Set Cover with Delay – Clairvoyance Is Not Required

Yossi Azar
Tel Aviv University, Israel
azar@tau.ac.il

Ashish Chiplunkar
Indian Institute of Technology Delhi, India
ashishc@iitd.ac.in

Shay Kutten
Technion – Israel Institute of Technology, Haifa, Israel
kutten@technion.ac.il

Noam Touitou
Tel Aviv University
noamtouitou@mail.tau.ac.il

Abstract

In most online problems with delay, clairvoyance (i.e. knowing the future delay of a request upon its arrival) is required for polylogarithmic competitiveness. In this paper, we show that this is not the case for set cover with delay (SCD) – specifically, we present the first non-clairvoyant algorithm, which is $O(\log n \log m)$-competitive, where n is the number of elements and m is the number of sets. This matches the best known result for the classic online set cover (a special case of non-clairvoyant SCD). Moreover, clairvoyance does not allow for significant improvement – we present lower bounds of $\Omega(\sqrt{\log n})$ and $\Omega(\sqrt{\log m})$ for SCD which apply for the clairvoyant case.

In addition, the competitiveness of our algorithm does not depend on the number of requests. Such a guarantee on the size of the universe alone was not previously known even for the clairvoyant case – the only previously-known algorithm (due to Carrasco et al.) is clairvoyant, with competitiveness that grows with the number of requests.

For the special case of vertex cover with delay, we show a simpler, deterministic algorithm which is 3-competitive (and also non-clairvoyant).

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

Keywords and phrases Set Cover, Delay, Clairvoyant

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.8

Related Version A full version of the paper is available at `https://arxiv.org/abs/1807.08543`.

Funding *Yossi Azar*: Supported in part by the Israel Science Foundation (grant No. 1506/16).
Ashish Chiplunkar: Supported by Pankaj Gupta Young Faculty fellowship.
Shay Kutten: Supported by the Ministry of Science and Technology together with the JSPS, as well as the BSF.

1 Introduction

In problems with delay, requests are released over a timeline. The algorithm must serve these requests by performing some action, which incurs a cost. While a request is pending (i.e. has been released but not yet served), the request accumulates delay cost. The goal of the algorithm is to minimize the sum of costs incurred in serving requests and the delay costs of requests.

28th Annual European Symposium on Algorithms (ESA 2020).
Editors: Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders; Article No. 8; pp. 8:1–8:21
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

There are two variants of such problems. In the clairvoyant variant, the delay function of a request (which determines the delay accumulation of that request over time) is revealed to the algorithm upon the release of the request. In the non-clairvoyant variant, at any point in time the algorithm is only aware of delay accumulated up to that point.

Most online problems with delay do not admit competitive non-clairvoyant algorithms – namely, there exist lower bounds for competitiveness which are polynomial in the size of the input space (e.g. the number of points in the metric space upon which requests are released). This is the case, for example, in the multilevel aggregation problem [10, 14], the facility location problem [7] and the service with delay problem [6]. However, these problems do admit *clairvoyant* algorithms which are polylog-competitive. An additional such problem is that of matching with delay (presented in [20]), for which the only known algorithms are for when all requests have an identical, linear delay function (and are in particular clairvoyant). Rather surprisingly, we show in this paper that the online set cover with delay problem *does* admit a competitive non-clairvoyant algorithm.

In the online set cover with delay problem (SCD), a universe of elements and a family of sets are known in advance. Requests then arrive over time on the elements, and accumulate delay cost until served by the algorithm. The algorithm may choose to buy a set at any time, at a cost specific to that set (and known in advance to the algorithm). Buying a set serves all pending requests (requests released but not yet served) on elements of that set; future requests on those elements, that have yet to arrive when the set is bought, must be served separately at a future point in time. For that reason, a set may be bought an unbounded number of times over the course of the algorithm. The goal of an algorithm is to minimize the sum of the total buying cost and the total delay cost. We note that one could also consider the problem in which sets are bought permanently, and cover future requests; however, it is easy to see that this problem is equivalent to the classic online set cover, and is thus of no additional interest. In the full version of this paper, we show that this problem is a special case of our problem.

As a variant of set cover, the SCD problem is very general, capturing many problems. Nevertheless, we give two possible motivations for the problem.

Summoning experts. consider a company which occasionally requires the help of experts. At any time, a problem may arise which requires external assistance in some field, and negatively impacts the performance of the company while unresolved. At any time, the company may hire any one of a set of experts to come to the company, solve all standing problems in that expert's fields of expertise, and then leave. The company aims to minimize the total cost of hiring experts, as well as the negative impact of unresolved problems.

