Computation under Uncertainty: From Online Algorithms to Search Games and Interruptible Systems

Spyros Angelopoulos

Computation under Uncertainty: From Online Algorithms to Search Games and Interruptible Systems

Spyros Angelopoulos

Habilitation Thesis

Defended on: December 14, 2020

Committee members

Steve Alpern, Warwick Business School
Nikhil Bansal, TU Eindhoven
Christoph Dürr, CNRS and Sorbonne Université
Pierre Fraigniaud, CNRS and Université de Paris
Lisa Hellerstein, New York University
Patrick Jaillet, MIT
Claire Mathieu, CNRS and Université de Paris

Laboratoire d'Informatique de Paris 6
Sorbonne Université
France

Computation under uncertainty: From Online Problems to Search Games and Interruptible Systems

Spyros Angelopoulos

Abstract

This Habilitation thesis explores aspects of computation in a status of uncertainty. First, we study *online* problems, in which the input is not known in advance, and the algorithm must make decisions for each input item without knowledge of future input items. Second, we consider *search games*, in which a mobile searcher must locate an immobile hider that lies in some unknown point of the search domain. Last, we study the design and analysis of *interruptible* systems, in which the system must be able to produce a reasonably efficient output even if interrupted at some arbitrary point in time.

A unifying theme in the study of the above topics is the power and limitations of well-known, worst-case measures such as the *competitive ratio* and the *acceleration ratio*. We argue that in certain cases, more nuanced performance measures are required, and to this end we introduce new models and techniques of analysis. We also demonstrate and exploit connections between these measures and approaches, even though they apply to seemingly unrelated domains.

Resumé

Ce mémoire d'Habilitation explore les aspects de la computation dans un statut d'incertitude. En premier lieu nous étudions les problèmes en-ligne, dans lesquels les données (input) ne sont pas connues à l'avance et où l'algorithme doit prendre des décisions pour chaque donnée en l'absence de connaissance des données apportées par la suite. En deuxième lieu nous considérons les jeux de recherche, dans lesquels un acteur en démarche de recherche doit localiser un acteur caché immobile qui se trouve à un point inconnu du domaine de recherche. En dernier lieu, nous examinons la conception et l'analyse de systèmes interruptibles, dans lesquels le système doit être en capacité de produire un résultat de manière raisonnablement efficace même s'il est interrompu à un instant arbitraire.

Un fil rouge dans l'étude des sujets succinctement présentés ci-dessus est l'examen de la puissance et des limites de mesures de performance dans le cas le plus défavorable, tels le rapport de compétitivité et le rapport d'accélération. Nous avançons que dans certains cas, des mesures de performance plus nuancées s'imposent, et à cet effet nous introduisons de nouveaux modèles et techniques d'analyse. Nous démontrons et exploitons aussi des relations entre ces mesures et ces approches, même si elles s'appliquent en apparence à des domaines distincts.

Contents

1 Introduction **5**
- 1.1 Online computation and competitive analysis . . . 6
  - 1.1.1 The setting . . . 6
  - 1.1.2 Competitive analysis of online algorithms . . . 7
  - 1.1.3 Beyond competitive analysis . . . 8
  - 1.1.4 Online computation with advice . . . 8
  - 1.1.5 Examples of online problems . . . 9
- 1.2 Search games and competitive analysis of online searching . . . 11
  - 1.2.1 The setting . . . 11
  - 1.2.2 Online search and competitive analysis of search strategies . . . 12
  - 1.2.3 Related work . . . 14
- 1.3 Interruptible systems and contract algorithms . . . 15
  - 1.3.1 The setting . . . 15
  - 1.3.2 The acceleration ratio of a schedule . . . 16
  - 1.3.3 Related work . . . 17
- 1.4 Contribution of this thesis . . . 17
  - 1.4.1 Online Computation . . . 17
  - 1.4.2 Online searching . . . 19
  - 1.4.3 Scheduling of contract algorithms and interruptible computation . . . 20

2 New measures and models in online computation **23**
- 2.1 The bijective ratio of online algorithms . . . 23
  - 2.1.1 Background . . . 23
  - 2.1.2 Related work . . . 25
  - 2.1.3 A general framework for establishing bijective optimality . . . 26
  - 2.1.4 Beyond (exact) bijective analysis: The bijective ratio of online algorithms 29
  - 2.1.5 An application: The bijective ratio of k-server problems . . . 29
- 2.2 Online computation with untrusted advice . . . 32
  - 2.2.1 Background . . . 33
  - 2.2.2 A new model for online computation with untrusted advice . . . 33
  - 2.2.3 Applications of the framework . . . 34
  - 2.2.4 Randomized online algorithms with untrusted advice . . . 39
- 2.3 Discussion . . . 39

3 Online search **41**
- 3.1 Online ray search with turn cost . . . 41
  - 3.1.1 Background . . . 41
  - 3.1.2 LP formulations of star search with turn cost . . . 42
  - 3.1.3 Finding a feasible solution to the dual LP . . . 43
  - 3.1.4 Searching with multiple searchers in the presence of turn cost . . . 45
- 3.2 Weighted m-ray search . . . 47

3.2.1 Background . . . 47
3.2.2 The uniform (unweighted) setting . . . 48
3.2.3 The general weighted setting . . . 49
3.3 Bijective analysis of linear search . . . 53
3.3.1 Background . . . 53
3.3.2 Strategies of optimal discovery ratio in Σ . . . 54
3.3.3 Strategies of optimal discovery ratio in Σ_9 . . . 56
3.4 Competitive analysis of expanding search . . . 58
3.4.1 Background . . . 58
3.4.2 Expanding search in a graph . . . 59
3.4.3 Expanding search in a network . . . 64
3.5 Discussion . . . 68

4 Contract scheduling and interruptible algorithms **71**
4.1 Scheduling contract algorithms with end guarantees . . . 71
4.1.1 Background . . . 71
4.1.2 Cyclic schedules and the LP formulation . . . 72
4.1.3 Obtaining an optimal schedule . . . 73
4.1.4 Computational evaluation . . . 75
4.2 New measures for evaluating contract schedules . . . 76
4.2.1 Background . . . 76
4.2.2 Problem formulation and comparison of measures . . . 77
4.2.3 A schedule of small deficiency for general m . . . 80
4.2.4 Lower bounds on the deficiency of a schedule for a single processor . . . 82
4.3 Connections between contract scheduling and ray searching . . . 84
4.3.1 Search with probabilistic detection and scheduling of randomized contract algorithms . . . 85
4.3.2 Randomized scheduling of contract algorithms . . . 87
4.3.3 Further connections . . . 87
4.4 Discussion . . . 89

5 Conclusion **91**
5.1 Online computation . . . 91
5.1.1 Bijective analysis of online algorithms . . . 91
5.1.2 Towards more realistic advice models . . . 91
5.2 Search games . . . 92
5.3 Interruptible algorithms and beyond . . . 92
5.4 Epilogue . . . 92

Chapter 1

Introduction

Computer Science, as we know it today, has been largely shaped through the early advancements in the field of Computational Complexity. The introduction of the *Turing machine* established a simple, yet powerful model of computation, and the *Cook-Levin theorem* and its subsequent applications demonstrated that an entire class of computational problems can be related in terms of hardness. Complexity theory provided the tools for the systematic study of a fundamental question: *How difficult is it to solve efficiently a given computational problem?*

A central concept in complexity theory is that the computational efficiency of an algorithm is determined by the *worst-case* instance. For example, the time complexity of an algorithm is the worst-case time required to solve the problem on a Turing machine; likewise, the space complexity of an algorithm is the worst-case tape length required, as function of the input size.

This concept of worst-case analysis is thus instrumental in complexity theory. It is worth observing that, at least to a certain degree, this "philosophy of analysis" marks a certain departure from other approaches that have been applied in the mathematical analysis of systems. For a notable example, *queuing theory*, which is the mathematical study of waiting lines, has traditionally applied a *probabilistic* approach in the analysis of queuing protocols. Namely, it studies concepts such as the average wait time in a queue, rather than the worst-case wait time.

Subsequently to the successes in the field of computational complexity in the '70s, several new computational models and areas emerged, largely motivated by real-life computational settings. For instance, *parallel and distributed computation* studies systems whose components are located on different networked computers; *online computation* studies settings in which the input is not known to the algorithm prior to its execution, but it is rather revealed in pieces throughout its execution; and the field of *approximation algorithms* studies the hardness of approximating computationally difficult problems.

It is interesting to note that these computational paradigms and areas inherited from complexity theory, and adapted accordingly, the concept of worst-case analysis. That is, the performance of a parallel, or distributed, or online, or approximation algorithm is evaluated on a worst-case, i.e, *adversarial* instance. A rather striking example of this analysis approach is the study of queuing systems in a worst-case setting (see the adversarial queuing theory paradigm [45]).

A basic tenet of worst-case analysis is relating the performance of the algorithm to an *ideal* solution. For instance, in approximation algorithms, the performance of the algorithm is compared to the optimal solution for the given instance, even (and especially if) the problem is NP-hard. For a different example, in online computation, the performance of the online algorithm is compared to an ideal algorithm which, unlike the online algorithm, has foreknowledge of the input.

At an intuitive level, there are two main aspects of worst-case analysis that justify its prevalence, and its success. First, it leads to strict, ubiquitous performance guarantees that do not rely to any assumptions or probabilistic properties. Second, and perhaps equally importantly, it is a clean-cut analysis approach to which a plethora of computational problems are amenable,

as amply demonstrated by the successes in the fields mentioned above.

Despite these appealing properties, it should be evident, even at an intuitive level, that worst-case analysis is not necessarily a panacea, especially when expressed via the "intermediary" that corresponds to the ideal solution. To illustrate this with an example, suppose that we would like to compare the football teams of two schools. One possibility, and probably the most sensible, is to have the two teams compete against each other. Another possibility is to have each team play against an "ideal team" (think of your favorite Champions League cup holder), and compare the school teams based on their individual performance. Arguably, the latter may not a good idea, since the two teams would probably perform equally bad with very little, if any, margin for drawing any useful conclusions. The example illustrates some pitfalls of worst-case analysis, and acts as a warning that more nuanced approaches may be required.

In this Habilitation thesis we study computational problems under uncertainty, with an emphasis on established and alternate worst-case measures of algorithmic performance. In this class of problems, the algorithm has to operate with limited information concerning the input or the problem setting. We will focus on three broad classes of such problems. We begin with *online computation*, in which the standard worst-case analysis is *competitive analysis*, and the corresponding worst-case measure the *competitive ratio* (Section 1.1). Next, we study *search games*, in which a mobile searcher has to locate, in an efficient manner, an immobile target that hides in some unknown position of the search domain (Section 1.2). Here, competitive analysis can been applied as a way for evaluating the performance of a search strategy relative to an ideal searcher that knows the precise hiding position of the target. The last part pertains to *anytime* computation, in which we are interested in designing algorithms that perform well even when interrupted during their execution (Section 1.3). Here, the associated worst case measure relates the performance of the interruptible algorithm to an ideal one that knows the precise time the interruption will occur.

An underlying theme that characterizes our contribution in the above topics is that, as we will argue later, worst-case analysis sometimes suffers from certain deficiencies. We remedy this by proposing new measures and analysis techniques, which are either applied complementary to the known measures, or entirely on their own. These measures and approaches have certain similarities that we exploit, despite the fact that they are applied in seemingly unrelated fields. In certain cases, we go one step further: we show that a certain model does not capture adequately a given setting or requirement, and propose a new model of study; or we show that a certain proof technique has led to erroneous results, and give a correct proof. A detailed list of the contribution of this thesis is given in Section 1.4.

1.1 Online computation and competitive analysis

1.1.1 The setting

Decisions in the real world usually have to be made in a status of incomplete information. A similar situation arises in many applications in computer science: an algorithm may not have complete knowledge of the environment, but we still require that it performs as efficiently as possible. Consider, for instance, an algorithm which has to act on a sequence of "events", however it knows only the events that occurred in the past. Can one design algorithms that are competitive with an omniscient agent that knows the entire future?

Online computation is a paradigm that applies precisely in settings in which the input to the algorithm is not known ahead of its execution, but is rather revealed incrementally. More formally, the input to the algorithm consists of a sequence of *requests*. Once a request is revealed to the algorithm, it must act on it, i.e., the algorithm must *serve* the request. This action comes with two constraints: First, it is *irrevocable*, in that the algorithm cannot revoke its action on a past request while serving the current request. Second, the model assumes that the algorithm has no knowledge about future requests, while serving the current request.

A typical online problem is often a combinatorial optimization problem in which the input is not known in advance. Hence, there is a concept of a *value* associated with the actions of the online algorithm. More specifically, in a cost minimization problem, serving a request incurs a *cost*, whereas in a profit maximization problem, serving a request accrues a *benefit*.

Example 1. *Consider the well-known* paging *or* cache replacement problem. *A fast cache of size k mediates between the CPU and the slow memory. Each of the k cache entries can store a single page, and suppose there are $N \gg k$ pages in total. Each request in the request sequence consists of a single page. An online algorithm for this problem must serve a request to a page p as follows. If p is in the cache, no action needs to be taken, and no cost is incurred. Otherwise, the algorithm must decide which page to evict from the cache, so as to bring in p. This action is done at unit cost. Therefore, the overall cost of the online algorithm on a given request sequence is the total number of page evictions for serving the sequence.*

It is not surprising that online algorithms have been a topic of study, implicitly and explicitly, for almost forty years: many computational problems are intrinsically online. The book [44] provides an in-depth analysis of advances in this field[1]. Equally importantly, online computation has had an impact in several domains of computer science such as machine learning [57], networks [56], and databases [77], for some representative examples.

1.1.2 Competitive analysis of online algorithms

How does one analyze and compare the performance of online algorithms? The standard method of such evaluation is by means of a worst-case measure, known as the *competitive ratio*, which was introduced in the seminal work of Sleator and Tarjan [125]. In words, the competitive ratio of an online algorithm is the worst-case ratio (among all possible request sequences) of the performance of the online algorithm (e.g., its cost in case of a minimization problem) over the corresponding performance of the best offline algorithm that has advance knowledge of the input. More formally, we have the following definition.

Definition 2. *Given a cost-minimization problem and a deterministic online algorithm A, the (strict) competitive ratio of A is defined by the quantity*

$$cr(A) = \sup_{\sigma} \frac{A(\sigma)}{\textsc{Opt}(\sigma)},$$

where $A(\sigma)$ denotes the cost of A on request sequence σ, and $\textsc{Opt}(\sigma)$ denotes the optimal cost for serving σ in an offline manner (i.e., assuming knowledge of σ).

A similar definition can apply to a profit-maximization online problem. The competitive ratio of an online problem is defined as the best competitive ratio achieved by an online algorithm for the said problem.

Occasionally, a slightly weaker version of Definition 2 is used, notably when the online problem cannot admit any efficient online algorithm according to the strict competitive ratio. More precisely, we say that algorithm A has *asymptotic* competitive ratio r if on every sequence σ it holds that $A(\sigma) \leq r \cdot \textsc{Opt}(\sigma) + c$, where c is a constant. Unless explicitly mentioned, any reference to "competitive ratio" in this thesis applies to the strict competitive ratio.

Definition 2 can be extended, in a natural way, to *randomized* online algorithms. Namely, it suffices to replace the algorithm's cost $A(\sigma)$ (which is now a random variable), with its expected cost $\mathbb{E}(\sigma)$, where the expectation is over the random choices of the algorithm.

The framework of competitive analysis has proven itself extremely fruitful in the development of online computation as an important field of theoretical computer science, and hundreds of

[1] There is a new book in preparation on online algorithms by Borodin and Pankratov, which is a testament to the ever increasing interest in online algorithms.

online problem have been studied under the competitiveness lens. This is due to the amenability of most online problems to this type of worst-case analysis. To be more specific, on the one hand, positive results, i.e., *upper bounds* on the competitive ratio, typically follow from combinatorial analysis that involves the worst-case behavior of a given online algorithm relative to the optimal solution. On the other hand, negative results, i.e., *lower bounds* on the competitive ratio, follow from a suitably constructed *two-person game* between the algorithm and an *adversary*. While such a game is conceptually simple to define for deterministic online algorithms, it is more intricate in the context of randomized online algorithms, and special care must be given to appropriately defining the power of the adversary [37].

1.1.3 Beyond competitive analysis

Notwithstanding the undeniable success of competitive analysis, certain drawbacks have long been known. Most notably, due to its pessimistic (i.e., worst-case) evaluation of algorithms, it often fails to distinguish between algorithms for which experimental evidence (or even plain intuition) would show significant differences in terms of performance. One definitive illustration of this undesirable situation is the paging problem, as defined in Example 1. In particular, strategies which are very efficient in practice, such as Least-Recently-Used and strategies that are naive and inefficient, such as Flush-When-Full are both optimal and thus indistinguishable using competitive analysis [125].

Generally speaking, competitive analysis is particularly meaningful when the obtained competitive ratios are small; however, when the competitive ratios are large (and even more so, in the case of an unbounded ratio) it risks no longer reflecting what is observed in practice. In these cases, a finer measure may very well be required. This disconnect between the empirical and the theoretical performance evaluation motivated a substantial line of research on measures alternative to the competitive ratio. Some of the known approaches are: the *Max-Max ratio* [36]; the *diffuse adversary model* [102, 129, 130]; *loose competitiveness* [128, 131]; the *random order ratio* [99]; the *relative worst-order ratio* [50, 48, 51]; the *accommodation function model* [55]; and *stochastic dominance* as well as *bijective and average analysis* [84, 58, 59, 123, 114, 108, 13, 81, 12, 25]. We refer the reader to the surveys [69, 83] for an in-depth discussion of the above techniques.

1.1.4 Online computation with advice

The standard model of online computation assumes that the online algorithm has no information concerning the requests it will have to serve in the future. In practice, however, online algorithms are often provided with some (limited) knowledge of the input. For example, the algorithm may have some lookahead on some of the upcoming requests, or it may know the size of the request sequence, or some specific characteristics of certain requests. While competitive analysis is still applicable, especially from the point of view of the analysis of a known, given algorithm, a new model was required to formally quantify the power and limitations of offline information.

The term *advice complexity* was first coined by Dobrev *et al.* [68], and subsequent formal models were presented by Böckenhauer *et al.* [41] and Emek *et al.* [70], with this goal in mind. More precisely, in the advice setting, the online algorithm receives some bits that encode information concerning the sequence of input items. As expected, this additional information can boost the performance of the algorithm, which is often reflected in better competitive ratios.

Under the current models, the advice bits can encode any information about the input sequence; indeed, defining the "right" information to be given to the algorithm plays an important role in obtaining better online algorithms. Clearly, the performance of the online algorithm can only improve by increasing the size of the advice. The objective is thus to identify trade-offs between the size of the advice and the performance of the algorithm. This is meant to provide a smooth transition between the purely online world (nothing is known about the input) and the purely "offline" world (everything is known about the input).

While a certain distinction can be made between several advice models (e.g, the advice may be given by an oracle either "per request", or rather "as a whole"), in this thesis, we focus on the *tape* model of advice [42], in which the algorithm has access to an advice string that is communicated by an oracle with access to the complete input. The advice complexity is precisely the length of this communication. More specifically, we have the following definition.

Definition 3. *We say that an online problem is c-competitive with advice of size $f(n)$, where f is a given function, and n is the size (i.e., length) of the request sequence, if there is a c-competitive algorithm for the problem with advice tape of size at most $f(n)$.*

In the last decade, a substantial number of online optimization problems have been studied in the advice model; we refer the reader to the survey of Boyar *et al.* [49] for an in-depth discussion of developments in this field.

1.1.5 Examples of online problems

In this section we present some important examples of online problems that are of interest in this thesis. We also survey some known results on the performance of online algorithms for these problems. In order to keep this section self-contained (and not overly long), we only survey results that apply to the standard model in which the algorithm has no advice on the input. In Chapter 2 we will come back to some of these problems, and address issues related to their advice complexity.

k-server

The k-server problem, originally proposed by Manasse et al. [110], involves k mobile servers over a metric space. Upon a request to a point in this space, a server must be moved to it. The incurred cost is the overall distance traversed by the servers. The k-server problem generalizes the paging problem and has motivated an outstanding body of research (see the surveys [63] and [101]). In general metric spaces, and in one of the most celebrated results in online computation, the Work Function Algorithm (WFA) of Koutsoupias and Papadimitriou is $(2k-1)$-competitive [103]; moreover, the best-known lower bound on the deterministic competitive ratio is k [110]. WFA is also known to be k-competitive on the line [32] as well as for two servers in general metric spaces [63], and it is also the best known algorithm for the circle metric [44]. Chrobak and Larmore showed that the algorithm DOUBLE COVERAGE, which moves certain servers at the same speed in the direction of the request until a server reaches the requested point, is k-competitive for the tree metric [62].

Weighted paging

In the standard (uniform) paging problem, each page is associated with the same cache eviction cost. Weighted paging is a generalization of the standard paging problem in which each page p is associated with an eviction cost c_p. This problem has generated an impressive body of work from the point of view of competitive analysis (see e.g., [60, 127, 31] and references therein). It is well known that the weighted paging problem for a cache of size k is equivalent to the k-server problem in a *discrete star graph*, assuming there are no requests to the center node of the star. More precisely, the star has as many edges as pages, and the weight of each edge is equal to half the eviction cost of the corresponding page; last, requests may appear on any leaf of the star.

For weighted paging, a tight k-competitive deterministic algorithm follows from the more general work of Chrobak *et al.* [62] for the k-server problem on trees. The randomized version of the problem has a much more fascinating history of research, which culminates with a randomized $O(\log k)$-competitive algorithm due to Bansal, Buchbinder and Naor [31].

Online reordering buffer management

In this online problem, the request sequence consists of points in a metric space, which have to be processed by a single server. The server comes with a buffer of size (capacity) k, and must decide, at each point in time, which request from its buffer to serve next. Serving a request entails moving the server to the location of the request. The objective is to minimize the total distance that the server has traveled within the metric space throughout serving the requests. This problem was introduced by Räcke et al. [116], and can model context switching costs that occur in various applications ranging from production engineering to disk management.

For this problem, Räcke et al. [116] gave a $O(\log^2 k)$-competitive algorithm for uniform metric spaces, which was subsequently improved to $O(\log k)$ by Englert and Westermann [71], and for general, non-uniform metrics. Avigdor-Elgrabli and Rabani [27] improved this bound to $O(\log k/\log\log k)$, which was in turn improved to a guarantee of $O(\sqrt{\log k})$ by Adamaszek *et al.* [1]. The latter is close to optimal due to a lower bound of $\Omega(\sqrt{\log k/\log\log k})$ shown in the same paper. All of the above results apply to deterministic algorithms.

Ski rental

The ski rental problem is a canonical example in online rent-or-buy problems. Here, the request sequence can be seen as vacation days, and on each day the vacationer (that is, the online algorithm) must decide whether to continue renting skis, or buy them. Without loss of generality we assume that renting costs a unit per day, and buying costs $B \in \mathbb{N}^+$. The number of days is unknown to the online algorithm. Generalizations of ski rental have been applied in many settings, such as dynamic TCP acknowledgment [95], the parking permit problem [113], and snoopy caching [94]. A simple online algorithm that rents until day $B-1$, and buys on day B achieves an optimal deterministic competitive ratio equal to $2-1/B$, whereas a more involved randomized strategy achieves optimal competitive ratio equal to $e/(e-1)$ [96].

Online bidding

In the online bidding problem, a player wants to guess a hidden, unknown value u, with $u \geq 1$. To this end, the player submits a sequence $X = (x_i)$ of positive, increasing *bids*, until one of them is at least u. The strategy of the player is defined by this sequence of bids, and the cost of guessing the hidden value u is equal to $\sum_{i=1}^{j} x_i$, where j is such that $x_{j-1} < u \leq x_j$. Hence the following natural definition of the competitive ratio of the bidder's strategy.

$$\mathrm{cr}(X) = \sup_u \frac{\sum_{i=1}^{j} x_i}{u}, \text{ where } j \text{ is such that } x_{j-1} < u \leq x_j.$$

The problem was introduced in [61] as a canonical problem for formalizing doubling-based strategies in online and offline optimization problems. A simple doubling strategy in which $x_i = 2^i$ yields an optimal deterministic competitive ratio equal to 4, whereas a more complex randomized strategy achieves an optimal competitive ratio of $e/(e-1)$.

It is worth noting that online bidding is essentially equivalent to the *linear search* problem (informally known as the *cow-path problem*), as we will see in Section 1.2. Moreover, it is identical to the problem of minimizing the *acceleration ratio of interruptible algorithms*, in its simplest setting, as we will discuss in Section 1.3.

Bin packing

This is the online variant of one of the first combinatorial optimization problems ever studied from the point of view of approximation algorithms; in fact, online bin packing was studied under worst-case measures even before competitive analysis was formally introduced in [125].

The problem models several applications [90, 64], and as such is of both theoretical and practical significance.

An instance of the online bin packing problem consists of a sequence of items with different *sizes* in the range $(0, 1]$, and the objective is to pack these items into a minimum number of *bins* of unit capacity. For each arriving item, the algorithm must either place it in one of the current bins, or open a new bin for the item. By the nature of the bin packing problem, it cannot be avoided that a constant number of bins are not well filled. Hence, it is standard practice to measure the performance of an online algorithm by means of the asymptotic competitive ratio.

A straightforward algorithm for online bin packing is First Fit. This algorithm maintains bins in the same order that they have been opened, and places an item into the first available bin with enough free space; if no such bin exists, it opens a new bin. Johnson [89] proved that the competitive ratio of First Fit is 1.7. Since this result, many other algorithms with improved competitive ratios have been studied. The best known algorithm was recently proposed by Balogh *et al.* [29] and has a competitive ratio of at most 1.5783. Moreover, it is known that no online algorithm can achieve a competitive ratio better than 1.54278 [30].

List update

The list update problem is one of the fundamental online problems, and among the first to be studied under the framework of competitive analysis [125]. The problem setting consists of a list of items, and a sequence of requests that must be served. Each request corresponds to an "access" to an item in the list. If the item is at position i of the list, then its access cost is i. After accessing the item, the online algorithm can move it closer to the front of the list at no cost: this operation is called a *free exchange*. In addition, at any point in time, the online algorithm can choose to swap any two consecutive items in the list: this operation is called a *paid exchange*, and incurs unit cost.

Move-To-Front (Mtf) is an algorithm that moves every accessed item to the front of the list using a free exchange. Sleator and Tarjan [125] proved that Mtf has a competitive ratio equal to 2. This ratio turns out to be the best that a deterministic algorithm can achieve [86]. Timestamp, introduced by Albers [2], is another algorithm that achieves the optimal competitive ratio of 2. This algorithm uses a free exchange to move an accessed item x to the front of the first item that has been accessed at most once since the last access to x. Another class of algorithms are Move-To-Front-Every-Other-Access (Mtf2) algorithms that move the accessed item to the front of the list on every other access. More precisely, these algorithms maintain a bit for each item in the list. Upon accessing an item x, the bit of x is flipped, and x is moved to front if its bit is 0 after the flip (otherwise the list is not updated). If all bits are initialized to 0 at the beginning of the algorithm's execution, Mtf2 is called called Move-To-Front-Even (Mtfe), otherwise, Mtf2 is called Move-To-Front-Odd (Mtfo). Both Mtfe and Mtfo algorithms have a competitive ratio of 2.5 [53]. Boyar et al [53] showed that, for any request sequence, one among Timestamp, Mtfo, and Mtfe must have a competitive ratio of at most 5/3, which, as we will see later, is very useful in the design of list update algorithms with advice.

1.2 Search games and competitive analysis of online searching

1.2.1 The setting

Searching for a target is a common task in everyday life, and an important computational problem with numerous applications. Problems involving search arise in such diverse areas as drilling for oil in multiple sites, the forest service looking for missing backpackers, search-and-rescue operations in the open seas, and navigating a robot between two points on the Mars terrain. All these problems involve a mobile *searcher* which must locate an immobile *target*, often also called *hider*, that lies in some unknown point in the *search domain*, i.e, the environment in which the

search takes place. The searcher starts from some initial placement within the domain, which we call the *origin*, or *root*. There is, also, some underlying concept of quality of search, in the sense that we wish, in informal terms, for the searcher to be able to locate the target as efficiently as possible. The *cost* incurred by the searcher expresses precisely the "effort" of the searcher, and is defined appropriately depending on the setting at hand. For example, the cost can be defined as the total distance traversed by the searcher until it reaches the hider.

Search theory is one of the original disciplines within the field of Operations Research. Typically, search problems are amenable to a game-theoretic approach, in which the searcher and the hider are players in a two-person zero-sum game over either pure or mixed strategies. More precisely, the searcher's strategy space consists of all possible trajectories it can follow in the search domain, whereas the hider's strategy space consists of all points in which it can hide. Given specific searcher and hider strategies, the cost assigned to the searcher is the time it takes to locate the hider, and can be seen as the game-theoretic payoff to the maximizing hider. The value of the game is then defined naturally as the minimax value at equilibrium. This game-theoretic framework offers a simple, yet very inclusive abstraction of search games that has led to an impressive body of work (see, e.g., the book [6], and more recently [5]).

It is worth noting that the Operations Research (OR) and the Theoretical Computer Science (TCS) communities follow somewhat different approaches on these problems. In OR, the emphasis is typically on the exact value of the game, usually for mixed strategies, and on a case-by-case analysis for a variety of domains (e.g., the infinite line, star graphs, trees, general networks). The asymptotic analysis and approximations of the value of the game tend to be of secondary importance. Thus, the problem may be deemed "unsolved" if the exact value of the game is yet unknown, and algorithmic considerations are often ignored, i.e., the searcher's optimal trajectories may be computationally hard to find. In contrast, in TCS, the focus is on explicit polynomial-time strategies, and for NP-hard search problems, approximation algorithms are sought. Asymptotic analysis is often the analysis of choice when it is hard to evaluate the exact value of the game.

1.2.2 Online search and competitive analysis of search strategies

Consider the case in which the search domain is *unbounded*. For a simple example, suppose that the domain consists of an infinite line that is unbounded both to the left and to the right of the origin. In this simple example, the search game has no equilibrium, since the hider can choose to hide arbitrarily far from the origin. To make the setting amendable to some kind of analysis, Beck and Newman [35] introduced the concept of *normalized* cost. More precisely, given a search strategy S, and a hider h, let $c(S, h)$ denote the total cost required for the searcher to locate h, and let $d(h)$ denote the distance of h from the origin. Beck and Newman defined the *normalized cost* $\hat{c}(S, h)$ as the ratio

$$\hat{c}(S, h) = \frac{c(S, h)}{d(h)},$$

and argued that for this payoff function, one can find both pure and mixed equilibria in the corresponding search game.

It is interesting to note that the definition of normalized cost is conceptually similar to a worst-case measure such as the competitive ratio. The numerator in the definition of the normalized cost is the cost of the strategy (i.e., the algorithm) whereas the denominator can be interpreted as the "optimal" cost for locating the hider, assuming that we know the precise position of the hider in the search domain. We can thus extend the definition of competitive analysis to the analysis of search games[2].

[2]It is worth noting that searching is not, *per se*, an online problem: there is no request sequence associated with the problem. Nevertheless, the term "competitive analysis of search games" seems to have been adopted without reservations by both the OR and the TCS communities.

Definition 4. *The competitive ratio of a (pure) search strategy S is defined as*

$$\sup_h \hat{c}(S, h).$$

A search strategy is called optimal *if it has minimum competitive ratio. The competitive ratio of an optimal strategy is defined as the competitive ratio of the search game.*

As in competitive analysis of online algorithms, the framework allows for both deterministic (pure) and randomized (mixed) strategies. In the latter case, it suffices to replace the numerator of the normalized cost with the *expected* cost incurred by the searcher. One can thus refer to the *deterministic* or the *randomized* competitive ratio of a search strategy.

We will be making a standard assumption in the field concerning the hider's position: namely, that it cannot hide arbitrarily close to the origin, but at least to some (unit) distance away from it. Otherwise, every strategy can have unbounded competitive ratio.

Example 5. *Consider the setting in which the search domain consists of an infinite line that is unbounded both to the left and to the right of the origin (we refer to the two sides of direction as the "0" and the "1" side). What are the optimal deterministic and randomized competitive ratios? This problem is known as the* linear search *problem, or informally* the cow path *problem.*

For this problem, Beck and Neuman [35] showed an optimal deterministic competitive ratio equal to 9, which is achieved by a simple doubling *strategy as follows. In iteration i, the searcher will start from the origin, traverse side $i \pmod 2$ to a length equal to 2^i, and return to the origin. A more complex randomized strategy shown and analyzed by Kao* et al *[93] has optimal randomized competitive ratio approximately equal to 4.5911.*

It is worth noting that linear search is *competitively equivalent* to the online bidding problem defined in Section 1.1.5. In particular, any strategy for online bidding that has competitive ratio C can be translated to a search strategy for linear search of competitive ratio $1 + 2C$, and vice versa.

Example 6. *Consider the setting in which the search domain consists of m semi-infinite, concurrent* rays *which intersect at a common origin O. This setting is known as the m-*ray search *or* star *search problem and generalizes linear search, since linear search is identical to 2-ray search. A mobile searcher, initially placed at O, can navigate through the star at unit speed. There is also a target that is hidden at some distance $d(h)$ from O, at a ray unknown to the searcher. A* search strategy *S determines, at every point in time at which the searcher is at the origin, a ray r and a depth l_r, such that the searcher will traverse r to depth at most l_r, and unless the target has been found, will return to O. Figure 1.1 illustrates the setting.*

Gal [73, 74] gave an optimal deterministic strategy for m-ray search, a result that was later rediscovered by Baeza-Yates et al. *[28]. Consider an arbitrary indexing of rays with number $0, 1, \ldots m-1$. In Gal's optimal strategy, in iteration i, the searcher starting from the origin traverses ray $i \pmod m$ to a length equal to $(m/(m-1))^i$, and returns to the origin. The resulting optimal competitive ratio is equal to*

$$1 + 2\frac{m^m}{(m-1)^{m-1}}.$$

Linear search, and its generalization, the star search, are among the most well-studied problems in search theory (see, for example, Chapters 7 and 9 in [5]). This is because star search has applications that are not necessarily confined to the concept of locating a target, which explains its significance and popularity. Indeed, star search offers an abstraction that applies naturally in settings in which we seek an intelligent allocation of resources to tasks. More precisely, it captures decision-making aspects when the objective is to successfully complete at least one

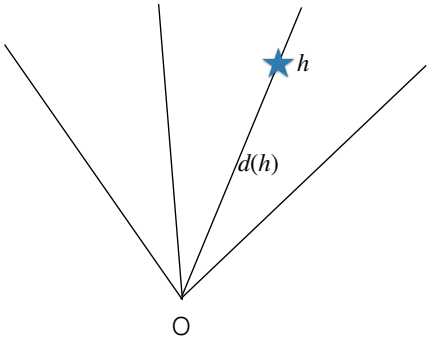


Figure 1.1: Illustration of ray search in a star of $m = 4$ rays.

task, without knowing in advance the completion time of each task. Some concrete applications include: drilling for oil in a number of different locations in [112]; the design of efficient *interruptible* algorithms, i.e., algorithms that return acceptable solutions even if interrupted during their execution (as we will discuss in Chapter 4); and database query optimization (in particular, pipelined filter ordering in [65]).

1.2.3 Related work

Optimal strategies for linear search under the (deterministic) competitive ratio were first given in [35]. As mentioned in Example 6, [74] gave optimal strategies for the generalized problem of ray searching, and the optimal competitive ratio can be written equivalently as

$$1 + 2M, \text{where } M = \frac{b^m}{b-1} \text{ and } b = \frac{m}{m-1}. \tag{1.1}$$

Gal's optimal strategy is an example of a *cyclic* strategy, namely a strategy that visit the rays in a round-robin fashion, according to a fixed permutation of the m rays. It is also an example of a *geometric* or *exponential* strategy, in that it is a cyclic strategy which in the i-th iteration traverses the corresponding ray to a length equal to b^i, for some appropriately chosen b that is called the *base*. For some (but not all) of the problems we will study, cyclic and geometric strategies are significant since they lead to optimal strategies, as in the case of the simplest variant of m-ray search.

Other related work includes the study of randomization [122] and [93]; multi-searcher strategies [107]; searching with turn cost [66]; the variant in which some probabilistic information on target placement is known [88, 91]; and the related problem of designing *hybrid algorithms* by [92].

Bounded star search, namely the case in which an upper bound is known on the distance of the target from the root was studied in [106] and [47]. New performance measures that are applicable in the context of multi-target searching were introduced in [100] and [112] (i.e., the setting in which there are multiple hiders and the searcher must locate one of them).

