Nearly-Tight Lower Bounds for Set Cover and Network Design with Deadlines/Delay

Noam Touitou
Tel Aviv University, Israel

Abstract

In network design problems with deadlines/delay, an algorithm must make transmissions over time to satisfy connectivity requests on a graph. To satisfy a request, a transmission must be made that provides the desired connectivity. In the deadline case, this transmission must occur inside a time window associated with the request. In the delay case, the transmission should be as soon as possible after the request's release, to avoid delay cost.

In FOCS 2020, frameworks were given which reduce a network design problem with deadlines/delay to its classic, offline variant, while incurring an additional competitiveness loss factor of $O(\log n)$, where n is the number of vertices in the graph. Trying to improve upon this loss factor is thus a natural research direction.

The frameworks of FOCS 2020 also apply to *set cover with deadlines/delay*, in which requests arrive on the elements of a universe over time, and the algorithm must transmit sets to serve them. In this problem, a universe of sets and elements is given, requests arrive on elements over time, and the algorithm must transmit sets to serve them.

In this paper, we give nearly tight lower bounds for set cover with deadlines/delay. These lower bounds imply nearly-tight lower bounds of $\Omega(\log n/\log\log n)$ for a few network design problems, such as node-weighted Steiner forest and directed Steiner tree. Our results imply that the frameworks in FOCS 2020 are essentially optimal, and improve quadratically over the best previously-known lower bounds.

2012 ACM Subject Classification Theory of computation; Theory of computation → Design and analysis of algorithms; Theory of computation → Online algorithms

Keywords and phrases Network Design, Deadlines, Delay, Online, Set Cover

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2021.53

1 Introduction

Network Design with Deadlines

In network design problems with deadlines, one is given a set of connectivity requests. Each connectivity request has an associated time window, starting with a release time and ending with a deadline. The input also contains items, with associated costs; usually, these items are some characteristic of a given graph, such as the edges or the nodes. A solution consists of a set of *transmissions*, taking place at various points in time, where each transmission is of some set of items. Such a solution is feasible if for every connectivity request, there exists a transmission which occurs within the request's time window, and provides the connectivity desired by the request.

This general description captures many network design problems. Two concrete examples are node-weighted Steiner forest and directed Steiner tree, both of which are considered in this paper.

- In *node-weighted Steiner forest with deadlines*, the items are the nodes of a given graph, with associated costs. A connectivity request is of some pair of terminal nodes in the graph, demanding that a connection be made between these nodes. A set of items (i.e.

nodes) which satisfies a request must contain a path connecting the request's terminal nodes; such a set must be transmitted within the request's time window to satisfy that request.

- In *directed Steiner tree with deadlines*, the items are the edges of a directed graph, with associated costs. A unique node in the graph is designated as the root of the graph; each connectivity request is of some terminal node, demanding its connection to the root node. A set of items (i.e. edges) which satisfies a request must contain a directed path connecting the request's terminal node to the root. Such a satisfying set must be transmitted within the request's time window for the solution to be feasible.

Network design with deadlines has been studied in the online setting, in which the algorithm constructs a solution as time advances, deciding at each point in time whether (and what) to transmit at that time. Each request is revealed to the algorithm at the request's release time. In line with previous work, we consider the clairvoyant model, where all parameters of the request are revealed at release time (in particular its deadline).

A more general model is *network design with delay*. In this model, each request has a (nondecreasing, continuous) delay function in lieu of a deadline. Each request can be served by a satisfying transmission after its release time. However, in addition to the cost of transmissions, a solution must also pay the *delay cost* of each request – the value of its delay function at the time of the earliest transmission which satisfies it (after its release time). This model can easily be seen to generalize the deadline model.

In [8], a framework for network design problems with deadlines is presented. This framework is in fact a polynomial-time reduction from online network design with deadlines to offline, classic network design: given a γ-approximation algorithm for the offline problem, the framework yields an $O(\gamma \log |U|)$-competitive algorithm for the online problem with deadlines, where U is the set of items. For most problems, in which the items are either the nodes or edges of a given simple graph $G = (V, E)$, we have that $\log |U| = O(\log n)$ – we only consider such problems in this paper. A similar framework for network design with delay is also given in [8], also based on a reduction with $O(\log n)$ loss[1].

In [8], a lower bound of $\Omega(\sqrt{\log n})$ is given on the competitiveness of any (randomized) algorithm for specific network design problems (node-weighted Steiner tree and directed Steiner tree). This implies that *every* reduction incurs a loss factor of $\Omega(\sqrt{\log n})$ (indeed, this information-theoretic lower bound applies even when we allow exponential time, which admits a 1-approximation for the offline problem). This leaves a quadratic gap between the upper and lower bounds; a natural research question would be to close this gap. In this paper, we give a negative answer to this question; in fact, we show that the framework of [8] is essentially tight.

Set Cover with Deadlines

The lower bounds of $\Omega(\sqrt{\log n})$-competitiveness for node-weighted Steiner tree and directed Steiner tree stem from a lower bound for the problem of set cover with deadlines, which is a special case of both problems. In the problem of set cover with deadlines, one is given a universe which consists of a set of elements E and a collection S of subsets from E, where the sets have costs. The input contains a set of requests, such that each request has an

[1] The reduction for the delay case given in [8] is to the prize-collecting offline problem, a similar offline problem which is almost always approximable to the same degree as the classic offline problem (up to some small constant).

associated element in E and a time window (release time and deadline). A solution consists of a set of transmissions, each of which occurs at some point in time, and consists of some set in S. A solution is feasible if the associated element of every request belongs to the set of some transmission which occurs during the request's time window. While not classically considered a network design problem, the set cover with deadlines problem does conform to the model of this paper: the sets are the items of the problem, and the requests are of specific elements.

Set cover with deadlines is indeed a special case of both node-weighted Steiner forest with deadlines and directed Steiner tree with deadlines, as seen by folklore reductions. In both reductions, the set cover input is reduced to graph comprising a root node, "set" nodes and "element" nodes, where the root node must connect to "element' nodes through the "set" nodes; see [8] for a full description of these reductions.

The best known lower bounds for competitiveness in the set cover with deadlines problem, described in [3], are $\Omega(\sqrt{\log \ell})$ and $\Omega(\sqrt{\log m})$, where ℓ and m are the number of elements and the number of sets in the universe, respectively. In this paper, we improve these lower bounds to $\Omega\left(\frac{\log \ell}{\log \log \ell}\right)$ and $\Omega\left(\frac{\log m}{\log \log m}\right)$ respectively.

1.1 Our Results

In this paper, we present lower bounds for three network design problems with deadlines which are tight up to a $\log \log$ factor. For set cover with deadlines or delay, where ℓ and m are the number of elements and sets respectively, we prove the following theorem.

▶ **Theorem 1.1.** *There exist $\Omega\left(\frac{\log \ell}{\log \log \ell}\right)$ and $\Omega\left(\frac{\log m}{\log \log m}\right)$ lower bounds on the competitiveness on any randomized online algorithm for set cover with deadlines (or delay).*

For node-weighted Steiner tree with deadlines and directed Steiner tree with deadlines or delay, for a graph with n vertices, we prove the following theorems.

