# Advice complexity of priority algorithms

Borodin, Allan; Boyar, Joan; Larsen, Kim S.; Pankratov, Denis

*Published in:*
Approximation and Online Algorithms - 16th International Workshop, WAOA 2018, Revised Selected Papers

*DOI:*
10.1007/978-3-030-04693-4_5

*Publication date:*
2018

*Document version:*
Accepted manuscript

*Citation for pulished version (APA):*
Borodin, A., Boyar, J., Larsen, K. S., & Pankratov, D. (2018). Advice complexity of priority algorithms. In L. Epstein, & T. Erlebach (Eds.), *Approximation and Online Algorithms - 16th International Workshop, WAOA 2018, Revised Selected Papers* (pp. 69-86). Springer VS. Lecture Notes in Computer Science Vol. 11312
https://doi.org/10.1007/978-3-030-04693-4_5

# Advice Complexity of Priority Algorithms*

Allan Borodin[1], Joan Boyar[2], Kim S. Larsen[2], and Denis Pankratov[3]

[1] University of Toronto, Toronto, Canada,
bor@cs.toronto.edu
[2] University of Southern Denmark, Odense, Denmark,
{joan,kslarsen}@imada.sdu.dk
[3] Concordia University, Montreal, Canada,
denis.pankratov@concordia.ca

**Abstract.** The priority model of "greedy-like" algorithms was introduced by Borodin, Nielsen, and Rackoff in 2002. We augment this model by allowing priority algorithms to have access to advice, i.e., side information precomputed by an all-powerful oracle. Obtaining lower bounds in the priority model without advice can be challenging and may involve intricate adversary arguments. Since the priority model with advice is even more powerful, obtaining lower bounds presents additional difficulties. We sidestep these difficulties by developing a general framework of reductions which makes lower-bound proofs relatively straightforward and routine. We start by introducing the Pair Matching problem, for which we are able to prove strong lower bounds in the priority model with advice. We develop a template for constructing a reduction from Pair Matching to other problems in the priority model with advice – this part is technically challenging since the reduction needs to define a valid priority function for Pair Matching while respecting the priority function for the other problem. Finally, we apply the template to obtain lower bounds for a number of standard discrete optimization problems.

## 1 Introduction

Greedy algorithms are among the first class of algorithms studied in an undergraduate computer science curriculum. They are among the simplest and fastest algorithms for a given optimization problem, often achieving a reasonably good approximation ratio, even when the problem is NP-hard. In spite of their importance, the notion of a greedy algorithm is not well defined. This might be satisfactory for studying upper bounds; when an algorithm is suggested, it does not matter much whether everyone agrees that it is greedy or not. However, lower bounds (inapproximation results) require a precise definition. Perhaps giving a precise definition for all greedy algorithms is not possible, since one can provide examples that seem to be outside the scope of the given model.

* The full version of the paper is available on arXiv [6]. For the first author, research is supported by NSERC. The second and third authors were supported in part by the Independent Research Fund Denmark, Natural Sciences, grant DFF-7014-00041.

Setting this philosophical question aside, we follow the model of greedy-like algorithms due to Borodin, Nielsen, and Rackoff [9]. The *fixed priority model* captures the observation that many greedy algorithms work by first sorting the input items according to some priority function, and then, during a single pass over the sorted input, making online irrevocable decisions for each input item. This model is similar to the online algorithm model with an additional preprocessing step of sorting inputs. Of course, if any sorting function is allowed, this would trivialize the model for most applications. Instead, a total ordering on the universe of all possible input items is specified before any input is seen, and the sorting is done according to this ordering, after which the algorithm proceeds as an online algorithm. This model has been adopted with respect to a broad array of topics [19, 2, 16, 13, 18, 5, 8, 3]. In spite of its appeal, there are relatively few lower bounds in this model. There does not seem to be a general method for proving lower bounds; that is, the adversary arguments tend to be ad-hoc. In addition, the basic priority model does not capture the notion of side information. The assumption that an algorithm does not know anything about the input is quite pessimistic in practice. This issue has been addressed recently in the area of online algorithms by considering models with advice (see [10] for an overview). In these models, side information, such as the number of input items or a maximum weight of an item, is computed by an all powerful oracle and is available to an algorithm before seeing any of the input. This information is then used to make better online decisions. The goal is to study trade-offs between advice length and the competitive ratio.

We introduce a general technique for establishing lower bounds on priority algorithms with advice. These algorithms are a simultaneous generalization of priority algorithms and online algorithms with advice. Our technique is inspired by the recent success of the binary string guessing problem and reductions in the area of online algorithms with advice. We identify a difficult problem (Pair Matching) that can be thought of as a sorting-resistant version of the binary string guessing problem. Then, we describe the template of gadget reductions from Pair Matching to other problems in the world of priority algorithms with advice. This part turns out to be challenging, mostly because one has to ensure that priorities are respected by the reduction. We then apply the template to a number of classic optimization problems. We restrict our attention to the fixed priority model. We also note that we consider deterministic algorithms unless otherwise specified.

### Related Model

Fixed priority algorithms with advice can be viewed in terms of the fixed priority backtracking model of Alekhnovich et al [1]. That model starts by ordering the inputs using a fixed priority function and then executes a computation tree where different decisions can be tried for the same input item by branching in the tree, and then choosing the best result. The lower bound results generally consider how much width (maximum number of nodes for any fixed depth in the tree) is necessary to obtain optimality where the width proven is often of

the form $2^{\Omega(n)}$. In contrast, our results give a parameterized trade-off between the number of advice bits and the approximation (competitive) ratio. However, given an algorithm in the fixed priority backtracking model, the logarithm of the width gives an upper bound on the number of bits of advice needed for the same approximation ratio. Similarly, a lower bound on the advice complexity gives a lower bound on width.

### Organization

We give a formal description of the models in Section 2. We motivate the study of the priority model with advice in Section 3. We introduce and analyze the Pair Matching problem in Section 4. We describe the reduction framework for obtaining lower bounds in Section 5 and apply it to classic problems in Section 6. We conclude in Section 7. Omitted proofs can all be found in the full version of the paper [6].

## 2 Preliminaries

We consider optimization problems for which we are given an objective function to minimize or maximize, and measure our success relative to an optimal offline algorithm.

### Online Algorithms with Advice

In an online setting, the input is revealed one item at a time by an adversary. An algorithm makes an irrevocable decision about the current item before the next item is revealed. For more background on online algorithms, we refer the reader to the texts by Borodin and El-Yaniv [7] and Komm [15].