Competitive analysis has been applied beyond the linear and star search, for example in searching within a graph [104, 72, 17]. In particular, Koutsoupias *et al.* [104] showed that minimizing the competitive ratio is NP-hard for general graphs, and gave strategies that achieve an $O(1)$-approximation of the optimal deterministic and randomized competitive ratio.

1.3 Interruptible systems and contract algorithms

1.3.1 The setting

One of the central objectives in the design of intelligent systems is the provision for *anytime* capabilities. In particular, several applications such as medical diagnostic systems and motion planning algorithms require that the system outputs a reasonably efficient solution even if interrupted during its execution. Hence the following problem arises: can we transform a given algorithm to its interruptible equivalent, without compromising the algorithm's performance?

Anytime algorithms are algorithms whose quality of output improves gradually as the amount of available computation time increases. This class of algorithms was introduced first by Dean and Boddy [43] in the context of time-depending planning, as well as by Horvitz [85] in the context of flexible computation. Anytime algorithms occur naturally in settings where a computationally intensive problem is addressed under uncertainty with respect to the available computation time. An example of this is a problem in which the answer must be provided when determined by an external input over which we have no control. For instance, consider an automated trading program for the stock market. These programs run time-intensive simulations to price various financial instruments. When a change in the bid price of a given stock occurs, the algorithm must produce a decision (buy/sell/hold) at that very instant, using whatever information it had garnished over the course of the simulations, so as to take advantage of the newly posted price. Another example is given by real-time applications. For instance, consider a motion planning algorithm for a robot in which a solution must be produced within a certain, but varying, amount of time: for example, if a robot is about to collide, a move is required instantaneously even if the algorithm is to produce a suboptimal move, while in others, there is sufficient time to carefully compute the next step. In this situation, the amount of allotted time is given to the algorithm beforehand.

According to Russell and Zilberstein [121], a further distinction can be made between two different types of anytime algorithms. On the one hand, *interruptible* algorithms are algorithms whose allowable running time is not known in advance, and thus can be interrupted (queried) at any given point throughout their execution. A typical example is algorithms based on local search, e.g., simulated annealing and hill climbing. On the other hand, the class of *contract* algorithms consists of algorithms which are given the amount of allowable computation time (i.e, the intended query time) as part of the input. However, if the algorithm is interrupted at any point before the contract time expires, the algorithm may not return a meaningful output. A typical example of contract algorithms is polynomial-time approximation schemes (PTAS) based on Dynamic Programming (DP): the larger the amount of computation time, the better the achieved approximation ratio. However, the algorithm may require all the available computation time in order to fill in the important entries of the DP table, otherwise the solution returned may be useless.

Although contract algorithms are obviously less appealing than interruptible algorithms, due to their lack of interruptible capabilities, they still can be quite useful on their own. For example, they often use simpler data structures, and thus tend to be easier to implement and maintain than interruptible algorithms [38]. Thus the following questions arise: Given a contract algorithm for a computational problem, can we transform it to an interruptible one? And how can one quantify the performance of the resulting interruptible algorithm.

Let us begin with addressing the first question. Here, one could opt for an *ad-hoc* method that is problem-specific. Another, and far more general technique is based on repeated executions of the contract algorithms with increasing execution times. This technique was first given in [121], and is based on iteratively doubling the available execution times (i.e., the contract lengths) of the contract algorithm. This can be described as a *schedule* of executions of the same algorithm in which the i-th execution is run for a time equal to 2^i. In this schedule, if an interruption occurs at time t, it is guaranteed that a contract of length at least $t/4$ has completed its execution, in

worst-case.

A generalization of the above setting was studied in [133] in which n different *instances* of an optimization problem are given, along with the statement of a contract algorithm for the problem. The objective is to design a schedule of interleaved executions of the contract algorithm for each instance. More formally, we consider the following setting. We assume a single processor and n *problem instances* or simply *instances*, numbered $0, \ldots, n-1$. A schedule X can be described as a sequence of the form $((x_i, p_i))_i$, meaning that the i-th scheduled contract in X has length x_i and is assigned to problem $p_i \in \{0, \ldots, n-1\}$. Suppose that an interruption takes place at time t, and let $\ell_{p,t}(X)$ denote the length (duration) of the longest execution of a contract algorithm for problem p that has completed by time t in X. Intuitively, this quantity $\ell_{p,t}(X)$ describes the "progress" that the interruptible system (i.e., the schedule X) has achieved by time t.

Example 7. *Suppose that we are given two problem instances, i.e., $n = 2$, and consider the schedule of Figure 1.2. The contracts for each instance are depicted in light (yellow) and dark (blue) colors, respectively, and the associated numbers describe their lengths. For an interruption that occurs at time $t = 12$, the largest contracts completed for the two problem instances have lengths 4 and 2, respectively.*

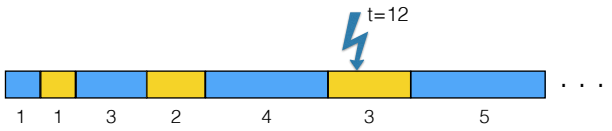


Figure 1.2: Illustration of a schedule of contracts for two problem instances.

We now move on to the second question. Given a schedule X, how can we evaluate its efficiency? We address this question in the following section.

1.3.2 The acceleration ratio of a schedule

Let us go back to the example of a single problem instance ($n = 1$) and a schedule of contract lengths described by the sequence $x_i = 2^i$. As we observed earlier, there is a multiplicative gap of at most 4 between the interruption t and the largest contract length completed by time t, and this is the optimal such gap, i.e, no other schedule can reduce this gap [133].

There is a natural interpretation to this quantity. It implies that an increase of the processor speed (in which the schedule is run) by a factor equal to 4 yields a system which is as efficient as one in which the interruption time is known in advance. Russell and Zilberstein refer thus to this measure as the *acceleration ratio* of the schedule.

For a formal definition, let us consider the general setting that involves n problem instances. Then the acceleration ratio of a schedule X [121], denoted by acc (X), is defined as

$$\mathrm{acc}(X) = \sup_t \max_p \frac{t}{\ell_{p,t}(X)}, \tag{1.2}$$

where recall that $\ell_{p,t}(X)$ is defined as the length of the largest contract for instance p that has completed in X by time t.

There are a few observations that we need to make. First, note that the interruption t can be arbitrarily small, then no schedule can have finite acceleration ratio. Hence we will be making the usual assumption in the field that t is at least 1. Second, all previous work on the acceleration ratio of a schedule assumes that t can be arbitrarily large, in that an interruption may appear arbitrarily far ahead in the future. This also implies that, as far as the theoretical analysis is concerned, the schedule is supposed to be *infinite*.

Furthermore, it needs to be emphasized that the acceleration ratio is a resource augmentation measure that relates processor speed to clairvoyance with respect to interruptions. It does not imply, in general, any further strict guarantees on the performance of the interruptible system. For example, if the contract algorithm is an approximation algorithm for an NP-hard problem, the acceleration ratio does not necessarily provide any useful information concerning the approximation ratio upon interruption.

Last, we observe that for a single problem instance $n = 1$, the problem is identical to online bidding (defined in Section 1.1.5) and essentially equivalent to the linear search problem (defined in Section 1.2). More precisely, a C competitive algorithm for online bidding gives rise to a schedule of the same acceleration ratio and vice versa, and a schedule with acceleration ratio C translates to strategy for linear search that has competitive ratio $1 + 2C$, and vice versa.

1.3.3 Related work

Contract scheduling has been studied in a variety of settings. Optimal schedules for a single processor and multiple instances were given in [133]. In particular, for n problem instances, there is a simple, optimal, cyclic strategy, in which the i-th contract in the schedule is for problem $i \pmod n$, and has length $(n + 1)/n)^i$. This optimal acceleration ratio is equal to

$$\frac{b^{n+1}}{b-1}, \text{ with } b = \frac{n+1}{n}. \tag{1.3}$$

Note that cyclic and geometric (or exponential) schedules can be defined along the same lines as cyclic and geometric strategies in ray search.

A more general setting involves scheduling the executions of contract algorithms in *multiple* processors, instead of a single one. Schedules of optimal acceleration ratio for a single instance and multiple processors were shown in [39]. The most general setting of multiple instance and multiple processors was first studied in [38], which gave an optimal schedule assuming the restrictive class of cyclic schedules. Optimal schedules for the multi-instance, multi-processor setting, without restrictions, were established in [105]. [21] studied contract scheduling in the presence of soft deadlines, namely when the interruption is not a strict deadline, but there is rather a "grace period" within which the interruptive system must output a solution. The setting in which the interruption is stochastic, instead of adversarial was studied in [39].

There are some interesting connections between the problem of minimizing the acceleration ratio of a (single processor) schedule for n instances, and the problem of minimizing the competitive ratio of (single searcher) m-ray search. More specifically, [38] showed that a cyclic strategy for ray search can be translated to a cyclic strategy for contract scheduling and vice versa, and gave an expression that relates the competitive and acceleration ratios of the corresponding solutions.

1.4 Contribution of this thesis

In this thesis we study problems related to online computation, algorithmic search theory, and interruptible computation. Our focus is on new models, measures, techniques and settings, as well as on connections between the above fields. Below we give an outline of our contributions.

1.4.1 Online Computation

Bijective analysis and the bijective ratio of online algorithms

In previous work [13] we introduced bijective analysis as a technique for pair-wise comparison of online algorithms based on the concept of *dominance*. Unlike worst-case measures such as the competitive ratio, bijective analysis compares online algorithms on their performance over

the entire spectrum of request sequences. This technique has been applied in problems such as paging, list update, bin coloring, routing in array mesh networks, and in connection with Bloom filters, and has often provided a clear separation between algorithms whose performance varies significantly in practice.

Despite its appealing properties, bijective analysis is quite restrictive: it seeks a relation between online algorithms that may be either too difficult to establish analytically, or worse, may not even exist. We thus propose remedies to these shortcomings. Our objective it to make all online algorithms amenable to this type of analysis.

First, we establish sufficient conditions that allow us to prove the bijective optimality of a certain class of algorithms for a wide range of problems; we demonstrate this approach in the context of well-studied online problems such as weighted paging, reordering buffer management, and 2-server on the circle.

Second, to account for situations in which either two algorithms are incomparable, or there is no clear optimum, we introduce the *bijective ratio* as a natural extension of (exact) bijective analysis. This makes it possible to compare two arbitrary online algorithms for an arbitrary online problem. In addition, the bijective ratio is a generalization of the Max/Max ratio [36], and allows for the incorporation of other useful techniques such as amortized analysis. We demonstrate the applicability of the bijective ratio to one of the fundamental online problems, namely the continuous k-server problem on metrics such as the line, the circle, and the star. Among other results, we show that the greedy algorithm attains bijective ratios of $O(k)$ across these metrics. This is in contrast to competitive analysis, according to which the greedy algorithm has unbounded competitive ratio, but also closer to empirical evidence that the greedy algorithm is not woefully inefficient.

Online computation with untrusted advice

This work proposes a new model of online computation with advice, and its starting point is a rather simple and intuitive observation. Namely, unlike real life in which advice is a recommendation that we can choose to follow or to ignore based on trustworthiness, in the current advice model, the online algorithm treats it as infallible. This means that if the advice is corrupt or, worse, if it comes from a malicious source, the algorithm may perform poorly. In this work, we study online computation in a setting in which the advice is provided by an untrusted source.

Our objective is to quantify the impact of untrusted advice so as to design and analyze online algorithms that are robust and perform well even when the advice is generated in a malicious, adversarial manner. It is also the first work that attempts to bridge the rather large gap between the information theoretic model of advice complexity and the machine learning world, where the advice (i.e., the predictor) is always considered to be noisy.

Our new model takes into consideration the competitive performance in both settings of trusted and untrusted advice, and leads naturally to *Pareto*-optimality as the desired efficient concept. We show how the new model can be applied to online problems such as ski rental, online bidding, bin packing, and list update, defined in Section 1.1.5. For ski-rental and online bidding, we show how to obtain algorithms that are Pareto-optimal with respect to the competitive ratios achieved; this improves upon the framework of Purohit [115] in which Pareto-optimality is not necessarily guaranteed. For bin packing and list update, we give online algorithms with worst-case tradeoffs in their competitiveness, depending on whether the advice is trusted or not; this is motivated by work of Lykouris and Vassilvitskii [109] on the paging problem, but in which the competitiveness depends on the reliability of the advice. Furthermore, we demonstrate how to prove lower bounds, within this model, on the tradeoff between the number of advice bits and the competitiveness of any online algorithm. Last, we study the effect of randomization: here we show that for ski-rental there is a randomized algorithm that Pareto-dominates any deterministic algorithm with advice of any size. We also show that a single random bit is not always inferior to a single advice bit, as it happens in the standard advice model.

1.4.2 Online searching

Online searching with turn cost

In this work we consider the problem of m-ray search in a setting in which every time the searcher turns direction, it incurs a fixed cost. This is motivated by applications such as robot search, in which turning is a costly operation that should not be ignored (e.g., it may take quite some time for the robot to turn). The objective is to optimize the competitive ratio, as function of the number of rays and the turn cost. This was a problem that was introduced and studied by Demaine *et al.* [66], who gave an elegant approach based on infinite linear programming. Solving this infinite LP gives a lower bound on the competitive ratio, which matches an upper bound of a simple, cyclic strategy.

While this approach is elegant and yields eventually the optimal algorithm, we show a flaw in the proof of [66]; namely, the technique used in [66] cannot prove the feasibility, much less the optimality of the claimed solution. More specifically, in our work we first demonstrate that the definition of duality in infinite LPs, as given in [66] can lead to erroneous results. We then provide a non-trivial correction which establishes the optimality of a certain round-robin search strategy. The main idea here is to find first a solution for each finite LP with k constraints, each corresponding to an adversarial target placement, and then evaluate the limit solution as k tends to infinity.

We then consider a generalization of this problem, which combines turn cost considerations (es defined above) and the presence of *multiple* searchers that must locate a target which lies in an unknown position of an m-ray. The setting is a combination of the multi-searcher problem with no turn cost [107] and the single-searcher problem with turn cost [66]. It is also the problem that lead us to discover the subtleties, and pitfalls, in the analysis of infinite LPs.

We obtain a near-optimal strategy for this combined problem, using again an approach based on infinite linear-programming formulations. Our solution to these problems follows, at a high level, the approach for the single-searcher setting. Namely, one would like to show that *any* strategy can be described by an infinite LP, in which, roughly speaking, each constraint corresponds to an adversarial placement of the target. There is a considerable complication here: The above statement is not necessarily true. We bypass this problem by creating a *lower-bound* series of LPs, which are not tied to a real strategy, but can be used to bound the performance of any strategy.

Best-of-both-worlds analysis of linear search

As mentioned in Section 1.2, there is a simple doubling strategy that achieves an optimal competitive ratio equal to 9 for linear search. This is not the unique competitively optimal strategy; in fact there is an infinite number of 9-competitively strategies. Perhaps surprisingly, and to the best of our knowledge, this simple observation has not attracted sufficient attention. To be precise, one could ask the same question that was asked for online problems with a multitude of optimal algorithms: Is there a strategy that *intuitively* could be better, or at least no worse, than the doubling strategy? And if yes, can we establish a theoretical separation between competitively equivalent strategies?

In this work we investigate this question by applying essentially the framework of bijective analysis to linear search. More precisely, we propose a measure that reflects the rate at which the line is being explored by the searcher, and which can be seen as an extension of the bijective ratio over an uncountable set of requests. Using this measure, we show that a natural strategy that explores the line aggressively (i.e., the searcher moves to the furthest possible extent each time around) is optimal among all 9-competitive strategies. This provides, in particular, a strict separation from the competitively optimal doubling strategy, which is much more conservative in terms of exploration.

Weighted m-ray search

In this line of work we extend online search, and more specifically m-ray search, to the setting in which there is not a single hider, but rather *multiple hiders* that lie in unknown positions of the search domain. Each hider has a weight that is revealed to the searcher when it is reached, at which point the searcher accrues the weight. There is also a *target weight* W that represents the total weight that the searcher needs to attain. We are thus interested in minimizing the competitive ratio of m-ray search, where each ray may have at most one hider. More specifically, the competitive ratio compares the cost incurred by the searcher to the optimal cost of an omniscient searcher that knows the exact positions of all hiders. This generalization is a formulation of weighted resource allocation: namely, how one should allocate resources (i.e., computational time) among m different, weighted tasks, without knowing in advance the task weights.

We distinguish between two settings: The *uniform* setting, in which all the hiders are identical, i.e., they all have unit weights, and the *non-uniform* setting in which each hider has its own weight, represented by a non-negative real number. For the uniform setting, we show that there is an optimal cyclic strategy which increases the search lengths by a constant factor every time a hider is located. In contrast, the non-uniform setting turns out to be much more complex, since simple strategies as the above are very inefficient. Instead, we have to rely to a strategy that modifies the search lengths adaptively, depending on the number of hiders located so far. We give a *parameterized* analysis of this adaptive strategy that establishes near-optimality with respect to the parameter $W/w_{\max}$, where $w_{\max}$ is the maximum hider weight.

Competitive analysis of expanding search

The usual search paradigm when seeking an immobile hider in a search domain is what we now call *pathwise* search, in that the searcher follows a continuous unit speed path, and the search cost is the total time that the searcher spends. More recently, Alpern and Lidbetter [7] introduced a new paradigm which they term *expanding search*, in which the searcher may restart the search at any time from any previously reached point, or more informally, the searcher does not incur cost for revisiting previously explored territory. This is a useful paradigm when modeling problems such as de-mining a field (ensuring that an area is mine-free is the costly operation as opposed to navigating through an area that has been de-mined) and drilling for coal (again, drilling is the costly operation).

In our work we study expanding search under the framework of competitive analysis. We consider two settings. First, the setting in which the target hides in the vertex of an edge-weighted graph, and second, the setting in which the target hides over any point of a given network, which may be bounded or unbounded. We present hardness and approximation results that pertain to both the deterministic and the randomized competitive ratios, with the randomized setting being, as expected, the more challenging one.

1.4.3 Scheduling of contract algorithms and interruptible computation

Earliest-completion scheduling of contract algorithms with end guarantees

All previous work on contract scheduling has provided strict performance guarantees for several variants of this setting, assuming that an interruption can occur arbitrarily ahead in the future. This means that the schedule is, at least theoretically, infinite. In practice, however, one expects that the schedule will reach a point beyond which further progress will only be marginal, hence it can be deemed complete. This occurs when the schedule has completed a sufficiently long contract for each problem instance, i.e., when each problem has attained a desired *end guarantee*.

In our work we study contract scheduling in a setting that is motivated by the above observation. In particular, we seek a *finite* schedule with the following properties: i) it attains the

optimal acceleration ratio, as in the standard model; and ii) it minimizes the time required to satisfy the end guarantees, among all schedules that obey property (i). In other words, if an interruption occurs during the execution of the schedule, the schedule has optimal performance; otherwise, if no interruption has occurred, it outputs a solution that meets the end guarantees the earliest possible, among all schedules optimal with respect to the acceleration ratio.

We show how to obtain an optimal schedule that satisfies the above properties. We complement the theoretical analysis with a computational evaluation that quantifies the gains with respect to the standard, infinite schedules.

A new measure for the analysis of interruptible systems

In this work we take a closer look into the performance measures for interruptible systems based on schedules of contract algorithms for n problem instances. We observe that the acceleration ratio compares the performance of the schedule to an ideal solution that knows both the interruption time and the queried instance. Thus, the ideal solution effectively ignores all but one of the problem instances. This yields an overly powerful ideal solution, which implies, in turn, a weak performance guarantee for the contract schedule.

We thus ask the question: can we define an extension of the acceleration ratio in which the quality of the schedule is compared to an ideal solution that only knows the interruption time, but must produce solutions to all n problem instances? We answer this question by introducing the *deficiency* of a schedule as a performance measure that meets these requirements. Beyond this conceptual contribution, minimizing the definency turned out to be a challenging optimization problem on its own, and much harder than minimizing the acceleration ratio. We showed several upper and lower bounds on the deficiency of schedules, and the takaway message is that there exist schedules with small, constant deficiency, independent of n. To highlight the applicability of our techniques, in the journal version we also showed how to obtain a much simpler proof of the main resul of [38].

Further connections between ray searching and contract scheduling

Bernstein *et al.* [38] were the first to demonstrate that ray searching and contract scheduling have certain similarities. In particular, they showed that cyclic strategies for one problem yield cyclic strategies for the other, and gave a relation for the corresponding competitive and acceleration ratios. In our work, we argue that the connections are deeper, in that these problems capture fundamental aspects of resource allocation under uncertainty.

In particular, we study several variants of these families of problems, such as searching and scheduling with probabilistic considerations, redundancy and fault-tolerance issues, randomized strategies, and trade-offs between performance and preemptions. For many of these problems we present the first known results that apply to multi-ray and multi-problem domains. Our objective is to demonstrate that several well-motivated settings can be addressed using the same underlying approach.

Chapter 2

New measures and models in online computation

In this chapter we focus on online computation, and we discuss contributions towards two directions. First, we extend a measure alternative to competitive analysis so as to make it applicable to all online problems; more precisely we introduce and apply an extension to *bijective analysis* which we call the *bijective ratio* (Section 2.1). Next, we introduce and apply an extension to the standard advice model of online computation in which the advice may originate from an untrustworthy source (Section 2.2).

The material of Section 2.1 is based on work that appeared recently in *Algorithmica* [24], but also discusses briefly previous work in the *Journal of the ACM* [25] and *Algorithmica* [13]. Section 2.2 is based on material that appeared in the *11th International Conference on Innovations in Theoretical Computer Science* [15].

2.1 The bijective ratio of online algorithms

2.1.1 Background

As discussed in the introduction, competitive analysis is a worst-case measure that builds on two main premises: First, the performance of an online algorithm is determined by a worst-case request sequence; and second, it is determined by a comparison to the offline optimal solution.

A different approach would be based on a *pair-wise comparison* of two online algorithms. More precisely, one would like to compare two online algorithms based on their performance on *all possible sequences*. In this context, and assuming a cost-minimization problem, the performance of an algorithm can be represented by a *multiset* of costs incurred by the corresponding request sequences. This naturally leads to a *dominance-based* criterion for comparing the cost-sets of two online algorithms.

Definition 8 ([13])**.** *Let $\mathcal{I}_n$ denote the set of all request sequences of length n. The online algorithm A is* no worse *than the online algorithm B according to bijective analysis if there exists an integer $n_0 \in \mathbb{N}^+$ such that, for all $n \geq n_0$, there exists a bijection $\pi : \mathcal{I}_n \to \mathcal{I}_n$ satisfying $A(\sigma) \leq B(\pi(\sigma))$ for each $\sigma \in \mathcal{I}_n$. Moreover, A is bijectively* optimal *if the above holds for all online algorithms B.*

Figure 2.1 illustrates the concept of bijective analysis.

There are several reasons one could opt for bijective analysis, as opposed to competitive analysis:

- Bijective analysis does not evaluate the performance of an algorithm on a single "worst-case" sequence, but instead takes into consideration the overall performance of the algorithm over all request sequences.

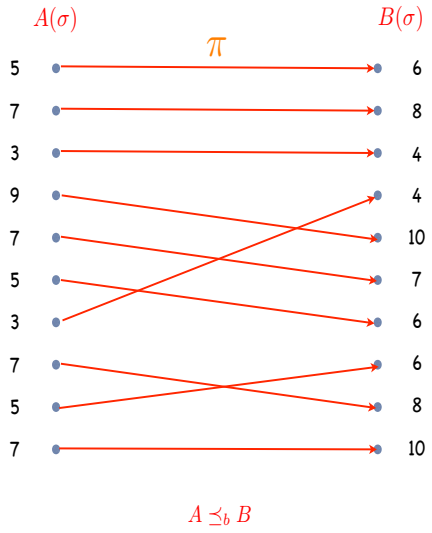


Figure 2.1: An illustration of the application of bijective analysis. Here, the numbers refer to the costs incurred by algorithms A and B on all possible request sequences of the same size.

- It allows direct comparison of two algorithms, hence there is neither the need to appeal to an offline algorithm, nor to analyze the offline optimum.
- It is consistent with some natural, "to-be-expected" properties of efficient online algorithms which competitive analysis fails to yield. For instance, in [13] we showed that paging with any amount of *lookahead* is strictly better (according to bijective analysis) than paging without lookahead. In contrast, as argued in [37], no finite lookahead can improve the competitive ratio of any paging algorithm.
- It can incorporate assumptions concerning the space of request sequences, which can lead to refined results. More precisely, suppose that we restrict the set of sequences to those exhibiting certain natural or established properties. Applying bijective analysis over the restricted input set can lead to a better understanding of the performance of algorithms.

To illustrate the last point, in particular, consider the paging problem. Typical request sequences for this problem are known to exhibit (temporal) *locality of reference* [67]; informally, this means that a page that is requested at some point in time is also likely to be requested in the short future. There are several models that capture properties of sequences with locality of reference such as [3]. By restricting the set of all possible request sequences $\mathcal{I}_n$ to precisely sequences with locality of reference, one can still apply Definition 8 and bijectively analyze the performance of online algorithms on such typical sequences.

There exist both relaxations and generalizations of bijective analysis. On the one hand, a substantially weaker measure than bijective analysis is *average analysis* [13], which compares the average cost of two online algorithms over request sequences of the same length. Formally, we say that online algorithm A is no worse than online algorithm B on request sequences of size n if $\sum_{\sigma\in\mathcal{I}_n} A(\sigma) \leq \sum_{\sigma\in\mathcal{I}_n} B(\sigma)$.

On the other hand, bijective analysis is a special case of (first order) *stochastic dominance* which defines a partial order on random variables. A random variable X is stochastically domi-

nated by a random variable Y if, for all $c \in \mathbb{R}$, we have $\Pr[X < c] \geq \Pr[Y < c]$. Clearly, if X is stochastically dominated by Y, then $\mathbb{E}(X) \leq \mathbb{E}(Y)$; however a much stronger conclusion can be drawn since F has more probability mass towards lower values than G. Stochastic dominance has been very useful in the context of decision theory and microeconomics, with applications varying from portfolio selection to measuring income inequality in society. For a comprehensive discussion, see Chapters 4 and 5 in the textbook [126]. It is interesting to note that under the assumption that all request sequences are drawn according to the uniform probability distribution, bijective analysis is identical to stochastic dominance.

The remainder of this section is structured as follows. In Section 2.1.2 we review some previous results on bijective and stochastic dominance in online computation. In Section 2.1.3 we obtain a general framework for establishing the bijective optimality of certain online algorithms. Last, in Section 2.1.4 we show how to extend bijective analysis so as to establish not only an exact, but also an approximate concept of dominance. This is very useful in cases in which it is either very difficult, or even impossible to appeal to (exact) bijective optimality.

2.1.2 Related work

The first consideration of bijective dominance for the analysis of online algorithms can be traced back to [84, 123] in the context of the *two-headed disk* problem. This problem is related to the k-server problem but with a different cost function. Given k mobile servers on a metric space, requests appear on the points of the metric space and the goal is to minimize the distance traveled to serve these requests. During the time a request is being served, the other servers can re-position themselves at no cost. This renders the decision of which server to use trivial (it will always be the closer server) and puts focus on the question of where to place the other server. Consider a line metric of unit length, and let x denote the location of a request, in terms of its distance from the left-most point. Hofri showed that the greedy algorithm, which positions the server that does not serve the request at $x/3$ if $x \geq 1/2$, and at $2/3 + x/3$ otherwise, is optimal in average, and conjectured that it is stochastically dominated by every other algorithm under a uniform distribution [84] which was proven later by Seshadri and Rotem [123].

Concerning average analysis, Calderbank *et al.* studied the 2-server problem on the line and circle [58], as well as the n-dimensional sphere [59]. They focused on the average case and, in particular, calculated the expected cost of the greedy algorithm on the circle. Here, the greedy algorithm is the simple algorithm that serves a request with the server closest to it. Moreover, [58] presents experimental data that show that the greedy algorithm is relatively close in performance to the offline optimal algorithm on the line. Similar experiments, for a variety of metric spaces and algorithms, including the greedy algorithm, are presented in [33, 119, 120]. In a related work, Anagnostopoulos *et al.* [8] studied the steady-state distribution of the greedy algorithm for the k-server problem on the circle.

Boyar *et al.* [52] provided a systematic study of several measures for a simple version of the k-server problem, namely, the two server problem on three colinear points, which they name the *baby server* problem. In particular, they showed that the greedy algorithm is bijectively optimal.

We introduced bijective analysis in [13], and we showed that without any assumptions on the request sequences, a large class of natural paging strategies known as *lazy* strategies are bijectively optimal (these are strategies that only evict a page upon a fault). In contrast, under a model of locality of reference of Albers *et al.* [3], we showed that a strategy that is known to be efficient in practice, namely Least-Recently-Used (LRU) is the *unique* optimal strategy according to average analysis. We later strengthened this result to to bijective optimality [25]. We showed similar results in [13, 25] concerning the list update problem and the Move-To-Front algorithm (MTF).

Stochastic dominance, assuming certain pertinent distributions, has been applied to certain online problems such as the paging problem [82], routing in array mesh networks [114], bin coloring [81] and in the online construction of Bloom filters [108].

2.1.3 A general framework for establishing bijective optimality

In this section we demonstrate a general technique for showing that a certain online problem admits a *bijectively optimal* algorithm. This extends a technique that was applied in [25] in the context of paging and list update. In particular, we identify some essential conditions under which a certain greedy-like algorithms (as formally defined in [46]) can be optimal. We will then apply this general framework to the *2-server problem on the continuous circle*, the *weighted paging* problem, and the *reordering buffer management* problem.

To this end, we first need a criterion that establishes, in a formal manner, the greedy characteristic. More precisely, consider an online algorithm which, on a sequence of requests σ, must serve the i-th request σ_i after having served the sequence of the first $i-1$ requests, denoted by $\sigma[1, i-1]$. We say that an algorithm is *greedy-like* if it serves each request σ_i in a way that minimizes the cost objective, assuming this request σ_i is the last request. This definition is motivated by a similar characterization of "greediness" in the context of *priority algorithms* as defined by Borodin *et al.* [46].

Naturally, not all greedy-like algorithms are expected to be bijectively optimal. For instance, for the classic paging problem, all lazy algorithms are greedy-like, however, as shown in [13, 25], assuming locality of reference (as defined therein), only LRU is optimal. Therefore, one needs first to choose a "good" algorithm in this class of greedy-like algorithms for the specific problem definition. Second, for the purposes of the proofs, we will need to appeal to an algorithm that treats *individual* requests as a "good" greedy-like algorithm. We can formalize these ideas as follows. Let G denote a greedy-like algorithm. Given a sequence σ, we say that A is *G-like on σ_j* if, after serving $\sigma[1, j-1]$, A serves request σ_j as G would. Due to the problem specific nature of the definition, the G-like notion cannot be characterized in general for all online problems. Instead, it needs to be defined for specific problems and specific greedy-like algorithms (e.g., the definition of an "LRU-like" algorithm in [25]). We say that algorithm A is *G-like on the interval* $[j, n]$ if, for every sequence $\sigma \in \mathcal{I}_n$, after serving $\sigma[1, j-1]$, A serves all requests $\sigma_j, \ldots, \sigma_n$ in a G-like manner. The following definition formally describes algorithms for which the G-like decision can be moved "one step earlier" without affecting performance with respect to bijective analysis.

Definition 9. *Suppose that A is an online algorithm for the sequences in $\mathcal{I}_n$ such that A is G-like on the interval $[j+1, n]$. We say that A is* G-like extendable *on j if there exists a bijection $\pi : \mathcal{I}_n \to \mathcal{I}_n$ and an online algorithm B with the following properties. For every $\sigma \in \mathcal{I}_n$,*

- *B makes the same decisions as A on the first $j-1$ requests of σ.*
- *B is G-like on σ_j.*
- *$\pi(\sigma)[1, j] = \sigma[1, j]$ and $B(\pi(\sigma)) \leq A(\sigma)$.*

Informally, A is G-like extendable if it can be transformed to another algorithm B that is "closer" to the statement of a G-like algorithm and is not inferior to A according to bijective analysis. Definition 9 is instrumental in proving the optimality of G; in particular, we obtain the following theorem.

Theorem 10. *If every online algorithm A (over requests in $\mathcal{I}_n$) that is G-like on the interval $[j+1, n]$ is also G-like extendable on j, for all $1 \leq j \leq n$, then G is bijectively optimal.*

Let us give some intuition about the above definitions, and the proof of Theorem 10. Let G be a specific greedy-like algorithm that we would like to prove that it is bijectively optimal (e.g., LRU for the paging problem). For any request sequence of size n, G makes G-like decisions on every request; we can think of these decisions as an n-bit string where each bit is 1, and in which the i-th bit is the indicator of the event that the i-th request is served in a G-like manner.

Definition 9 says that if a G-like extendable algorithm A is such that the last $n-j$ bits are all 1, then we can find another algorithm B which is bijectively no worse than A, and in which the j-th bit is equal to 1. Then the proof of Theorem 10 is based on the following idea: starting with any online algorithm A (i.e., with any n-bit string), transform A to G (a string of only 1s) using the transitivity of the bijective relation (akin to incrementing an n-bit binary counter).

We will demonstrate the applicability of this framework by showing optimality of greedy-like online algorithms for the 2-server problem on the circle. We will also mention the results we obtain, by a similar application of this framework, concerning the weighted paging problem and the reordering buffer management problem.

The 2-server problem on the circle

Before we proceed with the optimality proof, a word of caution: This is a "continuous" problem, in that the set of all request sequences $\mathcal{I}_n$ is uncountably infinite, unlike problems such as paging and list update in which $\mathcal{I}_n$ is discrete, and finite. This means that one needs to be careful about the allowable bijections. For simplicity, we model the continuous k-server problem using discrete metrics in which nodes are placed in an equispaced manner; as the number of nodes approaches infinity, this model provides a satisfactory approximation of the continuous problem. We thus approximate the continuous line (resp. circle) by a path (resp. cycle) in which vertices are uniformly spaced, i.e., all edges have the same length which may be arbitrarily close to zero. However, we note that the techniques we use in this work can most likely be made applicable even for the formal definition of the continuous problem, i.e., even when the set of all request sequences of length n is uncountably infinite, by using measure-preserving transformations such as *interval exchange transformations* [98].

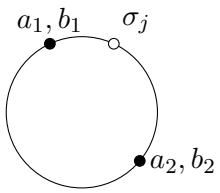
Let us now move to our applications. For 2-server on the circle, the candidate algorithm G is the obvious GREEDY algorithm (with an arbitrary tie-breaking rule) that serves a request by moving the server closer to the request, and the G-like notion is well-defined.

Theorem 11. GREEDY *is bijectively optimal for 2-server on the circle.*

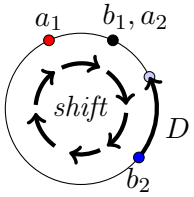
Proof. Let A denote any online algorithm that is G-like on the interval $[j+1, n]$, for some $j \in [1, n]$. From Theorem 10, it suffices to prove that A is G-like extendable on j. We will show the existence of an appropriate online algorithm B and a bijection π, according to Definition 9. In particular, since the definition requires that $\pi(\sigma)[1, j] = \sigma[1, j]$, and that B makes the same decisions as A on $\sigma[1, j-1]$, we only need to define $\pi(\sigma)[j+1, n]$, as well as the decisions of B while serving the latter sequence of requests.

Consider the request σ_j: if A serves this request in a G-like manner (i.e., greedily), then the theorem holds trivially. Otherwise, after serving $\sigma[1, j-1]$ and $\pi(\sigma)[1, j-1]$, respectively, A and B have the same configuration. Namely, if a_1, a_2 and b_1, b_2 denote the servers for the two algorithms at this configuration, we have that $a_1 \equiv b_1$ and $a_2 \equiv b_2$. Since A does not serve σ_j greedily, we can assume, without loss of generality, that $d(a_2, \sigma_j) \geq d(a_1, \sigma_j)$ and that A serves the request using a_2 (see Figure 2.2 for an illustration). Here, we denote by $d(x, y)$ the distance of two points x and y in the circle. Let $D = d(a_2, \sigma_j) - d(a_1, \sigma_j)$, and let $\overline{\sigma}[j+1, n]$ denote the sequence which is derived from $\sigma[j+1, n]$ by *shifting* each request by $\delta := d(a_1, \sigma_j)$ in the direction opposite to the move of a_2 (in the example of Figure 2.2, this is done clockwise). We then define the mapping $\pi(\sigma)$ as $\pi(\sigma) = \sigma[1, j] \cdot \overline{\sigma}[j+1, n]$. This mapping is bijective as every possible request (i.e. every node) at each index in a request sequence is mapped to a distinct value from the same set of nodes.