▶ **Theorem 1.2.** *There exists an $\Omega\left(\frac{\log n}{\log \log n}\right)$ lower bound on the competitiveness on any randomized online algorithm for node-weighted Steiner tree with deadlines (or delay).*

▶ **Theorem 1.3.** *There exists an $\Omega\left(\frac{\log n}{\log \log n}\right)$ lower bound on the competitiveness on any randomized online algorithm for directed Steiner tree with deadlines (or delay).*

The lower bounds of Theorems 1.1–1.3 are nearly tight – the framework of [8] implies $O(\log n)$-competitive algorithms for both node-weighted Steiner forest with deadlines and directed Steiner tree with deadlines, as well as $O(\log m)$- and $O(\log \ell)$-competitive algorithms for set cover with deadlines[2].

We prove the lower bounds of Theorems 1.1–1.3 for the deadline case only; as deadlines are a special case of delay, the lower bounds apply to the delay model as well.

1.2 Our Techniques

The lower bounds of this paper stem from describing a sequence of adversaries $(\mathcal{A}_0, \mathcal{A}_1, \cdots)$ for set cover with deadlines. As we advance in this sequence, we describe adversaries that use a larger universe, and force a worse a competitive ratio on the online algorithm. The growth rate of the universe (ℓ and m) together with the growth rate of the competitive ratio yields the desired lower bound.

[2] Note that the $O(\log \ell)$-competitive algorithm is not a direct application of the framework of [8], but requires a simple observation.

We construct each adversary $\mathcal{A}_i$ recursively from the description of $\mathcal{A}_{i-1}$. The universe of $\mathcal{A}_i$, the elements E_i and sets S_i, consists of multiple copies of E_{i-1} and S_{i-1}, but these copies are interleaved in a way that creates increasing difficulty for the algorithm with each iteration. As for requests, $\mathcal{A}_i$ makes recursive calls to $\mathcal{A}_{i-1}$ on copies of E_{i-1} (which appear inside E_i) which have the structure of $\mathcal{A}_{i-1}$ locally – specifically, restricting the sets of S_i to this copy of E_{i-1} yields a copy of S_{i-1}.

While the recursive nature of the construction is similar to the adversary in [3], the underlying idea is different: in [3], the main idea of the adversary was to present the algorithm with two options for sets, an "expensive" option (which serves some additional long-term requests) and a "cheap" option (which only serves the most urgent requests). The idea was to exploit the fact that the algorithm doesn't know whether investing in preparation for the future would pay off. However, our construction relies on a different principle – we present the algorithm with *many* different options for sets (the number of which grows logarithmically with the universe), the costs of which are identical (i.e. there is no cheap option). The online algorithm will either pick some small number of these options (in which case it would probably miss the correct one), or pick many of these options (which would be very expensive compared to the optimal solution).

1.3 Related Work

Classic online variants for network design problems have been studied extensively in the past. In such variants, the requests arrive over a sequence, and the items are bought rather than transmitted, such that a bought item can be used until the end of the input sequence. Some such problems were studied in [26, 23, 9, 29, 24, 1].

The relationship between this classic online and network design with deadlines is interesting: the clairvoyant deadlines model, considered in this paper, is more closely related to the offline problem (as shown in [8]), and can be much easier than the classic online variant. However, the *nonclairvoyant* deadlines model, where the deadline becomes known only at the end of the request's time window, is as hard as the classic online variant: the reduction showing this for set cover appears in [3].

The online set cover with deadlines/delay problem was first presented by Carrasco *et al.* [19], who gave tight upper- and lower-bounds of logarithmic competitiveness in the number of requests in the input. In [3], bounds referring to the size of universe (sets and elements) were given, including an $O(\log \ell \log m)$-competitive algorithm. This upper bound of $O(\log \ell \log m)$ applies to the nonclairvoyant setting. As nonclairvoyant set cover with deadlines/delay generalizes the classic online set cover, this algorithm is thus optimal for the class of polynomial, randomized algorithms, conditioned on NP $\subsetneq$ BPP [28].

A specific network design problem which was previously considered is facility location with deadlines/delay. In this problem, requests arrive on the nodes of a graph, and a transmission consists of a facility at some node, to which some pending requests are connected. In [7], $O(\log^2 n)$-competitive randomized algorithms for facility location with deadlines/delay were presented, which only worked for the uniform case (i.e. identical facility opening costs), where n is the number of nodes in the graph. These results were then improved to deterministic $O(\log n)$-competitive algorithms for both deadlines and delay in [8].

Another network design problem with deadlines/delay is multilevel aggregation, which is in fact Steiner tree with deadlines/delay where the underlying metric space is a tree. This problem was first presented by Bienkowski *et al.* [10], as a generalization to the previously studied TCP acknowledgement [20, 27, 17] and joint replenishment [18, 15, 11] problems. The algorithm of Bienkowski *et al.* had competitiveness which was exponential in the depth of the tree D. This was first improved to $O(D)$ for the deadline case by Buchbinder *et al.* [16] and then to $O(D^2)$ for the general delay case by [7].

Other problems with deadlines/delay, outside of network design, have also seen significant interest. The k-server problem with delay was presented in [5], and studied in [14, 7, 25]. Interestingly, this problem is related to the network design problem of multilevel aggregation, as discussed in [7].

Min-cost perfect matching with delays, presented in [21] is another such problem. This problem is only tractable for specific delay functions (e.g. linear and concave), and is studied in [2, 22, 21, 4, 12, 13, 6].

2 Preliminaries

Since our results for node-weighted Steiner forest and directed Steiner tree stem from our result for set cover through a folklore reduction, we only provide a formal description for set cover with deadlines.

Set cover with deadlines

In the set cover with deadlines problem, one is given a universe which consists of elements E and a collection of sets S, such that $s \subseteq E$ for every $s \in S$. In addition, each set $s \in S$ has a cost $c(S)$. Additionally, the input contains a set of requests Q. Each request $q \in Q$ has an associated element $e_q \in E$, a release time r_q and a deadline d_q (such that $r_q < d_q$).

A solution for the input is some collection of (instantaneous) transmissions $\{T_1, \cdots, T_k\}$ at various points in time, such that each transmission is of some set s_T. The cost of this solution is $\sum_{i=1}^{k} c(s_{T_i})$, i.e. the sum of costs of transmitted sets. Note that a set can be transmitted more than once in a solution; in this case, the set's cost is incurred more than once. A solution is *feasible* if for each request $q \in Q$, there exists a transmission T at some time between r_q and d_q such that $e_q \in s_T$. In words, a set containing the element requested by q must be transmitted within q's time window.

Rigorously, the events at time t occur in the following order: first, any transmitted sets at t serve pending requests; then, any request q with $r_q = t$ is released. Thus, in order to serve a request q, a transmission must take place in the half-open time window $(r_q, d_q]$. We remark that the results of this paper hold for any choice of order, and that our specific order is chosen for ease of presentation. (Indeed, our lower bound can be stated with distinct release/deadline times, which would bypass this issue.)

The online setting

An online algorithm receives the universe (i.e. E, S and $\{c(s)\}_{s \in S}$) up front, while the requests Q are revealed to the algorithm as time advances. Specifically, each request $q \in Q$ appears to the online solution only upon its release time r_q. We consider the *clairvoyant* model, in which all parameters of the request q are revealed at r_q (in particular its deadline d_q).

At any point in time, the algorithm is allowed to make a transmission T of any set s (thus incurring a cost of $c(s)$).

3 Lower Bound for Set Cover

In this section, we describe and analyze an oblivious adversary for set cover with deadlines, which we then use to prove Theorems 1.1–1.3.

