View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Dagstuhl Research Online Publication Server

Online Computation with Untrusted Advice

Spyros Angelopoulos
Sorbonne Université, CNRS, Laboratoire d'informatique de Paris 6, LIP6, Paris, France
spyros.angelopoulos@lip6.fr

Christoph Dürr
Sorbonne Université, CNRS, Laboratoire d'informatique de Paris 6, LIP6, Paris, France
christoph.durr@lip6.fr

Shendan Jin
Sorbonne Université, CNRS, Laboratoire d'informatique de Paris 6, LIP6, Paris, France
shendan.jin@lip6.fr

Shahin Kamali
Department of Computer Science, University of Manitoba, Winnipeg, Canada
shahin.kamali@umanitoba.ca

Marc Renault
Computer Sciences Department, University of Wisconsin – Madison, Madison, WI, USA
mrenault@cs.wisc.edu

Abstract

The advice model of online computation captures the setting in which the online algorithm is given some partial information concerning the request sequence. This paradigm allows to establish tradeoffs between the amount of this additional information and the performance of the online algorithm. However, 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. To this end, we focus on well- studied online problems such as ski rental, online bidding, bin packing, and list update. 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 et al. [NeurIPS 2018] 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 [ICML 2018] 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 model.

2012 ACM Subject Classification Theory of computation → Online algorithms

Keywords and phrases Online computation, competitive analysis, advice complexity, robust algorithms, untrusted advice

Digital Object Identifier 10.4230/LIPIcs.ITCS.2020.52

Related Version A full version of the paper is available at `https://arxiv.org/abs/1905.05655`.

Funding Research supported by the CNRS-PEPS Project ADVICE.

© Spyros Angelopoulos, Christoph Dürr, Shendan Jin, Shahin Kamali, and Marc Renault;
licensed under Creative Commons License CC-BY

1 Introduction

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 customer 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 the 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, with the objective of obtaining more realistic and robust online algorithms.

1.1 Online computation and advice complexity

In the standard model of online computation that goes back to the seminal work of Sleator and Tarjan [26], an online algorithm receives as input a sequence of *requests*. For each request in this sequence, the algorithm must make an irrevocable decision concerning the item, without any knowledge of future requests. The performance of an online algorithm is usually evaluated by means of the competitive ratio, which is the worst-case ratio of the cost incurred by the algorithm (assuming a minimization problem) to the cost of an ideal solution that knows the entire sequence in advance.

In practice, however, online algorithms are often provided with some (limited) knowledge of the input, such as lookahead on some of the upcoming requests, or knowledge of the input size. 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. [12], and subsequent formal models were presented by Böckenhauer et al. [6] and Emek et al. [13], 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 conveyed to the algorithm plays an important role in obtaining better online algorithms. Clearly, the performance of the online algorithm can only improve with larger number of advice bits. The objective is thus to identify the exact 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). 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. [7] for an in-depth discussion of developments in this field.

As argued in detail in [7], there are compelling reasons to study the advice complexity of online computation. Lower bounds establish strict limitations on the power of any online algorithm; there are strong connections between randomized online algorithms and online algorithms with advice (see, e.g., [16]); online algorithms with advice can be of practical

interest in settings in which it is feasible to run multiple algorithms and output the best solution (see [17] about obtaining improved data compression algorithms by means of list update algorithms with advice); and the first complexity classes for online computation have been based on advice complexity [8].

Notwithstanding such interesting attributes, the known advice model has certain drawbacks. The advice is always assumed to be some error-free information that may be used to encode some property often explicitly connected to the optimal solution. In many settings, one can argue that such information cannot be readily available, which implies that the resulting algorithms are often impractical.

1.2 Online computation with untrusted advice

In this work, we address what is a significant drawback in the online advice model. Namely, all previous works assume that advice is, in all circumstances, completely trustworthy, and precisely as defined by the algorithm. Since the advice is infallible, no reasonable online algorithm with advice would choose to ignore the advice.