Next, we define the actions of algorithm B over the sequence $\pi(\sigma)[j+1, n]$. Algorithm B serves the request $\sigma_j = \pi(\sigma_j)$ greedily; moreover we define B to move the server b_2 by a distance equal to D so as to ensure that the distance between the servers in B match the distance between the servers of A prior to serving the next request (in the example of Figure 2.2, this is



(a) The configurations right before σ_j.



(b) The configurations and actions after serving σ_j.

Figure 2.2: An illustration of the bijection that *shifts* the requests around the circle by a distance of $\delta := d(a_1, \sigma_j)$ and the first action of A after the configurations of A and B diverge on request σ_j. In the figures, a white dot represents the request, a black dot represents a node with servers of A and B, a red dot represents a node with only a server of A, a blue dot represents a node with only a server of B, and a cyan dot represents the position of server b_2 after the non-lazy move.

done counter-clockwise). We will attribute this cost to the j-th request and, hence, the cost of A on σ_j is the same as the cost of B on $\pi(\sigma_{j+1})$.

We will show, by induction on ℓ from $j+1$ to n, that the distances of the servers of A, $d(a_1, a_2)$, is the same as the distance between the servers of B, $d(b_1, b_2)$, and they are offset by a shift of δ, prior to serving the ℓ-th request. Moreover, the cost of A and B to serve their respective ℓ-th requests will be the same. For any request ℓ, if $d(a_1, a_2) = d(b_1, b_2)$ and the servers are offset by a shift of δ, B can serve the request $\pi(\sigma_\ell)$ by moving one of its servers that is in the same position, relative to the shift, as the server of A that serves σ_ℓ. This will ensure that the costs are identical for A and B, and, after serving r_ℓ, $d(a_1, a_2) = d(b_1, b_2)$ and that the servers are offset by a shift of δ. This proves both the base case and the inductive step of the induction. For the base case, recall that the actions of B on request $\pi(\sigma[j])$ ensures that $d(a_1, a_2) = d(b_1, b_2)$ and the servers are offset by a shift of δ prior to serving the $(j+1)$-th request.

In conclusion, the cost of A on σ_i is the same as B on $\pi(\sigma_i)$ which implies that $A(\sigma) = B(\pi(\sigma))$ and concludes the proof. □

Weighted paging and reordering buffer management

Recall the definitions of these two online problems, as given in Section 1.1.5. Again, an application of Theorem 10 can help us establish the bijective optimality of an appropriate greedy algorithm.

Consider first the weighted paging problem, and the simple greedy algorithm G which, upon a fault, evicts from the cache a page of smallest cost. This algorithm is greedy-like as it minimizes the cost at each request. Unlike Theorem 11 (and unlike the proof of the bijective optimality of greedy/lazy algorithms for unweighted paging in [13]), the proof is technically more involved, due to the asymmetry of the cost of requests (which complicates the argument for the G-like extendability of all possible online algorithms).

Theorem 12. *The above greedy algorithm is bijectively optimal for weighted paging.*

Consider next the reordering buffer management problem. For this problem, we define G as the greedy-like algorithm that switches (only if necessary) to a color c for which the number of items of color c in the buffer is maximized among all colors. The nature of this problem gives rise to some technical complications in the optimality proof, in the sense that an algorithm may delay processing a request, (an option that is not meaningful in the context of paging/k-server problems), which in turn complicates the comparison of $A(\sigma)$ and $B(\pi(\sigma))$ on a request-by-request manner. Nevertheless we can prove the following theorem.

Theorem 13. *The above greedy algorithm is bijectively optimal for reordering buffer management.*

2.1.4 Beyond (exact) bijective analysis: The bijective ratio of online algorithms

Despite the appealing properties of bijective analysis, its biggest deficiency is a rather serious one: It may be very difficult to compare two online algorithms, in that it may be very hard to prove analytically the existence of the required bijection; even worse, such a bijection may not even exist. Thus, this analysis technique may deem algorithms incomparable across a wide variety of problems, and, in this sense, it does not give rise to a real performance measure.

This drawback implies that bijective analysis lacks the most desirable property of the competitive ratio; namely the amenability of a given online problem to analysis. Such an observation could also help explain why these techniques did not become as popular as competitive analysis, even though the fundamentals and some limited applications can be traced to work contemporary with the early works on competitive analysis.

For these reasons, we propose (and apply) an extension of bijective analysis that makes the technique applicable to any given online problem, by relaxing the comparison of two algorithms. This extension gives rise to a performance measure which we call the *bijective ratio*.

Definition 14. *Given an online algorithm A and an algorithm B, we say that the* bijective ratio of A against B *is at most $\rho > 0$ if there exists an $n_0 \in \mathbb{N}^+$, such that, for all $n \geq n_0$, there exists a bijection $\pi : \mathcal{I}_n \to \mathcal{I}_n$ satisfying $A(\sigma) \leq \rho \cdot B(\pi(\sigma))$ for each $\sigma \in \mathcal{I}_n$. The bijective ratio of an online algorithm A is at most ρ if, for every algorithm B, the bijective ratio of A against B is at most ρ. The bijective ratio of an online cost-minimization problem is the infimum ρ for which there exists an online algorithm with bijective ratio at most ρ.*

Definition 14 is a natural extension of bijective optimality in the spirit of measures such as the competitive and the approximation ratio. It also upholds the essential aspect of bijective analysis in that every sequence for which A incurs a certain cost can be bijectively mapped to a sequence on which B is at least $\frac{1}{\rho}$ times as costly. Furthermore, a bijective ratio of ρ implies that the average-cost ratio of the two algorithms is at most ρ, but also the far stronger conclusion that the contribution of sequences to the average costs of the two algorithms can be attributed in a local manner, as argued earlier. Both properties are desired extensions of bijective optimality, in the sense that they provide a much stronger comparison than the one induced by average-case analysis. The bijective ratio is, by definition, an approximate stochastic dominance over a uniform distribution of the inputs. This definition readily extends to approximate stochastic dominance over any distribution.

It is also interesting to note that the bijective ratio is a generalization of the Max/Max ratio of Ben-David and Borodin [36]. This measure is defined as the ratio of the maximum-cost sequence for algorithm A over the maximum-cost sequence for algorithm B. Indeed, if A has bijective ratio ρ against B, then the i-th more costly sequence of A costs at most ρ times the i-th most costly sequence of B (assuming a given length of request sequences), *for every i*.

Last, in the above definitions, the performance ratios are *strict*; however, as with the competitive ratio, one can define *asymptotic* ratios. For instance, the asymptotic ratio of A against B is at most ρ if there exists a constant c such that $A(\sigma) \leq \rho \cdot B(\pi(\sigma)) + c$ for all $n \geq n_0$.

2.1.5 An application: The bijective ratio of k-server problems

As an application of the bijective ratio, we will consider the (continuous) k-server problem in metrics such as the line, the circle and the star. Our main focus is on the performance of the greedy algorithm which is motivated by the experimental evidence that this algorithm performs well in several settings [58, 33, 119, 120], in particular, when the requests follow a

uniform distribution and average cost is considered. However, these results are in contrast with competitive analysis since the greedy algorithm has an unbounded competitive ratio even on the line. As noted in [119], "the [experimental] results demonstrate that [the Work Function Algorithm (WFA)] performs similarly or only slightly better than a simple heuristic such as the greedy algorithm, although according to theory it should perform much better". In this sense, there is a big disconnect between competitive analysis and experimental behavior which, perhaps surprisingly, has not received as much attention from the theoretical computer science community as other problems such as the paging problem. Our results demonstrate that bijective analysis can help narrow this gap. More precisely, we show that the greedy algorithm has bijective ratio $O(k)$ in the considered metric spaces.

Exact bijective analysis, following an approach as in Section 2.1.3, is bound to fail: the greedy algorithm (denoted by GREEDY throughout this section) is not an optimal online algorithm for 2-server on the line, even for average-case analysis (which implies the same result for bijective analysis). This immediately raises the question: How good (or bad) is GREEDY? In the remainder of this section, we address this question.

Another algorithm we will consider is the algorithm k-CENTER [36], which anchors its servers at k points of the metric so as to minimize the maximum distance among points in the metric to a server; it then serves each request by moving the server closest to it, which subsequently returns to its anchor position. In contrast to the results for GREEDY, we will show that k-CENTER has an asymptotic bijective ratio of 2 for the line and the circle, which generalizes the known upper bound on the Max/Max ratio of this algorithm [36]. However, as we will see, k-CENTER has unbounded bijective ratio on a star metric.

It is worth mentioning that our results expand those of Boyar *et al.* [52] who showed that GREEDY is bijectively optimal for the 2-server problem on a very simple, albeit discrete metric consisting of three colinear points (termed the *baby server problem*).

In what follows we present, in more detail, our main results.

Results

We first begin with a discussion on *lower bounds* on the bijective ratio. The following simple proposition is on this direction, and its proof follows from a simple counting argument.

Proposition 15. *Suppose that there are $c > 0$ and n_0 such that, for all $n \geq n_0$, $|\{\sigma \in \mathcal{I}_n : A(\sigma) < \rho \cdot c\}| < |\{\sigma \in \mathcal{I}_n : B(\sigma) \leq c\}|$. Then the bijective ratio of A against B is at least ρ.*

Proposition 15 helps us establish the following lower bound, which extends to general k-server on the line and the circle.

Theorem 16. *Any deterministic online algorithm for 2-server even on a line metric of three equidistant points has bijective ratio at least 2.*

Another tool for showing a lower bound is the Max/Max ratio [37]. Clearly, the Max/Max ratio can be seen as a special case of the bijective ratio, and thus a lower bound on the Max/Max ratio implies the same lower bound on the bijective ratio. The following lower bound is based on this approach.

Theorem 17. *For any $\varepsilon > 0$, the bijective ratio of* GREEDY *is at least $\frac{k}{2} - \varepsilon$ for the line and at least $\frac{k}{3} - \varepsilon$ for the circle.*

We now turn our attention to *upper bounds*. We begin with showing sufficient conditions so that the bijective ratio of an online algorithm A against an algorithm B (that may be online or offline) is at most c; these conditions are formally described in Lemma 18 and Lemma 20. Both lemmas require two conditions stated in terms of individual requests, and their combination yields the desired bound.

Given a configuration C (i.e., a positioning of servers in the metric space), we use the notation $A((\sigma[i]|C)$ to denote the cost of A for serving request $\sigma[i]$ assuming the configuration C of servers. With a slight abuse of notation, we will denote by $A(\sigma[i]|B(\sigma[1,i-1]))$ the cost of A for serving request $\sigma[i]$ assuming a configuration resulting from B serving sequence $\sigma[1,i-1]$, where the suffix of the request sequence is implied. While this notation may not be well-defined for general algorithms, it is well-defined for the algorithms we analyze in this work.

The following simple, yet useful lemma is based on a *decoupling* approach. It identifies two properties which, if held simultaneously, lead to an upper bound on the bijective ratio of an algorithm A against an algorithm B.

Lemma 18 (Decoupling lemma)**.** *Suppose that there exists a $c > 1$, a $d > 0$ and a bijection π over $\mathcal{I}_n$ such that, given an online algorithm A and an algorithm B, for all $\sigma \in \mathcal{I}_n$ and all $i \leq n$, the following hold:*

(i) $A(\sigma[i]|B(\sigma[1,i-1])) \leq d \cdot B(\sigma[i])$, *and*

(ii) $A(\sigma[i]) - A(\pi(\sigma)[i]|B(\pi(\sigma)[1,i-1])) \leq (c-d) \cdot B(\pi(\sigma)[i])$,

then, $A(\sigma) \leq c \cdot B(\pi(\sigma))$.

Proof. Using $\pi(\sigma)$ as the request sequence for (i) and adding both inequalities, we get $A(\sigma[i]) \leq c \cdot B(\pi(\sigma)[i])$. The lemma follows by summing over all the requests. □

Let us explain first, on an intuitive level, the significance of the above lemma, and why it is important in the analysis of an algorithm such as the greedy algorithm. First, note that condition (i) holds for the greedy algorithm with $d = 1$. Given this fact, let us consider condition (ii), which we will argue that it holds for the line metric with $c = 2k$. There are two important observations to make concerning this condition. First, it applies to *single requests*, for both A and B. Second, it relates the performance of $A(\sigma[i])$ and $B(\pi(\sigma[i]))$ to the performance of $A(\pi(\sigma[i]))$ assuming it serves $\pi(\sigma[i])$ from a *specific configuration*, namely the configuration $B(\pi(\sigma)[1,i-1])$.

For an upper bound on the bijective ratio of A against B we can then assume that, in worst case, B always finds itself in an "optimal" configuration, whereas A always finds itself in the "worst" configuration. While such concepts do not necessarily exist for all metrics, we argue that they do exist for a metric such as the line. Here, an optimal configuration is the one in which the servers are at the k-Center anchoring positions (the positions that optimize coverage), whereas a worst-case configuration is one in which all servers happen to be cramped in one of the ends of the line. Assuming these pairs of worst and best case configurations for A and B, it is not difficult to see that if we order the costs that A and B pay for all possible single requests, we can order the costs such that the i-th most expensive request for A costs at most $2k$ times the cost of the i-th most expensive request for B. This helps define a bijection for each individual request, and thus a bijection for the entire sequence.

Formalizing the above ideas, we obtain the following upper bounds, by applying Lemma 18.

Theorem 19. *For the k-server problem,* greedy *has a bijective ratio of at most $2k$ on the circle and bijective ratio of at most k for the circle.*

Note that Theorem 19 shows an upper bound on the bijective ratio for the circle metric that is better than the best-known competitive ratio of $2k-1$ for the same metric (since the best-known competitive algorithm for this problem is the WFA algorithm, and no better bound on the competitive ratio is known for this algorithm on the circle).

One can argue that the decoupling lemma, Lemma 18 leaves quite some slack in the analysis, since it considers that A and B are at worst and best-case configurations at all times, which is quite pessimistic. This naturally leads to an extension that incorporates *amortized analysis*, as formalized in the following lemma.

Lemma 20 (Amortized decoupling lemma)**.** *Given an online algorithm A and an algorithm B, let Φ be any potential function such that the amortized cost of A for $\sigma[i]$ is $a_i = A(\sigma[i]) + \Delta\Phi_i$,*

where $\Delta\Phi_i = \Phi_i - \Phi_{i-1}$, and Φ_0 is the potential prior to serving the first request. Suppose also that there exist $c, d > 0$ and a bijection π over $\mathcal{I}_n$ such that, for all $\sigma \in \mathcal{I}_n$ and all $i \leq n$, the following hold:

(i) $A(\sigma[i]|B(\sigma[1, i-1])) \leq d \cdot B(\sigma[i])$, *and*
(ii) $a_i \leq c \cdot A(\pi(\sigma)[i]|B(\pi(\sigma)[1, i-1]))$,

then, $A(\sigma) \leq c \cdot d \cdot B(\pi(\sigma)) + \Phi_0 - \Phi_n$.

We can apply Lemma 20 and obtain the following improved upper bound on the bijective ratio of the greddy algorithm on the line

Theorem 21. GREEDY *has asymptotic bijective ratio at most $4k/3$ for k-server on the line.*

A much more straightforward analysis shows that k-CENTER is essentially optimal for the line and the circle metrics.

Theorem 22. k-CENTER *has an asymptotic bijective ratio of at most 2 for the k-server problem on the line and the circle.*

Theorem 22 leaves open the possibility that k-CENTER is bijectively efficient on more general metrics, especially given that its Max/Max ratio is at most 2 in arbirary metrics [37]. However, we show that this is not true. More precisely, we show that in *star* metrics, k-CENTER can be arbitrarily bad according to bijective analysis.

Theorem 23. *There exists a star S, and an online algorithm A such that* k-CENTER *has unbounded* asymptotic *bijective ratio against A on S.*

Proof sketch. The result follows from considering a star S that consists of $m-1$ rays of length d and one longer ray of length $4kd-d$. For this star, k-CENTER anchors its servers on the long ray (see Figure 2.3). In contrast, an efficient algorithm A would position one server in the centrer of the star. □

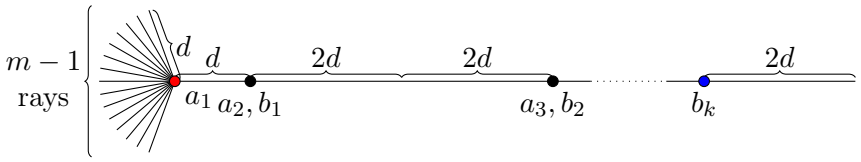


Figure 2.3: An illustration of the lower bound construction for the k-CENTER algorithm. Here, we denote by a_i, b_i the servers of A and k-CENTER, respectively.

In contrast, we can show that GREEDY has bijective ratio $4k$ for the star. This follows from an application of the amortized decoupling lemma (Lemma 20).

Theorem 24. GREEDY *has asymptotic bijective ratio $4k$ for k-server on a star metric.*

2.2 Online computation with untrusted advice

Suppose that you have an investment account with a significant amount in it, and that your financial institution advises you periodically on investments. One day, your banker informs you that company X will soon receive a big boost, and advises to use the entire account to buy stocks. If you were to completely trust the banker's advice, there are naturally two possibilities: either the advice will prove correct (which would be great) or it will prove wrong (which would be catastrophic). A prudent client would take this advice with a grain of salt, and would not be willing to risk everything. In general, our understanding of advice is that it entails *knowledge that is not foolproof.*

In this work we focus on online computation with advice. Our motivation stems from observing that, unlike the real world, the advice under the known models is often closer to "fiat" than "recommendation". Our objective is to propose a model which allows the possibility of incorrect advice, while still permitting to prove tradeoffs between the advice size and the performance of the online algorithm.

2.2.1 Background

As discussed in Section 1.1.4, the advice complexity model of online computation captures the interplay between the performance of the online algorithm and the amount of additional information given to the algorithm concerning its input. All previous work has assumed that advice is, in all circumstances, completely trustworthy, and precisely as defined by the algorithm. In particular, since the advice is assumed to be infallible, no online algorithm with advice would choose to ignore it.

It should be fairly clear that such assumptions are very unrealistic or undesirable. Advice bits, as all information, are prone to transmission errors. In addition, the known advice models often require that the information encodes some information about the input, which, realistically, cannot be known exactly (e.g., some bits of the optimal, offline solution). Last, and perhaps more significantly, a malicious entity that takes control of the advice oracle can have a catastrophic impact. For a very simple example, consider the ski rental problem, defined in Section 1.1.5. In the traditional advice model, one bit suffices to be optimal: 0 for renting throughout the horizon, 1 for buying right away. However, if this bit is wrong, then the online algorithm has unbounded competitive ratio, i.e., can perform extremely badly.

The above observations were recently made in the context of online algorithms with machine-learned predictions. Lykouris and Vassilvitskii [109] and Purohit *et al.* [115] show how to use predictors so as to design and analyze algorithms with two properties: (i) if the predictor is good, then the online algorithm should perform close to the best offline algorithm (what is called *consistency*); and (ii) if the predictor is bad, then the online algorithm should gracefully degrade, i.e., its performance should be close to that of the online algorithm without predictions (what is called *robustness*).

2.2.2 A new model for online computation with untrusted advice

Motivated by the above definitions from machine learning, in this work we analyze online algorithms based on their performance in both settings of trusted and untrusted advice. In particular, we will 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 achieved assuming that the advice encodes precisely what it is meant to capture; we call this ratio the competitive ratio with *trusted* (thus, always correct) advice. In contrast, w_A is the competitive ratio of A when the advice is *untrusted* (thus, potentially wrong). More precisely, in accordance with the worst-case nature of competitive analysis, we allow the incorrect advice to be chosen *adversarially*. Namely, assuming a deterministic online algorithm A, the incorrect advice string is generated by a malicious, adversarial entity.

To formalize the above concept, assume the standard advice model, in which a deterministic online algorithm A processes a sequence of requests $\sigma = (\sigma[i])_{i\in[1,n]}$ using an advice string ϕ. At each time t, A serves request $\sigma[t]$, and its output is a function of $\sigma[1.\ldots.t-1]$ and ϕ. Assuming a cost minimization problem, let $A(\sigma,\phi)$ denote the cost incurred by A on input σ, using an advice string ϕ. We can define r_A, and w_A as

$$r_A = \sup_{\sigma} \inf_{\phi} \frac{A(\sigma,\phi)}{\textsc{Opt}(\sigma)}, \quad \text{and} \quad w_A = \sup_{\sigma} \sup_{\phi} \frac{A(\sigma,\phi)}{\textsc{Opt}(\sigma)}, \tag{2.1}$$

where $\text{OPT}(\sigma)$ denotes the optimal offline cost for σ. Then we say that algorithm A is (r, w)-competitive for every $r \geq r_A$ and $w \geq w_A$. In addition, we say that A has advice complexity $s(n)$ if for every request sequence σ of length n, A depends only on the first $s(n)$ bits of the advice string ϕ. To illustrate this definition, the opportunistic 1-bit advice algorithm for ski rental that was described above is $(1, \infty)$-competitive. In contrast, the standard competitively optimal algorithm without advice is $(2, 2)$-competitive. In general, every online algorithm A without advice is trivially (w, w)-competitive, where w is the competitive ratio of A.

Hence, we can associate every algorithm A to a point in the 2-dimensional space with coordinates (r_A, w_A). These points are in general incomparable, e.g., it is difficult, or undesirable, to argue that a $(2, 10)$-competitive algorithm is better than a $(4, 8)$-competitive algorithm. However, one can appeal to the notion of *dominance*, by saying that algorithm A dominates algorithm B if $r_A \leq r_B$ and $w_A \leq w_B$. More precisely, we are interested in finding the Pareto frontier in this representation of all online algorithms. For the ski rental example, the two above mentioned algorithms belong to the Pareto set.

A natural goal is to describe this Pareto frontier, which in general, may be comprised of several algorithms with vastly different statements. Ideally, however, one 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. Regardless of optimality, given $\mathcal{A}$, we will describe its competitiveness by means of a function $f : \mathbb{R}_{\geq 1} \to \mathbb{R}_{\geq 1}$ such that for every ratio r there is an $(r, f(r))$-competitive algorithm in $\mathcal{A}$. This function will in general depend on parameters of the problem, such as, for example, the buying cost B in the ski rental problem.

We study various online problems in this new setting of untrusted advice. We also demonstrate that it is possible to establish both upper and lower bounds on the tradeoff between the size of the advice and the competitiveness in this advice model. In particular, we study the ski rental, online bidding, online bin packing, and the list update problems (Section 2.2.3). All the above results pertain to deterministic online algorithms. In Section 2.2.4, we make some further observations concerning the power of randomization in online computation with untrusted advice.

While our work addresses issues similar to [109] and [115], in that trusted advice is related to consistency whereas untrusted advice is related to robustness, it differs in two significant aspects: First, our ideal objective is to identify an optimal family of algorithms, and we show that in some cases (ski rental, online bidding), this is indeed possible; when this is not easy or possible, we can still provide approximations. Note that finding a Pareto-optimal family of algorithms presupposes that the exact competitiveness of the online problem with no advice is known. For problems such as bin packing, the exact optimal competitive ratios are not known. Hence, a certain degree of approximation is unavoidable in such cases. In contrast, [109, 115] focus on "smooth" tradeoffs between the trusted and untrusted competitive ratios, but do not address the issues related to optimality or the approximability of these tradeoffs.

Second, our model considers the size of advice and its impact on the algorithm's performance, which is the main focus of the advice complexity field. In general, we would like to parameterize advice by its size, i.e., to allow advice of a certain size. This opens up more possibilities to the algorithm designer in regards to the choice of an appropriate advice oracle, which may have further practical applications in machine learning.

2.2.3 Applications of the framework

In this section we show how the framework applies to several online problems. For definitions of the problems, and previous work on online algorithms without advice see Section 1.1.5.

Ski rental

We denote by D the number of skying days (unknown to the algorithm) and by B the cost of buying skis. Consider the single-bit advice setting. Suppose that the advice encodes whether to buy on day 1, or always rent. An algorithm that blindly follows the advice is optimal if the advice is trusted, but, if the advice is untrusted, the competitive ratio can be as high as D/B, if $D > B$. Hence, this algorithm is $(1, \infty)$-competitive, for $D \to \infty$.

We now move to the setting of untrusted advice. We define the family of algorithms A_k, with parameter $0 < k \leq B$ as follows. There is a single bit of advice, which is the indicator of the event $D < B$. If the advice bit is 1, then A_k rents until until day $B - 1$ and buys on day B. Otherwise, the algorithm buys on day k. It is easy to bound the performance of this class of algorithms as follows.

Proposition 25. *Algorithm A_k is $(1 + \frac{k-1}{B}, 1 + \frac{B-1}{k})$-competitive.*

Our algorithm A_k is slightly different from the one proposed in [115], which buys on day $\lceil B/k \rceil$ if the advice is 1 and is shown to be $(1 + k/B, 1 + B/k)$-competitive. More importantly, we show that A_k is Pareto-optimal in the space of all deterministic online algorithms with advice of *any size*. This implies that more than a single bit of advice will not improve the tradeoff between the trusted and untrusted competitive ratios.

Theorem 26. *For any deterministic $(1 + \frac{k-1}{B}, w)$-competitive algorithm A, with $1 \leq k \leq B$, with advice of any size, it holds that $w \geq 1 + \frac{B-1}{k}$.*

Proof. Let A be an algorithm with trusted competitive ratio at most $1 + \frac{k-1}{B}$. First, note that if the advice is untrusted, the competitive ratio cannot be better than the competitive ratio of a purely online algorithm. For ski-rental, it is known that no online algorithm can achieve a competitive ratio better than $1 + (B-1)/B$ [94]. So, in the case $k = B$, the claim trivially holds. In the remainder of the proof, we assume $k < B$.

We use σ_D to denote the instance of the problem in which the number of skiing days is D, and use $A_t(\sigma_D)$ to denote the cost of A for σ_D in case of trusted advice.

Consider a situation in which the input is σ_{B+k} and the advice for A is trusted. Let j be the day the algorithm will buy under this advice. Since the advice is trusted and thus $\textsc{Opt}(\sigma_{B+k}) = B$, it must be that

$$A_t(\sigma_{B+k}) \leq \left(1 + \frac{k-1}{B}\right) \textsc{Opt}(\sigma_{B+k}),$$

which implies $j < B + k$. In other words, A indeed buys on day j. We conclude that $A_t(\sigma_{B+k}) = j - 1 + B$ which further implies $j \leq k$.

Let x be the trusted advice A receives on input σ_{B+k} and suppose A receives the same advice x on input σ_j. Note that x can be trusted or untrusted for σ_j. The important point is that A serves σ_j in the same way it serves σ_{B+k}, that is, it rents for $j - 1$ days and buys on day j. The cost of A for σ_j is then $j - 1 + B$, while $\textsc{Opt}(\sigma_j) = j$. The ratio between the cost of the algorithm and Opt is therefore $1 + \frac{B-1}{j}$, which is at least $1 + \frac{B-1}{k}$ since $j \leq k$. Note that $1 + \frac{B-1}{k} > 1 + \frac{k-1}{B}$ (since we assumed $k < B$) and therefore the advice in this situation has to be untrusted, by the assumption on the trusted competitive ratio of A. We conclude that the untrusted competitive ratio must be at least $1 + (B-1)/k$. □

Online bidding

Recall the definition of the problem given in Section 1.1.5. Without advice, the best competitive ratio for this problem is 4, and can be achieved using the doubling strategy in which the i-th bid is equal to 2^i. If the advice encodes the value u, and assuming trusted advice, bidding $x_1 = u$

is a trivial optimal strategy. The above observations imply that there are simple strategies that are $(4, 4)$-competitive and $(1, \infty)$-competitive, respectively.

Let $w \geq 4$ is a fixed, given parameter. Our problem is stated as follows: What is the smallest r_w (as function of w) such that there is a (r_w, w)-competitive algorithm for online bidding? We will address this problem in two settings. First, we will assume that the advice encodes the exact value of the target u (Theorem 27), and thus can be unbounded in size. Then, we will consider the case in which the advice is a bounded string of k bits (Theorems 28 and 29).

We begin with the case in which the advice encodes u, and we summarize how to obtain a Pareto-optimal bidding strategy X^*_u. We will need some definitions. Since the index of the bid which reveals the value will be important in the analysis, we define the class $S_{m,u}$, with $m \in \mathbb{N}^+$ as the set of bidding strategies with advice u which are w-competitive (assuming no advice), and which, if the advice is trusted, succeed in finding the value with precisely the m-th bid. 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, it holds that $r \leq r'$.

The high-level idea is to identify, for any given m, a dominant strategy in $S_{m,u}$. 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 $L_{m,u}$, and which is shown below. For convenience, for any strategy X, we will always define x_0 to be equal to 1.

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

In this LP, the constraints (C_i) guarantee w-competitiveness if the advice is untrusted. The constraint $x_m = u$ follows from the fact that any dominant strategy discovers the target u with a bid exactly equal to u (otherwise we could improve the solution by scaling, a contradiction). The objective describes then the trusted competitive ratio.

Next, define $r^*_u = \inf_m r^*_{m,u}$, and $r^* = \sup_u r^*_u$. In words, 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.

It is indeed possible to compute $r^*_{m,u}$ and the corresponding strategy $X^*_{m,u}$, provided that $L_{m,u}$ is feasible. The main idea behind the technical proof is to show that in an optimal solution of $L_{m,u}$, all constraints C_i with $i \geq 2$ hold with equality. This allows us to describe the bids of the optimal strategy by means of a linear recurrence relation which we can solve so as to obtain an expression for the bids of $X^*_{m,u}$.

We can now describe, at a high level, the optimal strategy X^*_u. First, we can argue that the optimal objective value of $L_{m,u}$ is monotone increasing in m, thus 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 $w \geq 4$.

Putting everything together we obtain the following result.

Theorem 27. *Strategy X^*_u is Pareto-optimal and is $(\frac{w-\sqrt{w^2-4w}}{2}, w)$-competitive.*

For example, if $w = 4$, then the optimal bidding strategy is $(2, 4)$-competitive.

Strategy X_u^* requires u as advice, which can be unbounded. A natural question is what competitiveness can one achieve with only k advice bits, for some fixed k. We address this question both from the point of view of upper and lower bounds. Concerning upper bounds, we can use k bits to choose the best among 2^k appropriately defined bidding strategies, which leads to the following theorem.

Theorem 28. *For every $w \geq 4$, there exists a bidding strategy with k bits of advice which is (r, w)-competitive, where*

$$r = \begin{cases} \frac{\left(w+\sqrt{w^2-4w}\right)^{1+1/K}}{2^{1/K}(w+\sqrt{w^2-4w}-2)} & \text{if } w \leq (1+K)^2/K \\ \frac{(1+K)^{1+1/K}}{K} & \text{if } w \geq (1+K)^2/K. \end{cases}$$

and where $K = 2^k$.

In particular, for $w = 4$, the strategy of Theorem 28 is $(2^{1+\frac{1}{2^k}}, 4)$-competitive, whereas recall that X_u^* is $(2, 4)$-competitive. We complement this result with the following theorem, which gives a lower bound on the competitiveness of any bidding strategy with k bits. The result shows that one needs an unbounded number of bids to achieve $(2, 4)$-competitiveness. The main idea behind the proof is that with k bits of advice, the online algorithm can only differentiate between $K = 2^k$ online bidding sequences. Using a game between the algorithm and an adversary, we can then show that there has to be at least one choice for the hidden value for which all K bidding sequences are suboptimal.

Theorem 29. *For any bidding strategy with k advice bits that is $(r, 4)$-competitive it holds that $r \geq 2 + \frac{1}{3 \cdot 2^k}$.*

Online bin packing

For previous work on this problem in the setting of no advice, refer to Section 1.1.5. Online bin packing has also been studied in the advice setting [54, 117, 16]. In particular, a result of [16] shows that it is possible to achieve a competitive ratio of 1.4702 with only a constant number of (trusted) advice bits [16].

Our algorithm for the setting of untrusted advice uses the an algorithm introduced by Boyar et al. [54] which achieves a competitive ratio of 1.5 using $O(\log n)$ bits of advice [54]. We refer to this algorithm as Reserve-Critical in the remainder of this section, and describe it briefly in the next paragraph.

Reserve-Critical classifies items according to their size. Tiny items have their size in the range $(0, 1/3]$, small items in $(1/3, 1/2]$, critical items in $(1/2, 2/3]$, and large items in $(2/3, 1]$. In addition, the algorithm has four kinds of bins, called tiny, small, critical and large bins. Large items are placed alone in large bins, which are opened at each arrival. Small items are placed in pairs in small bins, which are opened every other arrival. Critical bins contain a single critical item, and tiny items up to a total size of $1/3$ per bin, while tiny bins contain only tiny items. The algorithm receives as advice the number of critical items, denoted by c, and opens c *critical bins* at the beginning. Inside each critical bin, a space of $2/3$ is reserved for a critical item, and tiny items are placed using First-Fit into the remaining space of these bins possibly opening new bins dedicated to tiny items. Each critical item is placed in one of the critical bins.

Note that the algorithm is heavily dependent on the advice being trusted. Imagine that the encoded advice is strictly larger than the real number of critical items. This results in critical bins which contain only tiny items. The worst case is reached when all tiny items have size slightly more than $1/6$ while there is no critical item. In this case, all critical bins are filled up to a level slightly more than $1/6$. Hence, untrusted advice can result in a competitive ratio as bad as 6 (which is quit bad for bin packing standards of approximation).

An idea for drastically reducing the number of advice bits, without compromising much the performance was used in [16]. Let t be the number of tiny bins opened by the algorithm, then the idea is to use the advice so as to approximate the fraction $c/(c+t)$ using only k bits (constant in number) instead of the exact quantity c. We call this fraction the *critical ratio*, and we refer to its approximation using k bits as γ. Then the algorithm would open critical and tiny bins as needed, maintaining a proportion between them close to the given critical ratio.

We are now ready to define the online bin packing algorithm for untrusted advice, to which we refer as the Robust-Reserve-Critical (RRC) algorithm. The RRC algorithm has a parameter $0 \leq \alpha \leq 1$, which together with the advice γ can be used to define a fraction $\beta = \min\{\alpha, \gamma\}$. The algorithm maintains a proportion close to β of critical bins among critical and tiny bins. Formally, on the arrival of a critical item, the algorithm places it in a critical bin, opening a new one if necessary. Each arriving tiny item x is packed in the first critical bin which has enough space, with the restriction that the tiny items don't exceed a fraction $1/3$ in these bins. If this fails, the algorithm tries to pack x in a tiny bin using First-Fit (this time on tiny bins). If this fails as well, a new bin B is opened for x. Now, B should be declared as a critical or a tiny bin. Let c' and t' denote the number of critical and tiny bins before opening B. If $c' + t' > 0$ and $\frac{c'}{c'+t'} < \beta$, then B is declared a critical bin; otherwise, B is declared a tiny bin. Large and small items are placed similarly to Reserved-Critical (one large item in each large bin and two small items in each small bin).

We obtain the following results concerning the performance of the RRC algorithm.

Theorem 30. *Algorithm Robust-Reserve-Critical with parameter $\alpha \in [0,1]$ and k bits of advice achieves a competitive ratio $r_{\text{RRC}} \leq 1.5 + \max\{\frac{1-\alpha}{4-3\alpha}, \frac{15}{2^{k/2+1}}\}$ when the advice is trusted and a competitive ratio $w_{\text{RRC}} \leq 1.5 + \max\{\frac{1}{4}, \frac{9\alpha}{8-6\alpha}\}$ when the advice is untrusted.*

Corollary 31. *For bin packing with untrusted advice, and assuming that the size k of the advice is a sufficiently large constant, RRC is $(r, f(r))$-competitive, where $r \geq 1.5$ and $f(r) = \max\{33 - 18r, 7/4\}$.*

List update

For previous work on this problem in the setting of no advice, and for algorithms to which we refer to in this section, see Section 1.1.5. Concerning (trusted) advice, Boyar *et al.* [53] showed that, for any request sequence, at least one of Timestamp, Mtfo, and Mtfe has a competitive ratio of at most 5/3. Thus, for a given request sequence, the best option among the three algorithms can be indicated with two bits of advice, giving a 5/3-competitive algorithm. However, if the advice is untrusted, the competitive ratio of this algorithm can be as bad as 5/2.