Cluster-covering with delay. suppose antennas generate data requests over time, which must be satisfied by an external server, with a cost to leaving a request pending. To satisfy an update request by an antenna, the server sends the data to a center antenna which transmits it at a certain radius, at a certain cost (which depends on the center antenna and the radius). All requests on antennas inside that radius are served by that transmission. This problem is a covering problem with delay costs, which can be described in terms of SCD. As an SCD instance, the elements are the antennas, and the sets are pairs of a center antenna and a (reasonable) transmission radius (the number of sets is quadratic in the number of antennas).

Carrasco et al. [15] provided a clairvoyant algorithm for the SCD problem, which is $O(\log N)$ competitive (where N is the number of the requests). However, as the number of requests becomes large, the competitive ratio of this algorithm tends to infinity – even for a very small universe of elements and sets. Thus, this algorithm does not provide a guarantee in terms of the underlying input space, as we would like. In addition, their algorithm has exponential running time (through making oracle calls which compute optimal solutions for NP-hard problems).

In this paper, we present the first algorithm for SCD which is polylog-competitive in the size of the universe, which is also the first algorithm for the problem which runs in polynomial time. Surprisingly, this algorithm is also non-clairvoyant, showing that the SCD problem admits non-clairvoyant competitive algorithms. Our randomized algorithm is $O(\log n \log m)$-competitive, where n is the number of elements and m is the number of sets. In this paper, we show a reduction from the classic online set cover to SCD, which implies (due to [27]) that our upper bound is tight for a polynomial-time, non-clairvoyant algorithm for SCD.

While our algorithm is optimal for the non-clairvoyant setting, one could wonder if there exists a *clairvoyant* algorithm which performs significantly better – especially considering the aforementioned problems, in which the gap between the clairvoyant and non-clairvoyant cases is huge. We answer this in the negative – namely, we show lower bounds of $\Omega(\sqrt{\log n})$ and $\Omega(\sqrt{\log m})$ on the competitiveness of any randomized clairvoyant algorithm, showing that there is no large gap which clairvoyant algorithms could bridge. Nevertheless, a quartic gap still exists, e.g. in the case that $m = \Theta(n)$. We conjecture that the gap is in fact quadratic, and leave this as an open problem.

In this paper, we also consider the problem of vertex cover with delay (denoted VCD). In the VCD problem, vertices of graph are given, with a buying cost associated with each vertex. Requests on the edges of the graph arrive over time, and accumulate delay until served by buying a vertex touching the edge (at the cost of that vertex's price). This problem corresponds to SCD where every element is in exactly two sets.

1.1 Our Results

We denote as before the number of elements in an SCD instance by n, and the number of sets by m. We also define $k \leq m$ to be the maximum number of sets to which a specific element may belong. We consider arbitrary (nondecreasing) continuous delay functions (not only linear functions).

In this paper, we present:

1. An $O(\log k \cdot \log n)$-competitive, randomized, non-clairvoyant algorithm for SCD, based on rounding of a newly-designed $O(\log k)$-competitive algorithm for the fractional version of SCD. The competitive ratio of this algorithm is tight – we show a reduction from (classic) online set cover to non-clairvoyant SCD.
2. Lower bounds of $\Omega(\sqrt{\log k})$ and $\Omega(\sqrt{\log n})$ on competitiveness for **clairvoyant** SCD, showing that clairvoyance cannot improve competitiveness beyond a quadratic factor.
3. A simple, deterministic, non-clairvoyant algorithm for vertex cover with delay (VCD) which is 3-competitive.

Our randomized algorithm for SCD is the first (sub-polynomial competitive) non-clairvoyant algorithm for this problem. Moreover, this is the first algorithm which is polylog-competitive in the size of the universe (even among clairvoyant algorithms).

In the process of obtaining our $\Omega(\sqrt{\log k})$ and $\Omega(\sqrt{\log n})$ lower bounds, we in fact obtain an $\Omega(\sqrt{\log m})$ lower bound (which immediately implies $\Omega(\sqrt{\log k})$ since $k \leq m$). The lower bounds also apply for the unweighted setting. These lower bounds improve over the lower bound of $\Omega(\log \log n)$ given in [15][1].

For VCD, while our algorithm is 3-competitive, note that there is a lower bound of 2. The lower bound uses a graph with a single edge which is requested multiple times; this graph corresponds to the TCP acknowledgment problem, analyzed in [19].