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 well-known ski rental problem: this is a simple, yet fundamental resource allocation, in which we have to decide ahead of time whether to rent or buy equipment without knowing the time horizon in advance. 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. In contrast, an online algorithm that does not use advice at all has competitive ratio at most 2, i.e., its output can be at most twice as costly as the optimal one.

The above observations were recently made in the context of online algorithms with machine-learned predictions. Lykouris and Vassilvitskii [21] and Purohit et al. [23] show how to use predictors 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*).

Motivated by these 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 tape. At each time t, A serves request $\sigma[t]$, and its output is a function of $\sigma[1..\ldots t-1]$ and

$\phi \in \{0,1\}^*$. Let $A(\sigma,\phi)$ denote the cost incurred by A on input σ, using an advice string ϕ. Denote by r_A, w_A as

$$r_A = \sup_{\sigma} \inf_{\phi} \frac{A(\sigma,\phi)}{\textsc{Opt}(\sigma)}, \quad \text{and} \quad w_A = \sup_{\sigma} \sup_{\phi} \frac{A(\sigma,\phi)}{\textsc{Opt}(\sigma)}, \tag{1}$$

where $\textsc{Opt}(\sigma)$ 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, the algorithm 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, whereas the standard competitively optimal algorithm without advice is $(2,2)$-competitive. In general, every online algorithm A without advice or ignoring its 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 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.

1.3 Contribution

We study various online problems in the 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 new advice model. We begin in Section 2 with a simple, yet illustrative online problem as a case study, namely the *ski-rental* problem. Here, we give a Pareto-optimal algorithm with only one bit of advice. We also show that this algorithm is Pareto-optimal even in the space of all (deterministic) algorithms with advice of *any* size.

In Section 3 we study the *online bidding* problem, in which the objective is to guess an unknown, hidden value, using a sequence of bids. This problem was introduced in [11] as a vehicle for formalizing efficient doubling, and has applications in several important online and offline optimization problems. As with ski rental, this is another problem for which a trivial online algorithm is $(1,\infty)$-competitive. We first show how to find a Pareto-optimal strategy, when the advice encodes the hidden value, and thus can have unbounded size. Moreover, we study the competitiveness of the problem with only k bits of advice, for some fixed k, and show both upper and lower bounds on the achieved competitive ratios. The results illustrate that is is possible to obtain non-trivial lower bounds on the competitive ratios, in terms of the advice size. In particular, the lower bound implies that, unlike the ski rental problem, Pareto-optimality is not possible with a bounded number of advice bits.

In Sections 4 and 5, we study the *bin packing* and *list update* problems; these problems are central in the analysis of online problems and competitiveness, and have numerous applications in practice. For these problems, 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 observe that the existing algorithms with advice perform poorly in the case the advice is untrusted. To address this, we give 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. Our $(r, f(r))$-competitive algorithm has $f(r) = \max\{33 - 18r, 7/4\}$ 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 [10]. 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. Likewise, for the list update problem, our $(r, f(r))$-competitive algorithm has $f(r) = 2 + \frac{10-3r}{9r-5}$ for $r \in [5/3, 2]$. If the algorithm takes maximum risk, i.e., if r is smallest, the algorithm is equivalent to an existing $(5/3, 5/2)$-competitive algorithm [9]. Again, by increasing r, we better safeguard against the event of untrusted advice.

All the above results pertain to deterministic online algorithms. In Section 6, we study the power of randomization in online computation with untrusted advice. First, we show that the randomized algorithm of Purohit et al. [23] for the ski rental problem Pareto-dominates any deterministic algorithm, even when the latter is allowed unbounded advice. Furthermore, we show an interesting difference between the standard advice model and the model we introduce: in the former, an advice bit can be at least as powerful as a random bit, since an advice bit can effectively simulate any efficient choice of a random bit. In contrast, we show that in our model, there are situations in which a randomized algorithm with L advice bits and one random bit is Pareto-incomparable to the Pareto-optimal deterministic algorithm with $L + 1$ advice bits. This confirms the intuition that a random bit is considered trusted, and thus not obviously inferior to an advice bit.