To address this issue, and obtain a better tradeoff, we introduce an algorithm which we call Toggle (Tog) that has a parameter $\beta \in [0, 1/2]$, and uses 2 advice bits to select one of the algorithms Timestamp, Mtfe or Mtfo. This algorithm achieves a competitive ratio $r_{\text{Tog}} = 5/3 + \frac{5\beta}{6+3\beta}$ when the advice is trusted and a competitive ratio at most $w_{\text{Tog}} = 2 + 2/(4+5\beta)$ when the advice is untrusted. The parameter β can be tuned and should be smaller when the advice is more reliable. In particular, when $\beta = 0$, we obtain a $(5/3, 5/2)$-competitive algorithm.

More precisely, given the parameter β, Tog works as follows. If the advice indicates Timestamp, the algorithm runs Timestamp. If the advice indicates either Mtfo or Mtfe, the algorithm will proceed in phases (the length of which partially depend on β) alternating ("toggling") between running Mtfe or Mtfo, and Mtf. In what follows, we use Mtf2 to represent the algorithm indicated by the advice. Tog will initially begin with Mtf2 until the cost of the accesses of the phase reaches a certain threshold, then a new phase begins and Tog switches to Mtf. This new phase ends when the access cost of the phase reaches a certain threshold, and Tog switches back to Mtf2. This alternating pattern continues as Tog serves the requests, namely Tog will use Mtf2 for the odd phases, and Mtf for the even phases.

The following two results summarize the performance of our algorithm.

Theorem 32. *Algorithm* TOG *with parameter* $\beta \in [0, 1/2]$ *and* 2 *bits of advice achieves a competitive ratio at most* $5/3 + \frac{5\beta}{6+3\beta}$ *when the advice is trusted and a competitive ratio at most* $2 + \frac{2}{4+5\beta}$ *when the advice is untrusted.*

Corollary 33. *For list update with untrusted advice,* TOG *is* $(r, f(r))$*-competitive where* $r \in [5/3, 2]$ *and* $f(r) = 2 + \frac{10-3r}{9r-5}$.

2.2.4 Randomized online algorithms with untrusted advice

The discussion in all previous sections pertains to deterministic online algorithms. In this section we focus on randomization and its impact on online computation with untrusted advice. We will assume, as standard in the analysis of randomized algorithms, that the source of randomness is trusted (unlike the advice).

First, we will argue that randomization can improve the competitiveness of the ski rental problem. For this, we note that [115] gave a randomized algorithm with a single advice bit for this problem which is $\left(\frac{\lambda}{1-e^{-\lambda}}, \frac{1}{1-e^{-(\lambda-1/B)}}\right)$-competitive, where $\lambda \in (1/B, 1)$ is a parameter of the algorithm. For simplicity, we may assume that B is large, hence this algorithm is $\left(\frac{\lambda}{1-e^{-\lambda}}, \frac{1}{1-e^{-\lambda}}\right)$-competitive, which we can write in the equivalent form $(w \ln \frac{w}{w-1}, w)$. In contrast, Theorem 26 shows that any deterministic Pareto-optimal algorithm with advice of any size is $(1+\lambda, 1+1/\lambda)$-competitive, or equivalently $(\frac{w}{w-1}, w)$-competitive. Standard calculus shows that $w \ln \frac{w}{w-1} < \frac{w}{w-1}$; therefore we conclude that the randomized algorithm Pareto-dominates any deterministic algorithm, even when the latter is allowed unbounded advice.

A second issue we address in this section is related to the comparison of random bits and advice bits as *resource*. More specifically, in the standard model in which advice is always trustworthy, an advice bit can be at least as powerful as a random bit since the former can simulate the efficient choice of the latter, and thus provide a "no-loss" derandomization. However, in the setting of untrusted advice, the interplay between advice and randomization is more subtle. This is because random bits, unlike advice bits, are assumed to be trusted.

We show, using online bidding as an example, that there are situations in which a deterministic algorithm with $L + 1$ advice bits is Pareto-incomparable to a randomized algorithm with 1 random bit and L advice bits. In particular we focus on the *bounded online bidding* problem, in which $u \leq B$, for some given B.

Theorem 34. *For every* $\epsilon > 0$ *there exist sufficiently large* B *and* L *such that there is a randomized algorithm for bounded online bidding with* L *advice bits and 1 random bit which is* $(\frac{1+\rho_1}{2}\rho_1 + \epsilon, \frac{1+\rho}{2\rho}w + \epsilon)$*-competitive for all* $w > 4$*, where* $\rho = \frac{w-\sqrt{w^2-4w}}{2}$.

Note that when $B, L \to \infty$, the competitiveness of the best deterministic algorithm with L advice bits approaches the one of X_u^*, as expressed in Theorem 27, namely (ρ, w). Thus, Theorem 34 shows that randomization improves upon the deterministic untrusted ratio w by a factor $\frac{1+\rho}{2\rho} < 1$, at the expense of a degradation of the trusted competitive ratio by a factor $\frac{1+\rho}{2} > 1$. For instance, if w is close to 4, then the randomized algorithm has untrusted competitive ratio less than 4, and thus better than any deterministic strategy.

2.3 Discussion

In this chapter we introduced a new measure and a new model that extend their established counterparts in the field of online computation. We showed how the bijective ratio can extend the notion of bijective analysis and make it applicable to all online algorithms. We also showed how to extend the advice complexity model of online computation to a setting in which the advice is not necessarily trustworthy.

Concerning bijective analysis, we conjecture that the bijective ratio of GREEDY for continuous k-server on the line is k. One way of approaching this question is to find appropriate bijections that map sequences in such a way that if GREEDY is in a "good" configuration relative to another algorithm before request i, it remains in a similarly good configuration after request i. However, such an approach may be fairly involved, since it requires bijections with specific rules for breaking ties with respect to the costs of requests (unlike the bijection we rely to in this work).

The ultimate objective is to apply the bijective ratio in the analysis of k-server problems on discrete metric spaces. It is known that even for the discrete line, the max/max ratio of GREEDY is at least ϕ^k, where ϕ is the golden ratio [36], hence the same lower bound applies to its bijective ratio. But is the bijective ratio of GREEDY only exponential in k? Is it constant for constant k? Perhaps the most important open question is to design algorithms of bijective ratio better than $2k-1$, which is the best known competitive ratio of the work function algorithm. We note that in this work we already showed a setting in which this is indeed possible, namely for continuous k-server on the circle we proved that GREEDY has bijective ratio k, whereas WFA is not known to be better than $2k-1$ even on such metrics.

Concerning online computation with untrusted advice, we observed that an efficient advice scheme should address the issues of "what constitutes good advice" as well as "how the advice should be used by the algorithm". We observed that, for certain online problems, some of the known algorithms with advice perform poorly in the case the advice is untrusted. To address this, we gave algorithms that can be "tuned" based on how much we are willing to trust the advice. This enables us to show guarantees in the form $(r, f(r))$-competitiveness, where r is strictly better than the competitive ratio of all deterministic online algorithms and $f(r)$ smoothly decreases as r grows, while still being close to the worst-case competitive ratio.

To illustrate this, consider the bin packing problem, and in particular Corollary 31. Our $(r, f(r))$-competitive algorithm is such that $f(r) = \max\{33-18r, 7/4\}$ for any $r \geq 1.5$. If $r = 1.5$, our algorithm is $(1.5, 6)$-competitive, and matches the performance of a known algorithm [54]. However, with a slight increase of r, one can improve competitiveness in the event the advice is untrusted. For instance, choosing $r = 1.55$, we obtain $f(r) = 5.1$. In other words, the algorithm designer can hedge against untrusted advice, by a small sacrifice in the trusted performance. Thus we can interpret r as the "risk" for trusting the advice: the smaller the r, the bigger the risk.

The model we introduced is a first step towards narrowing the very big gap between the ideal world of advice complexity, and the realistic world of machine learning with predictions. A next step is to express the performance of the algorithm as a function of the *error* in the advice. As an example, in our Pareto-optimal online bidding of Theorem 27, if there is the slightest error in the advice u, the algorithm is as bad as any online algorithm without advice, for a worst-case, adversarial choice of the hidden value u. In future work, we would like to be able to express the trusted competitive ratio as a function of the advice error, while still accounting for the advice size.

Chapter 3

Online search

In this chapter we focus on online search. First, we study the setting in which the searcher incurs a fixed cost every time it changes direction, and give optimal and near-optimal solutions for m-ray search under both single and multiple searchers, respectively 3.1 (Section 3.1). Then we introduce and study a weighted variant of search games in which there are multiple targets, each one with its own weight, and the objective is to minimize the competitive ratio for accumulating a certain overall weight (Section 3.2). In Section 3.3 we address linear search from the point of view of bijective analysis. Last, in Section 3.4 we analyze the competitive ratio of expanding search.

The material of Section 3.1 is based on work that appeared in *Theoretical Computer Science* [10] and *Theory of Computing Systems* [11]. Section 3.2 is based on work that appeared in *Theoretical Computer Science* [22] as well as on work that is currently under review [23]. Section 3.3 appeared in the *36th International Symposium on Theoretical Aspects of Computer Science* [14]. Last, Section 3.4 describes work that appeared in *Discrete Applied Mathematics* [17] and the *European Journal of Operations Research* [19].

3.1 Online ray search with turn cost

3.1.1 Background

In this section we study m-ray searching in the setting in which the searcher incurs a fixed *turn cost* upon changing direction, and the overall *cost* incurred by the searcher is the sum of the total distance traversed by the searcher and the total turn cost. This formulation models the often-encountered setting in which changing a searcher's direction is a time-consuming operation which cannot be ignored. For instance, in robotics, turning is a complex operation, and a robot cannot turn instantaneously.

This setting was introduced and studied by Demaine *et al.* [66]. Let us begin with a formal definition of the problem. We consider first the case of a single searcher.

We will assume that there are costs d_1 and d_2 for turning at a ray and at the origin, respectively; hence the turn cost incurred by a searcher on a single ray exploration is $d = d_1 + d_2$. We say that a search algorithm is (α, β)-*competitive* if for any placement of the target at distance h from the origin, the search cost is at most $\alpha h + \beta$. In particular, under the assumption that the target is never placed within distances smaller than a specified fixed constant and for zero turn cost, previous work has established optimal $(1 + 2M, 0)$-competitive strategies, as explained in Section 1.2, and in particular in (1.1).

The question we address is then the following: What is the smallest B such that a a strategy is $(1+2M, B)$ competitive, with no assumptions on the target placement? Note that this question becomes non-trivial only in the presence of turn cost, since otherwise B is zero, as argued in [66].

In the context of this problem, we say that a strategy S is *no worse* than strategy S' if S is

$(1+2M, B)$-competitive and S' is $(1+2M, B')$-competitive, with $B' \geq B$. The *optimal* strategy is the strategy that achieves the minimum possible B. It is also easy to show that the worst-case positions for placing the target are right after the turn point of each ray exploration (since every other possible placement cannot affect the worst-case competitiveness).

3.1.2 LP formulations of star search with turn cost

Demaine *et al.* [66] addressed this problem using an approach based on infinite linear programming (LP) formulations. More specifically, in order to lower-bound the cost of any search strategy, [66] defines an infinite series of linear programs, with each linear program describing a (progressively better) set of adversarial target placements. At the limit, the optimal value of the infinite LP gives the strongest lower bound. The approach of [66] consists of solving experimentally this series of finite LPs, then guessing a solution to the infinite LP, and finally providing a proof of optimality based on appropriate duality properties of infinite LP formulations. More precisely, they claimed that for every search strategy there is a placement of the target at a certain distance h from the origin such that the strategy incurs a (tight) cost of $(1+2M)h + (M-m)d$. Furthermore, they showed that this bound is tight, by analyzing a specific cyclic strategy.

We begin with a review of the approach in [66]. First, it is easy to argue, by an exchange argument, that the optimal strategy must be round-robin (cyclic). Let $x_1, x_2, \ldots$ denote the (infinite) sequence of distances in which the strategy cycles through the m rays. Suppose that the adversary places the target just beyond the turn point that corresponds to distance x_{i+1}, with $i \geq 0$. In this case, the cost of locating the target is $2\sum_{j=1}^{m+i} x_j + x_{i+1} + (m+i)d$. Therefore, in order for the algorithm to be $(1+2M, B)$-competitive we require that $2\sum_{j=1}^{m+i} x_j + x_{i+1} + (m+i)d \leq (1+2M)x_{i+1} + B$, or equivalently, that $2\sum_{j=1}^{m+i} x_j + (m+i)d \leq 2Mx_{i+1} + B$. In addition, the adversary may choose to place the target arbitrarily close to the origin, on the last ray to be explored in the first round, which implies the condition $2(\sum_{j=1}^{m-1} x_j) + (m-1)d \leq B$. Combining the above requirements, we obtain a family of linear programs. The k-th LP of this family is

$$
\begin{array}{lll}
\min & B & (\mathcal{P}) \\
\text{s.t.} & 2\sum_{j=1}^{m-1} x_j - B \leqslant -d(m-1) & \\
& 2\sum_{j=1}^{m+i} x_j - 2Mx_{i+1} - B \leqslant -d(m+i) \qquad \forall i = 0 \ldots k & \\
& B, x_1, \ldots, x_{m+k} \geqslant 0, &
\end{array}
$$

with dual LP

$$
\begin{array}{lll}
\max & \left((m-1)z + \sum_{i=0}^{k} y_i(m+i)\right) d & (\mathcal{D}) \\
\text{s.t.} & z + \sum_{i=0}^{k} y_i \leqslant 1 & (B\text{-constraint}) \\
& \left\{ \begin{array}{ll} z\ , & j \leq m-1 \\ 0\ , & \text{otherwise} \end{array} \right\} + \sum_{i=\max(0,j-m)}^{k} y_i & \\
& \quad - \left\{ \begin{array}{ll} My_{j-1}\ , & j \leq k+1 \\ 0\ , & \text{otherwise} \end{array} \right\} \geqslant 0 \qquad \forall j = 1 \ldots m+k & (x_j\text{-constraint}) \\
& z, y_0, \ldots y_k \geqslant 0 &
\end{array}
$$

We call k the *index* of the LP.

For every fixed k, the objective value of any feasible solution to the LP $(\mathcal{D})$ is a lower bound on B. Thus, the best lower bound on B is obtained by the optimal solution to the dual LP, when $k \to \infty$. In [66], a different approach is proposed: it is argued that a lower bound on B can be

obtained by finding a feasible solution to the following LP, which we call the *infinite dual LP*. Essentially, $(\mathcal{D}^\infty)$ is obtained as the dual of a (primal) infinite LP $(\mathcal{P}^\infty)$ which in turn is derived from $(\mathcal{P})$ by setting $k = \infty$.

$$\begin{aligned} \max \quad & \Big((m-1)z + \textstyle\sum_{i=0}^{\infty} y_i(m+i)\Big)\, d && (\mathcal{D}^\infty) \\ \text{s.t.} \quad & z + \textstyle\sum_{i=0}^{\infty} y_i \leqslant 1 \\ & \left\{ \begin{array}{ll} z\ , & j \leq m-1 \\ 0\ , & \text{otherwise} \end{array} \right\} + \textstyle\sum_{i=\max(0,j-m)}^{\infty} y_i - M y_{j-1} \geqslant 0 \qquad \forall j = 1, 2, \ldots \\ & z, y_0, y_1, \ldots \geqslant 0 \end{aligned}$$

In particular, [66] proposes the following recursively defined solution

$$\begin{aligned} z \quad &= \quad \frac{m}{M}, \quad y_0 = y_1 = y_{m-2} = \ldots = \frac{1}{M}, \quad y_{m-1} = \frac{1}{M}(1-z), \\ &\text{and} \quad y_i = y_{i-1} - \frac{1}{M} y_{i-m}, \text{ for all } i \geq m, \end{aligned} \tag{3.1}$$

which is then shown to be feasible for the infinite dual LP, and has objective value equal to $(M - Mz)d = (M - m)d$.

We now argue that relying on a solution that is only feasible for the infinite dual LP $(\mathcal{D}^\infty)$ can lead to erroneous results. In particular, Theorem 35 proves that one can give a solution that is feasible for the infinite dual LP, and whose objective value equals Md, which exceeds the upper bound of $(M - m)d$ shown in [66]. This demonstrates that one cannot necessarily rely on solutions to the infinite dual LP, and instead must first provide feasible solutions for finite LP formulations, then evaluate their objective value when $k \to \infty$.

Theorem 35. *Consider the solution that is recursively defined as in* (3.1)*, with the exception that $z = 0$. Then this solution is feasible for the infinite dual LP, and its objective value is equal to Md.*

It is worth mentioning that the reason we arrive at this contradictory result is that the dual solution defined in the statement of Theorem 35, while feasible for $\mathcal{D}^\infty$, does not give rise to a feasible solution for any finite dual LP (i.e., for any given k, the solution $\{z = 0, y_0, \ldots, y_k\}$ is infeasible for the dual LP of index k). It is also worth pointing out that strong duality, and often weak duality, are not always satisfied in infinite LPs. We refer the reader to the work of Karney [97] and Romeijn *et al.* [118] for a discussion of conditions under which (weak or strong) duality can be established for certain families of infinite LPs.

3.1.3 Finding a feasible solution to the dual LP

In this section we argue how to find a feasible dual solution to the LP $(\mathcal{D})$ for every index $k \geq m$. This will allow us to establish in a correct manner the main result of [66]. The best dual solution, namely the one that yields the best lower bound will be derived when $k \to \infty$. What complicates things is that, given indices k_1, k_2, with $k_1 < k_2$, the dual solution to the LP of index k_1 cannot be obtained, in a straightforward manner, from the dual solution to the LP of index k_2 (informally, the latter is not "contained" in the former). We thus fix k and propose a feasible dual solution that is parameterized on k. To this end, we need first to establish certain properties of the recurrence relation

$$y_i = y_{i-1} - \frac{1}{M} y_{i-m}, \qquad \text{with } i \in \mathbb{N}, \tag{3.2}$$

where $M = \frac{m^m}{(m-1)^{m-1}}$ and the initial data are

$$y_0 = y_1 = \ldots = y_{m-2} = \frac{1}{M}, \qquad y_{m-1} = \frac{1-z}{M}, \qquad \text{with } 0 \leq z \leq 1. \tag{3.3}$$

We define y as the sequence $\{y_i\}_{i=0}^{\infty}$. Note that since m and M are fixed, the only parameter that influences the initial data, and, implicitly, the asymptotic behavior of the sequence y, is z, which we call the *seed* of y.

The following is an important technical theorem concerning the recurrence relation (3.2). The theorem shows that we can choose appropriate seed values z such that either y is a positive sequence, or y eventually becomes negative, respectively. This will allow us to argue that $y_{k+m} = 0$ for some seed value, which in turn yields the tightness of the crucial B-constraint (Theorem 37).

Theorem 36. *Let $0 < \zeta < 1$ be given by*

$$\zeta = \left(\frac{m-1}{m}\right)^{m-1} = \frac{m}{M}, \tag{3.4}$$

and consider the infinite sequence $y = \{y_i\}_{i=0}^{\infty}$ with seed $0 \le z \le 1$.

Then, $y_i > 0$ for every $i \in \mathbb{N}$ if and only if $0 \le z \le \zeta$. Furthermore, for every $i_0 \ge m$, there exists z with $\zeta < z < 1$, such that $y_{i_0} = 0$, $y_i > 0$ for all $i < i_0$, and $y_i < 0$ for all $i > i_0$.

We now show how to apply Theorem 36 so as to obtain a feasible solution to the dual LP of index k.

Theorem 37. *For every fixed index $k \ge m$, there exists $\zeta < z^* \le 1$ such that the solution $\{z^*, y_0, \ldots, y_k\}$ is feasible for the dual LP of index k.*

Proof. First, from Theorem 36, we know that there is some $z^* \in (\zeta, 1)$ for which $y_{m+k} = 0$ and $y_0, \ldots, y_k > 0$, whence the solution $\{z^*, y_0, \ldots, y_k\}$ satisfies the non-negativity constraints.

Next, we show that the B-constraint is tight. To this end, by summing up the equations of the form $y_i = y_{i-1} - \frac{1}{M} y_{i-m}$ for $i = m, m+1, \ldots, m+k$ we obtain that $\sum_{i=0}^{k} y_i = M(y_{m-1} - y_{k+m})$. Moreover, we have that $y_{k+m} = 0$ for seed $z = z^*$. Hence, $\sum_{i=0}^{k} y_i = M y_{m-1} = M\left(\frac{1}{M} - \frac{1}{M} z^*\right) = 1 - z^*$. Thus we obtain

$$z^* + \sum\nolimits_{i=0}^{k} y_i = z^* + 1 - z^* = 1, \tag{3.5}$$

thus the B-constraint is tight.

It remains to show that the j-th constraint is satisfied for all $j \in [1, m+k]$. Let us denote by $\mathrm{LHS}(j)$ the LHS of the j-th constraint; we also define

$$\mathrm{L}(j) := \left\{ \begin{array}{l} z\ , \ \ j \le m-1 \\ 0\ , \ \ \text{otherwise} \end{array} \right\} + \sum\nolimits_{i=\max(0, j-m)}^{k} y_i - M y_{j-1}.$$

Clearly $\mathrm{LHS}(j) \ge \mathrm{L}(j)$, therefore in order to show that that the j-th constraint is satisfied, it suffices to show that $\mathrm{L}(j) \ge 0$. When $j \le m-1$, then $\mathrm{L}(j) = z^* + \sum_{i=0}^{k} y_i - M y_{j-1} = z^* + \sum_{i=0}^{k} y_i - M\frac{1}{M} = 1 - 1 = 0$, from (3.5). Similarly, when $j = m$, we have $\mathrm{L}(m) = \sum_{i=0}^{k} y_i - M y_{m-1} = \sum_{i=0}^{k} y_i - M\frac{1-z^*}{M} = 0$, again from (3.5).

Last, we will use induction to show that $\mathrm{L}(j) = 0$ when $j > m$. For any $j > m$ we have

$$\begin{aligned} \mathrm{L}(j+1) - \mathrm{L}(j) &= -M y_j + M y_{j-1} - y_{j-m} = \\ &- M(y_{j-1} - \frac{1}{M} y_{j-m}) + M y_{j-1} - y_{j-m} = 0, \end{aligned}$$

where the second-to-last equation follows from the definition of the recurrence relation. □

We are now ready to formally prove the main claim of [66], namely that for every $(1+2M, B)$-competitive strategy, $B \ge (M-m)d$. Consider the dual LP of a given index $k \ge m$, and the feasible dual solution $\{z^*, y_0, \ldots, y_k\}$ obtained in the proof of Theorem 37. It is easy to see that the sequence y is strictly decreasing in the seed z, which also implies that z^* is also decreasing in k. Thus, when $k \to \infty$, z^* converges to ζ from above, for ζ as defined in the statement

of Theorem 36. In other words, for $k \to \infty$, $z^* \to \frac{m}{M}$. Informally, we have obtained a dual solution which is identical to the dual solution given for the infinite-index LP in [66]. A formal evaluation of the dual objective can be done as in Theorem 7 of [66], from which we obtain that $B \geq (M - m)d$.

It is interesting to note that, in order to establish the correct proof, we show that for fixed index k, there exists a seed for which $y_i \geq 0$ for all $i \leq k$, but also $y_j < 0$ for some $j \geq k + m$. In contrast, [66] relies on a sequence which remains non-negative.

3.1.4 Searching with multiple searchers in the presence of turn cost

In this section we show how to extend the infinite LP framework to the problem of m-ray searching with turn cost and *multiple* searchers. More precisely, there are $p \geq 1$ searchers, and the total cost of the search is the cost paid by the first searcher who locates the hider. The competitive ratio of the search is then defined analogously to the case of a single searcher.

For the case of no turn cost, i.e., assuming $d = 0$, the problem was studied by López-Ortiz and Schuierer [107] who showed an optimal strategy with competitive ratio

$$1 + 2M_{m,p}, \text{ with } M_{m,p} = \frac{a^m}{a^p - 1} \text{ and } a = \left(\frac{m}{m-p}\right)^{\frac{1}{p}}, \tag{3.6}$$

which is a generalization of (1.1) for $p > 1$. In the remainder of this section we will denote $M_{m,p}$ by M, for convenience, since m and p will be implied from the setting.

In the presence of turn cost, i.e, in the case $d \neq 0$, we can formulate the problem of minimizing the competitive ratio along the same lines as for the case of a single searcher. Namely, we want to find a strategy that is $(1 + 2M, B)$-competitive such that B is minimum. Thus, this problem models the combination of the settings studied in [66] and [107].

At first glance, the problem may appear to be a rather straightforward, albeit more technical application of the infinite LP technique we discussed in the case of a single searcher. However, there is a big difference that makes the problem interesting on its own. In contrast to the single-processor case, one can no longer exploit the simple structure of cyclic solutions, since optimal strategies are not necessarily cyclic. This implies that one cannot obtain a straightforward LP formulation of the problem at hand. Note that the non-cyclicality of optimal solutions is often a critical difficulty in the context of search and scheduling problems (see [88] for a discussion on this issue, even for the substantially easier setting of a single processor).

We bypass this obstacle by establishing some useful properties of optimal search strategies in the presence of turn cost similar to [107]; we then exploit these properties so as to derive *lower-bound* LP formulations without knowledge of the exact structure of the optimal solution. This means that every strategy must satisfy the constraints of the LP, but a solution to the LP does not necessarily define a strategy. Nevertheless, we can obtain matching and near-matching upper bounds (depending on whether p divides m) by analyzing specific strategies.

In what follows we focus on how to derive the family of lower-bound LP formulations. We also briefly explain the challenges in solving the LPs and obtaining good lower bounds.

We will first establish some useful properties of optimal strategies (namely, Lemma 38 and Lemma 39). The proofs of these lemmas follow using a refinement of techniques from [107], namely by a series of transformations that preserve the competitive ratio of the search, and also do not increase the contribution of the turn cost to the overall search cost. We say that a ray r is *occupied* at time T if there is a searcher on r at time T, and that it is *busy* if is there is a searcher on r that is moving away from the origin at that time.

Lemma 38. *For any search strategy S, there is a strategy S' which is no worse than S with the following properties: i) At any time, there is at most one searcher on any given ray; ii) If a searcher moves towards the origin on some ray, then it continues until it reaches the origin; and iii) no searcher is immobile, at any given time.*

Lemma 39. *There is an optimal strategy that satisfies Lemma 38 such that if a searcher is located at the origin at a given point in time, then it chooses to explore the ray that has been explored the least among all non-busy rays.*

We call an optimal strategy that satisfies Lemma 39 *normalized*. We emphasize that Lemma 38 and Lemma 39 describe the same properties of (optimal) normalized schedules for the case of zero turn cost and for the measure of competitive ratio [107]. This allows us to derive further key properties of normalized schedules (using the techniques of Lemma 3 in [107]), tailored to the setting of non-zero turn cost, and for any $(1+2M, B)$-competitive strategy. More precisely, let S denote an optimal, normalized $(1+2M, B)$-competitive strategy, and let $X = x_1, x_2, \ldots$ denote the (infinite) sequence of the distances in S, *in increasing order of length* (that is, each x_i corresponds to the distance in which a ray is searched during some traversal of a certain searcher). We first obtain constraints that roughly correspond to adversarial target placements after each turn.

Lemma 40. *For any normalized optimal strategy S, $\sum_{j=1}^{m+i} x_j + d(m+i) \leq 2M \sum_{j=i+1}^{i+p} x_j + Bp$, for all $i \geq 0$.*

We need to add one more essential constraint so as to obtain our lower-bound LP.

Lemma 41. *For the normalized optimal strategy S, $2\sum_{j=1}^{m-p} x_j + d(m-p) \leq Bp$.*

From Lemma 40 and Lemma 41 it follows that for every $(1+2M, B)$-competitive strategy, $B \geq \lim_{k\to\infty} \mathrm{obj}_k$, where obj_k is the objective value of the following linear program.

$$
\begin{aligned}
\min \quad & B && (P)\\
\text{s.t.} \quad & 2\sum\nolimits_{j=1}^{m-p} x_j - pB \leqslant -d(m-p)\\
& 2\sum\nolimits_{j=1}^{m+i} x_j - 2M\sum\nolimits_{j=i+1}^{i+p} x_j - pB \leqslant -d(m+i)\\
& \qquad\qquad \forall i = 0 \ldots k\\
& B, x_1, \ldots, x_{m+k} \geqslant 0.
\end{aligned}
$$

We will again call k the *index* of the dual LP.

$$
\begin{aligned}
\max \quad & \left((m-p)z + \sum\nolimits_{i=0}^{k} y_i(m+i)\right) d && (D)\\
\text{s.t.} \quad & z + \sum\nolimits_{i=0}^{k} y_i \leqslant \frac{1}{p}\\
& \left\{ \begin{array}{ll} z\ , & j \leq m-p \\ 0\ , & \text{otherwise} \end{array} \right\} + \sum\nolimits_{i=\max(0,j-m)}^{k} y_i - M\sum\nolimits_{i=\max(0,j-p)}^{\min(j-1,k)} y_i \geqslant 0 \quad \forall j = 1 \ldots m+k\\
& z, y_0, \ldots, y_k \geqslant 0.
\end{aligned}
$$

We have now set up the primal and dual families of lower-bound LP formulations. The question is, where to go from here. One could try an approach similar to the one of Section 3.1.3. However, there are some significant obstacles. Specifically, in contrast to single-searcher variant, we cannot guess the optimal dual solution from computational experiments, and even if we could, it would be excruciatingly difficult to prove optimality. To better illustrate this difficulty, we emphasize that for the characteristic polynomial of the recurrence relation that is central to the analysis we cannot give a closed-form expression of its principal root, as we did in Theorem 36 for the case $p = 1$. Instead, we use tools from linear algebra and complex analysis so as to prove the existence of a feasible dual solution (without giving an explicit closed form) which is very close to optimal in the general case, and optimal for the case $p \mid m$.

The above approach leads us to the following lower bound.

Theorem 42. *For any constant $\epsilon > 0$, and any search algorithm, there exists a target placement at distance h from the origin such the algorithm incurs cost at least*

$$(1+2M)h + d\left(M - (1+\epsilon)\frac{m}{p} - O(1)\right).$$

In particular, when $p \mid m$, the corresponding cost is at least $(1+2M)h + d\left(M - \frac{m}{p}\right)$.

We complement this lower bound with an upper bound that follows from analyzing a specific cyclic strategy.

Theorem 43. *There exist search strategies that are $(1+2M, d(M - \frac{m}{p}))$-competitive, if $p \mid m$, and $(1+2M, d(M - \frac{m}{p} + 4))$-competitive, if $p \nmid m$.*

3.2 Weighted m-ray search

3.2.1 Background

All previous work related to searching on the line, and searching on the m-ray star has focused on the case of a single hider. In this section, we will consider the generalization in which there exist more than one hiders, and the searcher must locate a certain subset of them, according to a certain criterion.

In particular, we introduce and study the following setting. We are given an m-ray star with origin O. For each ray $i \in \{0, \ldots, m-1\}$, there is a hider of weight $w_i \geq 0$ that is hidden at some distance $d_i \geq 0$ from O. Note that the setting allows cases such as $d_i = \infty$ (i.e., there is no target hidden on ray i) and $w_i = 0$ (i.e., the target has no weight). A mobile searcher is initially located at O, and its objective is to locate a subset of targets whose aggregate weight is at least a specified value W. The searcher knows the number of rays m, however, it has no further knowledge about the instance, namely the weights of the targets and their distances from O.

We can extend the concept of the competitive ratio of searching in this setting. Specifically, we would like to compare the performance of a search strategy that is oblivious of the instance (that is, the exact position of the targets and their weights) to an omniscient searcher that has full information of the instance. To formalize this concept, given a number of rays $m \geq 2$, let us denote by $\mathcal{I}_m$ the set of all instances to the search problem, namely

$$\mathcal{I}_m = \left\{ \left(W, (d_i, w_i)_{0 \leq i \leq m-1\}}\right) : W \geq 0, w_i \geq 0, d_i \geq 1, \text{ and } \sum_{0 \leq i \leq m-1} w_i \geq W \right\}.$$

We can naturally define an extension of the optimal (offline) cost in this setting, which is the minimum cost for discovering a total weight W assuming full knowledge of the setting. Let T_I denote the set of all targets of an instance $I \in \mathcal{I}_m$, and for $X \subseteq T_I$, let w_X the weight of this subset, then we have

$$\textsc{Opt}(I) = \min_{X \subseteq T_I} \{d_X : w_X \geq W\}, \text{ where } d_X = 2\sum_{i \in X} d_i - \max_{i \in X} d_i. \tag{3.7}$$

A strategy S is called *ρ-competitive on instance I* for some $\rho \geq 1$, if $c(S, I) \leq \rho \cdot \textsc{Opt}(I)$, where $c(S, I)$ denotes the cost of strategy S on instance I. The competitive ratio of S is then defined as

$$\rho(S) = \sup_{I \in \mathcal{I}_m} \frac{c(S, I)}{\textsc{Opt}(I)}. \tag{3.8}$$

Figure 3.1 illustrates an example of an instance.

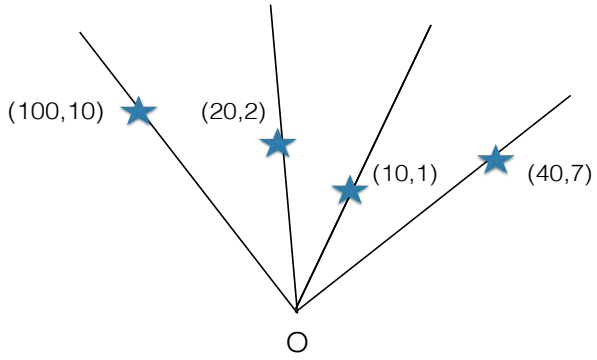


Figure 3.1: Illustration of the weighted star search problem, for an instance I with $m = 4$ rays. The target at ray i is represented by the pair (d_i, w_i), where d_i denotes its distance from O and w_i its weight. Assume an indexing of rays as $0 \dots 3$, from left to right, and a value of W equal to 10. Then an optimal strategy will either search the ray 0 at a cost of 100 or the rays 1,2,3 (in this order), which also yields the optimal cost of $2 \cdot 20 + 2 \cdot 10 + 40 = 100$.

The weighted setting is motivated by many factors. It can model decision-making processes when several tasks need to be pursued concurrently, without knowing in advance neither which ones will be fruitful, nor to which extent they will be so. For instance, a researcher may want to determine how to allocate her time among m different projects, without knowing in advance how useful the outcome of each project will be towards a combined publication of the potential results. Here, the projects can be modeled as weighted targets, with the distance to a target representing the time needed for the successful completion of the corresponding project. As another potential application, one may define hybrid algorithms [92] in which each heuristic execution is assigned a score related to the quality of the returned solution, or a system of interleaved executions of interruptible algorithms in which each algorithm is assigned a priority based on its importance. Such settings can be formulated as a weighted star-search problem.

3.2.2 The uniform (unweighted) setting

We first address the simpler variant in which all hiders have uniform, i.e., unit weights, and the searcher must locate W hiders in total, where W is integral.

Here, we rely on a geometric (cyclic) strategy, for an appropriate choice of its base b. We note that if a target is found during iteration i (i.e., when searching up to distance b^i), then the next ray (in the round-robin order) is searched up to distance b^i. The strategy is described in Algorithm 1.

It is not too difficult to show that Algorithm 1 is optimal. The idea behind the analysis is that the presence of multiple targets is, in a sense, beneficial as far as the competitive ratio is concerned. This is because both the online and the optimal offline search strategies will pay costs for having to discover multiple targets. There is, in fact, another way of interpreting this result: uniform multitarget search for W targets has the same competitive ratio as searching for a single target in $m - W + 1$ rays.

Theorem 44. *The competitive ratio of Algorithm 1 is minimized for* $b = \frac{m-(W-1)}{m-W}$*, for which it is equal to*

$$1 + 2\frac{(m-(W-1))^{m-(W-1)}}{(m-W)^{m-W}}.$$

Moreover, this is optimal for all deterministic strategies.

Algorithm 1: An optimal geometric strategy for searching for W hiders in m rays

```
1  i → 0
2  r → 0
3  f → 0
4  b → (m−(W−1))/(m−W)
5  repeat
6  |  explore the rth ray up to distance b^i
7  |  if target found then
8  |  |  f → f + 1
9  |  |  remove the rth ray
10 |  |  relabel the rays canonically from 0 ... m − f − 1
11 |  else
12 |  |  i → i + 1
13 |  |  r → r + 1 mod (m − f)
14 |  end
15 until f = W
```

3.2.3 The general weighted setting

We now turn our attention to the general setting, which is far more challenging than the unweighted setting. There are two main issues that one needs to address. The first issue pertains to the design of the strategy itself, and more precisely in determining the appropriate sequence of search lengths. Unlike most previous work in which it suffices to increase the search lengths by the same factor at each step, in order to obtain an optimal strategy, in our setting we can argue that such simple rules are very inefficient. We thus introduce an *adaptive* strategy in which the search lengths increase depending on the total number of targets found by the end of each step.