▶ Remark 1. While our $O(\log k \cdot \log n)$-competitive algorithm is presented for the case in which the sets and elements are known in advance, it can easily be modified for the case in which each element, as well as which of the sets contain it, becomes known to the algorithm only after the arrival of a request on that element. Moreover, the algorithm can in fact operate in the original setting of Carrasco et al. [15], as it does not need to know the family of sets itself, but rather the family of *restrictions* of the sets to the elements that have already arrived. This can be done through standard doubling techniques applied to $\log n$ and $\log k$ (i.e. squaring of n and k).

1.2 Our Techniques

In the course of designing a non-clairvoyant algorithm for the SCD problem, we also consider a fractional version of SCD. In this version, an algorithm may choose to buy a fraction of a set at any moment. Buying a fraction of a set partially serves requests present on an element of that set, which causes them to accumulate less future delay. As with the original version, a request is only served by fractions bought after its arrival. Hence, the sum of fractions bought for a single set over time is unbounded (i.e. a set may be bought many times).

In the **fractional $O(\log k)$-competitive algorithm**, each request that can be served by a set contributes some amount to the buying of that set. This amount depends exponentially on the delay accumulated by that request, as well as the delay of previous requests. Typically in algorithms with exponential contributions, these contributions are summed. Interestingly, our algorithm instead chooses the maximum of the contributions of the requests as the buying function of the set. The choice of maximum over sum is crucial to the proof (using sum instead of maximum would lead to a linear competitive ratio).

The analysis of this algorithm is based on dual fitting: we first present a linear programming representation of the fractional SCD problem, then use a feasible solution to the dual problem to charge the delay of the algorithm to the optimum. This is the reason for using the maximum in the buying function; each quantity satisfies a different constraint in the dual, and choosing the maximum satisfies all constraints. We then charge the buying cost of the algorithm to $O(\log k)$ times its delay, which concludes the analysis.

Next, we design a randomized competitive algorithm for the integer version of SCD using *2-level* randomized rounding of the fractional algorithm. At the *top level*, we construct a **randomized $O(\log k \cdot \log N)$-competitive algorithm for the integer version**, with N the number of requests. The top-level rounding consists of maintaining for each set a random threshold, and buying the set when the total buying of that set in the *fractional* algorithm exceeds the threshold. In addition, special service of a request is performed in the probabilistically unlikely event that the request is half-served in the fractional algorithm

[1] The lower bound of [15] shows $\Omega(\log N)$-competitiveness, but relies on a universe which is exponentially larger than the number of requests. As they mention in their paper, this therefore translates to an $\Omega(\log \log n)$ lower bound on competitiveness.

but is still pending in the rounding. Since in our problem we may buy a set an unbounded number of times, we require use of multiple subsequent thresholds. To analyze this, we make use of Wald's equation for stopping time.

We add the *bottom level* to improve the $O(\log k \cdot \log N)$-competitive algorithm to a **randomized $O(\log k \cdot \log n)$-competitive algorithm for the integer version**. The bottom level partitions time into phases for each element separately, and aggregates requests on that element that are released in the same phase. The competitive ratio of the resulting algorithm is asymptotically optimal for solving non-clairvoyant SCD in polynomial time, as shown by the reduction from the classic online set cover to non-clairvoyant SCD given in the full version of this paper.

Perhaps the most novel techniques in this paper are used for **the lower bounds of $\Omega(\sqrt{\log k})$ and $\Omega(\sqrt{\log n})$** for the clairvoyant case. The lower bounds are obtained by a recursive construction. Given a recursive instance for which any algorithm has a lower bound on the competitive ratio, we amplify that bound by duplicating every set in the recursive instance into two sets, one slightly more expensive than the other. Both sets perform the same function with respect to the recursive instance, but the algorithm also has an incentive to choose the expensive family of sets, since they serve some additional requests. If the algorithm chooses to buy a lot of expensive sets, the optimum releases another copy of the recursive instance, serviceable only by expensive sets. This forces the algorithm to buy the expensive sets twice; the optimum only buys them once. If, on the other hand, the algorithm chooses the inexpensive sets, it misses the opportunity to serve the additional requests and the recursive instance simultaneously, and must serve them separately.

The recursive description of our construction for the lower bounds is significantly more natural than its iterative description. Few lower bounds in online algorithms have this property – another such lower bound is found in [8].

The 3-competitive deterministic algorithm for VCD is simple and based on counters. This algorithm is only $k+1$ competitive for general SCD, and is thus significantly worse than the previous randomized algorithm that we have shown for general SCD.