While our work addresses issues similar to [21] and [23], 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, [21, 23] focus on "smooth" tradeoffs between the trusted and untrusted competitive ratios, but do not address the issues related to optimality and 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. For all problems we study, we parameterize advice by its size, i.e., we allow advice of a certain size k. Specifically, the advice need not necessarily encode the optimal solution or the request sequence itself. 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 A warm-up: the ski rental problem

2.1 Background

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 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 skiing days, which we denote by D, is unknown to the algorithm, and we observe that the optimal offline cost is $\min\{D, B\}$. Generalizations of ski rental have been applied in many settings, such as dynamic TCP acknowledgment [19], the parking permit problem [22], and snoopy caching [18].

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 is as high as D/B, if $D > B$. Hence, this algorithm is $(1, \infty)$-competitive, for $D \to \infty$.

2.2 Ski rental with 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 algorithms buys on day k.

▶ **Proposition 1.** *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 [23], 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 2.** *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 $1 + (B-1)/B$ [18]. 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 $1+(B-1)/k$. ◀

3 Online bidding

3.1 Background

In the *online bidding* problem, a player wants to guess a hidden, unknown real value $u\geq 1$. To this end, the player submits a sequence $X=(x_i)$ of 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.

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

The problem was introduced in [11] as a canonical problem for formalizing doubling-based strategies in online and offline optimization problems, such as searching for a target on the line, minimum latency, and hierarchical clustering. It is worth noting that online bidding is identical to the problem of minimizing the *acceleration ratio of interruptible algorithms* [25]; the latter and its generalizations are problems with many practical applications in AI (see, for instance [20]).

Without advice, the best competitive ratio is 4, and can be achieved using the doubling strategy $x_i=2^i$. If the advice encodes[1] 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.

3.2 Online bidding with untrusted advice

Suppose that $w\geq 4$ is a fixed, given parameter. We will show a Pareto-optimal bidding strategy X^*_u, assuming that the advice encodes u, which is $(\frac{w-\sqrt{w^2-4w}}{2},w)$-competitive (Theorem 5).

We begin with 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, 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, $r\leq r'$ holds.

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 $P_{m,u}$, and which is shown below. For convenience, for any strategy X, we will always define x_0 to be equal to 1.

[1] We assume that the advice provides the exact value u to the algorithm. For practical considerations, it suffices to assume an oracle that provides an $(1+\epsilon)$-approximation of the hidden value, for sufficiently small $\epsilon>0$. This will only affect the competitive ratios by the same negligible factor.

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

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

We also observe that an optimal solution $X^*_{m,u} = (x^*_i)_{i\ge 1}$ for $P_{m,u}$ must be such that $x_m = u$, otherwise one could define a strategy $X'_{m,u}$ in which $x'_i = x^*_i/\alpha$, for all $i \ge 1$, with $\alpha = u/x^*_m$, which is still feasible for $P_{m,u}$, is such that $x'_m = u$, and has better objective value than $X^*_{m,u}$, a contradiction. Furthermore, in an optimal solution, the constraint $\sum_{i=1}^{m} x_i \le r_{m,u} \cdot u$ must hold with equality. Therefore, $X^*_{m,u}$ and $r^*_{m,u}$ are also solutions to the linear program $L_{m,u}$.

Next, define $r^*_u = \inf_m r^*_{m,u}$, and $r^* = \sup_u r^*_u$. Informally, r^*_u, r^* are the optimal competitive ratios, assuming trusted advice. More precisely, the dominant strategy in the space of all w-competitive strategies is (r^*_u, w)-competitive, and r^* is an upper bound on r^*_u, assuming the worst-case choice of u.

We first argue how to compute $r^*_{m,u}$ and the corresponding strategy $X^*_{m,u}$, provided that $L_{m,u}$ is feasible. This is accomplished in Lemma 3. The main idea behind the technical proof is to show that in an optimal solution of $L_{m,u}$, all constraints C_i 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 $X^*_{m,u}$.