We define the sequence $(\ell_i)_{i=0}^{\infty}$ recursively by $\ell_0 := 1$ and by $\ell_i := (6i+1)\ell_{i-1}$ for $i > 0$. Similarly, we define the sequence $(m_i)_{i=0}^{\infty}$ recursively by $m_0 := 1$ and by $m_i := 4i \cdot m_{i-1}$ for $i > 0$. We prove the following lemma.

▶ **Lemma 3.1.** *For every index i, there exists an oblivious adversary $\mathcal{A}_i$ which generates a set cover with deadlines instance on a universe with ℓ_i elements and m_i sets, such that any deterministic algorithm is at least $\Omega(i)$-competitive against $\mathcal{A}_i$.*

Lemma 3.1, together with Yao's principle and some folklore reductions, is later used to prove Theorems 1.1–1.3.

3.1 The Set Cover Adversary

To prove Lemma 3.1, we now introduce the oblivious adversary $\mathcal{A}_i$. This adversary $\mathcal{A}_i$ provides a universe with elements E_i and set collection S_i. The adversary always provides unweighted instances, i.e. the cost of every set in S_i is always 1.

The Universe of $\mathcal{A}_i$

In the base case of $i = 0$, we have a universe of a single element, ($E_0 = \{e\}$) and a single set ($S_0 = \{s\}$).

For $i > 0$, we construct the universe of $\mathcal{A}_i$ recursively from the universe of $\mathcal{A}_{i-1}$. Define $b_i := 2i$. The set E_i contains copies of elements from E_{i-1}. Specifically, each element $e \in E_{i-1}$ has the following copies:

1. The copy $e^{\mathbf{s}}$ (define $E_{i-1}^{\mathbf{s}} := \{e^{\mathbf{s}} | e \in E_{i-1}\}$). This copy is called the *special* copy.
2. The b_i copies $e^{\mathbf{a},1}, \cdots, e^{\mathbf{a},b_i}$ (for every j, define $E_{i-1}^{\mathbf{a},j} := \left\{e^{\mathbf{a},j} \middle| e \in E_{i-1}\right\}$). These are called the *ancillary* copies of elements.
3. The b_i copies $e^{\mathbf{p},1}, \cdots, e^{\mathbf{p},b_i}$ (for every j, define $E_{i-1}^{\mathbf{p},j} := \left\{e^{\mathbf{p},j} \middle| e \in E_{i-1}\right\}$). These are called the *positive* copies of elements.
4. The b_i copies $e^{\mathbf{n},1}, \cdots, e^{\mathbf{n},b_i}$ (for every j, define $E_{i-1}^{\mathbf{n},j} := \left\{e^{\mathbf{n},j} \middle| e \in E_{i-1}\right\}$). These are called the *negative* copies of elements.

The elements of the instance of $\mathcal{A}_i$ are defined as

$$E_i := E_{i-1}^{\mathbf{s}} \cup \left(\bigcup_{j=1}^{b_i} E_{i-1}^{\mathbf{a},j}\right) \cup \left(\bigcup_{j=1}^{b_i} E_{i-1}^{\mathbf{p},j}\right) \cup \left(\bigcup_{j=1}^{b_i} E_{i-1}^{\mathbf{n},j}\right)$$

We can now observe that $|E_i| = (3b_i + 1)|E_{i-1}| = (6i+1)|E_i|$. Since $|E_0 = 1|$, for every i we have $|E_i| = \ell_i$, as required by Lemma 3.1.

Figure 1 shows the elements of $\mathcal{A}_0$ (upper left), $\mathcal{A}_1$ (lower left) and $\mathcal{A}_2$ (right). Each element in $\mathcal{A}_1$ and $\mathcal{A}_2$ is a copy of an element from the universe of the previous adversary; each copy is special (purple), ancillary (brown), positive (blue) or negative (red).

As for the sets, each set $s \in S_{i-1}$ has the following copies in S_i:

1. The b_i copies $s^{\mathbf{p},1}, \cdots, s^{\mathbf{p},b_i}$, such that

$$s^{\mathbf{p},j} = \bigcup_{e \in s} \left\{e^{\mathbf{s}}, e^{\mathbf{a},j}, e^{\mathbf{p},j}\right\}$$

(for each j, we define $S_{i-1}^{\mathbf{p},j} := \left\{s^{\mathbf{p},j} \middle| s \in S_{i-1}\right\}$). These copies are called the *positive* copies of sets.

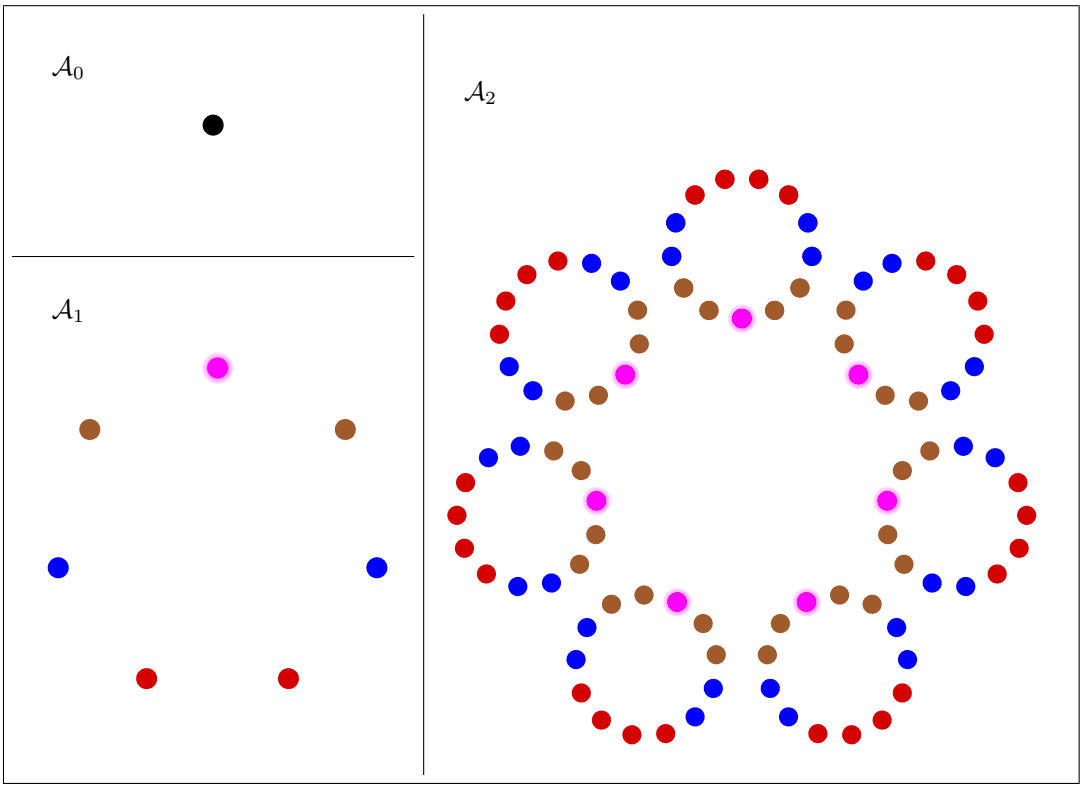

Figure 1 The Elements of $\mathcal{A}_0$, $\mathcal{A}_1$ and $\mathcal{A}_2$.