The second issue lies in analyzing this strategy, since the setting is substantially more complex than the unweighted one. To this end, we relate the competitiveness of the weighted search problem to that of a related problem which we term *subset search*. The objective in the latter problem is, informally, to locate a certain subset of the targets, without advance knowledge of the specific subset. Deriving an optimal strategy for this problem is the main technical challenge.

Results

We present and analyze a search strategy called AdaptiveSearch (or AdSch for brevity) for weighted ray search that attains a (near-)optimal competitive ratio. Define the function ϕ as

$$\phi(x) = 1 + 2\,(1+x)\left(1+\frac{1}{x}\right)^{x},\ x > 0.$$

Note that $\phi(x) = \Theta(x)$ as x gets large. As we will see, the function ϕ plays a central role in the analysis and reveals connections to previous studies of the simpler variants of search problems; we will elaborate more on that in due course. Our first theorem provides a tight, worst-case analysis of the competitive ratio of weighted search.

Theorem 45. *The competitive ratio of* AdSch *is* $\phi(m-1)$*, and this is the optimal competitive ratio for weighted search.*

Even though Theorem 45 is tight for some instances, a moment of thought reveals that it actually provides a rather too pessimistic worst-case guarantee. In fact, the results on the unweighted variant we discussed in Section 3.2.2 leave open the possibility that more refined guarantees than $\phi(m-1)$ may be attainable for weighted search; in particular, note that Theorem 44 shows that the competitive ratio of uniform search is $\phi(m-W)$. In order to make this intuition precise, we follow a *parameterized* approach, in which we study the performance of a strategy not only in terms of m, but also in terms of some natural parameters that reflect the "hardness" of the

instance. More specifically, given an instance $I = \{(W, (d_i, w_i)_{0\le i\le m-1})\}$, define $W_I = W$ and $w_{\max,I} = \max_{0\le i\le m-1} w_i$. Then any strategy, including the offline optimal one that knowns I, must locate at least $\lceil W_I/w_{\max,I}\rceil$ targets. The question then is: what is the competitiveness of weighted search, in terms of both m and the above bound on the number of targets that must be found? Our next result gives a near-tight answer, and generalizes the results on unweighted search.

Theorem 46. *Let $I \in \mathcal{I}_m$ be such that $w_{\max,I} > 0$. If $\lceil W_I/w_{\max,I}\rceil < m$, then*

$$c(\textsc{AdSch}, I) \le \phi\left(m - \lceil W_I/w_{\max,I}\rceil\right) \cdot \textsc{Opt}(I),$$

and $c(\textsc{AdSch}, I) \le (3+2e)opt(I)$ otherwise. Moreover, for every strategy Σ, there exists $I \in \mathcal{I}_m$ with $\frac{W_I}{w_{\max,I}} < m$ such that

$$c(\Sigma, I) \ge \phi\left(m - \lceil W_I/w_{\max,I}\rceil\right) \cdot \textsc{Opt}(I),$$

and there exists $I \in \mathcal{I}_m$ with $W_I/w_{\max,I} = m$ such that $c(\Sigma, I) \ge 2\textsc{Opt}(I)$.

While this result is essentially tight with respect to the parameter $W_I/w_{\max,I}$, we can, obtain an even stronger performance guarantee. As we show in the next theorem, we can surprisingly relate the competitiveness of weighted search to the *maximum* number of targets that an ideal solution of optimal cost must locate; in contrast, in Theorem 46, the competitive ratio is expressed in terms of the *minimum* number of targets in any ideal solution (more precisely: a lower bound on this number). For a general instance $I \in \mathcal{I}_m$, let s_I denote the maximum value of this parameter. To illustrate this concept, consider the instance of Figure 3.1. There are two optimal solutions: one locating the target $(100, 10)$, and one that locates all other targets. Thus, for this instance, we have that $s_I = 3$.

Theorem 47. *Let $I \in \mathcal{I}_m$. Then*

$$c(\textsc{AdSch}, I) \le \phi(m - s_I) \cdot \textsc{Opt}(I), \text{ if } s_I < m,$$

and $c(\textsc{AdSch}, I) \le (3 + 2\mathrm{e})\textsc{Opt}(I)$ otherwise. Moreover, for every strategy Σ and for every $s \in \{1, \ldots, m-1\}$, there exists $I \in \mathcal{I}_m$ such that

$$c(\Sigma, I) \ge \phi(m - s_I) \cdot \textsc{Opt}(I),$$

and there exists I with $s_I = m$ such that $c(\Sigma, I) \ge (2 - \epsilon)\textsc{Opt}(I)$, for arbitrarilly small $\epsilon > 0$.

The extreme cases $W_I/w_{\max,I} = m$ and $s_I = m$ in the statements of Theorems 46 and 47 respectively, describe the outlier cases in which the strategy must locate all m targets. In this case, a small additive gap of at most $(1 + 2\mathrm{e}) \cdot \textsc{Opt}(I)$ remains.

Strategies and the overall approach

In this section we present the strategy for weighted star search, and the main approach to the analysis. The approach builds on the relation between the competitiveness of the weighted search problem (WS, for brevity) and a problem we call SubsetSearch (SS, for brevity). We formally define the latter problem as follows. The instance to the problem consists of m unweighted targets (one per ray) with the target at ray i being at distance d_i from the origin of the star, as well as a subset $S \subseteq T_I$ of the targets. The distances, as well as the subset S are not known to the searcher. The search terminates when all targets in S have been discovered; we can assume the presence of an oracle that announces this event to the searcher and thus prompts the termination. The cost of the search is defined as the total cost incurred at termination, whereas the cost of the optimal solution to the instance is the cost for locating all targets in S assuming full information of the instance, i.e., equal to d_S, as defined in (3.7). The following lemma establishes a useful association between the two problems.

Lemma 48. *Suppose there is a strategy Σ_s for SS such that for any instance $I_s = (S, d_0, \dots d_{m-1})$ we have that $c(\Sigma_s, I_s) \le \rho(|S|, m) \cdot d_S$, where $\rho(|S|, m)$ is a function of $|S|, m$. Then there is a strategy Σ_w for* WS *such that for any instance I_w of this problem, $c(\Sigma_w, I_w) \le \rho(s_I, m) \cdot \textsc{Opt}(I_w)$.*

Lemma 48 demonstrates that the competitiveness of SS is directly related to that of WS. We thus focus on SS; in particular, we will analyze a strategy which we call AdaptiveSubset (AdSub for brevity), and which is described by the statement of Algorithm 2. The strategy explores rays in cyclic order (line 13), and keeps track of the found targets, denoted by f (line 11). We use the following notation.

$$b_x = 1 + \frac{1}{x} \quad \text{and} \quad b_{m,t} = b_{m-t} = 1 + \frac{1}{m-t}. \tag{3.9}$$

On each iteration, the strategy will search a ray to a length equal to $b_{m,f}$ times the length of the last ray exploration that did not reveal a target (line 8), until a new target is discovered. Once a target is found on a ray, the ray is not considered again; namely, the remaining rays are relabeled so as to maintain a cyclic order (line 12).

Algorithm 2: Strategy AdSub for SS

```
1  Input: m rays labeled {0, ..., m − 1}, subset oracle O
2  f ← 1, r ← 0, D ← 1
3  repeat
4      repeat
5          explore the rth ray up to distance D · b_{m,f} or until a target is found
6          if no target was found then
7              r ← r + 1 (mod m − f + 1)
8              D ← D · b_{m,f}
9          end
10     until a target was found
11     f ← f + 1
12     remove the rth ray and relabel the rays canonically from 0 ... m − f
13     r ← r (mod m − f + 1)
14 until all targets, according to O, are found
```

Before we proceed with the key points behind the analysis, it is instructive to point out that the need for an adaptive strategy that modifies the search lengths as a function of the number of targets that have been found seems to be, in a sense, unavoidable. To see this, consider first a cyclic strategy that uses geometrically increasing lengths with a fixed base that may depend only on m. Consider also an instance in which $|S| - 1$ targets from S are very close to the origin, then the competitive ratio of this strategy is at least the competitive ratio of a strategy that searches one target in $m - |S| + 1$ rays. Crucially, this ratio can only be achieved if the base equals $b_{m-|S|+1,1}$. Since $|S|$ is not known in advance, the wrong choice of a base can have a detrimental effect on performance.

Another potential candidate is a cyclic, geometric strategy that changes the base of search lengths once a target is found; more precisely, a strategy that on the i-th step searches the corresponding ray up to depth $b^i_{m,t}$, where $t - 1$ represents the number of targets found at the beginning of the step. However, such a strategy has an unbounded competitive ratio. To see this, suppose that $|S| = 1$, and that in iteration i, the strategy finds on ray r a target that is not in S. Suppose also that the unique target in S lies on ray $(r - 1) \bmod m$ and at distance $b^{i-1}_{m,1} + \epsilon$, for some $\epsilon > 0$. The strategy will discover this target after having spent cost at least $2b^{i+m-2}_{m,2}$ (namely, the cost for searching ray $(r - 2) \bmod m$) on iteration $m + i - 2$). It follows

that the competitive ratio is at least $(b_{m,2}/b_{m,1})^i$, which is unbounded since i can be arbitrarily large.

By combining the insights from both previous examples, we see that not only we have to adapt the search lengths, but we also have to ensure a smooth transition when such a modification takes place. This is precisely accomplished in AdSub. Concerning weighted search, Lemma 48 guides our choice of strategy. Namely, we denote by AdSch the strategy for WS obtained by running AdSub, with the only difference that the oracle announces termination when a subset of targets of weight at least W has been found.

The following lemma, which bounds the competitive ratio of AdSub, is the main technical result of this work.

Lemma 49. *Let $m \geq 2$. Given an instance I of SS in which we seek a set $S \subseteq \{0, \ldots, m-1\}$, with $S \neq \emptyset$, we have*

$$c(\text{AdSub}, I) \leq \phi(m - |S|) \cdot d_S, \quad \text{if } |S| \leq m-1, \tag{3.10}$$

and $c(\text{AdSub}, I) \leq (3 + 2e) \cdot d_S$, if $|S| = m$.

Assuming Lemma 49, we show first how to obtain Theorem 45. This establishes a tight, albeit worst-case bound on the competitive ratio.

Proof of Theorem 45. From Lemmas 48 and 49, for all $I \in \mathcal{I}_m$,

$$c(\text{AdSch}, I) \leq \max\{3 + 2e, \phi(m-1)\} \cdot \text{Opt}(I),$$

since ϕ is increasing. Moreover, $\phi(1) = 9 > 3+2e$, hence the upper bound follows. This bound is tight for the instance I in which there is only one target of weight w, and $W_I = w$ (i.e., standard star search for a single target [74]). □

In order to prove Theorems 46 and 47, we will need a simple lemma that addresses the extreme cases.

Lemma 50. *Given an instance I of* WS *for which $s_I = m$ or $W/w_{\max,I} = m$, we have $c(\text{AdSch}, I) \leq (3+2e)\text{Opt}(I)$. Moreover, for every strategy Σ, there exists I such that $c(\Sigma, I) \geq (2-\epsilon)\text{Opt}(I)$, for arbitrarily small ϵ.*

Having established bounds for the extreme cases, we can now proceed with the proofs of Theorems 46 and 47. Recall that these theorems establish our main parametric results on the competitive ratio of weighted search.

Proof of Theorem 46. The upper bound in the general case follows from Lemmas 48 and 49. For the lower bound (the general case), consider an instance I in which all targets have weight $w > 0$, and $W = tw$, for some $t \in \mathbb{R}^+$. Then any strategy must locate $\lceil t \rceil$ targets; from [22], the competitive ratio is at least $\phi(m-t) = \phi(m - \lceil \frac{W}{w_{\max,I}} \rceil)$. The upper and lower bounds in the extreme case are given by Lemma 50. □

Proof of Theorem 47. The upper bound in the general case follows from Lemmas 48 and 49. For the lower bound in the general case, given $s < m$, consider an instance I in which $W = sw$, for some $w > 0$, and there are s targets of weight w, while all other targets have weight 0. Suppose that $s-1$ of positive-weight targets are very close to the origin, and can be found at negligible cost. Thus, on this instance, weighted search is as hard as locating one target in $m-(s-1)$ rays, and thus has competitive ratio

$$1 + 2\frac{b_{m-s+1,1}^{m-s+1}}{b_{m-s+1,1} - 1} = 1 + 2\frac{b_{m-s}^{m-s+1}}{b_{m-s} - 1} = \phi(m-s),$$

where the second equality follows from (3.9). The upper and lower bounds in the extreme case are given by Lemma 50. □

We conclude this section with a few words on the main technical lemma, namely Lemma 49. To arrive at the result, we first prove essentially tight bounds on the optimal cost, by identifying appropriate parameters related to the search, and by relaying the cost of our strategy to precisely those parameters. We then apply an inductive argument on the number of targets discovered by the strategy in the last m rays that have been explored by the searcher. Several technical difficulties arise from the fact that, unlike the single-target variant, there is no simple expression that describes the optimal cost, and that the cost of the searcher involves several parameters in a trade-off relation. The full proof is very technical, since it appears there is very little "slack" on how to upper bound the involved ratios.

3.3 Bijective analysis of linear search

3.3.1 Background

Consider the problem of linear search (or cow-path problem) as defined in Section 1.2. It has long been known [34, 73] that the competitive ratio of linear search is 9, and is achieved by a simple *doubling* strategy: in iteration i, the searcher starts from the origin O, explores branch $i \bmod 2$ at a length equal to 2^i, and then returns to O. However, this strategy is not uniquely optimal; in fact, it is known that there is an infinite number of competitively optimal strategies for linear search. In particular, consider an "aggressive" strategy, which in each iteration searches a branch to the maximum possible extent, while maintaining a competitive ratio equal to 9. A simple analysis shows that this can be achieved by searching, in iteration i, branch $i \bmod 2$ to a length equal to $(i+2)2^{i+1}$ (here we start at index $i = 0$).

While both strategies `doubling` and `aggressive` are optimal in terms of competitive ratio, there exist realistic situations in which the latter may be preferable to the former. Consider, for example, a search-and-rescue mission for a missing backpacker who has disappeared in one of two (very long) 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 tuned to discovering new territory (such as `aggressive`) rather than a conservative strategy that tends to often revisit already explored areas (such as `doubling`).

We thus encounter a familiar phenomenon: there are two strategies which have the same performance according to competitive analysis, but intuition says that they may not perform comparably in the real world. Although similar problems in online computation have been studied extensively (as we discussed in Section 1.1), to our knowledge, no previous work addressed this issue in the context of online searching.

With the above observations in mind, we first need to quantify what constitutes efficiency in exploration. To this end, given a strategy S and $l \in \mathbb{R}^+$, we define $D(S, l)$ as the cost incurred by S the first time the searcher has explored an aggregate length equal to l, combined in both branches. An efficient strategy should be such that $D(S, l)$ is small, for all l. Unfortunately, this criterion by itself is insufficient: Consider a strategy that first searches one branch to a length equal to L, where L is very large. Then $D(S, l)$ is as small as possible for all $l \leq L$, but very big for, say $l = L + \epsilon$, for small positive ϵ.

We can still make use of the discovery cost in a way that allows us a pairwise comparison of strategies. Specifically, we define the following:

Definition 51. *Let S_1, S_2 denote two search strategies, we define the* discovery ratio of S_1 against S_2, *denoted by $dr(S_1, S_2)$, as*

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

Moreover, given a class $\mathcal{S}$ of search strategies, the discovery ratio of S against the class $\mathcal{S}$ *is defined as*

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

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

Intuitively, the discovery ratio preserves the worst-case nature of competitive analysis, and at the same time bypasses the need for an "offline optimum" solution. Note that if a strategy S has competitive ratio c then it also has discovery ratio; however, the opposite is not necessarily true.

While the discovery ratio is an intuitive measure by itself, in the context of linear search, we can argue that it very much related to bijective analysis. Recall the definition of the bijective ratio $\mathrm{br}(A, B)$ of an online algorithm A against an online algorithm B (Definition 14), for a discrete, online problem in which the set of all requests of size n, $\mathcal{I}_n$, is finite. An equivalent definition of $\mathrm{br}(A, B)$ is as follows. Let $A(i, n)$ denote the i-th least costly request sequence for A among request sequences in $\mathcal{I}_n$. Then

$$\mathrm{br}(A, B) = \sup_n \max_i \frac{A(i, n)}{B(i, n)}.$$

Consider in contrast, the linear search problem. Here, there is only one request: the unknown position of the hider (i.e., $n = 1$). However, the set of all requests is not only infinite, but uncountable. Thus, the above definitions do not carry over to our setting, and we need to seek alternative definitions. One possibility is to discretize the set of all requests (as in Section 2.1). Namely, we may assume that the hider can hide only at integral distances from the origin. Then given strategies S_1, S_2, one could define the bijective ratio of S_1 against S_2 as $\sup_i \frac{S_1(i)}{S_2(i)}$, where $S(i)$ denotes the i-th least costly request (hider position) in strategy S.

While the latter definition may indeed be valid, it is still not a faithful representation of the continuous setting. For instance, for hiding positions "close" to the origin, the discretization adds an overhead that should not be present, and skews the expressions of the ratios. For this reason, we need to adapt the definition so as to reflect the continuous nature of the problem. Specifically, note that while the concept "the cost of the i-th least costly request in S" is not well-defined in the continuous setting, the related concept of "the cost for discovering a total length equal to l in S" is, in fact, well defined, and is precisely the value $D(S, l)$. We can thus define the bijective ratio of S_1 against S_2 as

$$\mathrm{br}(S_1, S_2) = \sup_l \frac{D(S_1, l)}{D(S_2, l)},$$

which is the same as the definition of the discovery ratio (Definition 14).

We will next show how to apply the discovery ratio to the linear search problem. We will consider two different problems. The first problem is finding strategies of optimal discovery ratio against *any* other strategy (Section 3.3.2). Then, we consider a more refined and interesting problem: Find a strategy of optimal discovery ratio against all *competitively optimal* strategies, i.e., against all strategies of competitive ratio equal to 9 (Section 3.3.3).

Definitions We will denote by Σ the set of all search strategies, and by Σ_9 the subset of Σ that consists of strategies with competitive ratio 9. For a given strategy $X = (x_0, x_1, \ldots)$, we will denote by $T_i(X)$ (or simply by T_i, when X is clear from context) the quantity $\sum_{j=0}^{i} x_j$.

3.3.2 Strategies of optimal discovery ratio in Σ

We begin with a study of the discovery ratio against all possible strategies.

Let X, Y, denote two strategies in Σ, with $X = (x_0, x_1, \ldots)$ (recall that these are the search lengths for X). 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,

$$\mathrm{dr}(X, Y) = \sup_{i \in \mathbb{N}^*} \sup_{\delta \in (0, x_i - x_{i-2}]} \frac{D(X, x_{i-1} + x_{i-2} + \delta)}{D(Y, x_{i-1} + x_{i-2} + \delta)}. \tag{3.11}$$

The following theorem provides an expression of the discovery ratio in terms of the search segments of the strategy, that follows easily from (3.11).

Theorem 52. *Let $X = (x_0, x_1, \ldots)$. Then*

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

In particular, note that for $i = 2$, Theorem 52 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 $\alpha > 1$. A simple calculation yields the following lemma.

Lemma 53. *For G_α defined as above, we have $dr(G_\alpha, \Sigma) = \frac{2\alpha^2 + \alpha - 1}{\alpha^2 - 1}$.*

In particular, Lemma 53 shows that the discovery ratio of G_α tends to 2, as $\alpha \to \infty$, hence G_α has asymptotically optimal discovery ratio. However, we can show a much stronger result, namely that G_α achieves the *optimal trade-off* between the discovery ratio and the competitive ratio. This is established in the following theorem. Note that the competitive ratio of G_α is easily verified to be $1 + 2\frac{\alpha^2}{\alpha - 1}$ (and is minimized for $\alpha = 2$).

Theorem 54. *For every strategy $X \in \Sigma$, there exists $\alpha > 1$ such that $dr(X, \Sigma) \geq \frac{2\alpha^2 + \alpha - 1}{\alpha^2 - 1}$ and $cr(X) \geq 1 + 2\frac{\alpha^2}{\alpha - 1}$.*

We omit the proof of Theorem 54; however we note that is is based on the application of a very useful result by Gal [76] which, informally, lower-bounds the supremum of an infinite sequence of functionals by the supremum of simple functionals of a certain geometric sequence, and which we state here in a simplified form. Given an infinite sequence $X = (x_0, x_1, \ldots)$, define $X^{+i} = (x_i, x_{i+1}, \ldots)$ as the suffix of the sequence X starting at x_i.

Theorem 55 ([76])**.** *Let $X = (x_0, x_1, \ldots)$ be a sequence of positive numbers, r an integer, and $\alpha = \limsup_{n \to \infty} (x_n)^{1/n}$, for $\alpha \in \mathbb{R} \cup \{+\infty\}$. Let F_i, $i \geq 0$ be a sequence of functionals which satisfy the following properties:*

1. *$F_i(X)$ only depends on $x_0, x_1, \ldots, x_{i+r}$,*
2. *$F_i(X)$ is continuous for all $x_k > 0$, with $0 \leq k \leq i + r$,*
3. *$F_i(\lambda X) = F_i(X)$, for all $\lambda > 0$,*
4. *$F_i(X + Y) \leq \max(F_i(X), F_i(Y))$, and*
5. *$F_{i+1}(X) \geq F_i(X^{k+1})$, for all $k \geq 1$,*

then

$$\sup_{0 \leq i < \infty} F_i(X) \geq \sup_{0 \leq i < \infty} F_i(G_\alpha).$$

As a side note, the above theorem is very useful in establishing lower bounds that relate to the performance of geometric strategies, as we will see further in Chapter 4.

3.3.3 Strategies of optimal discovery ratio in Σ_9

In this section we focus on strategies in Σ_9, namely the set of competitively optimal strategies. We first establish certain useful properties concerning this class which are useful in the proofs, and explain their connection to the problem we want to solve.

Properties 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, [88]).

Lemma 56. *The strategy $X = (x_0, x_1, x_2, \ldots)$ is in Σ_9 if and only if its segments satisfy the following inequalities*

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

We now define a specific 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 function of t.

Lemma 57. *The strategy R_t is defined by the sequence $x_n = t(1 + \frac{n}{2})2^n$, for $n \geq 0$. Moreover, $T_n(R_t) = t(n+1)2^n$.*

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 and also minimizes the number of turns required to discover a certain length. We can thus say that $R_4 \equiv$ `aggressive`.

Next we need to be able to express the *number of turns* a competitive strategy performs. Specifically, 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. The following lemma gives an expression for $m^*(l)$, for general values of l.

Lemma 58. *For given $l > 4$, $m^*(l) = m(\texttt{aggressive}, l) = \min\{n \in \mathbb{N}_{\geq 1} : (3n+5)2^n \geq l\}$.*

Last, we need an important technical result that is instrumental in establishing the bounds on the discovery ratio, namely Lemma 59 below. This lemma combines several of the above facts. 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 59 below 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 in Section 3.3.2, we use a very simple lower bound on the cost for discovering a length l 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 problem 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, namely 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 59. *For $l > 4$, it holds*

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

We now have the necessary setup in order to prove the main results of this section. Recall that for any two strategies X, Y, $\mathrm{dr}(X, Y)$ is given by (3.11). Using 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

$$\mathrm{dr}(X, \Sigma_9) = \sup_{i \in \mathbb{N}^*} \sup_{\delta \in (0, x_i - x_{i-2}]} F_i(X, \delta), \quad \text{where } F_i(X, \delta) = \frac{2\sum_{j=0}^{i-1} x_j + x_{i-2} + \delta}{d^*(x_{i-1} + x_{i-2} + \delta)}. \tag{3.12}$$

The above properties are the tools need to prove upper and lower bounds on the discovery ratio. We summarize the main results below.

The discovery ratio of strategies in Σ_9

Theorem 60. *For the strategy* `aggressive` *it holds that* $dr(\texttt{aggressive}, \Sigma_9) = 8/5$.

Proof sketch. Let $(\bar{x}_0, \bar{x}_1, \ldots)$ denote the search lengths of `aggressive`. Using (3.12) we can show that for each $i \geq 2$ it holds that

$$\sup_{\delta \in (0, \bar{x}_i - \bar{x}_{i-2}]} F_i(\texttt{aggressive}, \delta) = \frac{99}{64} < \frac{8}{5}, \tag{3.13}$$

using a combination of the properties we established in the previous section. For $i = 1$, and $\delta \in (0, \bar{x_1}]$ the corresponding quantity is precisely equal to $8/5$. □

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

Theorem 61. *For every strategy $X \in \Sigma_9$, we have $dr(X, \Sigma_9) \geq \frac{8}{5} - \epsilon$, for arbitrarily small $\epsilon > 0$.*

Proof sketch. By looking only at the first segment of a strategy, namely by showing that

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

which is arbitrarily close to $\frac{8}{5}$ since δ can be arbitrarily small. □

Recall that `doubling` $\equiv (2^0, 2^1, 2^2, \ldots)$. 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 60, where instead of using the search segments $\bar{x}_i$ of `aggressive`, we use the search segments $x_i = 2^i$ of `doubling`.

Theorem 62. *For the doubling strategy it holds that* $dr(\texttt{doubling}, \Sigma_9) = \frac{7}{3}$.

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 63. *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* $0 \leq i \leq 4$.

3.4 Competitive analysis of expanding search

3.4.1 Background

So far in this chapter, we considered a rather standard formulation of what constitutes *cost* in searching. Namely, the searcher incurs cost every time it moves, or more precisely, the searcher's cost is the total distance traversed, or the total time spent assuming unit speed. This paradigm is known as *pathwise search* [7], and is arguably the "classic" paradigm under which search games have been studied.

There are settings, however, in which a different cost formulation is required. Consider, for example, mining coal: here digging into a new site is far more costly than moving the drill through an area that has already been dug. Another application is securing a dangerous area from hidden explosives; once the area is deemed clear, the searchers can navigate through it at a much faster. These examples motivate the *expanding search* paradigm, introduced by Alpern and Lidbetter [7], which applies situations where the cost of "re-exploration" is negligible compared to the cost of "first-time" searching. In particular, the re-exploration cost is assumed to be zero.

Example 64. *We illustrate the concept of expanding search on a graph with an example. Consider the graph* G *depicted in Figure 3.2 with root* O*; vertices* A, B, C *and* D*; and edges* OA, OB, BC *and* BD *of lengths* 3, 2, 2 *and* 1, *respectively. An example of an expanding search on* G, *which we will denote by* S, *can be described as a sequence of edges,* OB, OA, BD, BC. *Under* S, *the search time of vertex* D *is* $2 + 3 + 1 = 6$.

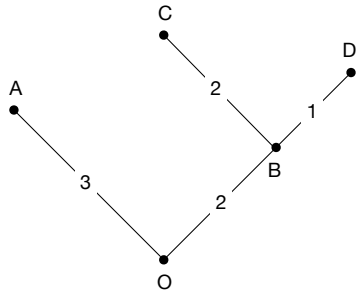


Figure 3.2: A graph G with root O.

Alpern and Lidbetter [7] take the approach of seeking randomized search strategies that minimize the expected search time in the worst case: that is, the maximum expected search time over all vertices. They also consider the problem of determining the search that minimizes some weighted average of the search times of the vertices. In our work, we focus on the *competitive ratio* of expanding search; that is, we consider a normalized version of the search time obtained by dividing the search time of a vertex v by the length of the shortest path from the origin O to v. For example, in the graph G depicted in Figure 3.2, the normalized search time under strategy S of vertex D is $6/3 = 2$ since D is at a distance of $2 + 1 = 3$ from the root. The maximum the normalized search time takes over all vertices of the graph is then the competitive

ratio of the search. In G, the normalized search time of S is maximized at B, where it is equal to $(3+2)/2 = 2.5$, so this is the competitive ratio of S. Note that this is a natural extension of the definition of competitive ratio in pathwise search.

In the spirit of the work of [104], which studied the competitive ratio of pathwise search, in this work we focus on computational and algorithmic issues of online expanding search. We note that [7] follow a purely mathematical approach to analyzing expanding search, with an emphasis on evaluating the value of the corresponding zero-sum games; computational and algorithmic issues are not considered. Table 3.1 illustrates the context of our work with respect to previous work. We note that the problem of minimizing the average search time of the vertices of a graph assuming the pathwise search formulation is precisely the well-known problem of minimizing the *latency* of a graph, also known as the *Traveling Repairman* problem (see [40, 78, 26, 124] for some representative results on this problem). The problem of choosing the randomized (pathwise) search that minimizes the *maximum expected search time* of points of a network was formalized by [75].

Table 3.1: Previous work and relations between search paradigms and objectives.

		Objective		
		Average search time	*Max expected search time*	*Competitive ratio*
Paradigm	*Pathwise*	Min. Latency problem ([40])	Gal's search game ([75])	Graph Searching ([104])
	Expanding	Expanding search ([7])	Expanding search ([7])	This work

In our work we study the competitive ratio of expanding search in two search domains. First, we study the setting in which the hider can hide in a vertex of a given edge-weighted and finite graph (Section 3.4.2). Second, we study the setting in which the hider can hide anywhere within a given edge-weighted, but potentially unbounded network (Section 3.4.3). Although the two setting share some similarities, they also exhibit some interesting differences, as we will discuss in the next sections.

3.4.2 Expanding search in a graph

We begin with some preliminary definitions concerning the setting.

Preliminaries

Let $G = (\mathcal{V}, \mathcal{E})$ be an undirected, connected, edge-weighted graph with $|\mathcal{V}| = n + 1$, and a distinguished root vertex $O \in \mathcal{V}$. The weight or *length* of edge $e \in \mathcal{E}$, denoted by $\lambda(e)$, represents the time required to search that edge (we assume, via normalization, that $\lambda(e) \geq 1$ for all edges e). For subgraphs or subsets of edges X, we write $\lambda(X)$ for the sum of the lengths of all the edges in X. We will call a graph of unit edge weights unweighted, otherwise it is weighted.

An expanding search strategy, or simply *search* on G is a sequence of edges, starting from the root, chosen so that the set of edges that have been searched at any given point in the sequence is a connected, increasing set. More precisely:

Definition 65. *An expanding search S on a graph G with root vertex O is a sequence of edges $(e_1, \ldots, e_n)$ such that every* prefix *$\{e_1, \ldots, e_k\}$, $k = 1, \ldots, n$ is a subtree of G rooted at O. We denote the set of all expanding searches on (G, O) by $\mathcal{S} = \mathcal{S}(G, O)$.*

For a given vertex $v \in \mathcal{V}$ and a given search strategy $S = (e_1, \ldots, e_n)$, denote by S_v the first prefix $\{e_1, \ldots, e_k\}$ that covers v. The *search time*, $T(S, v)$ of v is the total time $\lambda(S_v)$ taken to search all the edges before v is discovered. Let $d(v)$ denote the length of the shortest path from O to v, which is the minimum time for the Searcher to discover v. For $v \neq O$ the *normalized search time* is denoted by $\hat{T}(S, v) = T(S, v)/d(v)$.

Definition 66. *The (deterministic) competitive ratio $\sigma_S = \sigma_S(G)$ of a search strategy S for the graph G is defined as*

$$\sigma_S(G) = \max_{v \in \mathcal{V} - \{O\}} \hat{T}(S, v).$$

The (deterministic) competitive ratio, $\sigma = \sigma(G)$ of G is defined as

$$\sigma(G) = \min_{S \in \mathcal{S}} \sigma_S(G).$$

If $\sigma_S = \sigma$ we say S is optimal.

We will also consider *randomized search strategies*, that is some probabilistic choice of search strategies. Following the standard notation, we denote randomized strategies by lower case letters, and for a randomized search strategy s and a vertex v, we extend the notation $T(s, v)$ to denote the expected search time of v. Similarly we write $\hat{T}(s, v)$ for the *expected normalized search time* $T(s, v)/d(v)$.

The randomized competitive ratio ρ_S of a randomized search strategy S is then defined along the lines of Definition 66, and the same holds for the randomized competitive ratio of a graph G, which we will denote by $\rho(G)$, or simply ρ when G is clear from context.

We will view the randomized competitive ratio ρ through the lens of a finite zero-sum game between a Searcher and a Hider. The Searcher's pure strategy set is the set $\mathcal{S}$ of expanding searches and the Hider's pure strategy set is the set $\mathcal{V} - \{O\}$ of non-root vertices of G. For a Hider strategy $v \in \mathcal{V} - \{O\}$ and a Searcher strategy $S \in \mathcal{S}$, the payoff of the game is $\hat{T}(S, v)$, which the Hider wishes to maximize and the Searcher wishes to minimize. Since the strategy sets are finite, the game has a value and optimal mixed strategies for both players. By the standard minimax theorem for zero-sum games, the value of the game is equal to the randomized search ratio and an optimal randomized search strategy is an optimal mixed strategy for the Searcher in the game. A mixed strategy for the Hider is a probability distribution h over the vertices $\mathcal{V} - \{O\}$, and for mixed strategies h and s of the Hider and Searcher respectively, we write $T(s, h)$ and $\hat{T}(s, h)$ for the corresponding expected search time and expected normalized search time.

We will obtain lower bounds for $\rho(G)$ by giving explicit Hider strategies. More precisely, if h is a given mixed Hider strategy, the minimax theorem implies that $\rho(G) \geq \min_{S \in \mathcal{S}} \hat{T}(S, h)$.

General graphs

We begin by studying the complexity of optimizing the (deterministic) competitive ratio of expanding search in a given weighted graph G. We first show that the problem is NP-hard, using a reduction from 3-SAT.

Theorem 67. *Given a graph G with root O and $R \geq 0$, it is NP-Complete to decide whether $\sigma(G) \leq R$.*

Using an approach similar to the doubling heuristic of [104], we obtain a strategy that achieves a $O(1)$-approximation of the optimal competitive ratio. This strategy works by iterative doubling the radius of the disc around the origin that is to be searched, and explores the resulting graph by computing a Steiner tree of the corresponding vertex set (in contrast to pathwise search, in which the resulting graph is simply explored depth-first).

Theorem 68. *There is a polynomial-time search algorithm that approximates $\sigma(G)$ within a factor of $4\ln(4) + \epsilon < 5.55$.*

Trees and Unweighted Graphs

In this section we present our main technical results that apply to unweighted graphs and (weighted) trees. If G is a graph with root O, for any $r > 0$ let $\mathcal{V}_r$ be the set of vertices in G at distance no more than r from the root O and let G_r be the induced subgraph of G with vertex set $\mathcal{V}_r$.

For both classes of graphs, the deterministic case is easy. Namely, there is an optimal strategy that searches the vertices in non-decreasing order of their distance from the root.

We will thus focus on the randomized setting. We will first show that the optimal deterministic search approximates the optimal randomized competitive ratio by a factor of 2. To prove this, we use the following collection of lower bounds on the randomized competitive ratio ρ. For each non-root vertex v of G, let λ_v denote the length of the unique edge incident to v on some shortest path between O and v (so if G is an unweighted graph then $\lambda_v = 1$) and let $\lambda_O = 0$. For a set $\mathcal{A}$ of vertices, let $\lambda(\mathcal{A}) = \sum_{v\in\mathcal{A}} \lambda_v$ and let $\Delta(\mathcal{A}) = \sum_{v\in\mathcal{A}} \lambda_v d(v)$.

Lemma 69. *Suppose $\mathcal{A} = \{v_1, \ldots, v_m\}$ is a set of non-root vertices of G, and suppose the Hider chooses each $v_i \in \mathcal{A}$ with probability $p_i = \lambda_{v_i} d(v_i)/\Delta(\mathcal{A})$. Then the normalized search time $\hat{T}(S, p)$ of any search S against the Hider strategy p satisfies*

$$\hat{T}(S,p) \geq \frac{\sum_{i\leq j} \lambda_{v_i}\lambda_{v_j}}{\Delta(\mathcal{A})} \geq \frac{\lambda(\mathcal{A})^2}{2\Delta(\mathcal{A})}.$$

The next proposition follows directly from Lemma 69.