The assumption that an online algorithm does not know anything about the input is quite often too pessimistic in practice. Depending on the application domain, the algorithm designer may have access to knowledge about the number of input items, the largest weight of an input item, some partial solution based on historical data, etc. The advice tape model for online algorithms captures the notion of side information in a purely information-theoretic way as follows. An all-powerful oracle that sees the entire input prepares the infinite advice tape with bits, which are available to the algorithm during the entire process. The oracle and the algorithm work in a cooperative mode – the oracle knows how the algorithm will use the bits and is trying to maximize the usefulness of the advice with regards to optimizing the given objective function. The advice complexity of an algorithm is a function of the input length and is the number of bits read by the algorithm in the worst case for inputs of a given size. For more background on online algorithms with advice, see the survey by Boyar et al. [10].

**Fixed Priority Model with Advice**

Fixed priority algorithms can be formulated as follows. Let $\mathcal{U}$ be a universe of all possible input items. An input to the problem consists of a finite set of items $\mathcal{I} \subset \mathcal{U}$ satisfying some consistency conditions. The algorithm specifies a total order on $\mathcal{U}$ before seeing the input. Then, a subset of the possible input items is revealed (by an adversary) according to the total order specified by the algorithm. The algorithm makes irrevocable decisions about the items as they arrive.[4] The overall set of decisions is then evaluated according to some objective function. The performance of the algorithm is measured by the asymptotic approximation ratio with respect to the value provided by an optimal offline algorithm. The notion of advice is added to the model as follows. After the algorithm has chosen a total order on $\mathcal{U}$, an all-powerful oracle that has access to the entire input $\mathcal{I}$ creates a tape of infinitely many bits. The algorithm knows how the advice bits are created and has access to them during the online decision phase. Our interest is in how many bits of advice the algorithm uses compared with the result it obtains.

We consider only countable universes $\mathcal{U}$. In this case, having a total order on elements in $\mathcal{U}$ is equivalent (via a simple inductive argument) to having a priority function $P : \mathcal{U} \to \mathbb{R}$. The assumption of the universe being countable is natural, but also necessary for the above equivalence: there are uncountably many totally ordered sets that do not embed into the reals with the standard order.

**Definition 1.** *Let $\mathcal{U}$ be the universe of input items and let $P : \mathcal{U} \to \mathbb{R}$ be a priority function. For $u_1, u_2 \in \mathcal{U}$, we write $u_1 <_P u_2$ to mean $P(u_1) < P(u_2)$. We will say that larger priority means that the item appears earlier in the input, i.e., $u_1 <_P u_2$ means that $u_2$ appears* before *$u_1$ when the input is given according to $P$.*

*Example.* Kruskal's optimal algorithm for the minimum spanning tree problem is a fixed priority algorithm without advice. The universe of items is $\mathcal{U} = \mathbb{N} \times \mathbb{N} \times \mathbb{Q}$. An item $(i, j, w) \in \mathcal{U}$ represents an edge between a vertex $i$ and a vertex $j$ of weight $w$. The consistency condition on the input is that the edge $\{i, j\}$ can be present at most once in the input. The total order on the universe is specified by all items of smaller weight having higher priority than all items of larger weight, breaking ties, say, by lexicographic order on the names of vertices. Kruskal's algorithm processes input items in the given order and greedily accepts those items that do not result in cycles.

In this paper, we only consider the following input model for graph problems in the priority setting:

*Vertex arrival, vertex adjacency*: an input item consists of a name of a vertex together with a set of names of adjacent vertices. There is a consistency condition on the entire input: if $u$ appears as a neighbor of $v$, then $v$ must appear as a neighbor of $u$.

[4] In the adaptive priority model, the algorithm is allowed to specify a new ordering depending on previous items and decisions before a new input item is presented.

### Binary String Guessing Problem

Later we introduce the Pair Matching problem that can be viewed as a priority model analogue of the following online binary string guessing problem.

**Definition 2.** *The Binary String Guessing Problem [4] with known history (2-SGKH) is the following online problem. The input consists of* $(n, \sigma = (x_1, \ldots, x_n))$*, where* $x_i \in \{0, 1\}$*. Upon seeing* $x_1, \ldots, x_{i-1}$ *an algorithm guesses the value of* $x_i$*. The actual value of* $x_i$ *is revealed after the guess. The goal is to maximize the number of correct guesses.*

Böckenhauer et al. [4] provide a trade-off between the number of advice bits and the approximation ratio for the binary string guessing problem.

**Theorem 1 (Böckenhauer et al. [4]).** *For the 2-SGKH problem and any* $\varepsilon \in (0, \frac{1}{2}]$*, no online algorithm reading fewer than* $(1 - H(\varepsilon))n$ *advice bits can make fewer than* $\varepsilon n$ *mistakes for large enough* $n$*, where* $H(p) = H(1-p) = -p\log(p) - (1-p)\log(1-p)$ *is the binary entropy function.*

### Competitive and Approximation Ratios

The performance of online algorithms is measured by their competitive ratios. For a minimization problem, an online algorithm ALG is said to be *c-competitive* if there exists a constant $\alpha$ such that for all input sequences $I$ we have $\mathrm{ALG}(I) \leq c\,\mathrm{OPT}(I) + \alpha$, where $\mathrm{ALG}(I)$ denotes the cost of the algorithm on $I$ and $\mathrm{OPT}(I)$ is the value achieved by an offline optimal algorithm. The infimum of all $c$ such that ALG is $c$-competitive is ALG's *competitive ratio*. For a maximization problem, $\mathrm{ALG}(I)$ is referred to as profit, and we require that $\mathrm{OPT}(I) \leq c\,\mathrm{ALG}(I) + \alpha$. In this way, we always have $c \geq 1$ and the closer $c$ is to 1, the better. Priority algorithms are thought of as approximation algorithms and the term (asymptotic) approximation ratio is used (but the definition is the same).

## 3 Motivation

In this section we present a motivating example for studying the priority model with advice. We present a problem that is difficult in the pure priority setting or in the online setting with advice, but easy in the priority model with advice. Furthermore, the advice is easily computed by an offline algorithm.

The problem of interest is called Greater Than Mean (GTM). In the GTM problem, the input is a sequence $x_1, \ldots, x_n$ of rational numbers. Let $m = \sum_i x_i/n$ denote the sample mean of the sequence. The goal of an algorithm is to decide for each $x_i$ whether $x_i$ is greater than the mean or not, answering 1 or 0, respectively. We can also assume that the length of the sequence $n$ is known to the algorithm in advance. We start by noting that there is a trivial optimal priority algorithm with little advice for this problem.