2. The b_i copies $s^{\mathbf{n},1}, \cdots, s^{\mathbf{n},b_i}$, such that

$$s^{\mathbf{n},j} = \bigcup_{e \in s} \left(\left\{ e^{\mathbf{n},j} \right\} \cup \left(\bigcup_{j' \neq j} \left\{ e^{\mathbf{p},j'} \right\} \right) \right)$$

(for each j, we define $S_{i-1}^{\mathbf{n},j} := \left\{ s^{\mathbf{n},j} \middle| s \in S_{i-1} \right\}$). These copies are called the *negative* copies of sets.

Note that holds that $|S_i| = 2b_i \cdot |S_{i-1}| = 4i \cdot |S_{i-1}|$. Combined with the fact that $|S_0| = 1$, for every i we have $|S_i| = m_i$, as required by Lemma 3.1.

Figure 2 shows some (not all) of the sets in $\mathcal{A}_0$, $\mathcal{A}_1$ and $\mathcal{A}_2$. The gray set in S_0 is the only set of that universe. The blue and red sets in S_1 are positive and negative copies of the gray set, respectively. The purple set in S_2 is a positive copy of the red set. The green set in S_2 is a negative copy of the blue set.

Request sequence of $\mathcal{A}_i$

We now describe the sequence of requests generated by $\mathcal{A}_i$.

Recursive calls to $\mathcal{A}_{i-1}$

For $i > 0$, the adversary $\mathcal{A}_i$ relies on recursive calls to $\mathcal{A}_{i-1}$. Note that the elements E_i comprise copies of E_{i-1} ($E_{i-1}^{\mathbf{s}}, E_{i-1}^{\mathbf{p},j}$, et cetera). Thus, for any copy E'_{i-1} of E_{i-1}, it is well-defined to call $\mathcal{A}_{i-1}$ on E'_{i-1}: we release a request on a copy $e' \in E'_{i-1}$ of $e \in E_{i-1}$ whenever a request is released on e by $\mathcal{A}_{i-1}$.

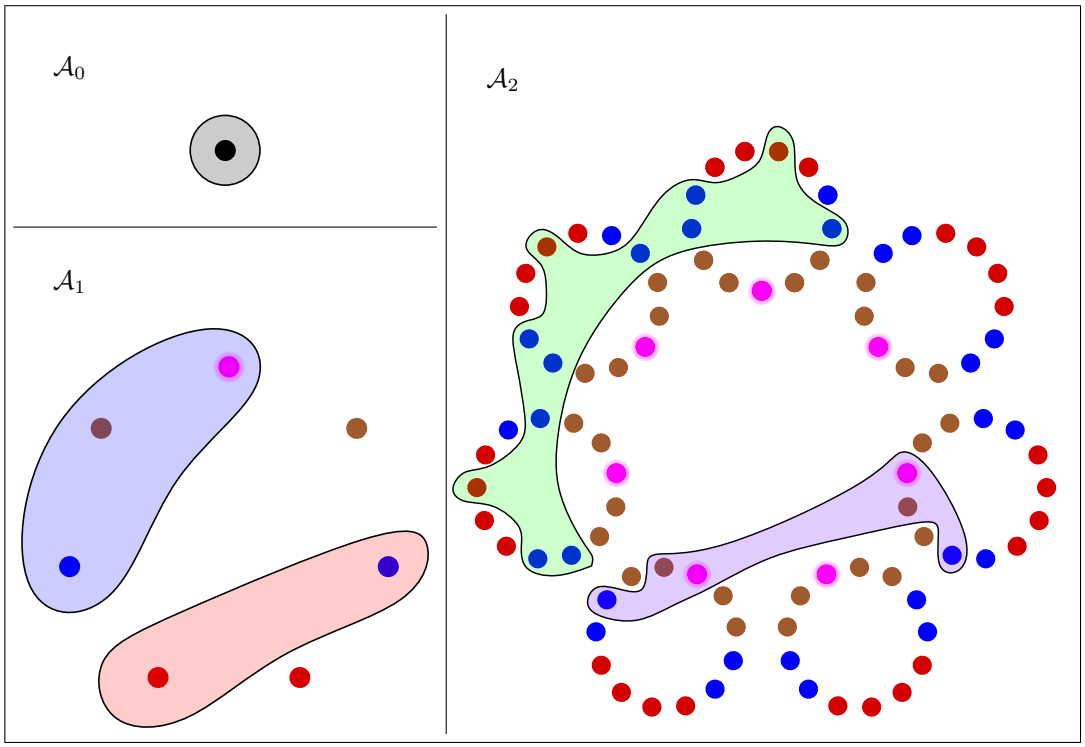

Figure 2 Some Example Sets in $\mathcal{A}_0$, $\mathcal{A}_1$ and $\mathcal{A}_2$.

For abbreviation, we use the superscript denoting the copy of the elements on the adversary. For example, calling $\mathcal{A}_{i-1}^{\mathbf{p},j}$ means calling $\mathcal{A}_{i-1}$ on the copy $E_{i-1}^{\mathbf{p},j}$.

Adversary time bounds

As we would like to pack multiple recursive $\mathcal{A}_{i-1}$ adversaries into one timeline, we would like to bound each $\mathcal{A}_{i-1}$ by some time interval, so that we can charge the algorithm for solving each $\mathcal{A}_{i-1}$ disjointly.

Define T_i for $i \geq 0$ recursively by $T_0 = 1$ and $T_i = 2b_i \cdot T_{i-1}$. Informally, T_i defines a time interval which contains $\mathcal{A}_i$; formally, each request q which $\mathcal{A}_i$ can possibly release has $[r_q, d_q] \subseteq [0, T_i]$ (this can be verified for the construction of $\mathcal{A}_i$ we describe next).

Requests of $\mathcal{A}_i$

For the case that $i = 0$, the adversary releases at time 0 a single request q on the single element in E_0, such that $d_q = 1$.

For the case of $i > 0$, the adversary $\mathcal{A}_i$ is given in Procedure 1; we now provide a verbal description of that procedure. The adversary $\mathcal{A}_i$ uses b_i phases, each of which takes $2T_{i-1}$ time, and makes two recursive calls to $\mathcal{A}_{i-1}$. In the beginning of each phase k, "background" requests are released on the elements in some positive copies of E_{i-1}, with deadlines at the end of the phase (i.e. $2T_{i-1}$ time after release). These requests are released on copies of E_{i-1} whose index is in some set M, where M initially consists of all b_i indices of positive copies of E_{i-1}. In the first half of each phase, i.e. the first T_{i-1} time units, $\mathcal{A}_i$ calls $\mathcal{A}_{i-1}^{\mathbf{s}}$ (and waits T_{i-1} time for its completion). Then, $\mathcal{A}_i$ chooses an index j_k to remove from M, and calls $\mathcal{A}_{i-1}^{\mathbf{n},j_k}$ which occurs in the second half of the phase.

Intuitive Explanation

The intuition behind this construction of $\mathcal{A}_i$ is the following. When the online algorithm handles the first recursive call in a phase, it has some choices to make. First, it has the standard choices to make when addressing $\mathcal{A}_{i-1}$ – that is, which sets from S_{i-1} to transmit. In addition, for each set in S_{i-1} that it wishes to transmit, it must choose the appropriate copy from S_i. Note that the only copies of a set $s \in S_{i-1}$ which can be used to solve $\mathcal{A}^{\mathbf{s}}_{i-1}$ are the positive copies, i.e. $s^{\mathbf{p},j}$ for some index j; moreover, since this first recursive call is on the special copies, *all* choices of j are possible. (The purpose of the special copies is exactly this: to be at the intersection of positive copies of sets, and force the algorithm to make a choice.)

