$$

If $w = 4$, then the worst case ratio is

$$\begin{aligned} r^* &= \sup_u r_u^* \\ &= \sup_u r_{m^*,u}^* \\ &= \lim_{m^* \to +\infty} 2 - \frac{2}{m^* + 1} \\ &= 2. \end{aligned}$$

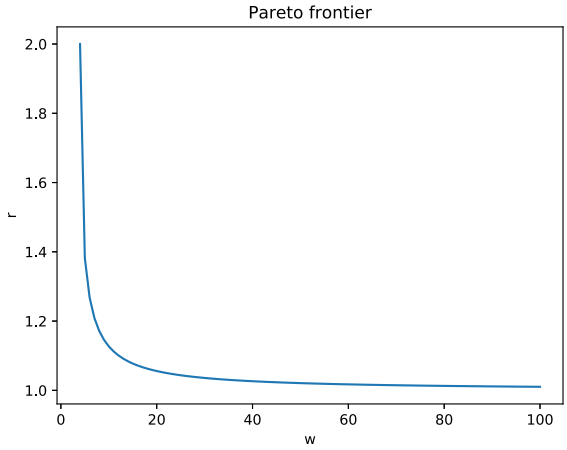


FIGURE 5.3 – The Pareto curve $f(w) = \frac{w^2-\sqrt{w^2-4w}}{2}$, for $w \geq 4$.

If $w > 4$, then the worst case ratio is

$$\begin{aligned} r^* &= \sup_u r_u^* \\ &= \sup_u r_{m^*,u}^* \\ &= \lim_{m \to +\infty} 1 + p - \frac{p^m(p^2-1)}{p^{m+1}-1} \\ &= 1 + p \\ &= \frac{w - \sqrt{w^2-4w}}{2}. \end{aligned}$$

This concludes the proof. □

By Lemma 5.7 we obtain the main result in this chapter.

Theorem 5.8. *The strategy X_u^* is Pareto-optimal and is $(\frac{w-\sqrt{w^2-4w}}{2}, w)$-competitive.*

5.3 Conclusion

In this chapter, we studied the online bidding problem in the untrusted advice model. We identified a Pareto-optimal strategy for a given fixed $w \geq 4$, which is $(\frac{w-\sqrt{w^2-4w}}{2}, w)$-competitive, thus forming a Pareto frontier in the 2-dimensional space (see Section 5.1.1). Figure 5.3 illustrates the above results.

In particular, for $w = 4$, which is the best competitive ratio without advice, we identified a strategy which is $(2, 4)$-competitive. However, the algorithm requires u as advice, which can be unbounded. A natural question here is what competitiveness can we achieve with only k advice bits, for some fixed k. we answered this question both from the point of view of upper and lower bounds in our paper [5]. For the upper bounds, we showed that for every $w \geq 4$, there exists a strategy with k bits of advice which is (r, w)-competitive, where r is expressed by some more complicated expression on w. In particular, for $w = 4$, we have $(2^{1+\frac{1}{2^k}}, 4)$-competitive, whereas X_u^* is $(2, 4)$-competitive. This result makes sense, since X_u^* requires infinite

precision. We also gave a lower bound on the competitiveness of any bidding strategy with k bits. The results confirm that indeed an infinite number of advice bits is necessary to achieve $(2, 4)$-competitiveness, for $w = 4$.

In terms of techniques, we used same of the proof ideas from Chapters 3 and 4. For these problems that we studied, the corresponding LPs can be solved by using the fact that optimal solutions saturate some constraints and we do not need any LP solvers to solve the LP.

Chapter 6

Conclusion

6.1 Summary of Results

In this thesis, we have studied some new ways of looking at online problems. More precisely, we have explored the following three topics:

1. Online computation with recourse (Chapter 2);
2. Performance measures other than standard competitive analysis (Chapter 3);
3. Online computation with advice (Chapter 5).

More precisely, concerning the study on the online computation under the recourse model, we studied the online matching problem with edge k-bounded recourse under the edge arrival model. We provided improvements on both upper and lower bounds on the competitive ratio. More specifically, we analyzed two online algorithms for the edge arrival model, namely AMP and L-GREEDY which seem to be incomparable: the former is asymptotically superior, in terms of k, but the latter has a better performance analysis for small k. Unfortunately, there is still a gap between the lower and upper bound on the competitive ratio. The optimal competitive ratio for any fixed $k \geq 4$ still remains unknown.