**Theorem 2.** *For Greater Than Mean, there exists a fixed priority algorithm reading at most* $\lceil \log n \rceil$ *advice bits, solving the problem optimally.*

*Proof.* The priority order is such that $x_1 \geq x_2 \ldots \geq x_n$. Thus, the numbers arrive in the order from largest to smallest. The advice specifies the earliest index $i \in [n]$ such that $x_i \leq m$. □

In the full version, we show that a priority algorithm without advice has to make many errors.[5]

**Theorem 3.** *For Greater Than Mean and any* $\varepsilon \in (0, \frac{1}{2}]$*, no fixed priority algorithm without advice can make fewer than* $(1/2 - \varepsilon)n$ *mistakes for large enough* $n$*.*

Finally, we show that an online algorithm requires a lot of advice to achieve good performance for the GTM problem. The proof is a minor modification of a reduction from 2-SGKH to the Binary Separation Problem (see [11] for details). We present the proof in its entirety for completeness.

**Algorithm 1** Reduction from 2-SGKH to GTM

```
procedure REDUCTION-2-SGKH-TO-GTM
    ℓ_1 ← 0, u_1 ← 1
    for i = 1 to n do
        y_i ← (ℓ_i + u_i)/2
        if A predicts y_i is greater than mean then
            predict x_i = 1
        else
            predict x_i = 0
        receive actual x_i
        if actual x_i = 1 then
            u_{i+1} ← y_i, ℓ_{i+1} ← ℓ_i
        else
            u_{i+1} ← u_i, ℓ_{i+1} ← y_i
    y_{n+1} ← ((n+1)/2)(ℓ_{n+1} + u_{n+1}) − Σ_{i=1}^{n} y_i
```

**Theorem 4.** *For the Greater Than Mean problem and any* $\varepsilon \in (0, \frac{1}{2}]$*, no online algorithm reading fewer than* $(1 - H(\varepsilon))(n-1)$ *advice bits can make fewer than* $\varepsilon n$ *mistakes for large enough* $n$*.*

*Proof.* We present a reduction from the 2-SGKH problem to the GTM problem. Let $A$ be an online algorithm with advice for the GTM problem. Our reduction

[5] In Theorem 3 and in all of our lower bound advice results, we state the result so as to include $\varepsilon = \frac{1}{2}$, in which case the conditions "fewer than $(1/2 - \varepsilon)$" and "fewer than $(1 - H(\varepsilon))$" make the statements vacuously true.

is presented in Algorithm 1. In the course of the reduction, an online input $x_1, \ldots, x_n$ of length $n$ for the 2-SGKH problem is converted into an online input $y_1, \ldots, y_{n+1}$ of length $n+1$ for the GTM problem with the following properties: The reduction is advice-preserving (the number of advice bits is the same) and for each $i \in [n]$, our algorithm $A$ for 2-SGKH makes a mistake on $x_i$ if and only if $A$ makes a mistake on $y_i$. This would finish the proof of the theorem.

Let $S = \{i \in [n] \mid x_i = 1\}$ and $T = [n] \setminus S$. The reduction uses a technique similar to binary search to make sure that $\forall i \in S$ and $\forall j \in T$ we have $y_i > y_j$, i.e., all the $y_i$ corresponding to $x_i = 1$ are larger than all the $y_j$ corresponding to $x_j = 0$. Then $y_{n+1}$ is chosen to make sure that the mean of the entire stream $y_1, \ldots, y_{n+1}$ lies between the smallest $y_i$ with $i \in S$ and the largest $y_j$ with $j \in T$. This implies that $y_i$ is greater than the mean if and only if the corresponding $x_i = 1$.

The following invariants are easy to see and are left to the reader: (1) $u_i > \ell_i$; (2) if $x_i = 1$, then $u_i > y_i \geq u_{i+1}$; (3) if $x_i = 0$, then $\ell_i < y_i \leq \ell_{i+1}$.

The required properties of the reduction follow immediately from the invariants. Let $i \in S$ and $j \in T$. Then, $y_i \geq u_{n+1} > \ell_{n+1} \geq y_j$. Finally, observe that $y_{n+1}$ is chosen so that the mean is $\sum_{i=1}^{n+1} y_i/(n+1) = \sum_{i=1}^{n} y_i/(n+1) + y_{n+1}/(n+1) = (1/2)(\ell_{n+1} + u_{n+1})$. This mean correctly separates $S$ from $T$. □

## 4 Pair Matching Problem

We introduce an online problem called Pair Matching.[6] The input consists of a sequence of $n$ distinct rational numbers between 0 and 1, i.e., $x_1, \ldots, x_n \in \mathbb{Q} \cap [0, 1]$. After the arrival of $x_i$, an algorithm has to answer if there is a $j \in [n] \setminus \{i\}$ such that $x_i + x_j = 1$, in which case we refer to $x_i$ and $x_j$ as forming a pair and say that $x_i$ has a matching value, $x_j$. The answer "accept" is correct if $x_j$ exists, and "reject" is correct if it does not. Note that since the $x_i$ are all distinct, if $x_i = \frac{1}{2}$, the correct answer is "reject", since $\frac{1}{2}$ cannot have a matching value.

We let $pairs(x_1, \ldots, x_n)$ denote the number of pairs in the input $x_1, \ldots, x_n$.

### 4.1 Online Setting

Analyzing Pair Matching in the online setting is relatively straightforward for both deterministic and randomized algorithms.

**Theorem 5.** *For Pair Matching, there exists a* 2*-competitive algorithm, answering correctly on* $n - pairs(x_1, \ldots, x_n)$ *input items.*

**Theorem 6.** *For Pair Matching, no deterministic online algorithm can achieve a competitive ratio less than* 2*.*

[6] There are similarities to the NP-Complete problems, Numerical Matching with Target Sums and Numerical 3-Dimensional Matching, though these problems ask if permutations of sets of inputs will lead to a complete matching.

**Theorem 7.** *For Pair Matching, there exists a randomized online algorithm that in expectation answers correctly on* $2n/3$ *input items.*

**Theorem 8.** *For Pair Matching, no randomized online algorithm can achieve a competitive ratio less than* $3/2$*.*

Lastly, we prove that online algorithms need a lot of advice in order to start approaching a competitive ratio of 1 for Pair Matching.

**Theorem 9.** *For Pair Matching and any* $\varepsilon \in (0, \frac{1}{2}]$*, no deterministic online algorithm reading fewer than* $(1 - H(\varepsilon))n/2$ *advice bits can make fewer than* $\varepsilon n$ *mistakes for large enough* $n$*.*

### 4.2 Priority Setting

In this section, we show that Theorem 9 also holds in the priority setting. The proof becomes a bit more subtle, so we give it in full detail.