However, transmitting these positive copies of sets serves an additional purpose – serving the background requests on M. For this reason, the online algorithm should choose to transmit $s^{\mathbf{p},j}$ for some $j \in M$. But note that for earlier phases, the set M is rather large – its size is $\Omega(i)$. The size of M prohibits the algorithm from transmitting every positive copy in M of every set – otherwise, the algorithm would incur a great cost. Instead, the algorithm must focus on a small subset of M, which leaves many background requests unsatisfied.

The optimal solution, on the other hand, is provided a "shortcut": it uses the second recursive call of the phase, which is to $\mathcal{A}^{\mathbf{n},j_k}_{i-1}$, to serve all background requests except for those on $E^{\mathbf{p},j_k}_{i-1}$. The requests on $E^{\mathbf{p},j_k}_{i-1}$ are served by the optimal solution in the first recursive call to $\mathcal{A}^{\mathbf{s}}_{i-1}$, where the solution only transmits copies from $S^{\mathbf{p},j_k}_{i-1}$. Note that such efficient handling of the background requests cannot be achieved by the online algorithm, since it does not have knowledge of j_k during the first half of phase k.

Near the end of a phase, when the online algorithm has (with high probability) a large amount of pending background requests to serve, the algorithm must incur the cost of an (offline) set cover for those pending requests; the only purpose of ancillary element copies is to keep the cost of this offline cover large, which ensures high costs for the algorithm. (Specifically, the ancillary element copies ensure that each positive set copy s has an element unique to s, which forces s to be part of the offline set cover; this is used in Proposition 3.4.)

Procedure 1 Adversary $\mathcal{A}_i$.

Function CreateInstance
- Start with $M \leftarrow \{1, 2, \cdots, b_i\}$.
- **for** k *from* 1 *until* b_i **do**
  - *// Start phase k at time $2(k-1)T_{i-1}$, by releasing background requests and calling $\mathcal{A}^{\mathbf{s}}_{i-1}$.*
  - For every $j \in M$, release a request on every element in $E^{\mathbf{p},j}_{i-1}$, with a deadline that's $2T_{i-1}$ time in the future.
  - Call $\mathcal{A}^{\mathbf{s}}_{i-1}$ and wait T_{i-1} time until its completion.
  - *// T_{i-1} time after the start of phase k, call the second $\mathcal{A}_{i-1}$ on a random negative copy of E_{i-1}.*
  - Choose $j_k \in M$ uniformly at random from M.
  - Call $\mathcal{A}^{\mathbf{n},j_k}_{i-1}$ and wait T_{i-1} time until its completion.
  - Set $M \leftarrow M \backslash \{j_k\}$.

The timeline of a possible instance created by $\mathcal{A}_2$ is shown in Figure 3. This figure shows $b_2 = 4$ phases, each of which contains two calls to $\mathcal{A}_{i-1}$. In this figure, $\mathcal{A}_2$ randomly chose the permutation $(4, 2, 1, 3)$ of the elements of $[b_2]$. The arrows above each phase show the release

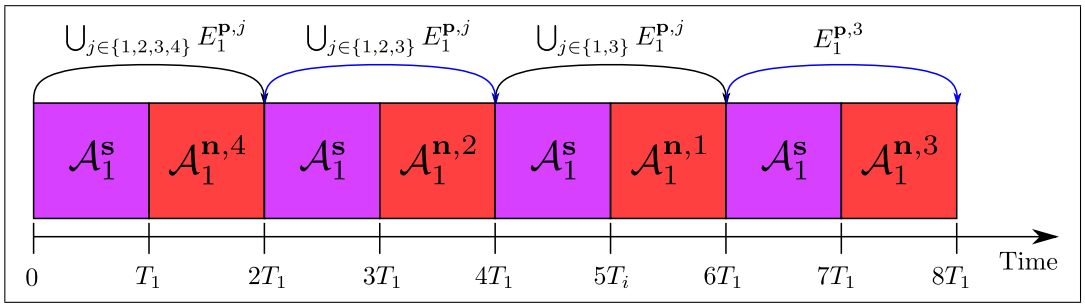

Figure 3 Request Timeline of $\mathcal{A}_2$.

times and deadlines of background requests – these requests are released at the beginning of each phase, and have deadlines at the end of the phase. Above the arrows are the sets of elements on which background requests are released.

3.2 Analysis

We now analyze the adversary described above, thus proving Lemma 3.1.

The Optimal Solution

The following lemma describes the optimal solution for instances generated by the adversary $\mathcal{A}_i$. Not only do they have a low cost relative to the algorithm, but they also buy every set exactly once, which is useful due to recursion.

▶ **Lemma 3.2.** *For every i, and for every instance generated by $\mathcal{A}_i$, there exists an offline solution that buys every set in S_i exactly once, and thus has cost m_i.*

Proof. The proof is by induction on i. For $i = 0$, the adversary $\mathcal{A}_i$ releases a single request on a single element at time 0, with deadline at time 1. Thus, transmitting the single set in m at any time during the interval $(0, 1]$ is a feasible solution.

Assume that the lemma holds for $\mathcal{A}_{i-1}$ – i.e. there exists an offline solution SOL which buys every set in S_{i-1} exactly once.

Now, consider an instance generated by $\mathcal{A}_i$. This instance was generated in b_i phases, each associated with some index in $[b_i]$ (the index chosen randomly by $\mathcal{A}_i$ in this phase). These indices form a permutation on the elements of $[b_i]$, and we write them as a sequence $(j_k)_{k=1}^{b_i}$, such that j_k is the index chosen by $\mathcal{A}_i$ in phase k.

We now describe an offline solution for the instance generated by $\mathcal{A}_i$. Upon phase k, consider the call to $\mathcal{A}_{i-1}^{\mathbf{s}}$ in the beginning of the phase. In the original universe of $\mathcal{A}_{i-1}$, i.e. E_{i-1} and S_{i-1}, the induction hypothesis implies that there exists an offline solution SOL for this instance which transmits every set of S_{i-1} exactly once. Thus, a solution for $\mathcal{A}_{i-1}^{\mathbf{s}}$ would be to transmit any positive copy of the set s whenever SOL transmits s. Our offline solution to $\mathcal{A}_i$ will transmit only the positive copies of index j_k, i.e. $s^{\mathbf{p},j_k}$ instead of s.

As for the call to $\mathcal{A}_{i-1}^{\mathbf{n},j_k}$, which is the second recursive call in phase k, we use a similar argument: we use the offline solution SOL for $\mathcal{A}_{i-1}$ in the original universe of $\mathcal{A}_{i-1}$, and transmit $s^{\mathbf{n},j_k}$ whenever SOL transmits j.

Observe that this offline solution for the instance generated by $\mathcal{A}_i$ is feasible:

- The requests inside recursive calls to $\mathcal{A}_i$ are satisfied from the induction hypothesis.
- The requests outside recursive calls, i.e. the background requests released in the beginning of the phase, are also satisfied: in each phase k, we transmit all sets in $S_{i-1}^{\mathbf{p},j_k}$ (which satisfy all requests on $E_{i-1}^{\mathbf{p},j_k}$) and $S_{i-1}^{\mathbf{n},j_k}$ (which satisfy all requests on $E_{i-1}^{\mathbf{p},j}$ for every $j \neq j_k$).