Proposition 70. *The randomized competitive ratio ρ of a tree or unweighted graph satisfies $\sigma/2 \leq \rho \leq \sigma$. Hence the optimal deterministic search is a 2-approximation of the optimal randomized search.*

It should be fairly clear that the above result is far from optimal; after all, one cannot expect to approximate efficiently a measure that applies to randomized strategies using a deterministic one. We will show next that we can indeed obtain improved approximations; more precisely we will present and analyze a randomized search for trees or unweighted graphs with approximation ratio asymptotically equal to 5/4. In the case that G is an unweighted graph, we will define the search on some shortest path tree (that is, a spanning tree of G comprising shortest paths from O to each vertex); if G is a weighted tree then we define the search on the whole of G. The idea of the strategy is to partition the vertices of the tree into subsets $\mathcal{V}_0, \mathcal{V}_1, \ldots$, each of which contains vertices whose distances from O are within some interval $[x_i, x_{i+1}]$, where the x_i are chosen randomly according to the method described later in Definition 72. The subsets are then searched one at a time, in increasing order of distance from O. Note that after visiting all the vertices in $\mathcal{V}_1 \cup \ldots \cup \mathcal{V}_j$ we can contract all the edges searched so far to the root vertex O and consider the problem of how to search the induced subtree G_{i+1} with vertex set $\mathcal{V}_{j+1} \cup O$.

The method of searching each of these subtrees is according to a *random depth-first search* (or *RDFS*), which we define as follows. Given a set of vertices H and a depth-first search S of H starting at the root, consider the search S^{-1} which is the depth-first search of H that arrives at the leaf vertices of H in the reverse order to S. An equiprobable choice of S and S^{-1} is a RDFS of H. It is straightforward to calculate the maximum expected search time of a RDFS.

Lemma 71. *Under any RDFS of a rooted tree H, a vertex v is found in expected time $(\lambda(H) + d(v))/2$.*

Proof. Suppose S is some depth-first search of H. Let A be the subset of edges searched by S up to and including when v is reached and let B be the subset of edges searched by S^{-1} up to and including when v is reached. It is easy to see that $A \cap B$ is the unique path from v to the

root. If s is the RDFS that chooses S and S^{-1} equiprobably, then the expected time $T(s, v)$ of s to reach v is

$$T(s, v) = (\lambda(A) + \lambda(B))/2 = (\lambda(A \cup B) + \lambda(A \cap B))/2 = (\lambda(H) + d(v)/2.$$

□

We can now define the randomized deepening strategy. Let t be the smallest integer such that every vertex of G is at distance less than 2^t from O.

Definition 72 (**Randomized deepening strategy**). *Suppose G is a tree with root O. For $i = 1, \ldots, t$, choose some x_i uniformly at random from the interval $[2^{i-1}, 2^i]$ and let $x_0 = 1$ and $x_{t+1} = 2^t$. For $i = 0, \ldots, t$, let $\mathcal{V}_i$ be the set of vertices of G whose distance from O lies in the interval $[x_i, x_{i+1})$. We call $\mathcal{V}_0, \ldots, \mathcal{V}_t$ the* levels *of the search, so that $\mathcal{V}_i$ is level i of the search. Let G_0 be the induced subtree of G with vertex set $\mathcal{V}_0 \cup O$ and we define $G_i, i > 0$ recursively as the induced subtree with vertex set $\mathcal{V}_i \cup O$ of the graph obtained by contracting $G_0 \cup \ldots \cup G_{i-1}$ to the root O. The **randomized deepening strategy** performs a RDFS of each of the trees G_i in the order $G_0, \ldots, G_t$.*

We need two results that are helpful in proving our main theorem, that the random deepening strategy has approximation ratio 5/4. Let $\mathcal{A}_i$ be set of vertices of G whose distance from O is in the interval $[2^{i-1}, 2^i)$, and let $\mathcal{A}^i = \cup_{j \le i} \mathcal{A}_j$. (For the purposes of writing the proof of Theorem 75 we allow i to take any integer value, but note that $\mathcal{A}_i$ is only non-empty for $i = 1, \ldots, t$.)

Lemma 73. *The expected sum over all vertices $v \in \mathcal{A}_i \cap \mathcal{V}_{i-1}$ of edge lengths λ_v is $2\lambda(\mathcal{A}_i) - \Delta(\mathcal{A}_i)/2^{i-1}$.*

We also make two simple observations about the parameters $\Delta(\mathcal{A}^i)$.

Lemma 74. *For any $i = 1, \ldots, t$ we have*

(i) $\Delta(\mathcal{A}^i) \le 2^i \lambda(\mathcal{A}^i)$ *and*

(ii) $\frac{\lambda(\mathcal{A}^i) - \Delta(\mathcal{A}^i)/2^i}{\rho} \le 2^{i-1}$.

The above lemmas can be used to show the following main result of this section.

Theorem 75. *Let G be a weighted tree or an unweighted graph. Let s be the randomized deepening strategy on G if G is a tree, or on some shortest path tree of G if G is an unweighted graph. Then the approximation ratio of s is asymptotically 5/4. In particular, $\rho_s \le (5/4)\rho + 1/2$.*

Bounding the randomized competitive ratio as function of the number of vertices

In this section we ask how large the randomized competitive ratio can be for a graph with n vertices plus the root. First note that the equivalent question for the (deterministic) competitive ratio is easily settled. Indeed, for an arbitrary weighted graph with minimum edge length normalized to 1, consider the search strategy that visits the vertices in non-decreasing order of their distance from the root. The normalized search time of the jth vertex to be visited is at most $j \le n$, so $\sigma \le n$. This is tight for a *uniform star*, that is a star with edges of equal length.

In contrast, it is not as easy to bound the *randomized* competitive ratio of an arbitrary weighted graph. Note that the randomized competitive ratio of a uniform star is $(n + 1)/2$, since an optimal Searcher strategy is to search the edges in a uniformly random order and an optimal Hider strategy is to choose one of the n vertices uniformly at random. In this section we will show that the uniform star is, in fact, the weighted graph with the largest randomized competitive ratio. In the process, we will explore connections with another optimization problem from databases.

We begin by focusing on stars; the general result then follows as a corollary. Here, the problem of finding the randomized competitive ratio is related to a problem in *pipelined filter ordering*. As mentioned in Section 1.2, star search is connected to the work of [65], who study an adversarial pipelined filter ordering problem. The model can be described as follows. An item, or *tuple* must be routed in some order through a set $\{O_1, \ldots, O_n\}$ of *operators*, each of which tests whether the tuple satisfies some predicate (or *filter*) of a conjunction. There is a known cost c_i of each operator O_i to process a tuple, and the tuple is routed through the operators until it fails one of the tests (and is eliminated) or it passes all of them. We may alternatively think of this as a product being subjected to several quality tests before being sent to the market.

In the problem of [65], an adversary chooses the set of filters which will eliminate the tuple. The aim is to choose a randomized routing of the tuple to minimize the *multiplicative regret*: that is the expected ratio of the total cost of eliminating the tuple to the cost of eliminating the tuple if the adversary's choice were known *a priori*. The authors argue that we may as well assume the adversary chooses only one filter to eliminate the tuple, as the adversary is not disadvantaged by this restriction. If the ordering chosen is (without loss of generality) $O_1, \ldots, O_n$ and the adversary chooses the filter corresponding to operator O_j, then the multiplicative regret is $(c_1 + \cdots + c_j)/c_j$. It is easy to see that the problem is exactly equivalent to our problem of calculating the optimal randomized search for a star graph with edges of lengths $c_1, \ldots, c_n$.

Condon *et al.* [65] find a polynomial time algorithm for their problem, and we summarize their result in the context of the randomized competitive ratio of a star graph.

Theorem 76 (From Section 4 of [65])**.** *Let G be a star graph with root O and vertices $v_1, \ldots, v_n$ at distances $c_1, \ldots, c_n$ from O, where the c_i are non-decreasing. There is a polynomial-time search algorithm that calculates the optimal randomized search of G. The randomized search ratio $\rho(G)$ is given by*

$$\rho(G) = \max_{1 \le k \le n} \frac{\sum_{1 \le i \le j \le k} c_i c_j}{\sum_{i=1}^{k} c_i^2}. \tag{3.14}$$

Note that the right-hand side of (3.14) is the maximum over all choices of $\mathcal{A} = \{v_1, \ldots, v_k\}$ of the value of the first lower bound in Lemma 69. It follows that the optimal strategy of the Hider is to choose such a set $\mathcal{A}$ which maximizes this bound and hide at vertex $v_i \in \mathcal{A}$ with probability proportional to c_i^2. The optimal strategy for the Searcher cannot be described so succinctly, and we refer to [65] for details of their algorithm.

We now show how to apply this result so as to bound the randomized search ratio for weighted stars; we also show that the bound holds for arbitrary weighted graphs.

Proposition 77. *Let G be a weighted star with n vertices plus the root. The randomized competitive ratio ρ of G satisfies $\rho \le (n+1)/2$.*

Proof. By Theorem 76, the randomized competitive ratio is equal to the right-hand side of (3.14). Suppose this expression is maximized by $k = k^*$ and, without loss of generality, assume that $\sum_{i=1}^{k^*} c_i = 1$. Then

$$\rho = \frac{\sum_{1 \le i \le j \le k^*} c_i c_j}{\sum_{i=1}^{k^*} c_i^2} = \frac{\frac{1}{2}\left(\sum_{i=1}^{k^*} c_i\right)^2 + \frac{1}{2}\sum_{i=1}^{k^*} c_i^2}{\sum_{i=1}^{k^*} c_i^2} = \frac{1}{2\sum_{i=1}^{k^*} c_i^2} + \frac{1}{2}. \tag{3.15}$$

The right-hand side of (3.15) is maximized when all the c_i's are equal to $1/k^*$, so that $\rho \le (k^*+1)/2 \le (n+1)/2$. □

Using Proposition 77, we obtain the following result.

Theorem 78. *Let G be a weighted graph with n vertices plus the root O. The randomized competitive ratio ρ of G satisfies $\rho \le (n+1)/2$.*

3.4.3 Expanding search in a network

In this section we study the competitive ratio of expanding search in general networks. More precisely, we consider a search domain that is represented by a connected network Q which consists of vertices and arcs, and which has a certain vertex O designated as its root. Moreover, Q is endowed with Lebesgue measure corresponding to length. The immobile hider can then hide anywhere along the network, which can be bounded, or even unbounded.

There are several reasons that motivate this setting. First, searching in a network is a general problem that goes back to early work by Isaacs [87] and Gal [75] in the context of pathwise search. Moreover, as we will show, there are certain similarities, but also interesting differences when comparing to expanding search in a graph (which was discussed in the previous section). Last, we develop some useful tools that are applicable not only in expanding search, but also in pathwise search. In particular, we show that our techniques yield an improved bound on the randomized competitive ratio of pathwise search in general networks.

We begin with some preliminaries concerning the network setting.

Preliminaries

Given a network Q with root O, we will denote the measure of a subset A of Q by $\lambda(A)$, and in the case that Q has finite measure, we will denote by $\mu = \lambda(Q)$ the total measure of Q. This defines a metric on Q, where $d(x, y)$ is the length of the shortest path from x to y. We write $d(x)$ for the distance $d(O, x)$ from O to x. We denote by $\deg_Q(v)$ the degree of v in Q, namely the number of arcs incident to v.

Given a network Q, and any $r \geq 0$, we denote the closed disc of radius r around O by $Q[r] = \{x \in Q : d(x) \leq r\}$. Let $r_{\max} = \max_{x \in Q} d(x)$ be the distance of the furthest point in Q from O, where $r_{\max} = \infty$ if Q is unbounded. We define the real function $f_Q : [0, r_{\max}] \to \mathbb{R}$ given by $f_Q(r) = \lambda(Q[r])$, so $f_Q(r)$ is the measure of the set of points at distance no more than r from the root.

Example 79. *Figure 3.3 depicts a network Q and the graph of the function $f_Q(r)$. The numbers on the arcs denote the corresponding lengths.*

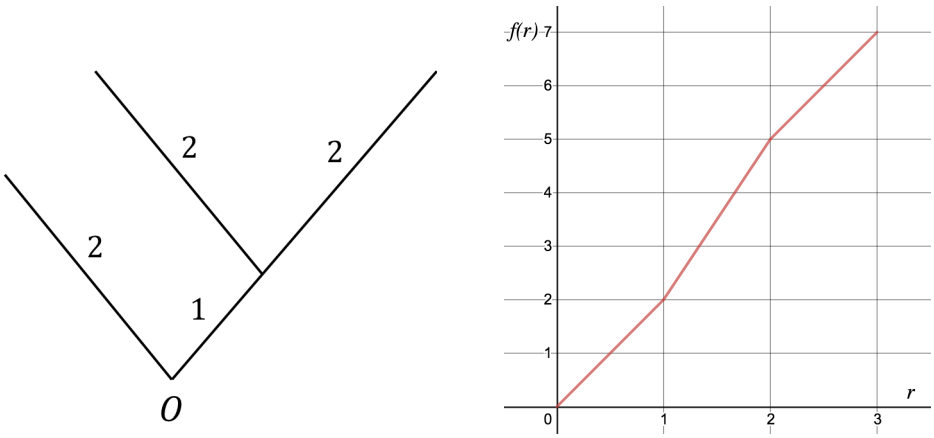


Figure 3.3: The calculation of the function $f_Q(r)$ (right) for a particular network Q (left).

We do not limit ourselves to bounded networks, but allow the network to be unbounded. In this case, we make the standing assumption that the network Q satisfies the condition that there exists some integer k such that for any $r > 0$,

$$|\{x \in Q : d(x) = r\}| \leq k. \tag{3.16}$$

That is, there are at most k points at distance r from O. Any network with a finite number of arcs automatically satisfies this condition. This condition ensures that the competitive ratio exists.

As an example of an unbounded network, if Q is an m-ray star, we have $|\{x \in Q : d(x) = r\}| = m$, for all r, hence this quantity is bounded. In contrast, if Q is an unbounded full binary tree in which there are 2^i vertices at distance i from the root, for all $i \in \mathbb{N}^+$, then this quantity is unbounded, and this implies the competitive ratio is also unbounded. Indeed, any deterministic search strategy (pathwise or expanding) on this network must take at least time 2^i to reach all points at distance i from the root, so the deterministic competitive ratio would be at least $2^i/i$, which is unbounded.

We can formally define expanding search in a network as an extension of the corresponding definition in the graph setting.

Definition 80 ([7])**.** *An expanding search on a network Q with root O is a family of connected subsets $S(t) \subset Q$ (for $0 \le t \le \mu$) satisfying: (i) $S(0) = O$; (ii) $S(t) \subset S(t')$ for all $t \le t'$; and (iii) $\lambda(S(t)) = t$ for all t.*

Given a strategy S and a hider H, we will use the same notation $T(S, H)$ to denote the search time for locating H in S as in the case of expanding search in a graph. The same holds for the concepts of the deterministic and randomized competitive ratios (see Section 3.4.2). In particular, we can view the randomized competitive ratio of a network as the *value* of a zero-sum game between the searcher and the hider.

Note that the competitive ratio of a strategy S may be infinite. For example, suppose that Q consists of two unit-length arcs a and b meeting at the root and suppose S searches a first and then b. If H lies on the arc b at distance x from the root then $\hat{T}(S, H) = (1+x)/x = 1+1/x \to \infty$ as $x \to 0$. It is not immediately obvious whether or not the competitive ratio of a network is finite in general, but we will show later that this is indeed the case, by explicitly giving the optimal search strategy for any network.

It is not so straightforward to show that the game has a value if Q is unbounded. Nonetheless, this is not important for our analysis, and we will rely on the following general result for zero-sum games that for any mixed Hider strategy h,

$$\rho(Q) \ge \inf_S \hat{T}(S, h), \tag{3.17}$$

where the infimum is taken over all search strategies S.

We can now proceed with our results.

Deterministic, expanding competitive ratio

We can obtain an expanding search of optimal deterministic ratio, using a "water filling" principle. Informally, the network is searched in such a way that the set of points that have been searched at any given time form an expanding disc around O.

Recall the definition of f_Q from the previous section. f_Q is piecewise linear and strictly increasing so has an inverse g_Q. The interpretation is that $g_Q(t)$ is the unique radius r for which $Q[r]$ has measure t.

Definition 81. *For a network Q with root O, consider the expanding search S^* defined by $S^*(t) = Q[g_Q(t)]$ for $0 \le t \le r_{\max}$.*

Thus, $S^*(t)$ is an expanding disc of radius $g_Q(t)$. It is easy to verify that S^* is indeed an expanding search. First, we note that $S^*(t)$ is connected, since $Q[r]$ is always connected. It also trivially satisfies (i) and (ii) from Definition 80, and (iii) is also satisfied since

$$\lambda(S^*(t)) = \lambda(Q[g_Q(t)]) = f_Q(g_Q(t)) = t.$$

It is also not difficult to show that S^* attains the optimal competitive ratio. First, note that the search time of a point $H \in Q$ under S^* is the unique time t such that $S^*(t) = Q[d(H)]$, so $T(S^*, H) = \lambda(Q[d(H)]) = f_Q(d(H))$. Hence, the competitive ratio of S^* is

$$\sigma_{S^*} = \sup_{H \in Q - \{O\}} \frac{f_Q(d(H))}{d(H)} = \sup_{r>0} \frac{f_Q(r)}{r}. \tag{3.18}$$

This has an intuitive interpretation as follows: if we draw the graph of $f_Q(r)$ then the competitive ratio is the slope of the steepest straight line through the origin that intersects with the graph of $f_Q(r)$. Condition (3.16) ensures that σ_{S^*} is finite for unbounded networks, since $f_Q(r) \leq kr$ for all r.

Theorem 82. *The expanding search S^* is optimal and the competitive ratio σ of a network Q with root O is given by*

$$\sigma = \sup_{r>0} \frac{f_Q(r)}{r}. \tag{3.19}$$

Randomized, expanding competitive ratio

In this section we study the randomized competitive ratio of expanding search, which is significantly more challenging to analyze than the deterministic one. We begin with a simple result: that the randomized competitive ratio is at least half the deterministic competitive ratio and that there exist networks for which this bound is tight. Recall that S^* denotes the optimal deterministic search strategy of the previous section.

Proposition 83. *For a network Q with root O, the randomized competitive ratio ρ satisfies*

$$\sigma/2 \leq \rho \leq \sigma.$$

Furthermore, the bounds are tight, in the sense that they are the best possible.

Proof sketch. The right-hand inequality is clear, since every deterministic search strategy is also a randomized search strategy. To prove the left-hand inequality, we first observe that since S^* is an optimal deterministic search, for any $\varepsilon > 0$, we can find some point H on Q such that $\hat{T}(S^*, H) \geq \sigma - \varepsilon$. Let $r = d(H)$ so that $\sigma \leq f_Q(r)/r + \varepsilon$. The inequality follows by considering the hider strategy that hides on $Q[r]$ uniformly, and applying Yao's principle. The tightness of this bound follows by considering the network depicted in Figure 3.4. □

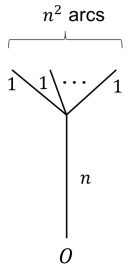


Figure 3.4: A network for which $\rho \approx \sigma/2$.

A corollary of Proposition 83 is that the "water-filling" search S^* approximates the optimal randomized search by a factor of 2.

As a next step, we will give a Hider strategy that allows us to get useful lower bounds on the randomized competitive ratio (Lemma 85 below). We also obtain upper bounds on ρ that are parameterized by the function f_Q (Corollaries 86 and 88 below) from which we can deduce the

randomized competitive ratio for certain classes of networks, for example when f_Q is a concave. This finding may have practical implications in the context of searching in a big city. This is because f_Q can be seen as a "density" measure, and a typical road network is usually more dense in its center than it its outskirts. Thus, one expects this density (and therefore f_Q as well) to decrease the further we move from the city center.

For a general network Q, let A be a connected subset. Let $d(A)$ be the distance from O to A and let u_A be the Hider strategy (probability measure) that hides uniformly on A, so that $u_A(X) = \lambda(X)/\lambda(A)$ for a measurable subset $X \subset A$. Then denote the average distance from O to points in A by $\overline{d}(A) = \int_{x\in A} d(x)du_A(x)$.

Theorem 84. *Consider the Hider strategy h_A given by*

$$dh_A(x) = \frac{d(x)}{\overline{d}(A)} du_A(x).$$

By adopting the strategy h_A, the Hider ensures that the randomized competitive ratio ρ satisfies

$$\rho \geq \frac{d(A) + \lambda(A)/2}{\overline{d}(A)}.$$

Theorem 84 allows us to obtain the following lower bound.

Lemma 85. *For any network Q with root O, it holds that $\rho \geq \deg_Q(O)$.*

The above lemma implies a tight bound on the randomized competitive ratio for all networks Q for which the function f_Q is concave, as shown in the following corollary.

Corollary 86. *For any network Q for which f_Q is concave, we have that $\sigma = \rho = \deg_Q(O)$, and strategy S^* is an optimal randomized strategy.*

Proof. The lower bound on ρ follows from Lemma 85. For the upper bound, by (3.19), we have $\rho \leq \sigma = \sup_{r>0} f_Q(r)/r$, and for any network for which f_Q is concave, it holds that $\sup_{r>0} f_Q(r)/r = \deg_Q(O)$. □

Note that Corollary 86 applies to star networks with m rays, since in this case, f_Q is linear; thus, $\sigma = \rho = m$.

Example 87. *An example of a network for which f_Q is concave is depicted in Figure 3.5, along with a plot of its function f_Q.*

More generally, we have established the following approximation.

Corollary 88. *Suppose that for the network Q it holds that $\sup_{r>0} f_Q(r)/r \leq \alpha \deg_Q(O)$, for some $\alpha > 1$. Then S^* approximates the optimal randomized ratio of Q within a factor at most α.*

A natural question that arises is what can we say for networks for which the function f_Q is not concave, and thus Corollary 86 does not apply. The answer is that finding optimal randomized strategies becomes a very complicated task. Consider the simplest such network, which is depicted in Figure 3.6. For this network, one can indeed find optimal strategies, which, however have a complex statement as we show in [19].

There is, however, some good news. We can obtain a good approximation of the optimal randomized ratio, by applying the randomized doubling strategy that we used in expanding search in a weighted tree, suitably adapted to the network setting. Using an analysis along the lines of the proof of Theorem 75, we obtain the following theorem.

Theorem 89. *The randomized doubling strategy s is a $\frac{5}{4}$-approximation of the optimal randomized search. In particular, $\rho_s \leq (5/4)\rho + 1/2$.*

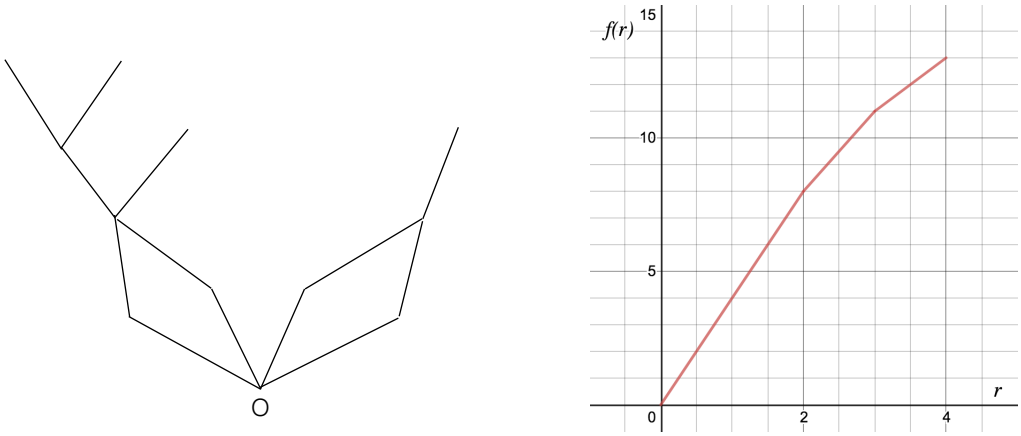


Figure 3.5: An example of a network Q (left) and the function $r \mapsto f(r) \equiv f_Q(r)$ (right). All arcs in Q are unit-length.

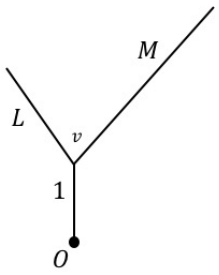


Figure 3.6: A simple network for which optimal randomized strategies are difficult to find. Here, L and M are general parameters.

Randomized pathwise competitive ratio

The techniques we developed in the previous section can be useful not only in expanding search, but in pathwise search as well. Consider the problem of searching for a hider in a network Q, in the pathwise paradigm. An obvious first approach is to apply the doubling technique of [104], in the context of a general, and possibly unbounded network rather than a fixed graph. We can show that this approach yields a $3 + 2\sqrt{2} \approx 5.828$-approximation of the randomized competitive ratio.

We can, however, obtain a better approximation, by adapting the randomized doubling strategy of the previous section to the pathwise setting.

Theorem 90. *The (pathwise) random doubling strategy s is a 5-approximation for the optimal randomized pathwise search. That is, $\rho_s \leq 5\rho$.*

3.5 Discussion

Concerning weighted search, our reduction from ßSubset Search could be useful in other settings. In particular, we believe it can be applied in the context of measures that are a relaxation of competitive analysis, such as the ones discussed by McGregor *et al.* [112] and Kirkpatrick [100]. Here, the offline algorithm knows the characteristics of the instance (the pairs of distances and weights for each target), but not the exact correspondence of targets to rays.

Concerning bijective analysis, one may ask whether there is additional evidence that bijective analysis points to the "right" strategy. Of course, the aggressive strategy has intuitive properties

that makes it a very attractive candidate among all competitively optimal strategies, but can we nevertheless argue about this using a different formal criterion? One could, for example, consider the *average* discovery cost of strategies, namely the average cost at which the strategy searches an aggregate length of L, for any given L. Computational experiments show that `aggressive` strictly dominates `doubling` for all values of L, and we believe this is a statement that should not be difficult to show. However, it does not look like average analysis by itself suffices to establish the dominance of `aggressive` against any other competitively optimal strategy.

Concerning expanding search, the relation between the problems of pipelined order filtering and expanding search that we established in Section 3.4.2 implies that our results apply to the following generalization of the model in [65]. Suppose the order that the tuple may be routed through the operators is subject to some precedence constraints. This is, there is a partial order $\prec$ on the set of operators such if $O_i \prec O_j$ then the tuple must be processed by O_i before it can be processed by O_j. The partial order defines a directed acyclic graph, and if that graph is a tree, then the problem of finding the randomized routing of the tuple to minimize the expected regret is equivalent to our problem of finding the optimal randomized search on a tree. By Theorem 75, the randomized deepening strategy has approximation ratio 5/4 for this problem.

Moreover, in our work we used Theorem 76 from [65], however we should emphasize that the resulting optimal strategy cannot be described succinctly, since [65] applies an involved, flow-based approach. For this reason, in [17] we gave an alternative proof of Proposition 77 that uses a simpler combinatorial strategy (based from intuition obtained from the unweighted case), and a game-theoretic approach.

Chapter 4

Contract scheduling and interruptible algorithms

In this chapter we focus on the problem of scheduling executions of contract algorithms so as to obtain an interruptible system. First, we introduce and study a new setting in which a contract algorithm is deemed complete once one of its executions has reached a certain end guarantee (Section 4.1). We then argue that there are settings in which the acceleration ratio may not truly reflect the quality of the schedule, and introduce and study new performance measures for contract scheduling (Section 4.2). Last, in Section 4.3 we explore connections between various variants of contract scheduling and ray searching, and argue that both settings can benefit from a common approach.

The material of Section 4.1 is based on work that appeared in the Proceedings of the 28-th International Joint Conference on Artificial Intelligence (IJCAI-19) [18]. Section 4.2 appeared in the *Journal of Scheduling* [20] (preliminary version in IJCAI-09). Section 4.3 appeared in the Proceedings of the 24-th International Joint Conference on Artificial Intelligence (IJCAI-15) [9].

We refer the reader to the discussion in Section 1.3 for background and previous work on contract scheduling.

4.1 Scheduling contract algorithms with end guarantees

4.1.1 Background

All previous work on contract scheduling based on the acceleration ratio has assumed that the interruption may appear arbitrarily far ahead in the future. This implies that the schedule has to be unbounded, and the theoretical analysis must incorporate such an assumption.

However, in practice, we expect that the execution of a schedule of contract algorithms will reach a point beyond which any progress over the problem instances will be only marginal. For instance, [43] show that functions of the form $Q(t) = 1 - e^{-\lambda t}$ can be used to model the expected performance of their anytime planner. This means that beyond a certain t_0, $Q(t)$ increases quite slowly (see also the discussion in [132]). Similar sharp thresholds can be observed on the performance profiles of anytime algorithms that relate computation time and precision of output in a medical diagnostic system [85]. Therefore, one can argue that a schedule of contracts may be deemed *complete* once certain *end criteria* on the efficiency of completed contracts have been met, in terms of the computational time that has been allotted to the problem instances.

We adopt the following definition of end guarantees, which is in line with the worst-case nature of the acceleration ratio as well as the previous observation on the performance profiles of typical contract algorithms: we allow the schedule to complete once each problem instance has finished a schedule of length at least a target value L. In particular, L can be defined in terms of the problem instance whose contract algorithm requires the largest deadline in order to achieve

a satisfactory performance, among all instances.

In our work we study contract scheduling in the above setting. In particular, we seek a *finite* schedule with the following properties: i) it attains the optimal acceleration ratio, as in the standard model; and ii) it minimizes the time required to satisfy the end guarantees, among all schedules that obey property (i). In other words, if an interruption occurs during the execution of the schedule, the schedule has optimal performance; otherwise, if no interruption has occurred, it outputs a solution that meets the end guarantees the earliest possible, among all schedules optimal with respect to the acceleration ratio.

Our contribution is an optimal schedule for the problem described above, namely for earliest completion scheduling of contract algorithms with end guarantees. We propose a schedule that is provably optimal, and can be computed in time polynomial in the size of the end guarantee L (and thus in the size of the input, under the reasonable assumption that the number of problem instances n is constant, independent of L). In addition, we present 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.

To formalize our setting, let us introduce some notation and definitions. Given a schedule X, and an end guarantee $L \in \mathbb{R}^+$, we say that X is *feasible for* L if for each instance p there is a contract for p in X that has length at least L. Let $T(X)$ denote the completion time of a finite schedule X. A schedule X that is feasible for L is called *earliest for* L if for any other schedule X' feasible for L, $T(X) \leq T(X')$. Let ρ_n^* denote the optimal acceleration ratio of an infinite schedule for n problems, and let S_n^* denote the set of all finite schedules that achieve this optimal acceleration ratio ρ_n^*.

Using the above 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 91. *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 optimal acceleration ratio equal to 4.*

In this work we demonstrate that techniques based on linear programming, which have not been explored in previous work, can be very useful. Indeed, to our knowledge, all previous work is based on a variant of iterative doubling; in contrast, we use the structure of optimal LP solutions in order to obtain schedules that are described by more complex, yet still efficiently computable recurrence relations. A main technical obstacle one has to overcome is to derive the recurrence relations from the saturated LP constraints. We emphasize that we do not need to solve any linear program, and that it is used only as a guide in the design and analysis of the schedule.

The remainder of this section is structured as follows. We begin in Section 4.1.2 by showing that there exist optimal schedules for our problem that are *cyclic*, i.e., contracts are assigned to problems in round-robin fashion. This allows us to formulate our setting by means of a linear program (LP). Section 4.1.3 describes the main technical steps in the design and analysis of our schedule. More specifically, we prove that an optimal cyclic strategy saturates the constraints of the LP. In turn, this allows us to define an appropriate recurrence relation over the contract lengths, which yields the optimal schedule. Section 4.1.4 provides a computational evaluation of the schedule.

4.1.2 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 means of an LP. To this end, we first define a useful property. Recall that $\ell_{q,t}(X)$ denotes the largest contract length completed in X at time t.

Definition 92. *A schedule $X = ((x_i, p_i))_{i=1}^m$ 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 [105].

Lemma 93. *For every schedule X feasible for L, there exists a normalized schedule X' feasible for L such that $acc(X') \leq acc(X)$ and $T(X') \leq T(X)$.*

The next theorem is central in that it allows us to obtain an LP formulation for our problem. The theorem implies that optimal schedules can be found in the space of *cyclic* and *monotone* schedules. In particular, monotonicity means that for any i, the length of the i-th contract in the schedule does not exceed the length of the $(i+1)$-th contract in the schedule.

Theorem 94. *Given the end guarantee L, there is a cyclic and monotone schedule that is optimal.*

We explain how to formulate an appropriate LP using Theorem 94. Suppose, first that the number of contracts in the optimal schedule is known, and equal to m (we will argue later how to remove this assumption). Let ρ_n^* denote the optimal acceleration ratio of an infinite schedule for n problems, as given by (1.3). Then the optimal contract lengths are the 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; 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 in order 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.

4.1.3 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 96). This lower bound is expressed in terms of T_{i-1} and two sequences (a_i) and (b_i), which are defined appropriately in order to satisfy the inductive arguments. Next, we need to find the right value of m, the number of contracts in an optimal schedule. 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 97). 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 98).

We define the sequence a and b recursively as follows:

$$a_i = \begin{cases} 1, & i \in [0, n-1] \\ \frac{\sum_{j=0}^{n-1} a_{i-n+j} \prod_{k=0}^{j-1}(b_{i-n+k}+1)}{\rho_n^* - \prod_{k=0}^{n-1}(b_{i-n+k}+1)}, & i \geq n \end{cases} \tag{4.1}$$

and

$$b_i = \begin{cases} 0, & i \in [0, n-1] \\ \frac{\prod_{k=0}^{n-1}(b_{i-n+k}+1)}{\rho_n^* - \prod_{k=0}^{n-1}(b_{i-n+k}+1)}, & i \geq n. \end{cases} \tag{4.2}$$

Lemma 95. *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 96. *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.*

Lemma 96 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 97. *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*

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

From Lemma 96 and 97, 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 96.

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 96, $x_i > 0$. 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 98. *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 97, 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 95, 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^*. 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 3 summarizes the steps needed to obtain the optimal schedule.

Algorithm 3: Earliest-completion scheduling of contract algorithms with end guarantee L

Input: $n \geq 1, L > 0$ and $\tau > 0$
$U \leftarrow \lceil \frac{\log L}{\log b} + n \rceil$
Compute $(a_i)_{i\in[1,U]}, (b_i)_{i\in[1,U]}$ using the recurrence relations (4.1) and (4.2)
for $m = n+1 \dots U$ **do**
    if $a_{m-1}L \leq \tau$ **then**
        Output x_i, using the recurrence relation $x_i = a_{m-i}L + b_{m-i}T_{i-1}(X)$, and stop.
    end
end

4.1.4 Computational evaluation

In this section we present computational results on the implementation of our schedule, whose completion time recall we denote 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 4.1 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.

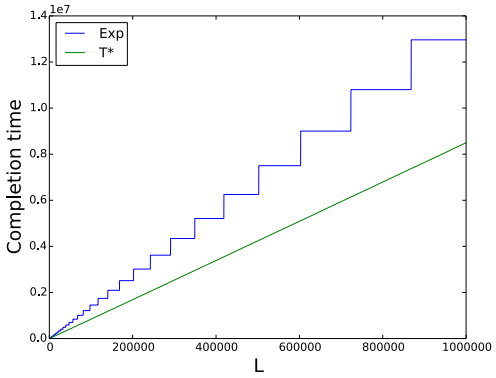


Figure 4.1: Completion times $T^*(L)$ and $T_{\texttt{exp}}(L)$ as function of L for $n = 5$.

Figure 4.2 illustrates the ratio $T_{\texttt{exp}}(L)/T^*(L)$ for $n \in \{1, 2, 20\}$. We observe that the fluctuations of the ratio, as function of L, tend to decrease in n.

Figure 4.2 motivates the evaluation of the ratio $T_{\texttt{exp}}(L)/T^*(L)$, for large L. This is shown in Figure 4.3. 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 run time of the implementation. In our computational evaluation, we observed that the values (a_i) described by (4.1) 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 $O(n \log(\log n + \log\log L))$.

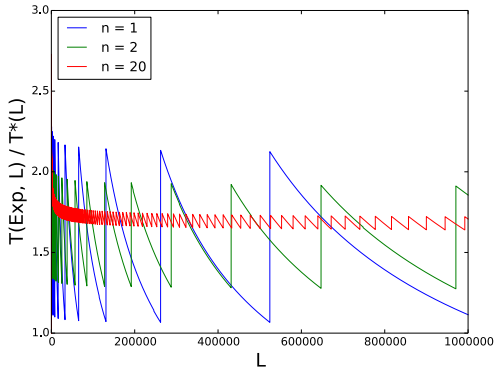


Figure 4.2: Ratio $T_{\texttt{exp}}(L)/T^*(L)$ as function of L.