Concerning the study on other performance measures than standard competitive analysis, we revisited the linear search problem. We introduced the discovery ratio, and apply it as supplementary to the competitive ratio in order to provide a separation on the performance of different competitively optimal strategies. More specifically, we analyzed `doubling` and `aggressive`, which are both competitively optimal. We showed that the strategy `aggressive` has an optimal discovery ratio $\frac{8}{5}$. In contrast, we showed that the strategy `doubling` has discovery ratio $\frac{7}{3}$. We also showed that any competitively optimal strategy that is also optimal with respect to the discovery ratio must have the exact same behavior as the aggressive strategy in the first five iterations. Moreover, we also showed that such competitively optimal strategies with optimal discovery ratio are not unique.

In terms of techniques, we observed that for some problems, it is convenient to formulate them using linear programming. Sometimes, we can look into the structure of the optimal solutions to linear programming to obtain an optimal solution. This motivated the study in Chapter 4, which can be considered as an application of the above idea from Artificial Intelligence. We studied contract scheduling in a model in which the interruptible system is deemed complete once certain performance guarantee has been reached on all problem instances. We designed a schedule that has best-of-both worlds guarantees: it completes the earliest possible and guarantees optimal interruptible performance according to the acceleration ratio.

Concerning the study on online computation with advice, we have studied the online bidding problem under the untrusted advice model. We identified a Pareto optimal strategy assuming that the advice encodes the hidden target.

6.2 What Next?

In this section, we discuss several open questions related to the above topics which we believe to be interesting.

1. *Design a higher lower bound for the online matching problem with recourse in the limited departure model.* In section 2.3, we obtained a lower bound using a two-phases game on strings played between the online algorithm and a specific adversary. We may design a higher lower bound using the same idea.

2. *Use randomization to improve the upper bound analysis for the online matching problem with k edge-recourse.* For $k \geq 4$, there is a gap between the lower bound and upper bound on the competitive ratio. It would be interesting to consider the case for $k \geq 4$, and improve the upper bound by some randomized algorithm. In fact, for $k \geq 4$, we have studied the *Min-Index* framework which has been originally developed by Buchbinder *et al*. [20] for the online standard matching problem. Unfortunately we only got limited results when the underlying graph is a single path. The analysis seems to be hard for a forest graph.

3. *Applying discovery ratio to the star search problem.* As presented in Section 3.1.1, the star search problem can be considered as a generalization of the linear search problem. We believe that our approach used in Chapter 3 can be still applicable. In contrast, it will require more complicated calculations to obtain the result in the context of the star search.

4. *Study the discovery cost in the average case analysis.* As discussed in Section 3.4, for a given total discovered length L, there is no dominance between the performance of `doubling` and `aggressive` using the discovery cost in the worst-case analysis. However, it would be very interesting to study the discovery cost in the average case in order to provide a strong separation on the performance of `doubling` and `aggressive` as illustrated in Figure 3.4. The main technical obstacle is to obtain a close formula of the average discovery cost for a given L and a given strategy.

5. *Study stochastic dominance using discovery cost for the bounded linear search problem.* This problem was presented in Section 3.5. We believe that the approach used in Chapter 3 is still applicable to obtain a stronger conclusion on stochastic dominance using the discovery cost. The main technical obstacle is that we have to consider the edge case very carefully, in which the target is located close to the endpoint of the line.

6. *Variants of contract scheduling with end guarantees.* We can consider a more general model of end guarantees. For instance, each problem instance may be associated with its own end guarantee. We believe that our approach is still applicable. In terms of techniques, some more complicated equations need to be solved, which may be a technical obstacle.

Bibliography

[1] S. Angelopoulos. Further connections between contract-scheduling and ray-searching problems. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1516–1522, 2015.

[2] S. Angelopoulos, D. Arsénio, and C. Dürr. Infinite linear programming and online searching with turn cost. *Theoretical Computer Science*, 670:11–22, 2017.

[3] S. Angelopoulos, C. Dürr, and S. Jin. Online maximum matching with recourse. In *Proceedings of the 43rd International Symposium on Mathematical Foundations of Computer Science (MFCS)*, 2018.

[4] S. Angelopoulos, C. Dürr, and S. Jin. Best-of-two-worlds analysis of online search. In *Proceedings of the 36th International Symposium on Theoretical Aspects of Computer Science (STACS)*, 2019.

[5] S. Angelopoulos, C. Dürr, S. Jin, S. Kamali, and M. Renault. Online computation with untrusted advice. *In Proceedings of the 11th Innovations in Theoretical Computer Science (ITCS)*, 2020.

[6] S. Angelopoulos and S. Jin. Earliest-completion scheduling of contract algorithms with end guarantees. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

[7] Tess Avitabile, Claire Mathieu, and Laura H. Parkinson. Online constrained optimization with recourse. *Information Processing Letters*, 113(3):81–86, 2013.