1.3 Other Related Work

A different problem called online set cover is considered in [3], in which the algorithm accumulates value for every element that arrives on a bought set, and aims to maximize total value. This problem appears to be fundamentally different from the online set cover in which we minimize cost, in both techniques and results.

The problem of set cover in the online setting has seen much additional work, e.g. in [22, 9, 18, 29, 1]. The set cover problem has also been studied in the streaming model (e.g. [21, 16]), stochastic model (e.g. [24]), dynamic model (e.g. [23]), and in the context of universal algorithms (e.g. [25, 22]) and communication complexity (e.g. [28]).

There are known inapproximability results for the (offline) set cover and vertex cover problems. In [17] it is shown that the offline set cover problem is unlikely to be approximable in polynomial time to within a factor better than $\ln n$. For the offline vertex cover, it is shown in [26] that it is NP hard to approximate within a factor better than 2, assuming the *Unique Games Conjecture*. These results apply to our SCD and VCD problems, as an instance of offline set cover (or vertex cover) can be released at time 0. Of course, these inapproximability results do not constitute lower bounds for the online model, in which unbounded computation is allowed – unlike the information-theoretic lower bound of $\Omega(\sqrt{\log n})$ for SCD which is given in this paper.

The field of online problems with delay over time has been of interest recently. This includes the problems of min-cost perfect matching with delays [20, 5, 2, 12, 11, 4], online service with delay [6, 13, 7] and multilevel aggregation [10, 14, 7].

Paper Organization

In Section 3, we present and analyze a fractional non-clairvoyant algorithm for SCD. In Section 4, we show how to round the previous algorithm in a non-clairvoyant manner to obtain our algorithm for the original (integral) SCD. In Section 5, we show lower bounds for clairvoyant SCD. In the full version of this paper, we show that the algorithm obtained in Section 4 is optimal for the non-clairvoyant case. In Section 6, we give a simple, deterministic, non-clairvoyant algorithm for vertex cover with delay.

2 Preliminaries

We denote the sets by $\{S_i\}_{i=1}^m$, with m the number of sets. We denote by n the number of elements. We define k to be the minimal number for which every element belongs to at most k sets. Requests q_j arrive on the elements. We denote the arrival time of request q_j by r_j, and write (with a slight abuse of notation) $q_j \in S_i$ if the element on which q_j has been released belongs to the set S_i.

Each request q_j has an arbitrary momentary delay function $d_j(t)$, defined for $t \geq r_j$. The accumulated delay of the request at time $t \geq r_j$ is defined to be $\int_{r_j}^{t} d_j(t)\,\mathrm{d}t$. At any time in which a request is pending, its momentary delay is added to the cost of the algorithm; that is, the algorithm incurs a cost of $\int_{r_j}^{\tau_j} d_j(t)\,\mathrm{d}t$ (the accumulated delay of q_j at time τ_j) for every request q_j, where τ_j is the time in which q_j is served. Each set S_i has a price $c(S_i) \geq 1$ which the algorithm must pay when it decides to buy the set. Buying a set serves all pending requests which belong to the set (but does not affect future requests). The *buying cost* of an online algorithm ON is $\mathrm{Cost}_{\mathrm{ON}}^p = \sum_i n_i \cdot c(S_i)$, where n_i is the number of times S_i has been bought by the algorithm. The *delay cost* of ON is $\mathrm{Cost}_{\mathrm{ON}}^d = \sum_j \int_{r_j}^{\tau_j} d_j(t)\,\mathrm{d}t$, where τ_j is the time in which q_j is served by the algorithm (τ_j is ∞ if q_j is never served by the algorithm)[2].

Overall, the cost of ON for the problem is $\mathrm{Cost}_{\mathrm{ON}} = \mathrm{Cost}_{\mathrm{ON}}^p + \mathrm{Cost}_{\mathrm{ON}}^d$

3 The Non-Clairvoyant Algorithm for Fractional SCD

We first describe a fractional relaxation of the (integer) set cover with delay problem. In this fractional relaxation, a set can be bought in parts. A fractional algorithm determines for each set S_i a nonnegative momentary buying function $x_i(t)$. The total buying cost a fractional online algorithm F incurs is $\mathrm{Cost}_F^p = \sum_i c(S_i) \cdot \int_0^\infty x_i(t)\,\mathrm{d}t$.