**Theorem 10.** *For Pair Matching and any* $\varepsilon \in (0, \frac{1}{2}]$*, no fixed priority algorithm reading fewer than* $(1 - H(\varepsilon))n/2$ *advice bits can make fewer than* $\varepsilon n$ *mistakes for large enough* $n$*.*

*Proof.* We prove the statement by a reduction from the *online problem* 2-SGKH. Let $A$ be a priority algorithm solving Pair Matching, and let $P$ be the corresponding priority function. (Note that we assume that the algorithm knows $P$; this is the case in all of our priority algorithm reductions.) The reduction follows the proof of Theorem 9 closely. The idea is to transform the online input to 2-SGKH into an input to Pair Matching. The difficulty arises from having to present the transformed input in the online fashion while respecting the priority function $P$.

Let $x_1, \ldots, x_n$ be the input to 2-SGKH. The online reduction works as follows. The online algorithm picks $n$ distinct numbers $y_1, \ldots, y_n$ from $[0, 1]$ and creates a list $z_1, \ldots, z_{2n}$ consisting of $y_i$ and $1 - y_i$ sorted according to $P$. The algorithm keeps a (max-heap ordered) priority queue $Q$ of elements from $z_i$ as well as a subsequence $Z$ of $z_1, \ldots, z_{2n}$. The reduction always picks the first element $z$ from $Z$. We maintain the invariant that $1 - z$ appears later in $Z$ according to $P$. If needed, the reduction algorithm will insert $1 - z$ into $Q$ to be simulated as an input to $A$ at the right time later on.

*Initialization.* Initially, $Q$ is empty and $Z$ is the entire sequence $z_1, \ldots, z_{2n}$. Before the element $x_1$ arrives, the algorithm feeds $z_1$ to $A$. If $A$ answers that $z_1$ is a part of a pair, then the online algorithm predicts $x_1 = 1$; otherwise the algorithm predicts $x_1 = 0$. Then the online algorithm finds $j$ such that $z_j = 1 - z_1$ and updates $Z$ by deleting $z_1$ and $z_j$. Then $x_1$ is revealed. If the actual value of $x_1$ is 1, the algorithm inserts $z_j$ into $Q$; otherwise the algorithm does not modify $Q$.

*Middle step.* Suppose that the algorithm has processed $x_1, \ldots, x_{i-1}$ and has to guess the value of $x_i$. The algorithm picks the first element $z$ from the subsequence $Z$. While the top element of $Q$ has higher priority than $z$ according to

$P$, the algorithm deletes that element from the priority queue and feeds it to $A$. Then, the algorithm feeds $z$ to $A$. The next steps are similar to the initialization case. If $A$ answers that $z$ is a part of a pair, then the online algorithm predicts $x_i = 1$; otherwise the algorithm predicts $x_i = 0$. The online algorithm finds $z'$ in $Z$ such that $z = 1 - z'$, and updates $Z$ by deleting $z$ and $z'$. Then $x_i$ is revealed. If the actual value of $x_i$ is 1, the algorithm inserts $z'$ into $Q$; otherwise the algorithm does not modify $Q$.

*Post-processing.* After the algorithm finishes processing $x_n$, it feeds the remaining elements (in priority order) from $Q$ to $A$.

It is easy to see that the online algorithm feeds a subsequence of $z_1, \ldots, z_{2n}$ to $A$ in the correct order according to $P$. In addition, the online algorithm makes exactly the same number of mistakes as $A$ (assuming that $A$ always answers correctly on the second element of a pair). The statement of the theorem follows since the size of the input to $A$ is at most $2n$. □

## 5 Reduction Template

Our template is restricted to binary decision problems since the goal is to derive inapproximations based on the Pair Matching problem. In reducing from Pair Matching to a problem $B$, we assume that we have a priority algorithm ALG with advice for problem $B$ with priorities defined by $P$. Based on ALG and $P$, we define a priority algorithm ALG$'$ with advice and a priority function, $P'$, for the Pair Matching problem. Input items $x_1, x_2, \ldots, x_n$ in $\mathbb{Q} \cap [0, 1]$ to Pair Matching arrive in an order specified by the priority function we define, based on $P$. We assume that we are informed when the input ends and can take steps at that point to complete our computation. Knowing the size $n$ of the input, which one naturally would in many situations after the initial sorting according to $P'$, would of course be sufficient.

Based on the input to the Pair Matching problem, we create input items to problem $B$, and they have to be presented to ALG, respecting the priority function $P$. Responses from ALG are then used by ALG$'$ to help it answer "accept" or "reject" for its current $x_i$. Actually, ALG will always answer correctly for a request $x_j = 1 - x_i$ when $i < j$, so the responses from ALG are only used when this is not the case. The main challenge is to ensure that the input items to ALG are presented in the order determined by $P$, because the decision as to whether or not they are presented needs to be made in time, without knowing whether or not the matching value will arrive.

Here, we give a high level description of a specific kind of gadget reduction. A gadget $G$ for problem $B$ is simply some constant-sized instance for $B$, i.e., a collection of input items that satisfy the consistency condition for problem $B$. For example, if $B$ is a graph problem in the vertex arrival, vertex adjacency model, $G$ could be a constant-sized graph, and the universe then contains all possible pairs of the form: a vertex name coupled with a list of possible neighboring vertex names. Note that each possible vertex name exists many times as a part of an input, because it can be coupled with many different possible lists of vertex

names. The consistency condition must apply to the actual input chosen, so for each vertex name $u$ which is listed as a neighbor of $v$, it must be the case that $v$ is listed as a neighbor of $u$.

The gadgets used in a reduction will be created in pairs (gadgets in a pair may be isomorphic to each other, so that they are the same up to renaming), one pair for each input item less than or equal to $1/2$ (for $x = 1/2$, the gadget will only be used to assign a priority to $x = 1/2$). One gadget from the pair is presented to ALG when $1 - x$ appears later in the input; and the other gadget when it does not. Using fresh names in the input items for problem $B$, we ensure that each input item less than $\frac{1}{2}$ to the Pair Matching problem has its own collection of input items for its gadgets for problem $B$. The pair of gadgets associated with an input item $x \leq 1/2$ can be written $(G_x^1, G_x^2)$. The same universe of input items is used for both of these gadgets.

We write $\max_P G$ to denote the first item according to $P$ from the universe of input items for $G$, i.e., the highest priority item. For now, assume that ALG responds "accept" or "reject" to any possible input item. This captures problems such as vertex cover, independent set, clique, etc.