Define the sequences a_i and b_i as follows:

$$
a_i = \frac{a_{i-1}}{w - 1 - b_{i-1}}, \text{ with } a_0 = 1, \text{ and } \quad b_i = \frac{1 + b_{i-1}}{w - 1 - b_{i-1}}, \text{ with } b_0 = 0, \tag{2}
$$

Moreover, for $w > 4$, let $\rho_1 = \frac{w - \sqrt{w^2-4w}}{2}$ and $\rho_2 = \frac{w + \sqrt{w^2-4w}}{2}$ denote the two roots of $x^2 - wx + w$, the characteristic polynomial of the above linear recurrence.

▶ **Lemma 3.** *For every m define $X_{m,u}$ as follows:*

- *If $w > 4$, then $x_{m,u,i} = \alpha \cdot \rho_1^{i-1} + \beta \cdot \rho_2^{i-1}$, where $\alpha = \frac{a_{m-1}\rho_2^{m-1} - 1}{\rho_2^{m-1} - \rho_1^{m-1}} \cdot u$, and $\beta = \frac{a_{m-1}\rho_1^{m-1} - 1}{\rho_1^{m-1} - \rho_2^{m-1}} \cdot u$,*
- *If $w = 4$, then $x_{m,u,i} = (\alpha + \beta \cdot i) \cdot 2^i$, where $\alpha = \frac{2^{m-1} \cdot m \cdot a_{m-1} - 1}{2^m (m-1)} \cdot u$, and $\beta = \frac{1 - 2^{m-1} \cdot a_{m-1}}{2^m (m-1)} \cdot u$.*

Then, $X_{m,u}$ is an optimal feasible solution if and only if $a_{m-1} \cdot u \le w$.

We can now give the statement of 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 $w \geq 4$; hence $m^* \leq \lceil \log u \rceil$. Then X_u^* is derived as in the statement of Lemma 3. The time complexity of the algorithm is $O(\log \log u)$, since we can describe each a_i, b_i, and hence a_{m-1} in closed form, avoiding the recurrence which would add a $O(\log u)$ factor. The technical details can be found in the long version of this paper [2].

Last, the following lemma allows us to express r^* as a function of the values of the sequence b, which we can further exploit so as to obtain the exact value of r_u^*.

▶ **Lemma 4.** *It holds that* $r^* = 1 + \sum_{i=1}^{\infty} \prod_{j=1}^{i-1} b_j$. *Furthermore,* $r^* = \frac{w - \sqrt{w^2 - 4w}}{2}$.

Combining Lemmas 3 and 4 we obtain following result:

▶ **Theorem 5.** *Strategy* X_u^* *is Pareto-optimal and is* $(\frac{w - \sqrt{w^2 - 4w}}{2}, w)$*-competitive.*

Strategy X_u^* requires u as advice, which can be unbounded. A natural question is what competitiveness can one achieve with 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 show the following:

▶ **Theorem 6.** *For every* $w \geq 4$*, there exists a bidding strategy with* k *bits of advice which is* (r, w)*-competitive, where*

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

and where $K = 2^k$.

In particular, for $w = 4$, the strategy of Theorem 6 is $(2^{1 + \frac{1}{2^k}}, 4)$-competitive, whereas X_u^* is $(2, 4)$-competitive. The following theorem gives a lower bound on the competitiveness of any bidding strategy with k bits. The result shows that one needs unbounded number of bids to achieve $(2, 4)$-competitiveness.

▶ **Theorem 7.** *For any bidding strategy with* k *advice bits that is* $(r, 4)$*-competitive it holds that* $r \geq 2 + \frac{1}{3 \cdot 2^k}$.

Proof sketch. We present only an outline. With k bits of advice, the online algorithm can differentiate only between $K = 2^k$ online bidding sequences, denoted by $X_1, \ldots, X_K$, each of which must have (untrusted) competitive ratio at least 4. Suppose, by way of contradiction, that the algorithm has trusted competitive ratio less than $2 + \frac{1}{3 \cdot 2^k}$. We reach a contradiction, by applying a game between the algorithm and the adversary, which proceeds in rounds. The adversary fixes a sufficiently large index $i \geq i_0$, for some i_0. In the first round, u is chosen by the adversary so as to be infinitesimally larger than $x_{K,i-1}$, namely the $(i-1)$-th bid of X_K. For the algorithm to guarantee the claimed r, we show that it will have to use the advice so as to "choose" one of the sequences $X_1, \ldots X_{K-1}$, say X_j. Then in the next round, the adversary will choose an appropriate u that is adversarial for X_j. The crux of the proof is to show that the algorithm's only response is to choose a sequence of index higher than j. Eventually, the only remaining choice for the algorithm is strategy X_K; moreover, we can show that throughout the execution of the algorithm the adversarial u is comparable to $x_{K,i-1}$, in particular, we show that $u \leq e^{1/3} x_{K,i-1}$. To conclude, the above argument shows that