In the fractional version, a request can be partially served. Under a fractional algorithm F, for any request q_j, and any set S_i such that $q_j \in S_i$, the set S_i covers q_j at a time $t \geq r_j$ by the amount $\int_{r_j}^{t} x_i(t')\,\mathrm{d}t'$ (which is obviously nondecreasing as a function of t). The total amount by which q_j is covered at time t is

$$\gamma_j(t) = \sum_{i|q_j \in S_i} \int_{r_j}^{t} x_i(t')\,\mathrm{d}t'.$$

[2] We solve the more general problem in which the algorithm doesn't have to serve all requests (observe that the adversary can still force the algorithm to serve all requests by adding infinite delay at time infinity). This allows the problem to capture additional problems (e.g. prize-collecting problems, in which a penalty could be paid to avoid serving a specific request)

If at time t we have $\gamma_j(t) \geq 1$, then q_j is considered served, and the algorithm does not incur delay. However, if $\gamma_j(t) < 1$, the algorithm F incurs delay proportional to the uncovered fraction of q_j. Formally, at time t the request q_j incurs $d_j^F(t)$ delay in F, where

$$d_j^F(t) = \begin{cases} d_j(t) \cdot (1 - \gamma_j(t)) & \text{if } \gamma_j(t) < 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

The delay cost of the algorithm is $\mathrm{Cost}_F^d = \sum_j \int_{r_j}^{\infty} d_j^F(t)\,\mathrm{d}t$. The total cost of the fractional algorithm is thus $\mathrm{Cost}_F = \mathrm{Cost}_F^p + \mathrm{Cost}_F^d$.

Description of the algorithm. We now describe an online algorithm called ONF for the fractional problem.

We define a total order $\preceq$ on requests, such that for any two requests q_{j_1}, q_{j_2} if $r_{j_1} < r_{j_2}$ we have $q_{j_1} \prec q_{j_2}$ (we break ties arbitrarily between requests with equal arrival time).

At any time t, the algorithm does the following.

1. For every request q_j, evaluate $d_j^{\mathrm{ONF}}(t)$ by its definition in Equation 3.1.
2. For every set S_i and request $q_j \in S_i$, define
$$D_i^j(t) = \sum_{j' | q_{j'} \in S_i \wedge q_{j'} \preceq q_j} d_{j'}^{\mathrm{ONF}}(t).$$
3. For every set S_i and request $q_j \in S_i$, define
$$x_i^j(t) = \frac{1}{k} \cdot \left(\frac{\ln(1+k)}{c(S_i)} \cdot D_i^j(t) \right) \cdot e^{\frac{\ln(1+k)}{c(S_i)} \int_{r_j}^t D_i^j(t')\,\mathrm{d}t'}.$$
4. Buy every set S_i according to $x_i(t)$, such that
$$x_i(t) = \max_j x_i^j(t).$$

This completes the description of the algorithm.

The intuition for the algorithm is that at any time t, the amount $\int_{r_j}^t D_i^j(t')\,\mathrm{d}t'$ is delay incurred by the algorithm until time t that the optimum possibly avoided by buying S_i at time r_j, and thus the algorithm wishes to minimize this amount. Thus, the request q_j places some "demand" on the algorithm to buy S_i. Since this is true for any $q_j \in S_i$, the algorithm chooses the maximum of the demands as the buying function of S_i.

This demand $x_i^j(t)$ placed on the algorithm by q_j to buy S_i is related to $\int_{r_j}^t D_i^j(t')\,\mathrm{d}t'$. If we wanted to make the total buying proportional to $\int_{r_j}^t D_i^j(t')\,\mathrm{d}t'$, it would sound reasonable to set $x_i^j(t)$ to be its derivative, namely $D_i^j(t)$. However, this would only be $\Omega(k)$-competitive, as demonstrated in Figure 3.1. We thus want the total buying to be proportional to an expression exponential in $\int_{r_j}^t D_i^j(t')\,\mathrm{d}t'$, which underlies the definition of $x_i^j(t)$ in our algorithm.

Denoting $X_i^j(t) = \int_{r_j}^t x_i^j(t')\,\mathrm{d}t'$, note that

$$X_i^j(t) = \frac{1}{k} \cdot \left[e^{\frac{\ln(1+k)}{c(S_i)} \int_{r_j}^t D_i^j(t')\,\mathrm{d}t'} - 1 \right]. \tag{3.2}$$

In the rest of this section, we prove the following theorem.

▶ **Theorem 2.** *The algorithm for fractional SCD described above is $O(\log k)$-competitive.*

We now analyze the algorithm for fractional SCD and prove Theorem 2.