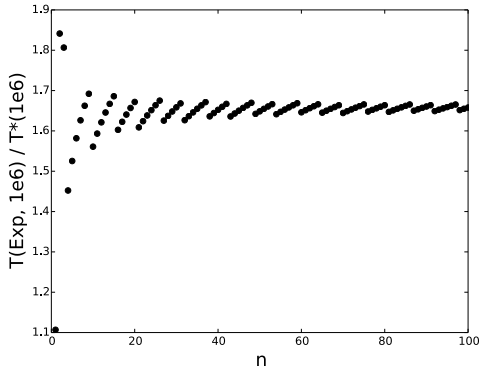


Figure 4.3: Ratio $T_{\texttt{exp}}(L)/T^*(L)$, for $L = 10^6$.

4.2 New measures for evaluating contract schedules

4.2.1 Background

In this section we have a closer look at the acceleration ratio as a performance measure. To illustrate better the arguments, we will assume the most general setting in which there is a set P of n problem instances, and the contracts are scheduled in m identical, parallel processors [38, 105]. Note that the definition of the acceleration ratio still applies in this setting: the measure compares the worst-case ratio of the interruption time divided by the largest contract completed by the interruption, among all processors. We will have a discussion about the intuition behind the measures, then the main measure will be formally defined in Corollary 103.

The central observation that motivates this work is that the acceleration ratio becomes problematic, as a performance measure, when at interruption time the algorithm is required to return a solution to *all* n *instances in* P instead of only to a specific queried problem instance. This arises, for example, in a medical diagnostic system which must perform concurrent evaluations for a number of medical issues. Here, the decision of the expert may very well have to take into account the entirety of these issues.

To explain why the acceleration ratio may not reflect the true performance of the system, suppose that we are given a set P of n problem instances (indexed $0, \ldots, n-1$) and a set M of m identical processors (indexed $0, \ldots, m-1$). Suppose, in particular, that $n > m$, and for the purposes of illustrating our argument, say that $n \gg m$. Note that it is not feasible for any ideal, optimal algorithm (that is, an algorithm with advance knowledge of the interruption time t) to schedule n contracts of length t to m processors, since $n > m$. In a sense, if we applied

the acceleration ratio to this domain, we would compare the performance of an interruptible algorithm which is expected to make progress on all n instances to an algorithm which only makes optimal progress on at most m: such a comparison is rather not fair. This shortcoming was noticed by Zilberstein *et al.* [133], who define a scaled-down variant of the acceleration ratio for the case of n problems and one processor as the quantity

$$\sup_t \max_{p \in P} \frac{t/n}{\ell_{p,t}}. \tag{4.3}$$

This measure describes, informally, an even distribution of the processor time among the n problem instances for the optimal (offline) schedule.

A different way of arguing about the above shortcoming is to consider the scenario in which at interruption time t a single problem instance $p \in P$ is queried. Naturally, the interruptible algorithm knows neither t or p in advance, but then the optimal offline algorithm should be oblivious of p as well, otherwise it would make progress only on p while ignoring all other instances in P. This suggests that the offline optimal algorithm implicit in the definition of the acceleration ratio is overly powerful, and better measures are needed for the setting that we study.

Motivated by the above observations, in this work we address the problem of designing interruptible algorithms using schedules of executions of contract algorithms, assuming m identical processors and n problem instances. We begin by considering measures alternative to the acceleration ratio (Section 4.2.2), and we propose the *deficiency* of a schedule as our measure of choice. In Section 4.2.3 we present a schedule whose deficiency is very small and rapidly decreasing in the ratio n/m (in contrast, the acceleration ratio of every schedule is known to approach infinity when $n/m \to \infty$). More precisely, we show analytically that its deficiency is at most 3.74 if $n \leq m$, and at most 4, if $n > m$. A numerical evaluation provides even smaller values.

Even though the deficiency of the proposed schedule is small, we provide further theoretical justification for the efficiency of our solution. Namely, in Section 4.2.4 we present several lower bounds on the deficiency of schedules in the single-processor setting ($m = 1$). More precisely, we prove a general lower bound of $(n + 1)/n$, an improved bound for the two-problem setting ($n = 2$), and a tight lower bound for round-robin schedules. The proofs are based on techniques originally developed in the context of search theory [76]; in particular we will make use of Gal's theorem (Theorem 55), which will allow us to lower-bound the performance of schedules with arbitrary contract lengths to that of schedules with geometrically increasing lengths.

It is worth noting that revisiting the definition of an ideal, optimal algorithm (and introducing new measures that capture this weakening) is conceptually related to similar considerations we discussed in Chapters 2 and 3. Namely, we would like to circumvent the concept of an offline optimal algorithm that is often overly powerful, and lead to pessimistic results.

4.2.2 Problem formulation and comparison of measures

Consider an (infinite) schedule X of executions of a contract algorithm over n problem instances. For an interruption time t, we will denote by $\ell(X, p, t)$ the length of the longest contract for instance $p \in P$ which is completed by time t in X (or simply $\ell_{p,t}$ when X is implied from context).

We need to formalize the question: what is the best way to assign the available resources (i.e., processing time) to the set of instances P? Towards this end, consider a schedule Y, which, in contrast to X, is finite: more precisely, Y schedules n distinct contracts, one for each instance in P (if Y schedules more than one contract per instance, than we can transform Y to a schedule Y' which is at least as good as Y by keeping only the largest contract per instance that appears in Y). Each contract in Y is scheduled in one of the m processors in M. We require that Y is *feasible with respect to* t, in the sense that the *makespan* of Y, namely the total sum of contract lengths on the most loaded processor used by Y does not exceed t. Let $\mathcal{Y}_t$ denote the class of all

schedules Y with the above properties. We will be calling Y an *offline solution* since it relies on advance knowledge of t.

Having defined $\mathcal{Y}_t$, we need a measure of how a schedule $Y \in \mathcal{Y}_t$ compares to X, which will also dictate which is the *best* schedule in $\mathcal{Y}_t$ compared to X. First observe that the acceleration ratio is not an appropriate measure for our setting, since under it the optimal offline schedule Y for interruption t dedicates all its resources to the contract that is worked on the least by X while failing to produce an answer for all other problems instances. This results in a large acceleration ratio which however does not truly reflect the quality of X (effectively, the optimal solution "cheats" by ignoring all but one problem instances, which is not acceptable in our setting).

Another alternative would be to compare the smallest contract completed in X by time t to the smallest contract completed in Y by time t. We will need some preliminary definitions first. Let S_X^t denote the set

$$\{\ell(X, j, t) | j \in [0, n-1]\},$$

that is, the set of the largest contracts per problem instance completed by time t) and let $S_X^t(i)$ denote the i-th smallest element of S_X^t. Similarly, for $Y \in \mathcal{Y}_t$ let S_Y^t denote the set of n contracts in Y, and $S_Y^t(i)$ be the i-th smallest contract in Y, respectively (ties are resolved arbitrarily).

Formally, we define the *performance ratio of X with respect to Y at time t* as:

$$\text{perf}(X, Y, t) = \frac{\min_i S_Y^t(i)}{\min_i S_X^t(i)} = \frac{S_Y^t(1)}{S_X^t(1)}. \tag{4.4}$$

The performance ratio of X at time t is then defined as

$$\text{perf}(X, t) = \sup_Y \text{perf}(X, Y, t),$$

where Y is a feasible schedule with respect to t. Last, the performance ratio of X is defined as

$$\text{perf}(X) = \sup_t \text{perf}(X, t).$$

The first observation is that under this measure there exists an "optimal" offline schedule in which all contracts have the same length: here, by "optimal" we mean a schedule $Y \in \mathcal{Y}_t$ against which the performance ratio of X is maximized. Indeed, given any offline schedule Y, consider any schedule Y' such that the length of all its contracts is $\min_i S_Y^t(i)$. Note that such a feasible Y' exists, since all contracts in Y are at least that long, and Y itself is feasible. Then it follows from the definition that $\text{perf}(X, Y, t) = \text{perf}(X, Y', t)$. We can show a close relationship of the performance ratio to the acceleration ratio for the standard setting (namely, when we seek the solution only to the queried problem).

Lemma 99. *There is a schedule X such that for any arbitrary interruption time t*

$$\textit{perf}(X, t) = \begin{cases} \frac{n}{m}\left(\frac{m+n}{n}\right)^{\frac{m+n}{m}} & \text{for } m \geq n \\ \frac{n}{m}\frac{1}{\lceil n/m \rceil}\left(\frac{m+n}{n}\right)^{\frac{m+n}{m}} & \text{for } m < n. \end{cases}$$

Furthermore, X is optimal with respect to this measure.

We note that for the case of one processor ($m = 1$) and n problem instances, Lemma 99 shows that the performance ratio of the optimal schedule matches the measure proposed by Zilberstein *et al.* [133], expressed by (4.3).

It is also not difficult to show that, unlike the optimal acceleration ratio (which is bounded by a function linear in $\frac{n}{m}$), the optimal performance ratio is bounded by a small constant; in other words, the schedule in the proof of Lemma 99 is very efficient under this refined measure.

Proposition 100. *The performance ratio of the schedule in the proof of Lemma 99 is bounded by $2e$, if $n > m$, and by 4 , if $n \leq m$.*

While the performance ratio seems to be a better candidate for a measure in the context of our setting than the acceleration ratio, it is far from being the best possible choice. A way to interpret this measure is that it every contract in the optimal offline solution may have fixed length, namely $t/\lceil n/m \rceil$. While it is guaranteed that the smallest contract in S_X^t indeed does not exceed $t/\lceil n/m \rceil$, the solution produced by X may be such that there exist several contracts in S_X^t which exceed this length. More formally, there may exist j such that $S_X^t(j) > S_Y^t(j)$ for the optimal offline solution Y. This is clearly undesirable, since it becomes difficult to argue that the optimal offline solution Y at time t is indeed better than the solution produced by the interruptible algorithm at time t, even though, supposedly, Y is optimal.

The above motivates the need for defining a further measure, one which takes into account the intuitive expectation that the optimal offline solution should perform better than the interruptible algorithm *on each problem instance*. To accomplish this, we allow the offline solution to observe the behavior of X at each point in time t, and then produce an appropriate "optimal" offline solution, tailored to the specifications of our setting. In a sense we allow the offline solution sufficient power in choosing its schedule, while at the same time we require that it produces solutions to all problem instances. This yields a measure which we call *deficiency*, and which is: i) consistent with the requirements of the setting we study; and ii) powerful enough, in the sense that if an algorithm performs well with respect to the new measure, then there are very strict guarantees about its performance.

Formally, we say that Y is *at least as good as* X, denoted by $Y \geq X$, if and only if $S_Y^t(i) \geq S_X^t(i)$, for all $i \in [1, n]$. Then for a schedule $Y \in \mathcal{Y}_t$ with $Y \geq X$, we say that the *deficiency of* X *with respect to* Y for interruption t is defined as

$$\mathrm{def}(X, Y, t) = \min_i \frac{S_Y^t(i)}{S_X^t(i)} \tag{4.5}$$

The *deficiency of* X *given* t is then defined as

$$\mathrm{def}(X, t) = \sup_{Y \in \mathcal{Y}_t, Y \geq X} \mathrm{def}(X, Y, t) \tag{4.6}$$

We define the deficiency of X simply as

$$\mathrm{def}(X) = \sup_t \ \mathrm{def}(X, t) \tag{4.7}$$

While (4.5) gives a formal definition of the deficiency, we will obtain an alternative definition that provides us with more flexibility in terms of the analysis of actual schedules. To this end, we will relate this measure to the *makespan* of a schedule.

Definition 101. *Suppose that we are given m identical processors and a set S of n jobs, where each job has a certain size (or length). For a given schedule of S (i.e., for an assignment of S to the processors), the* makespan *of the schedule is the load of the least loaded processor in this schedule, where the load of a processor is defined as the sum of the sizes of the jobs assigned to the said processor. We denote by $OPT(S)$ the optimal makespan of S, among all possible schedules.*

We will first show how to select a canonical representative from the set of schedules of optimal deficiency. The following lemma accomplishes this, and its proof is based on a simple scaling argument.

Lemma 102. *Given schedule X and interruption t, there exists an optimal schedule Y with the following properties:*

1. *Y is such that $S_Y^t(i) = d \cdot S_X^t(i)$, for all $1 \leq i \leq n$ and $d = \mathrm{def}(X, t) \geq 1$.*

2. *d is the largest possible value such that S_Y^t can be feasibly scheduled in M. i.e., the makespan of the corresponding schedule is exactly t.*

In words, the quantity d describes the maximum "stretch factor" so that the largest contracts per instance in Y can be scheduled in m processors at a makespan equal to t. Lemma 102 implies the following corollary which establishes a clean relation between the deficiency and the makespan of the schedule.

Corollary 103. $\text{def}(X,t) = \frac{t}{OPT(S_X^t)}$, *where $OPT(S_X^t)$ is the minimum makespan for scheduling n jobs in m processors, with job i having size (length) equal to $S_X^t(i)$.*

Example 104. *Consider the schedule X depicted in Figure 4.4. For $t = 10$, we have that $S_X^T = \{2,3,4\}$. Moreover, the optimal makespan for scheduling three jobs of sizes $2,3,4$ on two processors is 5. Thus, $OPT(S_X^t) = 5$ and therefore* $\text{def}(X,10) = 2$.

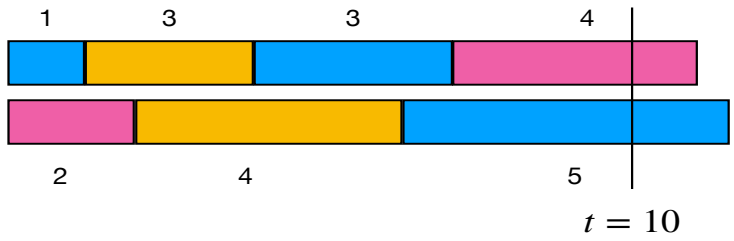


Figure 4.4: An illustration of the concept of deficiency, for a schedule for three instances on two processors. The numbers indicate the contract lengths.

Similarly to the acceleration ratio, one can think of the deficiency of a schedule as a resource-augmentation measure. Specifically, it guarantees that if the contracts of X are scheduled with a processor speedup of $\text{def}(X)$, then for any interruption t, no schedule that knows t can obtain a better feasible solution relatively to the one achieved by X.

To make the above more precise, recall that one can interpret the measures as the processor speedup that is required, in worst-case, in order for the system to be impervious to interruptions. In particular, by increasing the processor speed for the schedule X by a factor $\text{acc}(X)$, we ensure that at any interruption time t, a contract of length t has been completed for each of the n problem instances. A similar interpretation, as a speed augmentation measure, can be made for the deficiency of a schedule. More precisely, by increasing the processor speed for the schedule X by a factor $\text{def}(X)$, we ensure the following property: that for any possible assignment of an overall execution time t among the n problem instances, there will be at least one problem instance for which the schedule X with speedup factor $\text{def}(X)$ will complete a contract to a length that is at least as big as the corresponding length in that assignment.

4.2.3 A schedule of small deficiency for general m

In this section we propose a schedule for which we will show that its deficiency is bounded by a small constant, and is, thus, near-optimal. More precisely, we will identify an efficient schedule that belongs in the class of *exponential* schedules. The following definitions are natural extensions of those pertaining to the single-processor setting.

Definition 105 ([38])**.** *A schedule X with contract lengths $x_0, x_1, \ldots$ is called* round-robin *if it satisfies the following properties:*

1. Problem round-robin: *For every i, the contract x_i is assigned to problem instance $i \bmod n$.*

2. Processor round-robin: *For every i, the contract x_i is assigned to processor $i \bmod m$.*

Definition 106. *A round-robin schedule X with contract lengths $x_0, x_1, \ldots$ is called* exponential *if the contract length x_i is equal to b^i, for some $b > 1$.*

Since an exponential schedule is fully described by its base, the remainder of this section is devoted to finding a value b that yields a schedule of small deficiency. In the remainder of this analysis, we will denote by Exp this schedule. We will also denote by T_k the finish time of the k-th contract of the schedule, in the round-robin order, whereas $L_k = b^k$ denotes the length of the k-th contract in this order.

The following lemma shows that in order to evaluate the deficiency of any schedule (not necessarily exponential) it suffices to consider only interruption times right before a contract terminates. Let T_c^- denote a time infinitesimally smaller than the finish time T_c of contract $c \in X$. Recall also that according to Definition 101, $OPT(S)$ denotes the optimal makespan for a schedule of a set S of jobs in m processors.

Lemma 107. $\mathrm{def}(X) = \sup_{c \in X} \frac{T_C^-}{OPT(S_X^{T_c^-})}$.

Since Exp is a round-robin, exponential schedule, we can substitute the denominator in the statement of Lemma 107 as follows.

$$\mathrm{def}(\textsc{Exp}) = \sup_{k \geq 0} \frac{T_{n+k}^-}{OPT(\{L_k, \ldots L_{n+k-1}\})} \tag{4.8}$$

where $L_i = b^i$ is the length of the i-th contract in the round-robin order. Eq. (4.8) suggests that in order to bound the deficiency of Exp, one needs to obtain a lower bound on the makespan of n jobs, whose lengths are equal to $L_k, \ldots L_{n+k-1}$, respectively. This is accomplished in the next lemma.

Lemma 108. *For $L_i = b^i$ it holds that*

$$OPT(L_k, \ldots L_{n+k-1}) \geq \kappa \cdot b^k \frac{b^{n+m-1} - b^{(n-1) \bmod m}}{b^m - 1},$$

where κ is defined as $\kappa = \max\left\{\frac{1}{2-1/m}, \frac{b^m-1}{b^m}\right\}$.

Proof sketch. We can evaluate the makespan of the well-known greedy algorithm which schedules jobs of sizes $L_k, \ldots L_{n+k-1}$, in m identical processors numbered $0, \ldots m-1$, assuming the jobs are considered in this particular order (i.e., in increasing order of size). To do so we apply Graham's fundamental theorem on the performance of the greedy scheduling policy [80], which has approximation ratio $2 - 1/m$. □

We now proceed to bound the deficiency of the exponential schedule Exp. It is easy to show that for exponential schedules, $T_{n+k} = \frac{b^{k+n+m} - b^{(k+n) \bmod m}}{b^m - 1}$ [38]. Let

$$\lambda = 1/\kappa = \min\left\{2 - \frac{1}{m}, \frac{b^m}{b^m - 1}\right\}.$$

Combining with (4.8) and Lemma 108, and after some calculations, we obtain that

$$\mathrm{def}(\textsc{Exp}) \leq \lambda \cdot \frac{b^{n+m}}{b^{n+m-1} - b^\gamma}, \tag{4.9}$$

where γ is defined as $(n-1) \bmod m$.

We thus seek the value of b that minimizes the RHS of (4.9). It is not straightforward to do this analytically, nevertheless we obtain the following upper bound.

Proposition 109. *There exists an exponential schedule* Exp *for which* $\mathrm{def}(\textsc{Exp}) \leq 3.74$*, if* $n > m$*, and* $\mathrm{def}(\textsc{Exp}) \leq 4$*, if* $n \leq m$*.*

We conclude this section by observing that the schedule we obtained outperforms the strategy of optimal acceleration ratio, in all ranges of n and m. More precisely, the schedule of [38] and [105] has unbounded deficiency for $m \gg n$. For $n > m$, this schedule has constant deficiency, but larger than the deficiency of Exp (namely, a deficiency equal to 4.24 in the worst case).

4.2.4 Lower bounds on the deficiency of a schedule for a single processor

In the previous section we showed that it is possible to obtain schedules of small, and constant deficiency. In this section, we turn our attention to *lower bounds* on the deficiency of *any* schedule, for the case of a single processor ($m = 1$).

Before proceeding with the lower bounds, we begin with some preliminary observations. First, note that for $m = 1$, we obtain from (4.9) that an exponential schedule X with base b has deficiency

$$\mathrm{def}(X) \leq \frac{b^{n+1}}{b^n - 1}, \tag{4.10}$$

which is minimized for a value of b equal to $(n+1)^{\frac{1}{n}}$. Thus:

Corollary 110. *For* $m = 1$ *the deficiency of the best exponential schedule for* n *problem instances is at most* $\frac{(n+1)^{\frac{n+1}{n}}}{n}$*.*

As in the previous section, recall that we will denote by Exp the best exponential schedule whose deficiency is described by Corollary 110.

We will also use a simplified expression for the deficiency of a schedule on a single processor. Recall that we denote by $l_{p,t}$ the length of the longest contract for problem instance p that has completed by time t. We observe that for all schedules X on a single processor, Corollary 103 gives

$$\mathrm{def}(X) = \sup_t \frac{t}{\sum_{i=0}^{n-1} l_{i,t}} = \sup_{t \in T_X} \frac{t}{\sum_{i=0}^{n-1} l_{i,t}}, \tag{4.11}$$

where T_X is defined as the set of all times right before the completion of a contract in X.

A general lower bound

We can give a simple bound that applies to all schedules. The proof follows from some straightforward properties of the deficiency.

Theorem 111. *For any schedule* X *for* n *problem instances and a single processor, we have that*

$$\mathrm{def}(X) \geq \frac{n+1}{n}.$$

Round-robin schedules on a single processor

In this section we focus on the class of all *round-robin* schedules on a single processor. Namely, a schedule X in this class schedules contracts of lengths $x_0, x_1, \ldots$, in this order, such that the contract x_i is assigned to problem instance $i \bmod n$. We can assume, without loss of generality, than for all $i \geq n$, $x_i < x_{i+n}$, otherwise the contract x_{i+n} can be omitted from the schedule without increasing its deficiency.

This is a very natural class of candidate schedules that one should study, as we discussed in Section 1.3, and indeed under the acceleration ratio there exist optimal schedules that belong

in this class [105]. It is thus important to know their efficiency and limitations with respect to deficiency. We address this issue, and show that the exponential schedule Exp (as defined earlier in this schedule) is optimal for this class.

To prove this result, we apply a theorem by Gal (Theorem 55) that allows us to relate the performance of schedules that have certain structure (such as round-robin schedules) to the performance of exponential schedules, i.e., schedules whose contract lengths increase geometrically. At an intuitive level, Theorem 55 allows us to lower-bound the supremum of an infinite set of functionals by the supremum of simple functionals of a geometric sequence; the latter is typically much easier to compute than the former. In our context, the objective is then to express the deficiency as the supremum of a sequence of functionals. We will give the full proof as an illustration of Gal's theorem.

Theorem 112. *For every round robin schedule X on a single processor, we have that* $\mathrm{def(X)} \geq \frac{(n+1)^{\frac{n+1}{n}}}{n}$.

Proof. Let us denote by $(x_i)_{i\geq 0}$ the sequence of contract lengths in X, in the order they are scheduled. Consider a time t right before the completion of contract x_{k+n} of the round-robin schedule, with $k \geq 0$. The set S_X^t of the n largest contracts per problem is then the set $\{x_k, x_{k+1}, \ldots x_{k+n-1}\}$. Thus, from (4.11) we have

$$\mathrm{def}(X) \geq \frac{t}{\sum_{j=0}^{n-1} x_{k+j}} = \frac{\sum_{j=0}^{k+n} x_j}{\sum_{j=k}^{k+n-1} x_j}.$$

Define now the functional $F_k(X) = \frac{\sum_{j=0}^{k+n} x_j}{\sum_{j=k}^{k+n-1} x_j}$. It is easy to verify that $F_k(X)$ satisfies the conditions of Theorem 55. Therefore, there is $a > 0$ such that

$$\mathrm{def}(X) = \sup_{0\leq k<\infty} F_k(X) \geq \sup_{0\leq k<\infty} \frac{\sum_{j=0}^{k+n} a^j}{\sum_{j=k}^{k+n-1} a^j}. \tag{4.12}$$

Note that if $a = 1$, we have that $\frac{\sum_{j=0}^{k+n} a^j}{\sum_{j=k}^{k+n-1} a^j} = \frac{k+n+1}{n}$, which tends to infinity as $k \to \infty$, and thus $\mathrm{def}(X) = \infty$ in this case. Thus we can assume that $a \neq 1$, thus (4.12) implies

$$\mathrm{def}(X) \geq \sup_{0\leq k<\infty} \frac{a^{k+n+1} - 1}{a^{k+n} - a^k} = \sup_{0\leq k<\infty} \frac{a^{n+1} - \frac{1}{a^k}}{a^n - 1}. \tag{4.13}$$

Note that if $a < 1$, then the rhs of (4.13) is ∞. Thus, we can assume that $a > 1$. In this case,

$$\sup_{0\leq k<\infty} \frac{a^{n+1} - \frac{1}{a^k}}{a^n - 1} = \frac{a^{n+1}}{a^n - 1}, \tag{4.14}$$

and the latter attains its minimum at $a = (n+1)^{\frac{1}{n}}$, which suffices to prove the result. □

From Corollary 110, it follows that Exp is optimal for the class of round-robin schedules.

Schedule deficiency for $n \in \{1, 2\}$, $m = 1$

In Theorem 111 we showed a general lower bound of $(n+1)/n$ on the deficiency of any schedule (for $m = 1$). This bound is, however, not tight, and indeed we argue, in this section, that for $n \in \{1, 2\}$, better lower bounds can be obtained. First, it is straightforward to see that for $n = 1$, the deficiency is equivalent to the acceleration ratio, and the exponential schedule Exp is optimal. More importantly, we describe how to obtain a lower bound on the deficiency of a

schedule for $n = 2$ that improves upon the lower bound of Theorem 111. Namely, we obtain a lower bound of 2.115 whereas Theorem 111 gives a lower bound of only 1.5. Note that the corresponding upper bound due to EXP is 2.598.

Why should one be interested in improving the lower bound for the case $n = 2$? This is the simplest case which still captures the complexity of the problem. It is thus likely that an improved lower bound will provide insights about improving upon the exponential schedule. The proof shows that there is indeed a certain degree of difficulty involved in obtaining an optimal schedule, even for this simple case.

Theorem 113. *For $n = 2$ and $m = 1$, any schedule has deficiency at least 2.115.*

Proof sketch. We first argue that there exists an optimal schedule with the property that at each point in time it schedules a contract for the problem that has been worked on the least so far, and is such that at most two contracts per problem instance are scheduled consecutively. This allows us to lower-bound the deficiency of the optimal schedule as a supremum of functionals, and we can still apply Gal's theorem. □

4.3 Connections between contract scheduling and ray searching

Most of the previous work in contract scheduling has focused on a basic setting with several simplifying assumptions. Namely, it is assumed that each execution of the contract algorithm is error-free, that the scheduler is not prone to errors, that the schedule lengths are chosen deterministically (thus the scheduler has no access to random bits), and that the executions of contracts incur no cost due to context switching. In this work we study contract scheduling in more realistic settings that address these considerations. We do so by expanding the study of connections between the search and scheduling problems that was initiated in [38], thus addressing several extensions and generalizations of these two classes of problems. We assume the setting of n-instance contract scheduling and m-ray searching, with a single searcher/processor. More precisely, we study the following classes of problems:

Uncertain target detection and Monte Carlo contract algorithms: We study the stochastic setting in which the searcher detects the target with probability p during each visit, which models searching with *overlook*. In addition, we study the setting in which each contract algorithm is a randomized Monte Carlo algorithm with probability of success equal to p. For the former, we show that a cyclic strategy achieves an asymptotically optimal competitive ration of $\Theta(m/p)$. For the latter, we give a corresponding optimal bound on the acceleration ratio equal to $\Theta(n/p)$.

Redundancy and fault tolerance We seek search strategies under the constraint that at least r visits over the target are required in order to locate it. In a similar vain, we seek scheduling strategies under the assumption that at least r executions of a contract algorithm are required so as to benefit from its output. This models search and scheduling with uncertainty, when the probability of success is unknown to the algorithm designer. Concerning the search problem, we show that the best-possible cyclic strategy has competitive ratio $O(rm)$, which is asymptotically optimal. We also give a *non-cyclic* strategy that improves upon the cyclic ones. Concerning the scheduling problem, we consider two different models that capture redundancy in this setting, and we provide optimal and near-optimal schedules (of competitive ratio $O(rn)$), respectively. Our results reflect the search paradigm, in that there exist non-cyclic schedules that improve upon the best cyclic ones.

Randomized scheduling strategies We show how access to random bits can improve the expected performance of a scheduling strategy, in comparison to a deterministic schedule. Our approach here is motivated by the randomized search strategy of Kao *et al.* [92] for searching on m rays. A computational evaluation shows that the expected acceleration ratio is within a factor of approximately 0.6 of the worst-case deterministic acceleration ratio. For large n, we show analytically that this reduction factor converges to $(n+1)\frac{\mathrm{e}}{e-1}$, a value that is very close to the computational results.

Trade-offs between performance and the number of searches and contracts We quantify the trade-offs between the performance ratios and the number of turns by the searcher or the number of algorithm executions in the schedule. This is useful in situations in which the turn cost or the setup cost for the two settings, respectively, are considerable and must be taken into account. For instance, a robot incurring a significant cost each time it changes direction, or a contract incurring delay due to context switching.

Our broader objective is to bring attention to the fact that both classes of problems can be interpreted as *resource allocation* problems. More specifically, in searching, we seek an allocation of search time over the different rays, whereas in scheduling, we seek an allocation of execution time to different problems. Thus the two settings are comparable, although not identical: it is not at all obvious how to obtain a "black box" reduction from one to the other that applies to the different variants we study here. Nevertheless, our study of contract scheduling can certainly benefit from a variety of techniques and approaches.

We will focus, in this section, in two specific settings: the stochastic setting (Section 4.3.1), and the randomized setting (Section 4.3.2). In Section 4.3.3 we will give an overview of the results and techniques concerning further connections.

4.3.1 Search with probabilistic detection and scheduling of randomized contract algorithms

We begin with a study of the effect of uncertainty in searching and contract scheduling. In particular, we consider the setting in which the detection of a target is *stochastic*, in that the target is revealed with probability p every time the searcher passes over it, where p is known to the searcher. In a similar vein, we address the problem of scheduling randomized contract algorithms; namely, each execution of the (Monte Carlo) randomized algorithm succeeds with probability p.

Searching with *overlook* has been studied [111] on the discrete star, i.e., when each ray has a known length and the target can only hide at the end of a ray, and a strategy of optimal cost can be obtained in this setting. In contrast, when the search domain is unbounded, and for the competitive ratio, the problem has only been studied in [6] in the context of linear search (i.e., when $m = 2$), and the exact competitiveness of the problem is not known even in this much simpler case. No results are known for general m.

In our setting, the search cost is defined as the expected time of the first successful target detection. Moreover, for contract scheduling, and for every problem instance p and interruption t, we define $\mathbb{E}[\ell_{p,t}]$ as the expected longest contract completed for instance p by time t. The competitive and the acceleration ratios are then defined naturally as extensions of the corresponding definitions. We begin with a lower bound on the performance measures.

Lemma 114. *Every search strategy with probabilistic detection has competitive ratio at least* $\frac{m}{2p}$*, and every schedule of randomized contract algorithms has acceleration ratio at least* $\frac{n}{p}$*.*

Proof sketch. Consider first the search variant. Let S denote the set of all points at distance at most d from the origin, for a fixed d. Given a search strategy and a point $x \in S$, let t_x^k denote the time in which the searcher reaches x for the k-th time. We can show that for every $k \geq 1$,

there exists $x \in S$ such that the search cost at the time of the k-th visit of x is at least $kmd/2$, which further implies that there exists a point in S with the desired property.

Consider now the scheduling setting. For a given interruption time t and a given problem instance i, let $l_1^i, l_2^i, \dots l_{n_i}^i$ denote the lengths of the contracts for problem instance i that have completed by time t, in non-increasing order. Let the random variable $\ell_{i,t}$ denote the expected length of the longest contract completed for instance i by time t. Then

$$\mathbb{E}[\ell_{i,t}] = \sum_{j=1}^{n_i} p(1-p)^{j-1} l_j^i \leq p \sum_{j=1}^{n_i} l_j^i.$$

Since $\sum_{i=0}^{n-1} \sum_{j=1}^{n_i} l_j^i = t$, there exists an instance i for which $\mathbb{E}[\ell_{i,t}] \leq p\frac{t}{n}$. The claim follows from the definition of acceleration ratio. □

We now move to the upper bounds, and we begin with the search problem. We will first show how to analyze cyclic, exponential strategies. To this end, let $X = (x_i)_{i \geq 0}$ denote the lengths of this exponential strategy, where $x_i = b^i$, for some b that will be suitably chosen. Let d denote the distance of the target from the origin, then there exists index l such that $x_l < d \leq x_{l+m}$. We denote by P_k the probability that the target is found during the k-th visit of the searcher, when all previous $k-1$ attempts were unsuccessful, hence $P_k = (1-p)^{k-1}p$. Moreover, we have $\sum_{k=j}^{\infty} P_k = (1-p)^{j-1}$.

We can express the cost of the searcher for locating the target by considering the contribution to the expected cost when the searcher moves away from the origin and the contribution when the searcher moves towards the origin; let us denote by C_1, C_2 these two expected costs. Concerning C_1, we observe that the total distance traversed by the searcher at the moment of the k-th visit of the target in the "away from O" direction is equal to $2\sum_{j=0}^{l+km-1} x_j + d$, whereas for C_2, the corresponding distance at the k-th visit of the target in the "towards O" direction is $2\sum_{j=0}^{l+km} x_j - d$. Therefore, we have

$$C_1 = \sum_{k=1}^{\infty} P_{2k-1} \sum_{j=0}^{l+km-1} (2x_j + d) \text{ and } C_2 = \sum_{k=1}^{\infty} P_{2k} \sum_{j=0}^{l+km} (2x_j - d), \tag{4.15}$$

and the overall expected cost C of the searcher is the sum of C_1 and C_2.

Since $x_i = b^i$, by substitution we obtain

$$\begin{aligned} C_1 &= 2\sum_{k=1}^{\infty} P_{2k-1} \frac{b^{l+km}-1}{b-1} + d\sum_{k=1}^{\infty} P_{2k-1} \\ &\leq 2p\sum_{k=1}^{\infty} (1-p)^{2k-2} \cdot \frac{b^{l+km}}{b-1} + d \\ &= 2p\frac{b^{l+m}}{b-1} \sum_{k=1}^{\infty} ((1-p)^2 b^m)^{k-1} + d. \end{aligned} \tag{4.16}$$

For C_2 we can argue that its contribution to the competitive ratio is at most a constant times the contribution of C_1, hence we will not need to provide an exact expression. It remains to choose the right base b. We can show that there is such a value for which the cost C_1 is in $O(m/p)$. More specifically, we can obtain the following theorem.

Theorem 115. *There exists an exponential strategy for m-ray searching with probabilistic detection that has competitive ratio at most $\frac{7}{3} + \frac{56}{3}\frac{m}{p}$.*

Next, we consider the problem of scheduling contract algorithms in the stochastic setting. It turns out that in this setting it is much easier to obtain asymptotically optimal algorithms, as shown in the following theorem.

Theorem 116. *There exists an exponential schedule for randomized Monte Carlo contract algorithms that has acceleration ratio at most $\mathrm{e}\frac{n}{p} + \frac{\mathrm{e}}{p}$, where n is the number of problem instances.*

4.3.2 Randomized scheduling of contract algorithms

Here we study the power of randomization for scheduling executions of (deterministic) contract algorithms. The main idea is to allow contract lengths to be chosen according to a randomized method, and evaluate the *expected* acceleration ratio of the corresponding schedule. Our approach is motivated by the randomized strategy of [93] for searching on m rays. We emphasize, however, that our analysis differs in several key points, and most notably in the definition of appropriate random events.

We will analyze the following randomized schedule RANDOM$_b$ which is parameterized by a based b (we will show later how to choose an appropriate b.) RANDOM$_b$ chooses a random permutation $\pi : \{0, \dots n-1\} \to \{0, \dots n-1\}$ of the n problem instances, as well as a random ε uniformly distributed in $[0, 1)$. In every iteration $i \geq 0$, the strategy executes a contract for problem $\pi(i) \mod n$ with corresponding length $b^{1+\varepsilon}$, with $b > 1$. We denote by $r(n, b)$ the expected acceleration ratio of RANDOM$_b$.

Theorem 117. *Schedule* RANDOM$_b$ *has expected acceleration ratio* $r(n, b) = n\frac{b^{n+1} \ln b}{(b^n-1)(b-1)}$.

In order to evaluate the best randomized exponential schedule, i.e., the base b for which RANDOM$_b$ has optimal acceleration ratio, we must find the minimizer of the function $r(n, b)$, as expressed by Theorem 117. Using standard calculus, one can show that $r(n, b)$ has a unique minimum, for given n. However, unlike the deterministic case, it is not obvious that a closed form for the function $r^*(n) = \min_{b>1} r(n, b)$ can be found. Thus, we must resort to numerical methods.