$$r \geq \sup_{i \geq i_0} \frac{\sum_{j=1}^{i} x_{K,j}}{e^{\frac{1}{3}} x_{K,i-1}} = \frac{1}{e^{\frac{1}{3}}} \sup_{i=1} \frac{\sum_{j=1}^{i} x_{K,j}}{x_{K,i-1}} \geq \frac{4}{e^{\frac{1}{3}}} > 2 + \frac{1}{3K}.$$

where we used the fact that $\sup_{i=1} \frac{\sum_{j=1}^{i} x_{K,j}}{x_{K,i-1}} = 4$, since X_K is 4-competitive. ◀

4 Online bin packing

4.1 Background

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, each with a capacity of 1. For each arriving item, the algorithm must place it in one of the current bins or open a new bin for the item. We say that algorithm A has an asymptotic competitive ratio r if, on every sequence σ, the number of opened bins satisfies $A(\sigma) \leq r \cdot \textsc{Opt}(\sigma) + c$, where c is a constant. As standard in the analysis of bin packing problems, throughout this section, by "competitive ratio" we mean "asymptotic competitive ratio". The First Fit [15] algorithm maintains bins in the same order that they have been opened, and places an item into the first bin with enough free space; if no such bin exists, it opens a new bin. First Fit has a competitive ratio of 1.7 [15] while the best online algorithm has a competitive ratio of at least 1.54278 [5] and at most 1.57829 [4]. Online bin packing has also been studied in the advice setting [10, 24, 3]. In particular, it is possible to achieve a competitive ratio of 1.4702 with only a constant number of (trusted) advice bits [3].

In this section, we introduce an algorithm named Robust-Reserve-Critical (Rrc) which has a parameter $\alpha \in [0, 1]$, indicating how much the algorithm relies on the advice. Provided with $O(1)$ bits of advice, the algorithm is asymptotically $(r_{\text{Rrc}}, w_{\text{Rrc}})$-competitive for $r_{\text{Rrc}} = 1.5 + \frac{1-\alpha}{4-3\alpha}$ and $w_{\text{Rrc}} = 1.5 + \max\{\frac{1}{4}, \frac{9\alpha}{8-6\alpha}\}$. If the advice is reliable, we set $\alpha = 1$ and the algorithm is asymptotically $(1.5, 6)$-competitive; otherwise, we set α to a smaller value.

4.2 The Reserve-Critical algorithm

Our solution uses an algorithm introduced by Boyar et al. [10] which achieves a competitive ratio of 1.5 using $O(\log n)$ bits of advice [10]. We refer to this algorithm as Reserve-Critical in this paper and describe it briefly. The algorithm 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.

4.3 The Robust-Reserve-Critical (RRC) algorithm

Let t be the number of tiny bins opened by the Reserved-Critical algorithm. Recall that c is the number of critical bins. We call the fraction $c/(c+t)$ the *critical ratio*. The advice for Rrc is a fraction γ, integer multiple of $1/2^k$, that is encoded in k bits such that if the advice is trusted then $\gamma \leq c/(c+t) \leq \gamma + 1/2^k$. In case $c/(c+t)$ is a positive integer multiple of

$1/2^k$, we break the tie towards $\gamma < c/(c+t)$. Note that for sufficiently large, yet constant, number of bits, γ provides a good approximation of the critical ratio. Indeed having γ as advice is sufficient to achieve a competitive ratio that approaches 1.5 in the trusted advice model, as shown in [3].