In this figure, there are $k-1$ elements, where each element is contained in k sets of cost 1, one central set (which contains all elements) and $k-1$ peripheral sets (each contains exactly one element). Consider $k-1$ requests, one on each element, all arriving at time 0. Their delay functions are identical, and go to infinity as time progresses.
Consider an algorithm which buys sets linearly to the delay - that is,
$x_i(t) = \max_j D_i^j(t) = \sum_{j|q_j \in S_i} d_j^{\text{ONF}}(t)$. The momentary delay of every request contributes equally to the buying functions of the k containing sets. Thus, the total fraction bought of peripheral sets is exactly $k-1$ times the total fraction bought of the central set. Consider the point in time in which all requests are half-covered (through symmetry, this happens for all requests at the same time, and must happen since the requests gather infinite delay). We have that the central set was bought for a fraction of exactly $\frac{1}{4}$ (which can again be seen through symmetry of the requests and their delay). Thus, the peripheral sets were bought for a fraction of $\frac{k-1}{4}$, for a total of $\frac{k}{4}$. Consider that the optimal solution costs 1, as the optimum buys the central set at time 0.

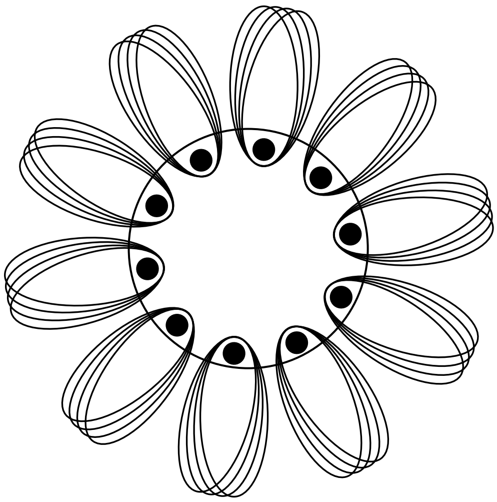

Figure 3.1 Linear Buying $\Omega(k)$ Example.

3.1 Charging Buying Cost to Delay

In this subsection we prove the following lemma.

▶ **Lemma 3.** $Cost^p_{\text{ONF}} \leq 2\ln(1+k) \cdot Cost^d_{\text{ONF}}$.

Proof. The proof is by charging the momentary buying cost at any time t to the $2\ln(1+k)$ times the momentary delay incurred by ONF at t. Let q_j be some request released by time t. For every i such that $q_j \in S_i$, we charge some amount $z_i^j(t)$ to $d_j^{\text{ONF}}(t)$. Denote by j_i the request in S_i such that

$$x_i(t) = x_i^{j_i}(t).$$

If $q_j \preceq q_{j_i}$, we choose

$$z_i^j(t) = \frac{\ln(1+k)}{k} \cdot d_j^{\text{ONF}}(t) \cdot e^{\frac{\ln(1+k)}{c(S_i)} \int_{r_{j_i}}^{t} D_i^{j_i}(t')\,\mathrm{d}t'}.$$

Otherwise, we choose $z_i^j(t) = 0$. Note that for every set S_i we have $\sum_{j|q_j \in S_i} z_i^j(t) = c(S_i) \cdot x_i(t)$, and thus the entire buying cost is charged.

The total buying cost charged to a request q_j at time t is $\sum_{i|q_j\in S_i} z_i^j(t)$. We show that for any j we have

$$\sum_{i|q_j\in S_i} z_i^j(t) \le 2\ln(1+k)\cdot d_j^{\text{ONF}}(t).$$

Summing the previous equation over requests q_j and integrating over time yields the lemma.

If $d_j^{\text{ONF}}(t)=0$ we have $z_i^j(t)=0$ for every i, as required. From now on, we assume that $d_j^{\text{ONF}}(t)>0$.