For each $x \leq 1/2$, the gadget pair satisfies two conditions: the first item condition, and the distinguishing decision condition. The *first item condition* says that the first input item $m_1(x)$ according to $P$ gives no information about which gadget it is in. To accomplish this, we define the priority function for ALG$'$ as $P'(x) = P(\max_P G_x^1)$ for all $x \leq 1/2$ and set $m_1(x) = \max_P G_x^1 = \max_P G_x^2$ (the second equality holds since we assume the two gadgets have the same input universe). The *distinguishing decision condition* says that the decision with regards to item $m_1(x)$ that results in the optimal value of the objective function in $G_x^1$ is different from the decision that results in the optimal value of the objective function in $G_x^2$. This explains why the one gadget is presented to ALG when $1 - x$ appears later in the input sequence and the other when it does not.

Now that the first item of the gadget associated with $x$ is defined, the remaining actual input items in the gadget pair for $x$ must be completely defined according to the distinguishing decision condition. This gives two sets (overlapping, at least in $m_1(x)$) of input items. The item with highest priority among all of the items in the actual gadget pair, ignoring $m_1(x)$, is called $m_2(x)$, and we define $P'(1 - x) = P(m_2(x))$ for $x < 1/2$. Thus, we guarantee the following list of properties: $x < 1/2$ will arrive before $1 - x$ in the input sequence for Pair Matching for ALG$'$, $m_1(x)$ will arrive for algorithm ALG at the same time, ALG's response for $m_1(x)$ can define the response of ALG$'$ to $x$, and the decision as to which gadget in the pair is presented for $x$ can be made at the time $1 - x$ arrives or ALG$'$ can determine that it will not arrive (because either the input sequence ended or an $x'$ with lower priority than $1 - x$ arrived).

To warm up, we start with an example reduction from Pair Matching to Triangle Finding; a somewhat artificial problem in this context, but well-studied in streaming algorithms [17], for instance. This reduction then serves as a model for the general reduction template.

### 5.1 Example: Triangle Finding

Consider the following priority problem in the vertex arrival, vertex adjacency model: for each vertex $v$, decide whether or not $v$ belongs to some triangle (a cycle of length 3) in the entire input graph. The answer "accept" is correct if $v$ belongs to some triangle, and otherwise the answer should be "reject". We refer to this problem as Triangle Finding. This problem might look artificial and it is optimally solvable offline in time $O(n^2)$, but as mentioned above, advice-preserving reductions between priority problems require subtle manipulations of a priority function. The Triangle Finding problem allows us to highlight this issue in a relatively simple setting.

**Theorem 11.** *For Triangle Finding and any $\varepsilon \in (0, \frac{1}{2}]$, no fixed priority algorithm reading at most $(1 - H(\varepsilon))n/8$ advice bits can make fewer than $\varepsilon n/4$ mistakes.*

*Proof.* We prove this theorem by a reduction from the Pair Matching problem. Let ALG be an algorithm for the Triangle Finding problem, and let $P$ be the corresponding priority function. Let $x_1, \ldots, x_n$ be the input to Pair Matching. We define a priority function $P'$ and a valid input sequence $v_1, \ldots, v_m$ to Triangle Finding. When $x_1, \ldots, x_n$ is presented according to $P'$ to our priority algorithm for Pair Matching, it is able to construct $v_1, \ldots, v_m$ for ALG, respecting the priority function $P$. Moreover, our algorithm for Pair Matching will be able to use answers of ALG to answer the queries about $x_1, \ldots, x_n$.

Now, we discuss how to define $P'$. With each number $x \in \mathbb{Q} \cap [0, 1/2]$, we associate four unique vertices $v_x^1, v_x^2, v_x^3, v_x^4$. The universe consists of all input items of the form $(v_x^i, \{v_x^j, v_x^k\})$ with $i, j, k \in [4]$, $i \notin \{j, k\}$ and $j < k$; there are 12 input items for each $x$: 4 possibilities for the vertex, and for each of the $\binom{3}{2} = 3$ possibilities for the ordered pair of neighbors. Let $m_1(x)$ be the first item according to $P$ among the 12 items. Using only the input items from the 12 items we are currently considering, we extend this item in two ways, to a 3-cycle $C_x^3$ and to a 4-cycle $C_x^4$. When we write $C_x^3$ or $C_x^4$, we mean the set of items forming the 3-cycle or 4-cycle, respectively. Now, $P'$ is defined as follows:

$$P'(x) = \begin{cases} P(m_1(x)), & \text{if } x \leq 1/2 \\ \max_{g \in (C_{1-x}^3 \cup C_{1-x}^4) \setminus \{m_1(1-x)\}} P(g), & \text{otherwise} \end{cases}$$

In other words, if $x > 1/2$, we set $P'(x)$ to be the first element other than $m_1(1-x)$ in $C_{1-x}^3 \cup C_{1-x}^4$. In terms of our high level description given at the beginning of this section, $(C_x^3, C_x^4)$ form the pair of gadgets – a triangle and a square (4-cycle). By construction, this pair of gadgets satisfies the first item condition. By the definition of the problem, the optimal decision for all vertices in $C_x^3$ is "accept" (belongs to a triangle) and the optimal decision for all vertices in $C_x^4$ is "reject" (does not belong to a triangle). Thus, these gadgets also satisfy the distinguishing decision condition.

Let $x_1, \ldots, x_n$ denote the order input items are presented to our algorithm as specified by $P'$. Our algorithm constructs an input to ALG which is consistent

with $P$ along the following lines: for each $x \leq 1/2$ that appears in the input, the algorithm constructs either a three-cycle or a four-cycle (disjoint from the rest of the graph). Thus, each $x \leq 1/2$ is associated with one connected component. During the course of the algorithm, each connected component will be in one of the following three states: undecided, committed, or finished. When $x \leq 1/2$ arrives, the algorithm initializes the construction with the item $m_1(x)$ and sets the component status to undecided. It answers "accept" (there will be a matching pair) for $x$ if ALG responds "accept" (triangle) for $m_1(x)$, and it answers "reject" if ALG responds "reject" (square).

Note that for any $x \leq 1/2$, $P'(x) > P'(1-x)$, so if $x' > 1/2$ arrives and $1 - x'$ has not appeared earlier, ALG$'$ can simply reject $x'$ and does not need to present anything to ALG. If $x$ has arrived and at some point, $1 - x$ arrives, the algorithm commits to constructing the 3-cycle $C_x^3$. If ALG$'$ had guessed correctly that $1 - x$ would arrive, it is because ALG responded "accept" for $m_1(x)$) and also guessed correctly. If ALG$'$ had guessed that $1 - x$ would not arrive, it is because ALG guessed that a square would arrive, and both guessed incorrectly. If some $x'$ arrives with $P'(x') < P'(1 - x)$ for some $x \neq x'$ and $x$ has arrived earlier, then ALG$'$ can be certain that $1 - x$ will not arrive. It commits to constructing the 4-cycle $C_x^4$. Thus, if ALG$'$ answered "reject" for $x$, it answered correctly, and a square makes ALG's decision for $m_1(x)$ correct. Similarly, if ALG$'$ answered "accept" for $x$, it answered incorrectly, so a square makes ALG's decision incorrect.