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 strategy (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 the Reserved-Critical algorithm (one large item in each large bin and two small items in each small bin).

4.4 Analysis

Intuitively, RRC works similarly to Reserved-Critical except that it might not open as many critical bins as suggested in the advice. The algorithm is more "conservative" in the sense that it does not keep two-third of many (critical) bins open for critical items that might never arrive. The smaller the value of α is, the more conservative the algorithm is. Our analysis is based on two possibilities in the final packing of the algorithm. In the first case (case I), all critical bins receive a critical item, while in the second case (case II) some of them have their reserved space empty. In case I, we show the number of bins in the packing of RRC is within a factor $1.5 + \frac{1-\beta}{4-3\beta}$ of the number of bins in the optimal packing. Note that this ratio decreases as the value of α (and β) grows. This implies a less conservative algorithm would be better packing in this case. Case II happens only if the advice is untrusted. In this case, the number of bins in the RRC packing is within a factor $1.5 + \frac{9\beta}{8-6\beta}$ of the number of bins in an optimal packing. This ratio increases with α (and β). This implies a more conservative algorithm would be better in this case as it would open less critical bins and, thus, would have fewer without critical items.

Assume the advice is trusted. Then either $\gamma \leq \alpha$ or $\gamma > \alpha$. In the former case, the algorithm maintains the same ratio as suggested by advice, and a result from [3] indicates that the competitive ratio is at most $1.5 + \frac{15}{2^{k/2+1}}$. In the former case, the algorithm maintains a smaller number of critical items than what the advice suggested; all these bins receive critical items and the final packing will be in Case I. Consequently, when the advice is trusted, the competitive ratio is at most $1.5 + \max\{\frac{1-\alpha}{4-3\alpha}, \frac{15}{2^{k/2+1}}\}$. If the advice is untrusted, both case I and case II can be realized for the final packing. The competitive ratio will be at most $1.5 + \max\{\frac{1}{4}, \frac{9\alpha}{8-6\alpha}\}$. We can conclude with the following theorem:

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

Assuming the size k of the advice is a sufficiently large constant, we conclude the following.

▶ **Corollary 9.** *For bin packing with untrusted advice, there is a $(r, f(r))$-competitive algorithm where $r \geq 1.5$ and $f(r) = \max\{33 - 18r, 7/4\}$.*

5 List update

5.1 Background

The list update problem consists of a list of items of length m, and a sequence of n requests that should be served with minimum total cost. Every 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 algorithm can move it closer to the front of the list with no cost using a "free exchange". In addition, at any point, the algorithm can swap the position of any two consecutive items in the list using a "paid exchange" which has a cost of 1. Throughout this section, we adopt the standard assumption that m is a large integer but still a constant with respect to n.

Move-to-Front (MTF) is an algorithm that moves every accessed item to the front of the list using a free exchange. MTF has a competitive ratio of at most 2 [27], which is the best that a deterministic algorithm can achieve [14]. Timestamp [1] 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. Move-To-Front-Every-Other-Access (MTF2) is a class of algorithms which 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 0 at the beginning, MTF2 is called called Move-To-Front-Even (MTFE), and if all bits are 1 at the beginning, MTF2 is called Move-To-Front-Odd (MTFO). Both MTFE and MTFO algorithms have a competitive ratio of 5/2 [9]. In [9] it is shown that, for any request sequence, at least one of Timestamp, MTFO, and MTFE has a competitive ratio of at most 5/3. 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 can be as bad as 5/2.

To address this issue, we introduce an algorithm named 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 of $r_{\text{TOG}} = 5/3 + \frac{5\beta}{6+3\beta}$ when the advice is trusted and a competitive ratio of 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 get a $(5/3, 2.5)$-competitive algorithm.

5.2 The Toggle algorithm

Given the parameter β, the Toggle algorithm (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. The algorithm 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. As such, TOG will use MTF2 for the odd phases which we will call *trusting phases*, and MTF for the even phases which we will call *ignoring phases*. The actions during each phase are formally defined below.