Figure 4.5 illustrates the performance of the randomized schedule $r^*(n)$ versus the optimal deterministic schedule, denoted by $d^*(n)$. We observe that $r^*(n) \leq 0.6d^*(n)$, for $n = 1, \dots 80$. In fact, we can show analytically that for $n \to \infty$, $r^*(n)$ converges to a value that does not exceed $\frac{e}{e-1}(n+1)$ (recall that $d^*(n)$ converges to $e(n+1)$). More precisely, choosing $b = \frac{n+1}{n}$ we obtain

$$r^*(n) \leq n(n+1)\frac{(1+1/n)^n \ln(1+1/n)}{((1+1/n)^n - 1)(1+1/n)},$$

which converges to $(n+1)\frac{e}{e-1}$, as $n \to \infty$, a value extremely close to the computational results.

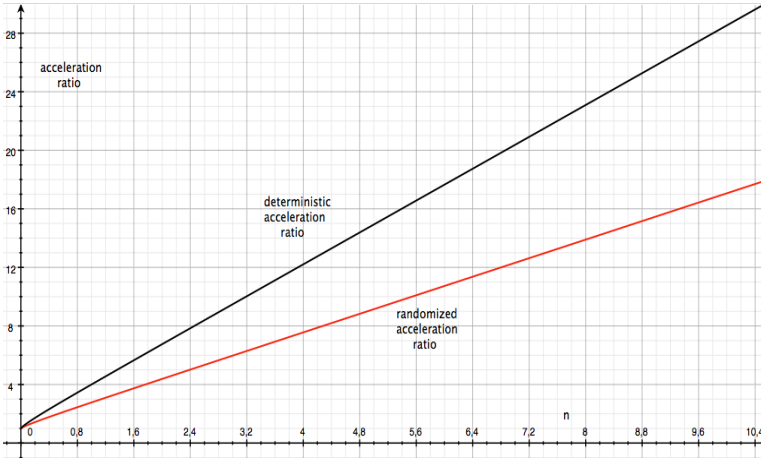


Figure 4.5: Plots of the randomized ($r^*(n)$) and the deterministic ($d^*(n)$) acceleration ratios, as functions of n.

4.3.3 Further connections

Fault tolerance and redundancy in search and scheduling

In Section 4.3.1 we studied searching and scheduling problems in a stochastic setting. But what if the success probability p is not known in advance? In the absence of such information, one could

opt for imposing a sufficiently good lower bound r on the number of times the searcher has to visit the target and, likewise, a lower bound r on the number of times a contract algorithm must be executed before its response can be trusted. In other words, this setting can model fault tolerance and redundancy in the search and scheduling domains. We thus call r the *redundancy parameter*. The search variant has been studied in [6] only in the context of linear search ($m = 2$); as in the case of probabilistic detection, even when $m = 2$ the exact optimal competitive strategies are not known.

We start with the search problem. The proof of the following lemma follows directly by the proof of Lemma 114.

Lemma 118. *Every search strategy on m rays with redundancy parameter $r \in \mathbb{N}^+$ has competitive ratio at least $\frac{rm}{2}$.*

Concerning upper bounds, we can first evaluate the competitiveness of the best exponential strategy.

Theorem 119. *The best exponential strategy for searching with redundancy parameter r has competitive ratio at most $2(\lceil \frac{r}{2} \rceil m - 1) \left(\frac{\lceil \frac{r}{2} \rceil m}{\lceil \frac{r}{2} \rceil m - 1}\right)^{\lceil \frac{r}{2} \rceil m} + 1 \leq 2\mathrm{e}(\lceil \frac{r}{2} \rceil (m-1)) + 1$.*

Interestingly, we can show that there exist non-cyclic strategies, which, for $r > 2$, improve upon the (best) exponential strategy of Theorem 119. Assume first that r is even, and consider the following strategy: In iteration i, the searcher visits ray $i \bmod m$ first up to the point at distance x_{i-m} from the origin, then performs r traversals of the interval $[x_{i-m}, x_i]$ (thus visiting r times each point of the said interval), then finally completes the iteration by returning to the origin (for notational convenience we define $x_j = 0$ for all $j < 0$). If r is odd, then the same strategy applies, but we charge an additional traversal of the interval $[x_{i-m}, x_i]$, in both directions of the search. We call this strategy NC-SEARCH (non-cyclic search).

Theorem 120. *Strategy* NC-SEARCH *has competitive ratio at most $r(m-1)\left(\frac{m}{m-1}\right)^m + 2 - r$, if r is even and at most $(r+1)(m-1)\left(\frac{m}{m-1}\right)^m + 2 - r$, if r is odd.*

We can show, by comparing the results of Theorems 119 and 120, that the non-cyclic strategy is better than the best exponential strategy for $r > 2$. For simplicity, let us assume that r is even.

Theorem 121. *For even $r > 2$,* NC-SEARCH *has better competitive ratio than the best exponential strategy.*

We now move to contract scheduling with redundancy parameter r. Here, there are two natural settings one may consider. In the first setting, we require that if there is an interruption at time t, the system may output the solution returned by a contract, only if the contract has been executed r times; we call this setting *r-repetitive*. This setting reduces to standard contract scheduling with n problem instances, as shown formally in the following theorem. Thus, there exists an optimal cyclic schedule.

Theorem 122. *For contract scheduling with n problem instances in the r-repetitive setting, the optimal acceleration ratio is $r(n+1)\left(\frac{n+1}{n}\right)^n$.*

A second setting that one may consider stipulates that the schedule returns, upon interruption t and for queried problem p, the r-th smallest contract for problem p that has completed its execution by time t; we call the setting the *r-smallest* setting. In this setting, we can still apply a non-cyclic schedule, motivated by Theorem 121. We can show, as in the search counterpart, that this schedule is better than the best exponential schedule, albeit only slightly so.

Theorem 123. *The best exponential schedule for contract scheduling with n problem instances in the r-smallest setting has acceleration ratio at most*

$$(rn+1)\left(1+\frac{1}{rn}\right)^{rn} \leq \mathrm{e}rn + e.$$

Furthermore, there exists a non-cyclic strategy which improves upon the best exponential strategy for all n, r.

Even though we cannot show that the non-cyclic strategy of Theorem 123 is optimal, we note that in the r-smallest setting there is a simple lower bound of rn on the acceleration ratio of any schedule. This follows easily from the fact that for any schedule and any interruption t, there is at least one problem instance whose r-th smallest contract is of length at most $t/(rn)$.

The strategies described above establish connections beyond those that result from the use of cyclic strategies. More precisely, we have shown that non-cyclic ray-searching algorithms have counterparts in the domain of contract-scheduling; furthermore, the non-cyclic strategies improve upon the best cyclic ones. We have thus addressed an open question from [38], who asked whether there exist connections between the two problems that transcend cyclic strategies.

Trade-offs between performance and executions of searches and algorithms

Most previous work on ray searching assumes that the searcher can switch directions at no cost. In practice, however, turning is often a costly operation and thus should not be ignored. In a similar vein, we usually assume that there is no setup cost upon execution of a contract algorithm, however some initialization cost may be incurred in practice. One could address this requirement by incorporating the turn/setup cost in the performance evaluation as we did in Section 3.1. Here we follow a different approach by studying the trade-off between performance, as expressed by the competitive and acceleration ratios, and the number of searches and/or executions of contract algorithms.

We show the following result concerning the tradeoffs in the contract scheduling setting.

Theorem 124. *For contract scheduling with n problem instances, the exponential schedule with base b has acceleration ratio $\frac{b^{n+1}}{b-1}$, and schedules at most $\log_b(t(b-1)+1)+1$ contracts by time t. Moreover, any schedule with acceleration ratio at most $\frac{b^{n+1}}{b-1}$ with $b>1$ must schedule at least $\log_b(t(b-1)+1)-n$ contracts by time t, for all t.*

The following theorem can be considered as the counterpart of Theorem 124 for ray searching.

Theorem 125. *For m-ray searching, the exponential strategy with base b has competitive ratio $1+2\frac{b^m-1}{b-1}$, and for any distance d traversed by the searcher it makes at most $\log_b(d(b-1)+1)+1$ turns. Moreover, any strategy with competitive ratio at most $1+2\frac{b^m-1}{b-1}$, for any $b>1$ incurs at least $\log_b(d(b-1)+1)+1-m$ turns.*

Even though the theorems follow the same approach, it is worth noting that they differ in certain technical details. A general observation is that the search variants tend to be somewhat more complex than the scheduling ones.

4.4 Discussion

Concerning contract scheduling with end guarantees, the approach we follow in solving the problem builds on some tools that we developed in the context of bijective analysis of linear search. In very informal terms, scheduling with end guarantees generalizes some of the ideas of bijective analysis in linear search, since we deal with many contract algorithms, whereas in linear search we "deal" only with two semi-infinite lines; but can also be seen as a restrictive case, in

that we are not interested in establishing the bijective ratio, but rather in studying directly the optimal, offline solution behind that measure.

Concerning the deficiency of a schedule, we note that by applying Gal's theorem (Theorem 55) in [20] we gave a much simpler proof of the main result in [38] concerning the acceleration ratio of optimal cyclic schedules in the multiprocessor setting. This is a further illustration of the significance of this theorem. It is also worth pointing out that our result established in Theorem 113 applies Gal's theorem in a manner that gives a lower bound, albeit a non-tight one. To our knowledge, this theorem was applied, in previous work, so as to establish optimality with respect to a given exponential strategy. Here we showed that it can be equally useful in situations in which an exponential strategy may not necessarily be optimal.

Last, in what concerns the connections between the search and the scheduling problems, it is worth mentioning that the search problem with turn cost (discussed in Chapter 3) has a natural counterpart in the scheduling world: more precisely, it is related to contract algorithms whose execution incurs a fixed, additive *setup cost*. In [9] we showed how the solution to the multi-search problem with turn cost effectively leads to a solution to this scheduling problem.

Chapter 5

Conclusion

In this Habilitation thesis we studied problems related to computation under uncertainty, and in particular problems related to online computation, search games, and interruptible algorithms. We introduced new performance measures, extended some of the known computational models, and applied some new analysis techniques. We conclude with some directions for future work concerning the topics we covered.

5.1 Online computation

5.1.1 Bijective analysis of online algorithms

The biggest challenge in this area is to develop new tools for the analysis of online algorithms. Our current arsenal of techniques is, still, rather limited. From the point of view of upper bounds, the decoupling technique we used in Section 2.1.4 may work for the problems at hand, but for more general problems and algorithms, it is clear that new methods will be required. From the point of view of lower bounds, we have used, so far, techniques based on counting (or probabilistic arguments) and the Max/Max ratio. But it looks like more nuanced techniques will be necessary in order to prove better lower bounds.

These difficulties could perhaps lead to the sobering realization that bijective analysis may just be too difficult as a technique, in what concerns at least the theoretical analysis. But we believe that the technique can be useful not only in terms of analysis, but also in terms of algorithm engineering. We have evidence that for other online problems, algorithms that we conjecture are bijectively efiicient, do indeed perform well in practice (and definitely better than the competitively optimal ones), even though it is not obvious how to analyze them. The true testament of the significance of bijective analysis would therefore be its capacity to inspire new algorithms that perform well in practice, and this is a direction that we are currently pursuing.

5.1.2 Towards more realistic advice models

The work we presented in Chapter 2 is a first effort to bridge the large gap between the ideal framework of advice complexity, and the more realistic world of machine learning. There are many open questions to pursue here.

First and foremost, in our setting we assumed that the advice can either be perfect, or adversarially generated. For this reason, some of the performance tradeoffs are only of theoretical significance. A more practical model would allow the possibility of *noisy advice*. The challenge here is to show impossibility results (i.e., limitations) on the performance of online algorithms. Another avenue for addressing this issue is to assume that the advice is correct according to some probability distribution, instead of being adversarially generated.

A second direction stems from observing that the machine learning framework so far has not placed any restrictions on the size of the advice. However, there are situations in which some

restriction is necessary in order to obtain any practical algorithms. We would like thus to study online algorithms in a setting in which there is (potentially erroneous) advice of small size.

5.2 Search games

The bijective analysis of linear search that we discussed in Section 3.3 pertains naturally to deterministic (pure) strategies. An obvious question is whether one could apply this approach to randomized (mixed) strategies. It would be interesting to explore an analysis based on stochastic dominance.

One serious difficulty in analyzing strategies for ray search problems is that, if it so happens that an optimal strategy cannot be found in the space of exponential strategies, then Gal's theorem (Theorem 55) cannot yield a tight lower bound. This situation arises for example in the context of stochastic search that we covered in Section 4.3.1. We thus need new tools to address this issue. One approach would be to describe the optimal strategies via quadratic programs and explore duality in an attempt to establish optimality.

Search problems can be studied in a natural setting in which there is some advice (or hint) about the hider's position. Thus, the ideas we developed in the context of online computation with untrusted advice can definitely be of interest. Note also that the setting needs to incorporate some concept of error with respect to the given advice. There are many specific settings one can study. For example, the advice can describe the direction towards which the searcher should move, or the whereabouts of the target.

5.3 Interruptible algorithms and beyond

While this is a much narrower field than online computation and search games, there are still some interesting problems to resolve. First, it would be interesting to obtain a strategy of optimal deficiency, even for the case of a single processor and n problems. Here we face the same complication as in the search games: it is not clear whether cyclic, exponential strategies are optimal, and we do not have good lower-bound tools for such cases.

Second, in the real world there is usually some partial information concerning the interruption. Consider the example of a medical diagnostic system. Here, the expert may know that the system will need to be queried around a *specific time*, with some degree of confidence (i.e., prior to a scheduled surgery). Again, the advice model can find a pertinent application in this context, and this is a direction that we are currently pursuing.

Whereas the acceleration ratio is a measure that is well-understood and well-accepted by the AI community, it is clear that it is only a resource allocation measure, from which one cannot infer conclusions concerning, say, the approximation power of the resulting system. There are some alternative, theoretical models that take this issue into account, see e.g., the class of *progressive* algorithms [4], which can establish some theoretical guarantees, but they have not yet been widely applied. There is also the class of *algorithm portfolios* [79] which has significant practical impact in AI. It would be interesting to explore connections between all the above models.

5.4 Epilogue

Competitive analysis, and other worst-case measures such as the acceleration ratio, emerged from the need to be able to say something about the algorithm's performance when we know little or nothing about the instance. Over the years, the theoretical evaluation of online algorithms evolved so as to capture not only the worst-case, but also the "realistic" case (whenever there is some tangible interpretation of it). If there is one lesson that the current year has to offer is that worst-case guarantees will not go out of fashion; instead, the age-old wisdom "hope for the

best, plan for the worst" will remain the sensible plan of action. It is very likely that similar considerations will become even more prevalent in the design and evaluation of algorithms and systems over the years to come. This is also the take-home message of this Habilitation thesis.

Bibliography

[1] Anna Adamaszek, Marc P. Renault, Adi Rosén, and Rob van Stee. Reordering buffer management with advice. *Journal of Scheduling*, 20(5):423–442, 2017.

[2] Susanne Albers. Improved randomized on-line algorithms for the list update problem. *SIAM J. Comput.*, 27:682–693, 1998.

[3] Susanne Albers, Lene M. Favrholdt, and Oliver Giel. On paging with locality of reference. *Journal of Computer and System Sciences*, 70(2):145–175, 2005.

[4] Sander P. A. Alewijnse, Timur M. Bagautdinov, Mark de Berg, Quirijn W. Bouts, Alex P. ten Brink, Kevin Buchin, and Michel A. Westenberg. Progressive geometric algorithms. *Journal of Ccomputational Geometry*, 6(2):72–92, 2015.

[5] Steve Alpern, Robbert Fokkink, L Gasieniec, Roy Lindelauf, and VS Subrahmanian. *Search theory: A Game Theoretic Perspective*. Springer, 2013.

[6] Steve Alpern and Shmuel Gal. *The theory of search games and rendezvous*. Kluwer Academic Publishers, 2003.

[7] Steve Alpern and Thomas Lidbetter. Mining coal or finding terrorists: The expanding search paradigm. *Operations Research*, 61(2):265–279, 2013.

[8] Aris Anagnostopoulos, Clément Dombry, Nadine Guillotin-Plantard, Ioannis Kontoyiannis, and Eli Upfal. Stochastic analysis of the k-server problem on the circle. In *Proceedings of the 21st International Meeting on Probabilistic, Combinatorial and Asymptotic Methods for the Analysis of Algorithms (AofA)*, pages 21–34, 2010.

[9] Spyros 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.

[10] Spyros Angelopoulos, Diogo Arsénio, and Christoph Dürr. Infinite linear programming and online searching with turn cost. *Theoretical Computer Science*, 670:11–22, 2017.

[11] Spyros Angelopoulos, Diogo Arsénio, Christoph Dürr, and Alejandro López-Ortiz. Multiprocessor search and scheduling problems with setup cost. *Theory of Computing Systems*, pages 1–34, 2016.

[12] Spyros Angelopoulos, Reza Dorrigiv, and Alejandro López-Ortiz. List update with locality of reference. In *Proceedings of the 8th Latin American Theoretical Informatics Symposium (LATIN)*, pages 399–410, 2008.

[13] Spyros Angelopoulos, Reza Dorrigiv, and Alejandro López-Ortiz. On the separation and equivalence of paging strategies and other online algorithms. *Algorithmica*, 81(3):1152–1179, 2019.

[14] Spyros Angelopoulos, Christoph Dürr, and Shendan Jin. Best-of-two-worlds analysis of online search. In *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019*, volume 126 of *LIPIcs*, pages 7:1–7:17, 2019.

[15] Spyros Angelopoulos, Christoph Dürr, Shendan Jin, Shahin Kamali, and Marc P. Renault. Online computation with untrusted advice. In *11th Innovations in Theoretical Computer Science Conference, ITCS 2020*, pages 52:1–52:15, 2020.

[16] Spyros Angelopoulos, Christoph Dürr, Shahin Kamali, Marc P. Renault, and Adi Rosén. Online bin packing with advice of small size. *Theory Comput. Syst.*, 62(8):2006–2034, 2018.

[17] Spyros Angelopoulos, Christoph Dürr, and Thomas Lidbetter. The expanding search ratio of a graph. *Discrete Applied Mathematics*, 260:51–65, 2019.

[18] Spyros Angelopoulos and Shendan Jin. Earliest completion scheduling of contract algorithms with end guarantees. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 5493–5499, 2019.

[19] Spyros Angelopoulos and Thomas Lidbetter. Competitive search in a network. *European Journal of Operational Research*, 286(2):781–790, 2020.

[20] Spyros Angelopoulos and Alejandro López-Ortiz. Interruptible algorithms for multi-problem solving. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, pages 380–386, 2009.

[21] Spyros Angelopoulos, Alejandro López-Ortiz, and Angele Hamel. Optimal scheduling of contract algorithms with soft deadlines. In *Proceedings of the 23rd National Conference on Artificial Intelligence (AAAI)*, pages 868–873, 2008.

[22] Spyros Angelopoulos, Alejandro López-Ortiz, and Konstantinos Panagiotou. Multi-target ray searching problems. *Theoretical Computer Science*, 540:2–12, 2014.

[23] Spyros Angelopoulos and Konstantinos Panagiotou. Optimal strategies for weighted ray search. *CoRR*, abs/1704.03777, 2017.

[24] Spyros Angelopoulos, Marc P. Renault, and Pascal Schweitzer. Stochastic dominance and the bijective ratio of online algorithms. *Algorithmica*, 82(5):1101–1135, 2020.

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

[26] Sanjeev Arora and George Karakostas. Approximation schemes for minimum latency problems. *SIAM J. Comput.*, 32(5):1317–1337, 2003.

[27] Noa Avigdor-Elgrabli and Yuval Rabani. An improved competitive algorithm for reordering buffer management. *ACM Transactions on Algorithms*, 11(4):35:1–35:15, 2015.

[28] Ricardo A. Baeza-Yates, Joseph C. Culberson, and Gregory G.E. Rawlins. Searching in the plane. *Information and Computation*, 106:234–244, 1993.

[29] János Balogh, József Békési, György Dósa, Leah Epstein, and Asaf Levin. A new and improved algorithm for online bin packing. In *26th Annual European Symposium on Algorithms (ESA 2018)*, pages 5:1–5:14, 2018.

[30] János Balogh, József Békési, György Dósa, Leah Epstein, and Asaf Levin. A new lower bound for classic online bin packing. In *Approximation and Online Algorithms - 17th International Workshop, WAOA 2019*, volume 11926 of *Lecture Notes in Computer Science*, pages 18–28. Springer, 2019.

[31] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. A primal-dual randomized algorithm for weighted paging. *Journal of the ACM*, 59(4):19:1–19:24, 2012.

[32] Yair Bartal and Elias Koutsoupias. On the competitive ratio of the work function algorithm for the k-server problem. *Theoretical Computer Science*, 324(2-3):337–345, 2004.

[33] Alfonzo Baumgartner, Robert Manger, and Zeljko Hocenski. Work function algorithm with a moving window for solving the on-line k-server problem. *Journal of Computing and Information Technology*, 15(4):325–330, 2007.

[34] Anatole Beck. On the linear search problem. *Naval Research Logistics*, 2:221–228, 1964.

[35] Anatole Beck and D.J. Newman. Yet more on the linear search problem. *Israel Journal of Mathematics*, 8:419–429, 1970.

[36] Shai Ben-David and Allan Borodin. A new measure for the study of on-line algorithms. *Algorithmica*, 11:73–91, 1994.

[37] Shai Ben-David, Allan Borodin, Richard Karp, Gabor Tardos, and Avi Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, 1994.

[38] Daniel S. Bernstein, Lev Finkelstein, and Shlomo 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.

[39] Daniel S. Bernstein, Theodore J. Perkins, Shlomo Zilberstein, and Lev Finkelstein. Scheduling contract algorithms on multiple processors. In *Proceedings of the Eighteenth AAAI Conference on Artificial Intelligence (AAAI)*, pages 702–706, 2002.

[40] Avrim Blum, Prasad Chalasani, Don Coppersmith, Bill Pulleyblank, Prabhakar Raghavan, and Madhu Sudan. The minimum latency problem. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing (STOC)*, pages 163–171, 1994.

[41] 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*, volume 5878, pages 331–340, 2009.

[42] Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, Richard Královič, and Tobias Mömke. Online algorithms with advice: the tape model. *Information and Computation*, 254:59–83, 2017.

[43] Mark Boddy and Thomas L. Dean. Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67(2):245–285, 1994.

[44] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[45] Allan Borodin, Jon M. Kleinberg, Prabhakar Raghavan, Madhu Sudan, and David P. Williamson. Adversarial queuing theory. *J. ACM*, 48(1):13–38, 2001.

[46] Allan Borodin, Morten N. Nielsen, and Charles Rackoff. (Incremental) priority algorithms. *Algorithmica*, 37(4):295–326, 2003.

[47] Prosenjit Bose, Jean-Lou De Carufel, and Stephane Durocher. Searching on a line: A complete characterization of the optimal solution. *Theoretical Computer Science*, 569:24–42, 2015.

[48] Joan Boyar, Martin R. Ehmsen, and Kim S. Larsen. A theoretical comparison of LRU and LRU-K. *Acta Informatica*, 47(7-8):359–374, 2010.

[49] Joan Boyar, Lene M. Favrholdt, Christian Kudahl, Kim S. Larsen, and Jesper W. Mikkelsen. Online algorithms with advice: A survey. *SIGACT News*, 47(3):93–129, 2016.

[50] Joan Boyar, Lene M. Favrholdt, and Kim S. Larsen. The relative worst-order ratio applied to paging. *Journal of Computer and System Sciences*, 73(5):818–843, 2007.

[51] Joan Boyar, Lene M. Favrholdt, and Kim S. Larsen. Relative worst-order analysis: A survey. In *Adventures Between Lower Bounds and Higher Altitudes*, pages 216–230. Springer, 2018.

[52] Joan Boyar, Sandy Irani, and Kim S. Larsen. A comparison of performance measures for online algorithms. *Algorithmica*, 72(4):969–994, 2015.

[53] Joan Boyar, Shahin Kamali, Kim S. Larsen, and Alejandro López-Ortiz. On the list update problem with advice. *Information and Computation*, 2016.

[54] Joan Boyar, Shahin Kamali, Kim S. Larsen, and Alejandro López-Ortiz. Online bin packing with advice. *Algorithmica*, 74(1):507–527, 2016.

[55] Joan Boyar, Kim S. Larsen, and Morten N. Nielsen. The accommodating function: A generalization of the competitive ratio. *SIAM Journal on Computing*, 31(1):233–258, 2001.

[56] Carlos Fisch Brito, Elias Koutsoupias, and Shailesh Vaya. Competitive analysis of organization networks or multicast acknowledgment: How much to wait? *Algorithmica*, 64(4):584–605, 2012.

[57] Niv Buchbinder, Shahar Chen, Joseph Naor, and Ohad Shamir. Unified algorithms for online learning and competitive analysis. *Math. Oper. Res.*, 41(2):612–625, 2018.

[58] A Robert Calderbank, Edward G Coffman Jr, and Leopold Flatto. Sequencing problems in two-server systems. *Mathematics of Operations Research*, 10(4):585–598, 1985.

[59] A Robert Calderbank, Edward G Coffman Jr, and Leopold Flatto. Sequencing two servers on a sphere. *Communications in Statistics. Stochastic Models*, 1(1):17–28, 1985.

[60] Marek Chrobak, Howard J. Karloff, T. H. Payne, and Sundar Vishwanathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4(2):172–181, 1991.

[61] Marek Chrobak and Claire Kenyon. Competitiveness via doubling. *SIGACT News*, pages 115–126, 2006.

[62] Marek Chrobak and Lawrence L. Larmore. An optimal on-line algorithm for k-servers on trees. *SIAM Journal on Computing*, 20(1):144–148, 1991.

[63] Marek Chrobak and Lawrence L Larmore. The server problem and on-line games. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 7:11–64, 1992.

[64] Edward G. Coffman Jr., János Csirik, Gabor Galambos, Silvano Martello, and Daniele Vigo. Bin packing approximation algorithms: survey and classification. In Panos M. Pardalos, Ding-Zhu Du, and Ronald L. Graham, editors, *Handbook of Combinatorial Optimization*, pages 455–531. Springer, 2013.

[65] Anne Condon, Amol Deshpande, Lisa Hellerstein, and Ning Wu. Algorithms for distributional and adversarial pipelined filter ordering problems. *ACM Transaction on Algorithms*, 5(2):24:1–24:34, 2009.

[66] Erik D. Demaine, Sándor Fekete, and Shmuel Gal. Online searching with turn cost. *Theoretical Computer Science*, 361:342–355, 2006.

[67] Peter J. Denning. Working sets past and present. *IEEE Trans. Softw. Eng.*, 6:64–84, 1980.

[68] Stefan Dobrev, Rastislav Královič, and Dana Pardubská. Measuring the problem-relevant information in input. *RAIRO Inform. Theor. Appl.*, 43(3):585–613, 2009.

[69] Reza Dorrigiv and Alejandro López-Ortiz. A survey of performance measures for on-line algorithms. *SIGACT News (ACM Special Interest Group on Automata and Computability Theory)*, 36(3):67–81, September 2005.

[70] Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. Online computation with advice. *Theoret. Comput. Sci.*, 412(24):2642 – 2656, 2011.

[71] Matthias Englert and Matthias Westermann. Reordering buffer management for non-uniform cost models. In *Proceedings of the 32nd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 627–638, 2005.

[72] Rudolf Fleischer, Tom Kamphans, Rolf Klein, Elmar Langetepe, and Gerhard Trippen. Competitive online approximation of the optimal search ratio. *SIAM Journal on Computing*, 38(3):881–898, 2008.

[73] Shmuel Gal. A general search game. *Israel Journal of Mathematics*, 12:32–45, 1972.

[74] Shmuel Gal. Minimax solutions for linear search problems. *SIAM Journal on Applied Mathematics*, 27:17–30, 1974.

[75] Shmuel Gal. Search games with mobile and immobile hider. *SIAM Journal on Control and Optimization*, 17(1):99–122, 1979.

[76] Shmuel Gal. *Search Games*. Academic Press, 1980.

[77] Minos Garofalakis, Yannis Ioannidis, Banu Özden, and Avi Silberschatz. Competitive online scheduling of continuous-media streams. *Journal of Computer and System Sciences*, 64(2):219–248, 2002.

[78] Michel X. Goemans and Jon M. Kleinberg. An improved approximation ratio for the minimum latency problem. *Mathematical Programming*, 82:111–124, 1998.

[79] Carla P. Gomes and Bart Selman. Algorithm portfolios. *Artificial Intelligence*, 126(1–2):43–62, 2001.

[80] Ronald L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical J.*, 45:1563–1581, 1966.

[81] Benjamin Hiller and Tjark Vredeveld. Probabilistic analysis of online bin coloring algorithms via stochastic comparison. In *Proceedings of the 16th Annual European Symposium on Algorithms (ESA)*, pages 528–539, 2008.

[82] Benjamin Hiller and Tjark Vredeveld. Simple optimality proofs for Least Recently Used in the presence of locality of reference. Technical report, Maastricht University of Business and Economics, 2009.

[83] Benjamin Hiller and Tjark Vredeveld. Probabilistic alternatives for competitive analysis. *Computer Science - R&D*, 27(3):189–196, 2012.

[84] Micha Hofri. Should the two-headed disk be greedy? - yes, it should. *Information Processing Letters*, 16(2):83–85, 1983.

[85] Erik Horvitz. Reasoning about beliefs and actions under computational resource constraints. *Int. J. Approx. Reasoning*, 2(3):337–338, 1988.

[86] Sandy Irani. Two results on the list update problem. *Inform. Process. Lett.*, 38:301–306, 1991.

[87] Rufus Isaacs. *Differential games*. John Wiley and Sons, New York, 1965.

[88] Patrick Jaillet and Matthew Stafford. Online searching. *Operations Research*, 49:234–244, 1993.

[89] David S. Johnson. *Near-optimal bin packing algorithms*. PhD thesis, MIT, Cambridge, MA, 1973.

[90] David S. Johnson. Bin packing. In *Encyclopedia of Algorithms*, pages 207–211. 2016.

[91] Ming-Yang Kao and Michael Littman. Algorithms for informed cows. In *Proceedings of the AAAI 1997 Workshop on Online Search*, 1997.

[92] Ming-Yang Kao, Yuan Ma, Michael Sipser, and Yiqun Yin. Optimal constructions of hybrid algorithms. *Journal of Algorithms*, 29(1):142–164, 1998.

[93] Ming-Yang Kao, John H Reif, and Stephen R Tate. Searching in an unknown environment: an optimal randomized algorithm for the cow-path problem. *Information and Computation*, 131(1):63–80, 1996.

[94] Anna Karlin, Mark Manasse, Larry Rudolph, and Daniel Sleator. Competitive snoopy caching. *Algorithmica*, 3:79–119, 1988.

[95] Anna R Karlin, Claire Kenyon, and Dana Randall. Dynamic tcp acknowledgment and other stories about e/(e- 1). *Algorithmica*, 36:209–224, 2003.

[96] Anna R. Karlin, Mark S. Manasse, Lyle A. McGeoch, and Susan Owicki. Competitive randomized algorithms for non-uniform problems. pages 301–309, 1990.

[97] Dennis F. Karney. Duality gaps in semi-infinite linear programming–an approximation problem. *Mathematical Programming*, 20:129–143, 1981.

[98] Michael Keane. Interval exchange transformations. *Mathematische Zeitschrift*, 141:25–31, 1975.

[99] Claire Kenyon. Best-fit bin-packing with random order. In *Proceedings of the 7th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 359–364, 1996.

[100] David G. Kirkpatrick. Hyperbolic dovetailing. In *Proceedings of the 17th Annual European Symposium on Algorithms (ESA)*, pages 616–627, 2009.

[101] Elias Koutsoupias. The k-server problem. *Computer Science Review*, 3(2):105–118, 2009.

[102] Elias Koutsoupias and Christos Papadimitriou. Beyond competitive analysis. *SIAM J. on Computing*, 30:300–317, 2000.

[103] Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *Journal of the ACM*, 42(5):971–983, 1995.

[104] Elias Koutsoupias, Christos H. Papadimitriou, and Michalis Yannakakis. Searching a fixed graph. In *Proc. of the 23rd Int. Colloq. on Automata, Languages and Programming (ICALP)*, pages 280–289, 1996.

[105] Alejandro López-Ortiz, Spryos Angelopoulos, and Angèle M. Hamel. Optimal scheduling of contract algorithms for anytime problems. *Journal of Artificial Intelligence Research*, (51):533–554, 2014.

[106] Alejandro López-Ortiz and Sven Schuierer. The ultimate strategy to search on m rays? *Theoretical Computer Science*, 261(2):267–295, 2001.

[107] Alejandro López-Ortiz and Sven Schuierer. On-line parallel heuristics, processor scheduling and robot searching under the competitive framework. *Theoretical Computer Science*, 310(1–3):527–537, 2004.

[108] Steven S. Lumetta and Michael Mitzenmacher. Using the power of two choices to improve bloom filters. *Internet Mathematics*, 4(1):17–33, 2007.

[109] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, pages 3302–3311, 2018.

[110] Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for on-line problems. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computation (STOC)*, pages 322–333, 1988.

[111] David Matula. A periodic optimal search. *The American Mathematical Monthly*, 71(1):15–21, 1964.

[112] Andrew McGregor, Krzysztof Onak, and Rina Panigrahy. The oil searching problem. In *Proceedings of the 17th European Symposium on Algorithms (ESA)*, pages 504–515, 2009.

[113] Adam Meyerson. The parking permit problem. In *46th Annual IEEE Symposium on Foundations of Computer Science*, pages 274–284. IEEE, 2005.

[114] Michael Mitzenmacher. Bounds on the greedy routing algorithm for array networks. *Journal of Computer and System Sciences*, 53(3):317–327, 1996.

[115] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In *Advances in Neural Information Processing Systems*, volume 31, pages 9661–9670, 2018.

[116] Harald Räcke, Christian Sohler, and Matthias Westermann. Online scheduling for sorting buffers. In *Proceedings of the 10th Annual European Symposium on Algorithms (ESA)*, pages 820–832, 2002.

[117] Marc P. Renault, Adi Rosén, and Rob van Stee. Online algorithms with advice for bin packing and scheduling problems. *Theor. Comput. Sci.*, 600:155–170, 2015.

[118] H Edwin Romeijn, Robert L Smith, and James C Bean. Duality in infinite dimensional linear programming. *Mathematical Programming*, 53:79–97, 1992.

[119] Tomislav Rudec, Alfonzo Baumgartner, and Robert Manger. Measuring true performance of the work function algorithm for solving the on-line k-server problem. *Journal of Computing and Information Technology*, 18(4), 2010.

[120] Tomislav Rudec, Alfonzo Baumgartner, and Robert Manger. A fast work function algorithm for solving the k-server problem. *Central European Journal of Operations Research*, 21(1):187–205, 2013.

[121] Stuart J. Russell and Shlomo Zilberstein. Composing real-time systems. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 212–217, 1991.

[122] Sven Schuierer. A lower bound for randomized searching on m rays. In *Computer Science in Perspective*, pages 264–277, 2003.

[123] Sridhar Seshadri and Doron Rotem. The two headed disk: Stochastic dominance of the greedy policy. *Information Processing Letters*, 57(5):273–277, 1996.

[124] René Sitters. The minimum latency problem is np-hard for weighted trees. In *Proceedings of the 9th Inernational Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 230–239, 2002.

[125] Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.

[126] Elmar Wolfstetter. *Topics in Microeconomics*. Cambridge University Press, Cambridge, 1999.

[127] Neal E. Young. On-line caching as cache size varies. In *Proceedings of the 2nd Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms (SODA)*, pages 241–250, 1991.

[128] Neal E. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11(6):525–541, 1994.

[129] Neal E. Young. Bounding the diffuse adversary. In *Proceedings of the 9th Annual ACM-SIAM symposium on Discrete Algorithms (SODA)*, pages 420–425, 1998.

[130] Neal E. Young. On-line paging against adversarially biased random inputs. *Journal of Algorithms*, 37(1):218–235, 2000.

[131] Neal E. Young. On-line file caching. *Algorithmica*, 33(3):371–383, 2002.

[132] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI Mag.*, 17(3):73–83, 1996.

[133] Shlomo Zilberstein, François Charpillet, and Philippe Chassaing. Optimal sequencing of contract algorithms. *Ann. Math. Artif. Intell.*, 39(1-2):1–18, 2003.