At the end of the input, ALG$'$ finishes off by checking which values of $x$ have arrived without $1-x$ arriving or some $x'$ with higher priority than $1-x$ arriving, and ALG again commits to the 4-cycle, as in the other case where $1 - x$ does not arrive.

Throughout the algorithm, there are several connected components, each of which can be undecided, committed, or finished. Note that an undecided component corresponding to input $x$ consists of a single item $m_1(x)$. Upon receiving an item $y$, the algorithm first checks whether some undecided components have turned into committed ones: namely if an undecided component consisting of $m_1(x)$ satisfies $P'(1 - x) > P'(y)$, it switches the status to a committed component according to the rules described above. Then, the algorithm feeds input items corresponding to committed yet unfinished connected components to ALG and does so in the order of $P$ up until the priority of such items falls below $P'(y)$ (this can be done by maintaining a priority queue). Finally, the algorithm processes the item $y$ by either creating a new component or by turning an undecided component into a decided one. Then, the algorithm moves to the next item. Due to our definition of $P'$ and this entire process, the input constructed for ALG is valid and consistent with $P$. Observe that the input to ALG$'$ is of size at most $4n$, so the number of advice bits must be divided by four relative to Theorem 10, and the theorem follows. □

### 5.2 General Template

In this subsection, we establish two theorems that give general templates for gadget reductions from Pair Matching – one for maximization problems and one for minimization problems. A high level overview is given at the beginning of this section.

We let $\mathrm{ALG}(I)$ denote the objective function for ALG on input $I$. The *size* of a gadget $G$, denoted by $|G|$, is the number of input items specifying the gadget. We write $\mathrm{OPT}(G)$ to denote the best value of the objective function on $G$. Recall that we focus on problems where a solution is specified by making an accept/reject decision for each input item. We write $\mathrm{BAD}(G)$ to denote the best value of the objective function attainable on $G$ after making the wrong decision for the first item (the item with highest priority, $\max(G)$), i.e., if there is an optimal solution that accepts (rejects) the first item of $G$, then $\mathrm{BAD}(G)$ denotes the best value of the objective function *given* that the first item was rejected (accepted). We say that the objective function for a problem $B$ is *additive*, if for any two instances $I_1$ and $I_2$ to $B$ such that $I_1 \cap I_2 = \emptyset$, we have $\mathrm{OPT}(I_1 \cup I_2) = \mathrm{OPT}(I_1) + \mathrm{OPT}(I_2)$.

**Theorem 12.** *Let $B$ be a minimization problem with an additive objective function. Let* ALG *be a fixed priority algorithm with advice for $B$ with a priority function $P$. Suppose that for each $x \in \mathbb{Q} \cap [0, 1/2]$ one can construct a pair of gadgets $(G_x^1, G_x^2)$ satisfying the following conditions:*

**The first item condition:** $m_1(x) = \max_P G_x^1 = \max_P G_x^2$.
**The distinguishing decision condition:** *the optimal decision for $m_1(x)$ in $G_x^1$ is different from the optimal decision for $m_1(x)$ in $G_x^2$ (in particular, the optimal decision is unique for each gadget). Without loss of generality, we assume $m_1(x)$ is accepted in an optimal solution in $G_x^1$.*
**The size condition:** *the gadgets have finite sizes; let $s = \max_x(|G_x^1|, |G_x^2|)$, where the cardinality of a gadget is the number of input items it consists of.*
**The disjoint copies condition:** *for $x \neq y$ and $i, j \in \{1, 2\}$, input items making up $G_x^i$ and $G_y^j$ are disjoint.*
**The optimal/bad condition:** *the values $\mathrm{OPT}(G_x^1)$, $\mathrm{BAD}(G_x^1)$ and $\mathrm{OPT}(G_x^2)$, $\mathrm{BAD}(G_x^2)$ are independent of $x$, and we denote them by $\mathrm{OPT}(G^1)$, $\mathrm{BAD}(G^1)$, $\mathrm{OPT}(G^2)$, and $\mathrm{BAD}(G^2)$; we assume that $\mathrm{OPT}(G^2) \geq \mathrm{OPT}(G^1)$.*

*Define $r = \min\left\{\frac{\mathrm{BAD}(G^1)}{\mathrm{OPT}(G^1)}, \frac{\mathrm{BAD}(G^2)}{\mathrm{OPT}(G^2)}\right\}$. Then for any $\varepsilon \in (0, \frac{1}{2}]$, no fixed priority algorithm reading fewer than $(1 - H(\varepsilon))n/(2s)$ advice bits can achieve an approximation ratio smaller than*

$$1 + \frac{\varepsilon(r-1)\,\mathrm{OPT}(G^1)}{\varepsilon\,\mathrm{OPT}(G^1) + (1-\varepsilon)\,\mathrm{OPT}(G^2)}.$$

The following theorem is for maximization problems.

**Theorem 13.** *Let $B$ be a maximization problem with an additive objective function. Let* ALG *be a fixed priority algorithm with advice for $B$ with a priority*

*function $P$. Suppose that for each $x \in \mathbb{Q} \cap [0, 1/2]$ one can construct a pair of gadgets $(G_x^1, G_x^2)$ satisfying the conditions in Theorem 12. Then for any $\varepsilon \in (0, \frac{1}{2}]$, no fixed priority algorithm reading fewer than $(1 - H(\varepsilon))n/(2s)$ advice bits can achieve an approximation ratio smaller than*

$$1 + \frac{\varepsilon(r-1)\,\mathrm{OPT}(G^1)}{\varepsilon\,\mathrm{OPT}(G^1) + (1-\varepsilon)r\,\mathrm{OPT}(G^2)},$$

*where* $r = \min\left\{\frac{\mathrm{OPT}(G^1)}{\mathrm{BAD}(G^1)}, \frac{\mathrm{OPT}(G^2)}{\mathrm{BAD}(G^2)}\right\}$.

We mostly use Theorems 12 and 13 in the following specialized form.

**Corollary 1.** *With the set-up from Theorems 12 and 13, we have the following:*

*For a minimization problem, if* $\mathrm{OPT}(G^1) = \mathrm{OPT}(G^2) = \mathrm{BAD}(G^1) - 1 = \mathrm{BAD}(G^2) - 1$*, then no fixed priority algorithm reading fewer than* $(1-H(\varepsilon))n/(2s)$ *advice bits can achieve an approximation ratio smaller than* $1 + \frac{\varepsilon}{\mathrm{OPT}(G^1)}$.