Trusting phase: In a trusting phase, TOG will use MTF2 to serve the requests. Let σ_i be the first request of some trusting phase j for $1 \leq i \leq n$ and an odd $j \geq 1$. Before serving σ_i, TOG modifies the list with paid exchanges to match the list configuration that would result

from running MTF2 on the request sequence $\sigma_1, \ldots, \sigma_i$. The number of paid exchanges will be less than m^2. In addition, TOG will set the bits of items in the list to the same value as at the end of this hypothetical run. As such, during a trusting phase, TOG incurs the same access cost as MTF2. The trusting phase continues until the cost to access a request σ_ℓ, $i < \ell \leq n$, for TOG would cause the total access cost for the phase to become at least m^3 (or the request sequence ends). The next phase, which will be an ignoring phase, begins with request $\sigma_{\ell+1}$.

Ignoring phase: In an ignoring phase, TOG will use the MTF rule to serve the request. Unlike the trusting phase, TOG does not use paid exchanges to match another list configuration. Let σ_i be the first request of some ignoring phase j for $1 \leq i \leq n$ and an even $j \geq 1$. The ignoring phase continues until the cost to access a request σ_ℓ, $i < \ell \leq n$, for TOG would cause the total access cost for the phase to exceed $\beta \cdot m^3$ (or the request sequence ends). The next phase, which will be a trusting phase, begins with request $\sigma_{\ell+1}$.

5.3 Analysis

The access cost in each phase of TOG is $\Theta(m^3)$, while the cost that it might pay at the beginning of each phase is $O(m^2)$. Consequently, the cost of the algorithm in a trusting phase is roughly $m^3 + o(m^3)$ and $\beta m^3 + o(m^3)$ for an ignoring phase. Moreover, the optimal algorithm incurs a cost of at least $m^3/2.5 - m^2$ in a trusting phase and at least $\beta m^3/2 - m^2$ in an ignoring phase; these results follow from the upper bounds of respectively 2.5 and 2 for the competitive ratios of MTF and MTF2, and the fact that the discrepancy in the initial configuration of each phase changes the cost of OPT in that phase by at most m^2. We can extend these results to show that, for sufficiently long lists, the competitive ratio of TOG (regardless of the advice being trusted or not) converges to at most $2 + \frac{2}{4+5\beta}$. We conclude that, when the advice is untrusted, the competitive ratio of TOG is at most $2 + \frac{2}{4+5\beta}$.

Now, assume the advice is trusted. If the advice indicates Timestamp as the best algorithm among MTFE, MTFO, and Timestamp, the algorithm uses Timestamp to serve the entire sequence, and since the advice is right, the competitive ratio will be at most 5/3 [9]. If the advice indicates MTF2 (either MTFE or MTFO), we compare the cost of TOG with that of MTF2 in each phase. A careful phase analysis, similar to the one for the competitive ratio, shows that the ratio between the costs of the two algorithms converges to at most $2 + \frac{2}{4+5\beta}$. We conclude that, when the advice is trusted, the competitive ratio of the TOG algorithm converges to $5/3 + \frac{5\beta}{6+3\beta}$ for sufficiently long lists. We can state the following theorem:

▶ **Theorem 10.** *Algorithm* TOG *with parameter* $\beta \in [0, 1/2]$ *and* k *bits of advice achieves a competitive ratio of at most* $5/3 + \frac{5\beta}{6+3\beta}$ *when the advice is trusted and a competitive ratio of at most* $2 + \frac{2}{4+5\beta}$ *when the advice is untrusted.*

▶ **Corollary 11.** *For list update with untrusted advice, there is a* $(r, f(r))$*-competitive algorithm where* $r \in [5/3, 2]$ *and* $f(r) = 2 + \frac{10-3r}{9r-5}$.

6 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). Given a randomized algorithm A, its trusted and untrusted competitive ratios are defined as in (1), with the difference that the cost $A(\sigma, \phi)$ is now replaced by the expected cost $\mathbb{E}(A(\sigma, \phi))$.