Denote by $T_j=\{i|q_j\in S_i \text{ and } z_i^j>0\}$. We have

$$\begin{aligned}\sum_{i|q_j\in S_i} z_i^j(t) &= \sum_{i\in T_j} z_i^j(t)\\ &= \ln(1+k)\cdot d_j^{\text{ONF}}(t)\cdot \sum_{i\in T_j}\frac{1}{k}\cdot e^{\frac{\ln(1+k)}{c(S_i)}\int_{r_{j_i}}^t D_i^{j_i}(t')\,\mathrm{d}t'}.\end{aligned}$$

Now note that

$$\begin{aligned}\frac{1}{k}\cdot e^{\frac{\ln(1+k)}{c(S_i)}\int_{r_{j_i}}^t D_i^{j_i}(t')\,\mathrm{d}t'} &= \frac{1}{k}+X_i^{j_i}(t)\\ &\le \frac{1}{k}+\int_{r_{j_i}}^t x_i(t')\,\mathrm{d}t'\\ &\le \frac{1}{k}+\int_{r_j}^t x_i(t')\,\mathrm{d}t'\end{aligned}$$

where the equality is due to equation 3.2, the first inequality is due to the definition of $X_i^{j_i}(t)$ and since $x_i(t)\ge x_i^{j_i}(t)$, and the last inequality is due to $q_j \preceq q_{j_i}$.

Thus

$$\sum_{i|q_j\in S_i} z_i^j(t) \le \ln(1+k)\cdot d_j^{\text{ONF}}(t)\cdot\sum_{i\in T_j}\left(\frac{1}{k}+\int_{r_j}^t x_i(t')\,\mathrm{d}t'\right) \le 2\ln(1+k)\cdot d_j^{\text{ONF}}(t)$$

where the last inequality follows from $|T_j|\le k$, and from $\sum_{i|q_j\in S_i}\int_{r_j}^t x_i(t')\,\mathrm{d}t'\le 1$ (due to the assumption that $d_j^{\text{ONF}}(t)>0$). ◀

3.2 Charging Delay to Optimum

In this subsection, we charge the delay of the algorithm to the optimum via dual fitting.

3.2.1 Linear Programming Formulation

We formulate a linear programming instance for the fractional problem, and observe its dual instance.

Primal. In the primal instance, the variables are:

- $x_i(t)$ for a set S_i and time t, which is the fraction by which the algorithm buys S_i at time t.
- $p_j(t)$ for a request q_j and time $t\ge r_j$, which is the fraction of q_j not covered by bought sets at time t.

The LP instance is therefore:

Minimize:

$$\sum_i \int_0^\infty c(S_i) \cdot x_i(t)\,\mathrm{d}t + \sum_j \int_{r_j}^\infty p_j(t) \cdot d_j(t)\,\mathrm{d}t$$

under the constraints:

$$\forall j, t \geq r_j : p_j(t) + \sum_{i|q_j \in S_i} \int_{r_j}^t x_i(t')\,\mathrm{d}t' \geq 1$$

$$p_j(t) \geq 0\,,\ x_i(t) \geq 0.$$

Dual. Maximize:

$$\sum_j \int_{r_j}^\infty y_j(t)\,\mathrm{d}t$$

under the constraints:

$$\forall i, t : \sum_{j|q_j \in S_i \wedge r_j \leq t} \int_t^\infty y_j(t')\,\mathrm{d}t' \leq c(S_i) \tag{C1}$$

$$\forall j, t \geq r_j : y_j(t) \leq d_j(t) \tag{C2}$$

$$y_j(t) \geq 0.$$

▶ Remark 4. As we chose to consider time as continuous, the linear program described here has an infinite number of variables and constraints. This is merely a choice of presentation, as discretizing time would yield a standard, finite LP. Nevertheless, weak duality for this infinite LP (the only duality property used in this paper) holds (see e.g. [30]).

3.2.2 Charging Delay to Optimum via Dual Fitting

We now charge the delay of the fractional algorithm to the cost of the optimum.

▶ **Lemma 5.** $Cost^d_{\mathrm{ONF}} \leq Cost_{OPT}$.

Proof. The proof is by finding a solution to the dual problem, such that the goal function value of the solution is equal to the delay of the algorithm.

For every request q_j and time t, we assign $y_j(t) = d_j^{\mathrm{ONF}}(t)$. This assignment satisfies that the goal function is the total delay incurred by the algorithm.

Note that the C2 constraints trivially hold, since $d_j^{\mathrm{ONF}}(t) \leq d_j(t)$ for any j, t. Now observe the C1 constraints. For any time t and a set S_i, the resulting C1 constraint is implied by the C1 constraint of time r_j and the set S_i, with q_j being the last request released by time t. We thus restrict ourselves to C1 constraints of time r_j for some j.