Now note that all sets in S_i are bought exactly once: this is since in phase k the algorithm buys all sets from $S_{i-1}^{\mathbf{p},j_k}$ and $S_{i-1}^{\mathbf{n},j_k}$ exactly once (and no other sets). Since $(j_k)_{k=1}^{b_i}$ are a permutation of $[b_i]$, this yields the desired claim. ◀

The Cost of the Algorithm

We now bound the expected cost of the algorithm. For every i, define $c_i := \frac{i}{8}$. The main result here is the following lemma.

▶ **Lemma 3.3.** *For every i, and for every deterministic online algorithm* ALG *against the adversary $\mathcal{A}_i$, it holds that the expected cost of the algorithm during the interval $(0, T_i]$ is at least $c_i \cdot m_i$ (where the expectation is over the random choices of $\mathcal{A}_i$).*

We prove Lemma 3.3 by induction. For the base case of $i = 0$, note that this trivially holds for $\mathcal{A}_i$: the algorithm must transmit a set during $(0, 1]$ at cost 1, which is more than $\frac{1}{8}$. For every $i > 0$, assuming that the lemma holds for $i - 1$, we prove the lemma for $\mathcal{A}_i$. We also henceforth fix ALG to be the online deterministic algorithm which runs against the adversary $\mathcal{A}_i$. Slightly abusing notation, we also use ALG to refer to the cost of the online algorithm during the interval $(0, T_i]$; indeed, costs outside this interval are irrelevant to Lemma 3.3.

First, we show an important property of the universe of $\mathcal{A}_i$. For every universe with elements E and sets S, denote by $\mathrm{sc}(E, S)$ the cost of the optimal (classic, offline) set cover solution for this universe. For all (reasonable) universes, buying all sets is a feasible set cover solution; Proposition 3.4 shows that for the universe of $\mathcal{A}_i$, buying all sets is in fact the *only* solution.

▶ **Proposition 3.4.** *For every i, it holds that* $\mathrm{sc}(E_i, S_i) = m_i$.

Proof of Proposition 3.4. Clearly, buying each set in S_i is a feasible solution of cost m_i. It now show that each set must be bought, which proves the proposition. To this end, we prove that for each set $s \in S_i$ there exists an element $e \in E_i$ such that e is in s, but in no other set in S_i. If this indeed holds for each $s \in S_i$, each set must be bought, completing the proof.

We prove this claim by induction on i. For the base case of $i = 0$ this trivially holds. Now, for $i > 0$, assume that for each set $\bar{s} \in S_{i-1}$ there exists an element in E_{i-1} which is in $\bar{s}$ and in no other set in S_{i-1}.

Now, consider any set $s \in S_i$. This set is a copy of some set $\bar{s} \in S_{i-1}$, for which the induction hypothesis provides an element $\bar{e} \in E_{i-1}$ which is in $\bar{s}$ and not in any other set in S_{i-1}.

1. If s is a positive copy of $\bar{s}$, i.e. $s = \bar{s}^{\mathbf{p},j}$ for some j, then observe the element $e := \bar{e}^{\mathbf{a},j} \in E_i$. It holds that e is in s but in no other set in S_i.
2. If s is a negative copy of $\bar{s}$, i.e. $s = \bar{s}^{\mathbf{n},j}$ for some j, then observe the element $e := \bar{e}^{\mathbf{n},j} \in E_i$. It holds that e is in s but in no other set in S_i

This completes the proof of the claim, and thus the proposition. ◀

Note again that Proposition 3.4 refers to the offline cost of covering *the universe* of $\mathcal{A}_i$; this is not the same as the cost of the optimal solution against $\mathcal{A}_i$ (for example, ancillary copies in the universe are never requested by $\mathcal{A}_i$, and are only useful for future recursion).

We would now like to give a lower bound for the expected cost of the algorithm at each phase. For every phase k, let ALG_k be the cost of ALG during the phase k, i.e. during the time interval $((2k-2)T_{i-1}, 2kT_{i-1}]$.

▶ **Lemma 3.5.** *For every phase k, it holds that:*
1. *If $k \leq \frac{b_i}{2}$, then $\mathbb{E}[\text{ALG}_k] \geq \left(c_{i-1} + \frac{1}{4}\right) \cdot 2m_{i-1}$*
2. *If $k > \frac{b_i}{2}$, then $\mathbb{E}[\text{ALG}_k] \geq c_{i-1} \cdot 2m_{i-1}$*

Proof. Fix any phase k, which starts at time $\tau := 2(k-1) \cdot T_{i-1}$. For this proof, fix the set of random choices made by $\mathcal{A}_i$ until the start of phase k; henceforth in the proof the expectations are thus only on random choices from phase k onwards. Since we bound the expected cost of the algorithm for every possible set of choices made up to phase k by $\mathcal{A}_i$, this lower bound also applies in expectation over those choices.

Let M denote the value of the variable of the same name in $\mathcal{A}_i$ at the beginning of phase k (recall that $M \subseteq [b_i]$ is a set of indices). Since we have fixed the random choices of $\mathcal{A}_i$ before phase k, the set M is some fixed set.

We divide the transmissions made by the algorithm during the phase into three disjoint parts:
- Part P_1: transmissions of positive sets during $(\tau, \tau + T_{i-1}]$ (the first half of the phase).
- Part P_2: transmissions of negative sets from $S_{i-1}^{\mathbf{n},j_k}$ during $(\tau + T_{i-1}, \tau + 2T_{i-1}]$ (the second half of the phase).
- Part P_3: transmissions not in P_1, P_2.

We denote the number of transmissions in P_ℓ (equivalently: the total cost of such transmissions) by C_ℓ.

Part P_1. Observe that the positive sets are the only sets that can be used for $\mathcal{A}_{i-1}^{\mathbf{s}}$ in the first part of the phase. Also note that intersecting all positive sets with $E_{i-1}^{\mathbf{s}}$ yields a collection of sets which is identical to S_{i-1}; that is, for every set $s \in \bigcup_j S_{i-1}^{\mathbf{p},j}$, there exists a set $s' \in S_{i-1}$ such that

$$s \cap E_{i-1}^{\mathbf{s}} = \{e^{\mathbf{s}} | e \in s'\}$$

(specifically, the set s' is such that s is a positive copy of s')

Thus, covering the elements of $E_{i-1}^{\mathbf{s}}$ with these sets is as hard as covering the elements of E_{i-1} with S_{i-1}. Now, recall the induction hypothesis made for Lemma 3.3, which implied that the expected cost of this part of the algorithm is at least $c_{i-1} \cdot m_{i-1}$.

Part P_2. Note that the sets of $S_{i-1}^{\mathbf{n},j_k}$ are the only sets that contain elements from $E_{i-1}^{\mathbf{n},j_k}$, and are thus the only sets that can be used to serve $\mathcal{A}_{i-1}^{\mathbf{n},j_k}$. Also note that intersecting the sets of $S_{i-1}^{\mathbf{n},j_k}$ with the elements $E_{i-1}^{\mathbf{n},j_k}$ yields a collection of sets which is identical to S_{i-1} (in a similar way to the argument for P_1) We can thus again apply the induction hypothesis, and see that the expected cost of the algorithm in buying the sets of $S_{i-1}^{\mathbf{n},j_k}$ must be at least $c_{i-1} \cdot m_{i-1}$.