[8] R. Baeza-Yates, J. Culberson, and G. Rawlins. Searching in the plane. *Information and Computation*, 106:234–244, 1993.

[9] A. Beck. On the linear search problem. *Naval Research Logistics*, 2:221–228, 1964.

[10] A. Beck and D.J. Newman. Yet more on the linear search problem. *Israel Journal of Mathematics*, 8(4):419–429, 1970.

[11] R. Bellman. An optimal search problem. *SIAM Review*, 5:274, 1963.

[12] Aaron Bernstein, Jacob Holm, and Eva Rotenberg. Online bipartite matching with amortized replacements. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA)*, pages 947–959, 2018.

[13] D.S. Bernstein, L. Finkelstein, and S. Zilberstein. Contract algorithms and robots on rays: unifying two scheduling problems. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1211–1217, 2003.

[14] D.S. Bernstein, T. J. Perkins, S. Zilberstein, and L. Finkelstein. Scheduling contract algorithms on multiple processors. In *Proceedings of the 18th AAAI Conference on Artificial Intelligence (AAAI)*, pages 702–706, 2002.

[15] Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, Richard Královič, and Tobias Mömke. On the advice complexity of online problems. In *International Symposium on Algorithms and Computation*, pages 331–340. Springer, 2009.

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

[17] Joan Boyar, Lene M. Favrholdt, Michal Kotrbčík, and Kim S. Larsen. Relaxing the irrevocability requirement for online graph algorithms. In *Proceedings of the 15th Workshop on Algorithms and Data Structures, (WADS)*, pages 217–228, 2017.

[18] Joan Boyar, Lene M Favrholdt, Christian Kudahl, Kim S Larsen, and Jesper W Mikkelsen. Online algorithms with advice: A survey. *ACM Computing Surveys (CSUR)*, 50(2):19, 2017.

[19] Niv Buchbinder, Joseph Seffi Naor, et al. The design of competitive online algorithms via a primal–dual approach. *Foundations and Trends® in Theoretical Computer Science*, 3(2–3):93–263, 2009.

[20] Niv Buchbinder, Danny Segev, and Yevgeny Tkach. Online algorithms for maximum cardinality matching with edge arrivals. In *Proceedings of the 25th Annual European Symposium on Algorithms, (ESA)*, pages 22:1–22:14, 2017.

[21] Ashish Chiplunkar, Sumedh Tirodkar, and Sundar Vishwanathan. On randomized algorithms for matching in the online preemptive model. In *Proceedings of the 23rd Annual European Symposium on Algorithms, (ESA)*, pages 325–336. Springer, 2015.

[22] M. Chrobak and C. Mathieu. Competitiveness via doubling. *SIGACT News*, pages 115–126, 2006.

[23] Marek Chrobak, Claire Kenyon, John Noga, and Neal E Young. Incremental medians via online bidding. *Algorithmica*, 50(4):455–478, 2008.

[24] Marek Chrobak and Claire Kenyon-Mathieu. Sigact news online algorithms column 10: competitiveness via doubling. *ACM SIGACT News*, 37(4):115–126, 2006.

[25] E.D. Demaine, S.P. Fekete, and S. Gal. Online searching with turn cost. *Theoretical Computer Science*, 361:342–355, 2006.

[26] Stefan Dobrev, Rastislav Královič, and Dana Pardubská. Measuring the problem-relevant information in input. *RAIRO-Theoretical Informatics and Applications*, 43(3):585–613, 2009.

[27] R. Dorrigiv and A. López-Ortiz. A survey of performance measures for on-line algorithms. *SIGACT News*, 36(3):67–81, 2005.

[28] Jack Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of research of the National Bureau of Standards B*, 69(125-130):55–56, 1965.

[29] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of mathematics*, 17:449–467, 1965.

[30] Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. Online computation with advice. *Theoretical Computer Science*, 412(24):2642–2656, 2011.

[31] Leah Epstein, Asaf Levin, Danny Segev, and Oren Weimann. Improved bounds for online preemptive matching. In *Proceedings of the 30th International Symposium on Theoretical Aspects of Computer Science, (STACS)*, pages 389–399, 2013.

[32] S. Gal. A general search game. *Israel Journal of Mathematics*, 12:32–45, 1972.

[33] S. Gal. Minimax solutions for linear search problems. *SIAM Journal on Applied Mathematics*, 27:17–30, 1974.

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

[35] Albert Gu, Anupam Gupta, and Amit Kumar. The power of deferral: maintaining a constant-competitive steiner tree online. *SIAM Journal on Computing*, 45(1):1–28, 2016.