First, we will argue that randomization can improve the competitiveness of the ski rental problem. For this, we note that [23] 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 2 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 much more intricate. 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 12.** *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, and which is $(\frac{1+\rho_1}{2}\rho_1 + \epsilon, \frac{1+\rho_1}{2\rho_1}w + \epsilon)$-competitive for all $w > 4$, where $\rho_1 = \frac{w-\sqrt{w^2-4w}}{2}$.*

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 5, namely (ρ_1, w). Thus, Theorem 12 shows that randomization improves upon the deterministic untrusted ratio w by a factor $\frac{1+\rho_1}{2\rho_1} > 1$, at the expense of a degradation of the trusted competitive ratio by a factor $\frac{1+\rho_1}{2} > 1$. 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.

References

1. Susanne Albers. Improved Randomized On-Line Algorithms for the List Update Problem. *SIAM J. Comput.*, 27:682–693, 1998.
2. Spyros Angelopoulos, Christoph Dürr, Shendan Jin, Shahin Kamali, and Marc P. Renault. Online Computation with Untrusted Advice. *CoRR*, abs/1905.05655, 2019. `arXiv:1905.05655`.
3. 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.
4. 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.
5. János Balogh, József Békési, György Dósa, Leah Epstein, and Asaf Levin. A new lower bound for classic online bin packing. *CoRR*, abs/1807.05554, 2018. `arXiv:1807.05554`.
6. Hans-Joachim Böckenhauer, Dennis Komm, Rastislav Královič, Richard Královič, and Tobias Mömke. On the Advice Complexity of Online Problems. In *Proceedings of the 20th International Symposium on Algorithms and Computation (ISAAC)*, pages 331–340, 2009.
7. 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.

8 Joan Boyar, Lene M Favrholdt, Christian Kudahl, and Jesper W Mikkelsen. Advice complexity for a class of online problems. In *32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

9 Joan Boyar, Shahin Kamali, Kim S. Larsen, and Alejandro López-Ortiz. On the List Update Problem with Advice. *Information and Computation*, 253:411–423, 2016.

10 Joan Boyar, Shahin Kamali, Kim S. Larsen, and Alejandro López-Ortiz. Online Bin Packing with Advice. *Algorithmica*, 74(1):507–527, 2016.

11 Marek Chrobak and Claire Kenyon. Competitiveness via Doubling. *SIGACT News*, pages 115–126, 2006.

12 Stefan Dobrev, Rastislav Královič, and Dana Pardubská. Measuring the problem-relevant information in input. *RAIRO Inform. Theor. Appl.*, 43(3):585–613, 2009.

13 Yuval Emek, Pierre Fraigniaud, Amos Korman, and Adi Rosén. Online computation with advice. *Theoret. Comput. Sci.*, 412(24):2642–2656, 2011.

14 Sandy Irani. Two results on the list update problem. *Inform. Process. Lett.*, 38:301–306, 1991.

15 David S. Johnson. *Near-optimal bin packing algorithms*. PhD thesis, MIT, Cambridge, MA, 1973.

16 Mikkelsen J.W. Randomization Can Be as Helpful as a Glimpse of the Future in Online Computation. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 39:1–39:14, 2016.

17 Shahin Kamali and Alejandro López-Ortiz. Better Compression through Better List Update Algorithms. In *Data Compression Conference*, pages 372–381, 2014.

18 Anna Karlin, Mark Manasse, Larry Rudolph, and Daniel Sleator. Competitive snoopy caching. *Algorithmica*, 3:79–119, 1988.

19 Anna R Karlin, Claire Kenyon, and Dana Randall. Dynamic TCP Acknowledgment and Other Stories about e/(e- 1). *Algorithmica*, 36:209–224, 2003.

20 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.

21 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.

22 Adam Meyerson. The Parking Permit Problem. In *46th Annual IEEE Symposium on Foundations of Computer Science*, pages 274–284. IEEE, 2005.

23 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.

24 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. doi:10.1016/j.tcs.2015.07.050.

25 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.

26 Daniel Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28:202–208, 1985.

27 Daniel D. Sleator and Robert Endre Tarjan. Self-adjusting Binary Search Trees. *Jour. of the ACM*, 32:652–686, 1985.