*For a maximization problem, if* $\mathrm{OPT}(G^1) = \mathrm{OPT}(G^2) = \mathrm{BAD}(G^1) + 1 = \mathrm{BAD}(G^2) + 1$*, then no fixed priority algorithm reading fewer than* $(1-H(\varepsilon))n/(2s)$ *advice bits can achieve an approximation ratio smaller than* $1 + \frac{\varepsilon}{\mathrm{OPT}(G^1)-\varepsilon}$.

Next, we describe a general procedure for constructing gadgets with the above properties. For simplicity, we do it for graph problems in the vertex arrival, vertex adjacency input model. Later we discuss what is required to carry out such general constructions for other combinatorial problems. In the case of graphs, an input item consists of a vertex name with the names of neighbors of that vertex. First, consider defining a single gadget instead of a pair. We define a gadget in several steps. As the first step, we define a graph $G = \left([n], E \subset \binom{[n]}{2}\right)$ over $n$ vertices. Then, when defining a gadget based on input $x$ to Pair Matching, we pick $n$ vertex names $V_x$ and give a bijection $f\colon V_x \to [n]$. Finally, we read off the resulting input items in the order given by the priority function. Thus, we think of $G$ as giving a topological structure of the instance, and it is converted into an actual instance by assigning new names to the vertices. The reason that the names from the topological structure are not used directly is that we want to define a separate gadget instance for each $x \in \mathbb{Q} \cap [0, 1/2]$. Thus, all gadgets instances are going to have the same topological structure,[7] but will differ in names of vertices.

For graphs in the vertex arrival, vertex adjacency model, we say that two input items are isomorphic if they have the same number of neighbors, i.e., they differ in just the names of the vertices and the names of their neighbors. A topological structure $G$ consisting only of isomorphic items is a regular graph. For any priority function $P$ and any vertex $v \in [n]$, we can force the corresponding item to appear first according to $P$ by naming vertices appropriately. Fix $x$ and consider all possible input items that can be formed from $V_x$ consistently with

[7] However, both gadgets within a pair do not necessarily have the same topological structure. In Triangle Finding, they did not.

$G$. One of those items appears first according to $P$. Define a bijection $f$ by first mapping that first item to $u$ and its neighbors in $G$, and extending this one-to-one correspondence to other vertices in $G$ in an arbitrary, consistent manner. In this case, the input item corresponding to $u$ would appear first according to $P$ in the input to the graph problem. Because all items are isomorphic, it is always possible to extend the bijection to all of $G$.

Now, suppose that two topological structures $G^1 = ([n], E^1)$ and $G^2 = ([m], E^2)$ consist only of isomorphic items. Using a similar idea, for each priority function $P$, each $x \in \mathbb{Q} \cap [0, 1/2)$, each $u \in [n]$, and each $v \in [m]$, one can assign names to vertices of $G^1$ and $G^2$ such that the first input item according to $P$ is associated with $u$ in $G^1$ and the same item is associated with $v$ in $G^2$. In particular, this means that as long as the two topological structures are regular, we can always convert them into gadgets satisfying the first item condition.

Suppose that there is a vertex $u$ in $G^1$ that appears in every optimal solution in $G^1$, i.e., a "reject" decision leads to non-optimality. Furthermore, suppose that there is a vertex $v$ in $G^2$ that is excluded from every optimal solution in $G^2$, i.e., an "accept" decision leads to non-optimality. Then for each $x$, using the above construction, we can make the first item according to $P$ be associated with $u$ in $G^1$ and with $v$ in $G^2$. This means that we can always convert the topological structures into gadgets satisfying the distinguishing decision condition. Finally, observe that the size condition is satisfied with $s = \max(|G^1|, |G^2|)$.

This gadget construction can be carried out in other input models. We need to have a notion of isomorphism between input items, and a notion of the topological structure of a gadget. Once we have the two notions, if we find topological structures consisting only of isomorphic items with uniquely identifiable "reject"/"accept" items in all optimal solutions, then we immediately conclude that the problem requires the trade-off between advice and approximation ratio as outlined in Theorems 12 and 13 and Corollary 1 with parameter $s$ equal to the size of the topological template.

We finish this section by remarking that one can perform similar reductions with gadgets where not all input items are isomorphic. Theorem 17, which is based on a lower bound construction from [5], is proven via a reduction for Vertex Cover using *two* gadget pairs with some vertices of degree 2 and others of degree 3. One simply needs that there is one gadget pair for the case where a vertex of degree 2 has the highest priority and another gadget pair for the case where a vertex of degree 3 has highest priority. For both gadget pairs, $s = 7$, the optimal value is 3, and the minimum possible objective value for the gadget in the pair is 4. Thus, the results of Theorem 12 (or Theorem 13 if it was a mazimization problem) and Corollary 1 can be applied. This idea can be extended to other input models where the gadgets have input items which are not isomorphic. For simplicity, we do not restate the two theorems or the corollary for the extension where there are $t$ different classes of isomorphic input items and thus $t$ pairs of gadgets.

## 6 Reductions to Classic Optimization Problems

In this section, we provide one detailed example of an application of the general reduction template, plus statement of results for other problems. With the exception of bipartite matching, all of these problems are NP-hard, as a consequence of the NP-completenes of their underlying decision problems, as established in the seminal papers by Cook [12] and Karp [14]. Furthermore, these problems are known to have various hardness of approximation bounds.

### 6.1 Detailed Example: Independent Set

We consider the maximum independent set problem in the vertex arrival, vertex adjacency input model. Consider the topological structure of a gadget in Figure 1. There are 5 vertices on the top and 3 vertices on the bottom. All top vertices are connected to all bottom vertices. Additionally, the 5 vertices on the top form a cycle. In this way, each vertex has degree 5 and hence all the input items are isomorphic. If we pick any vertex from the top to be in the independent set, then we forgo all the bottom vertices, and we are essentially restricted to picking an independent set from $C_5$, which has size at most 2. On the other hand, we could pick all 3 vertices from the bottom to form an independent set.

Suppose without loss of generality that the highest priority input item is $(1, \{4, 5, 6, 7, 8\})$. The optimal decision for the first vertex is unique: For $G^1$, one should accept, and for $G^2$, reject.