[36] Anupam Gupta, Amit Kumar, and Cliff Stein. Maintaining assignments online: Matching, scheduling, and flows. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA)*, pages 468–479, 2014.

[37] Xin Han and Kazuhisa Makino. Online minimization knapsack problem. In *Proceedings of the 7th International Workshop on Approximation and Online Algorithms, (WAOA)*, pages 182–193, 2009.

[38] C. Hipke, C. Icking, R. Klein, and E. Langetepe. How to find a point in the line within a fixed distance. *Discrete Applied Mathematics*, 93:67–73, 1999.

[39] Makoto Imase and Bernard M Waxman. Dynamic steiner tree problem. *SIAM Journal on Discrete Mathematics*, 4(3):369–384, 1991.

[40] Kazuo Iwama and Shiro Taketomi. Removable online knapsack problems. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming, (ICALP)*, pages 293–305, 2002.

[41] P. Jaillet and M. Stafford. Online searching. *Operations Research*, 49:234–244, 1993.

[42] Bala Kalyanasundaram and Kirk Pruhs. Online weighted matching. *Journal of Algorithms*, 14(3):478–488, 1993.

[43] M-Y. Kao and M.L. Littman. Algorithms for informed cows. In *Proceedings of the AAAI 1997 Workshop on Online Search*, 1997.

[44] M-Y. Kao, J.H. Reif, and S.R. Tate. Searching in an unknown environment: an optimal randomized algorithm for the cow-path problem. *Information and Computation*, 131(1):63–80, 1996.

[45] Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, (STOC)*, pages 352–358. ACM, 1990.

[46] D. G. Kirkpatrick. Hyperbolic dovetailing. In *Proceedings of the 17th Annual European Symposium on Algorithms (ESA)*, pages 616–627, 2009.

[47] A. López-Ortiz, S. Angelopoulos, and A.M. Hamel. Optimal scheduling of contract algorithms for anytime problems. *Journal of Artificial Intelligence Research*, 51:533–554, 2014.

[48] A. López-Ortiz and S. Schuierer. The ultimate strategy to search on *m* rays? *Theoretical Computer Science*, 261(2):267–295, 2001.

[49] A. López-Ortiz and S. Schuierer. On-line parallel heuristics, processor scheduling and robot searching under the competitive framework. *Theoretical Computer Science*, 310(1–3):527–537, 2004.

[50] L Lovasz and MD Plummer. Matching theory· new york, 1986.

[51] E.R. Love. Some logarithmic inequalities. *The Mathematical Gazette*, 64(427):55–57, 1980.

[52] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. *arXiv preprint arXiv:1802.05399*, 2018.

[53] A. McGregor, K. Onak, and R. Panigrahy. The oil searching problem. In *Proceedings of the 17th European Symposium on Algorithms (ESA)*, pages 504–515, 2009.

[54] Andrew McGregor. Finding graph matchings in data streams. In *Approximation, Randomization and Combinatorial Optimization, Algorithms and Techniques, (APPROX-RANDOM)*, pages 170–181, 2005.

[55] Nicole Megow, Martin Skutella, José Verschae, and Andreas Wiese. The power of recourse for online MST and TSP. *SIAM Journal on Computing*, 45(3):859–880, 2016.

[56] Aranyak Mehta. Online matching and ad allocation. *Foundations and Trends in Theoretical Computer Science*, 8(4):265–368, 2013.

[57] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. In *Advances in Neural Information Processing Systems*, pages 9661–9670, 2018.

[58] S. J. Russell and S. Zilberstein. Composing real-time systems. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 212–217, 1991.

[59] C.L. Siegel. *Topics in Complex Function Theory, Vol. 1: Elliptic Functions and Uniformization Theory*. New York: Wiley, 1988.

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

[61] Robert Endre Tarjan. Amortized computational complexity. *SIAM Journal on Algebraic Discrete Methods*, 6(2):306–318, 1985.

[62] Ashwinkumar Badanidiyuru Varadaraja. Buyback problem-approximate matroid intersection with cancellation costs. In *Proceedings of the 38th International Colloquium on Automata, Languages, and Programming, (ICALP)*, pages 379–390, 2011.

[63] Vijay V Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

[64] Andrew Chi-Chih Yao. New algorithms for bin packing. *Journal of the ACM (JACM)*, 27(2):207–227, 1980.

[65] Morteza Zadimoghaddam. Online weighted matching: Beating the 1/2 barrier. *CoRR*, abs/1704.05384, 2017. URL: http://arxiv.org/abs/1704.05384.

[66] S. Zilberstein, F. Charpillet, and P. Chassaing. Optimal sequencing of contract algorithms. *Ann. Math. Artif. Intell.*, 39(1-2):1–18, 2003.