Combining Parts P_1 and P_2, we have

$$\mathbb{E}[C_1] + \mathbb{E}[C_2] \geq c_{i-1} \cdot 2m_{i-1} \tag{1}$$

Equation (1) immediately implies the second claim of this lemma. It remains to prove the first claim.

Assume henceforth that $k \leq \frac{b_i}{2}$. If it holds that $\mathbb{E}[C_1] \geq \frac{b_i}{4} \cdot m_{i-1}$, then since $\frac{b_i}{4} \cdot m_{i-1} = \frac{i}{2} \cdot m_{i-1} \geq \left(c_{i-1} + \frac{1}{2}\right) m_{i-1}$, we have $\mathbb{E}[C_1] + \mathbb{E}[C_2] \geq \left(c_{i-1} + \frac{1}{4}\right) \cdot 2m_{i-1}$, which completes the proof of the second claim.

We therefore assume henceforth that $\mathbb{E}[C_1] < \frac{b_i}{4} \cdot m_{i-1}$.

Part P_3. Consider the background requests of phase k which were released on $E_{i-1}^{\mathbf{p},j}$, for any index $j \in M$. As the restriction of S_i to $E_{i-1}^{\mathbf{p},j}$ is identical to S_{i-1}, Proposition 3.4 implies $\mathrm{sc}\left(E_{i-1}^{\mathbf{p},j}, S_i\right) = m_{i-1}$. Thus, the algorithm has to transmit at least m_{i-1} sets that contain elements from $E_{i-1}^{\mathbf{p},j}$ during the phase.

Consider the sets transmitted in Part P_1. These sets are all positive sets. Each such positive set $s^{\mathbf{p},j}$, for some j, does not contain any positive elements outside $E_{i-1}^{\mathbf{p},j}$. Denote by C_1^j the number of sets from $S_{i-1}^{\mathbf{p},j}$ transmitted in P_1, such that $C_1 = \sum_j C_1^j$. Then we know that for each $j \in M$ we have that in P_2 and P_3 together, there must be at least $m_{i-1} - C_1^j$ transmissions of sets containing elements from $E_{i-1}^{\mathbf{p},j}$.

Now, observe that choosing $j = j_k$, Part C_2 transmits only sets from $S_{i-1}^{\mathbf{n},j_k}$, which do not contain elements from $E_{i-1}^{\mathbf{p},j_k}$. This thus yields a lower bound of $C_3 \geq m_{i-1} - C_1^{j_k}$.

Overall, we have that:

$$\begin{aligned}\mathbb{E}[C_3] &\geq \sum_{j\in M} \Pr(j_k = j) \cdot \mathbb{E}\left[m_{i-1} - C_1^j \middle| j_k = j\right] = \frac{1}{|M|}\sum_{j\in M}\left(m_{i-1} - \mathbb{E}\left[C_1^j | j_k = j\right]\right) \\ &= \frac{1}{|M|}\sum_{j\in M}\left(m_{i-1} - \mathbb{E}\left[C_1^j\right]\right) = m_{i-1} - \frac{1}{|M|}\mathbb{E}[C_1] \geq m_{i-1} - \frac{2}{b_i}\mathbb{E}[C_1] \geq \frac{m_{i-1}}{2}\end{aligned}$$

The second equality is due to the fact that C_1^j is independent of the choice of j_k (indeed, the choice of j_k only affects the input from time $\tau + T_{i-1}$). The second inequality is from the fact that $k \leq \frac{b_i}{2}$, which implies that $|M| \geq \frac{b_i}{2}$. The third inequality is from $\mathbb{E}[C_1] < \frac{b_i}{4} \cdot m_{i-1}$. Combining this with Equation (1), we obtain

$$\mathbb{E}[C_1] + \mathbb{E}[C_2] + \mathbb{E}[C_3] \geq 2c_{i-1} \cdot m_{i-1} + \frac{m_{i-1}}{2} = \left(c_{i-1} + \frac{1}{4}\right) \cdot 2m_{i-1}$$ ◀

Proof of Lemma 3.3. It holds that

$$\begin{aligned}\mathbb{E}[\mathrm{ALG}] &= \sum_{k=1}^{b_i} \mathbb{E}[\mathrm{ALG}_k] \geq \frac{b_i}{2}\left(c_{i-1} + \frac{1}{4}\right) \cdot 2m_{i-1} + \frac{b_i}{2} c_{i-1} \cdot 2m_{i-1} \\ &= \left(c_{i-1} + \frac{1}{8}\right) 2b_i \cdot m_{i-1} = c_i m_i\end{aligned}$$

where the first inequality is due to applying Lemma 3.5 to the phases (using the stronger claim for the earlier phases and the weaker claim for the later phases), and the final equality uses the fact that $m_i = 2b_i \cdot m_{i-1}$. ◀

Proof of Lemma 3.1. The lemma results immediately from Lemmas 3.2 and 3.3. ◀

3.3 Proofs of Theorems

We now use the construction above to prove the main theorems of this paper.

Proof of Theorem 1.1. Lemma 3.1 implies that any deterministic algorithm is $\Omega(i)$-competitive against $\mathcal{A}_i$, which uses a universe of ℓ_i elements and m_i sets. Yao's principle now implies that for every randomized online algorithm, there exists an instance with ℓ_i and m_i on which its competitive ratio is $\Omega(i)$.

For the set-based bound, note that $m_i = 4^i \cdot i! \leq (4i)^i$, which implies $i \geq \frac{\log m_i}{4 \log i}$. Now observe that $m_i \geq 2^i$ and thus $i \leq \log m_i$. Together with the previous observation, we have that $i = \Omega\left(\frac{\log m_i}{\log\log m_i}\right)$, which yields the desired $\Omega\left(\frac{\log m}{\log\log m}\right)$-competitiveness lower bound.

For the element-based bound, note that $2^i \leq \ell_i \leq (7i)^i$. A similar argument thus yields that $i = \Omega\left(\frac{\log \ell_i}{\log\log \ell_i}\right)$, which gives us the $\Omega\left(\frac{\log \ell}{\log\log \ell}\right)$-competitiveness lower bound. ◀

Proofs of Theorems 1.2 and 1.3. There exist folklore reductions from set cover to node-weighted Steiner tree and directed Steiner tree, which reduce a set cover instance with ℓ elements and m sets to graphs with $\ell + m + 1$ nodes. These reductions carry over to the deadline/delay setting (for a detailed description of these reductions, see e.g. [8]).

Now, $\mathcal{A}_i$ as described for set cover yields a graph with $\ell_i + m_i + 1$ nodes, which is at most $3 \cdot (7i)^i$ nodes (and more than 2^i nodes). Lemma 3.1, together with argument identical to the proof of Theorem 1.1, yield an $\Omega\left(\frac{\log n}{\log \log n}\right)$ on the competitiveness of any randomized algorithm. ◀

4 Discussion and Open Problems

In this paper, we presented nearly-logarithmic lower bounds on competitiveness for some network design problems with deadlines (which therefore also apply to the delay cases). In [8], a framework is shown which solves every network design with deadlines problem using an approximation algorithm for the corresponding offline problem, losing a logarithmic factor in competitiveness; our results thus show that this logarithmic factor is nearly optimal.