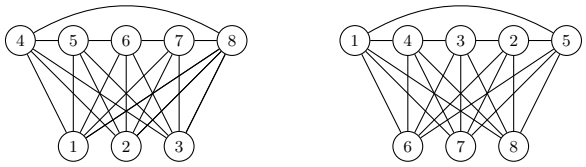


**Fig. 1.** Topological structure of the gadgets $(G^1, G^2)$ for independent set.

In this case, the maximum number $s$ of input items for a gadget is 8, $\mathrm{OPT}(G^1) = \mathrm{OPT}(G^2) = 3$, and $\mathrm{BAD}(G^1) = \mathrm{BAD}(G^2) = 2$. By Corollary 1, we can conclude the following:

**Theorem 14.** *For Maximum Independent Set and any $\varepsilon \in (0, \frac{1}{2}]$, no fixed priority algorithm reading fewer than $(1 - H(\varepsilon))n/16$ advice bits can achieve an approximation ratio smaller than $1 + \frac{\varepsilon}{3-\varepsilon}$.*

Theorem 14 is related to but incomparable with the inapproximation bound results on priority algorithms (without advice) of Borodin et al. [5] for weaker models.

### 6.2 Other Results

Detailed definitions of the problems below, input universes, general details on how to obtain the following results from the general template, and the relationship to the known literature can be found in the full version.

**Theorem 15.** *For Maximum Bipartite Matching and any $\varepsilon \in (0, \frac{1}{2}]$, no fixed priority algorithm reading fewer than $(1 - H(\varepsilon))n/6$ advice bits can achieve an approximation ratio smaller than $1 + \frac{\varepsilon}{3-\varepsilon}$.*

**Theorem 16.** *For Maximum Cut and any $\varepsilon \in (0, \frac{1}{2}]$, no fixed priority algorithm reading fewer than $(1-H(\varepsilon))n/16$ advice bits can achieve an approximation ratio smaller than $1 + \frac{\varepsilon}{15-\varepsilon}$.*

**Theorem 17.** *For Minimum Vertex Cover and any $\varepsilon \in (0, \frac{1}{2}]$, no fixed priority algorithm reading fewer than $(1 - H(\varepsilon))n/14$ advice bits can achieve an approximation ratio smaller than $1 + \frac{\varepsilon}{3}$.*

**Theorem 18.** *For Maximum 3-Satisfiability and any $\varepsilon \in (0, \frac{1}{2}]$, no fixed priority algorithm reading fewer than $(1 - H(\varepsilon))n/6$ advice bits can achieve an approximation ratio smaller than $1 + \frac{\varepsilon}{8-\varepsilon}$.*

**Theorem 19.** *For Job Scheduling of Unit Time Jobs with Precedence Constraints and any $\varepsilon \in (0, \frac{1}{2}]$, no fixed priority algorithm reading fewer than $(1 - H(\varepsilon))n/18$ advice bits can achieve an approximation ratio smaller than $1 + \frac{\varepsilon}{6-\varepsilon}$.*

## 7 Concluding Remarks

We have developed a general framework for showing linear lower bounds on the number of advice bits required to get a constant approximation ratio for fixed priority algorithms with advice. The framework relies on reductions from the Pair Matching problem — analogue of the Binary String Guessing problem from the online world, resistant to universe orderings. Many problems remain open:

- Can our framework (or a modification of it) show non-constant inapproximation results with large advice, for example, for independent set?
- In vertex coloring, any decision for the first item can be completed to an optimal solution. Can our framework be modified to handle such problems? For example, see an argument for the makespan problem in [19].
- An interesting goal is to study the "structural complexity" of online and priority algorithms. Can one define analogues of classes such as NP, NP-Complete, ♯P, etc. for online/priority problems? If so, are complete problems for these classes natural?

#### Acknowledgments

Part of the work was done when the first author was visiting Toyota Technological Institute at Chicago. The work was initiated while the second and third authors were visiting the University of Toronto. Most of the work was done when the fourth author was a postdoc at the University of Toronto.

## References


1. Michael Alekhnovich, Allan Borodin, Joshua Buresh-Oppenheim, Russell Impagliazzo, Avner Magen, and Toniann Pitassi. Toward a model for backtracking and dynamic programming. *Computational Complexity*, 20(4):679–740, 2011.
2. Spyros Angelopoulos and Allan Borodin. On the power of priority algorithms for facility location and set cover. *Algorithmica*, 40(4):271–291, 2004.
3. Bert Besser and Matthias Poloczek. Greedy matching: Guarantees and limitations. *Algorithmica*, 77(1):201–234, 2017.
4. Hans-Joachim Böckenhauer, Juraj Hromkovič, Dennis Komm, Sacha Krug, Jasmin Smula, and Andreas Sprock. The string guessing problem as a method to prove lower bounds on the advice complexity. *Theor. Comput. Sci.*, 554:95–108, 2014.
5. Allan Borodin, Joan Boyar, Kim S. Larsen, and Nazanin Mirmohammadi. Priority algorithms for graph optimization problems. *Theor. Comput. Sci.*, 411(1):239–258, 2010.
6. Allan Borodin, Joan Boyar, Kim S. Larsen, and Denis Pankratov. Advice complexity of priority algorithms. *ArXiv*, 2018. arXiv:1806.06223 [cs.DS].
7. Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
8. Allan Borodin and Brendan Lucier. On the limitations of greedy mechanism design for truthful combinatorial auctions. *ACM Trans. Economics and Comput.*, 5(1):2:1–2:23, 2016.
9. Allan Borodin, Morten N. Nielsen, and Charles Rackoff. (Incremental) priority algorithms. *Algorithmica*, 37(4):295–326, 2003.
10. Joan Boyar, Lene M. Favrholdt, Christian Kudahl, Kim S. Larsen, and Jesper W. Mikkelsen. Online algorithms with advice: A survey. *ACM Comput. Surv.*, 50(2):19:1–19:34, 2017.
11. Joan Boyar, Shahin Kamali, Kim S. Larsen, and Alejandro López-Ortiz. Online bin packing with advice. *Algorithmica*, 74(1):507–527, 2016.
12. Stephen A. Cook. The complexity of theorem-proving procedures. In *3rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 151–158. ACM, 1971.
13. Sashka Davis and Russell Impagliazzo. Models of greedy algorithms for graph problems. *Algorithmica*, 54(3):269–317, May 2009.
14. Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103, 1972.
15. Dennis Komm. *An Introduction to Online Computation – Determinism, Randomization, Advice*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2016.
16. Neal Lesh and Michael Mitzenmacher. Bubblesearch: A simple heuristic for improving priority-based greedy algorithms. *Inf. Process. Lett.*, 97(4):161–169, 2006.
17. Andrew McGregor. Graph stream algorithms: a survey. *ACM SIGMOD Record*, 43(1):9–20, 2014.
18. Matthias Poloczek. Bounds on greedy algorithms for MAX SAT. In *19th Annual European Symposium on Algorithms (ESA)*, volume 6942 of *LNCS*, pages 37–48. Springer, 2011.
19. Oded Regev. Priority algorithms for makespan minimization in the subset model. *Inf. Process. Lett.*, 84(3):153–157, 2002.