However, the problems we consider in this paper might be tougher than other network design problems with deadlines. While our paper shows that logarithmic loss in approximation ratio is necessary for the general case, there exist many network design problems for which no superconstant lower bound on competitiveness exists. Examples of such problems with deadlines are (edge-weighted) Steiner tree and Steiner forest, facility location, and multicut. Resolving the competitive ratio for these problems remains an interesting open problem.

References

1. Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. A general approach to online network optimization problems. *ACM Trans. Algorithms*, 2(4):640–660, 2006. doi:10.1145/1198513.1198522.
2. Itai Ashlagi, Yossi Azar, Moses Charikar, Ashish Chiplunkar, Ofir Geri, Haim Kaplan, Rahul M. Makhijani, Yuyi Wang, and Roger Wattenhofer. Min-cost bipartite perfect matching with delays. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2017, August 16-18, 2017, Berkeley, CA, USA*, pages 1:1–1:20, 2017. doi:10.4230/LIPIcs.APPROX-RANDOM.2017.1.
3. Yossi Azar, Ashish Chiplunkar, Shay Kutten, and Noam Touitou. Set cover with delay – clairvoyance is not required. In Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, pages 8:1–8:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.8.
4. Yossi Azar and Amit Jacob Fanani. Deterministic min-cost matching with delays. In *Approximation and Online Algorithms – 16th International Workshop, WAOA 2018, Helsinki, Finland, August 23-24, 2018, Revised Selected Papers*, pages 21–35, 2018. doi:10.1007/978-3-030-04693-4_2.
5. Yossi Azar, Arun Ganesh, Rong Ge, and Debmalya Panigrahi. Online service with delay. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 551–563, 2017. doi:10.1145/3055399.3055475.
6. Yossi Azar, Runtian Ren, and Danny Vainstein. The min-cost matching with concave delays problem. *CoRR*, abs/2011.02017, 2020. arXiv:2011.02017.
7. Yossi Azar and Noam Touitou. General framework for metric optimization problems with delay or with deadlines. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 60–71, 2019. doi:10.1109/FOCS.2019.00013.

8 Yossi Azar and Noam Touitou. Beyond tree embeddings – a deterministic framework for network design with deadlines or delay. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1368–1379. IEEE, 2020. doi:10.1109/FOCS46700.2020.00129.

9 Piotr Berman and Chris Coulston. On-line algorithms for steiner tree problems (extended abstract). In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 344–353, New York, NY, USA, 1997. ACM. doi:10.1145/258533.258618.

10 Marcin Bienkowski, Martin Böhm, Jaroslaw Byrka, Marek Chrobak, Christoph Dürr, Lukáš Folwarczný, Lukasz Jez, Jiri Sgall, Nguyen Kim Thang, and Pavel Veselý. Online algorithms for multi-level aggregation. In *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, pages 12:1–12:17, 2016. doi:10.4230/LIPIcs.ESA.2016.12.

11 Marcin Bienkowski, Jaroslaw Byrka, Marek Chrobak, Lukasz Jez, Dorian Nogneng, and Jirí Sgall. Better approximation bounds for the joint replenishment problem. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 42–54, 2014. doi:10.1137/1.9781611973402.4.

12 Marcin Bienkowski, Artur Kraska, Hsiang-Hsuan Liu, and Pawel Schmidt. A primal-dual online deterministic algorithm for matching with delays. In *Approximation and Online Algorithms – 16th International Workshop, WAOA 2018, Helsinki, Finland, August 23-24, 2018, Revised Selected Papers*, pages 51–68, 2018. doi:10.1007/978-3-030-04693-4_4.

13 Marcin Bienkowski, Artur Kraska, and Pawel Schmidt. A match in time saves nine: Deterministic online matching with delays. In *Approximation and Online Algorithms – 15th International Workshop, WAOA 2017, Vienna, Austria, September 7-8, 2017, Revised Selected Papers*, pages 132–146, 2017. doi:10.1007/978-3-319-89441-6_11.

14 Marcin Bienkowski, Artur Kraska, and Pawel Schmidt. Online service with delay on a line. In *Structural Information and Communication Complexity – 25th International Colloquium, SIROCCO 2018, Ma'ale HaHamisha, Israel, June 18-21, 2018, Revised Selected Papers*, pages 237–248, 2018. doi:10.1007/978-3-030-01325-7_22.

15 Carlos Fisch Brito, Elias Koutsoupias, and Shailesh Vaya. Competitive analysis of organization networks or multicast acknowledgment: How much to wait? *Algorithmica*, 64(4):584–605, 2012. doi:10.1007/s00453-011-9567-5.

16 Niv Buchbinder, Moran Feldman, Joseph (Seffi) Naor, and Ohad Talmon. O(depth)-competitive algorithm for online multi-level aggregation. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1235–1244, 2017. doi:10.1137/1.9781611974782.80.

17 Niv Buchbinder, Kamal Jain, and Joseph Naor. Online primal-dual algorithms for maximizing ad-auctions revenue. In *Algorithms – ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings*, pages 253–264, 2007. doi:10.1007/978-3-540-75520-3_24.

18 Niv Buchbinder, Tracy Kimbrel, Retsef Levi, Konstantin Makarychev, and Maxim Sviridenko. Online make-to-order joint replenishment model: primal dual competitive algorithms. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 952–961, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347186.

19 Rodrigo A. Carrasco, Kirk Pruhs, Cliff Stein, and José Verschae. The online set aggregation problem. In *LATIN 2018: Theoretical Informatics – 13th Latin American Symposium, Buenos Aires, Argentina, April 16-19, 2018, Proceedings*, pages 245–259, 2018. doi:10.1007/978-3-319-77404-6_19.

20 Daniel R. Dooly, Sally A. Goldman, and Stephen D. Scott. TCP dynamic acknowledgment delay: Theory and practice (extended abstract). In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 389–398, 1998. doi:10.1145/276698.276792.

21 Yuval Emek, Shay Kutten, and Roger Wattenhofer. Online matching: haste makes waste! In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 333–344, 2016. doi:10.1145/2897518.2897557.

22 Yuval Emek, Yaacov Shapiro, and Yuyi Wang. Minimum cost perfect matching with delays for two sources. In *Algorithms and Complexity – 10th International Conference, CIAC 2017, Athens, Greece, May 24-26, 2017, Proceedings*, pages 209–221, 2017. doi:10.1007/978-3-319-57586-5_18.

23 Dimitris Fotakis. On the competitive ratio for online facility location. *Algorithmica*, 50(1):1–57, 2008. doi:10.1007/s00453-007-9049-y.

24 Anupam Gupta, Ravishankar Krishnaswamy, and R. Ravi. Online and stochastic survivable network design. *SIAM J. Comput.*, 41(6):1649–1672, 2012. doi:10.1137/09076725X.

25 Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. Caching with time windows. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1125–1138. ACM, 2020. doi:10.1145/3357713.3384277.

26 Makoto Imase and Bernard M. Waxman. Dynamic steiner tree problem. *SIAM J. Discrete Math.*, 4(3):369–384, 1991. doi:10.1137/0404033.

27 Anna R. Karlin, Claire Kenyon, and Dana Randall. Dynamic TCP acknowledgment and other stories about e/(e-1). *Algorithmica*, 36(3):209–224, 2003.

28 Simon Korman. On the use of randomization in the online set cover problem. Master's thesis, Weizmann Institute of Science, 2005.

29 J. Naor, D. Panigrahi, and M. Singh. Online node-weighted steiner tree and related problems. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 210–219, 2011.