For a request q_j and a set S_i, we need to show:

$$\sum_{j'|q_{j'} \in S_i \wedge q_{j'} \preceq q_j} \int_{r_j}^\infty d_{j'}^{\mathrm{ONF}}(t')\,\mathrm{d}t' \leq c(S_i).$$

Using the definition of $D_i^j(t)$, we need to show:

$$\int_{r_j}^{\infty} D_i^j(t)\,\mathrm{d}t \le c(S_i).$$

Define t_0 to be the minimal time (possibly ∞) such that for all $t \ge t_0$ we have $D_i^j(t) = 0$. We must have that $\int_{r_j}^{t_0} x_i(t)\,\mathrm{d}t \le 1$; otherwise, all requests $q_{j'} \in S_i$ such that $q_{j'} \preceq q_j$ will be completed before t_0, in contradiction to t_0's minimality. Thus we have

$$\begin{aligned} 1 &\ge \int_{r_j}^{t_0} x_i(t)\,\mathrm{d}t \ge \int_{r_j}^{t_0} x_i^j(t)\,\mathrm{d}t \\ &= \frac{1}{k}\left[e^{\frac{\ln(1+k)}{c(S_i)}\int_{r_j}^{t_0} D_i^j(t)\,\mathrm{d}t} - 1\right] \end{aligned}$$

where the second inequality is due to the definition of $x_i(t)$, and the equality is due to equation 3.2. This yields

$$(1+k)^{\frac{1}{c(S_i)}\int_{r_j}^{t_0} D_i^j(t)\,\mathrm{d}t} \le 1+k$$

and thus

$$\int_{r_j}^{\infty} D_i^j(t)\,\mathrm{d}t = \int_{r_j}^{t_0} D_i^j(t)\,\mathrm{d}t \le c(S_i)$$

as required. ◀

We can now prove the main theorem.

Proof of Theorem 2. Using Lemmas 3 and 5, we have

$$\begin{aligned} \mathrm{Cost}_{\mathrm{ONF}} &= \mathrm{Cost}^p_{\mathrm{ONF}} + \mathrm{Cost}^d_{\mathrm{ONF}} \\ &\le (2\ln(1+k)+1)\cdot \mathrm{Cost}^d_{\mathrm{ONF}} \\ &\le (2\ln(1+k)+1)\cdot \mathrm{Cost}_{OPT} \end{aligned}$$

as required. ◀

▶ Remark 6. For the more difficult delay model in which a partially served request q_j incurs delay $d_j^{\mathrm{ONF}}(t) = d_j(t)$ instead of $d_j^{\mathrm{ONF}}(t) = d_j(t)\cdot(1-\gamma_j(t))$ in ONF, this algorithm is still $O(\log k)$ competitive against the fractional optimum in the easier delay model. The proof is identical.

4 Randomized Algorithm for SCD by Rounding

In this section, we describe a non-clairvoyant, polynomial-time randomized algorithm which is $O(\log k \cdot \log n)$-competitive for integral SCD. Our randomized algorithm uses randomized rounding of the fractional algorithm of Section 3. We describe the rounding in two steps. First, we show a somewhat simpler algorithm which is $O(\log k \cdot \log N)$-competitive. We then modify this algorithm to obtain a $O(\log k \cdot \log n)$-competitive algorithm.

The rounding of the fractional algorithm of section 3 costs the randomized integral algorithm of this section a multiplicative factor of $\log n$ over that fractional algorithm.

Denote by $x_i(t)$ the fractional buying function in the algorithm ONF of Section 3. For a request q_j, we denote by S_{i_j} the least expensive set containing q_j; that is, $i_j = \arg\min_{i|q_j\in S_i} c(S_i)$.

For every request q_j, we denote the total covering of q_j at time t in ONF by $\gamma_j(t)$, where

$$\gamma_j(t) = \sum_{i|q_j\in S_i} \int_{r_j}^{t} x_i(t')\,\mathrm{d}t'.$$

We denote by t_j the first time in which $\gamma_j(t) = \frac{1}{2}$.

$O(\log k \cdot \log N)$-Competitive Rounding

We now describe a randomized integral algorithm, called ONR, which is $O(\log k \cdot \log N)$ competitive with respect to the fractional optimum, with N the total number of requests. We assume a-priori knowledge of N for the algorithm.

The randomized integral algorithm runs the fractional algorithm of Section 3 in the background, and thus has knowledge of the function $x_i(t)$ for every i. The algorithm does the following.

1. At time 0:
   a. For every set S_i, choose Λ_i from the range $[0, \frac{1}{2\ln N}]$ uniformly and independently, and set $\tau_i = 0$.
2. At time t:
   a. For every i, if $\int_{\tau_i}^{t} x_i(t')\,\mathrm{d}t' \geq \Lambda_i$ then:
      i. Buy S_i.
      ii. Assign to Λ_i a new value drawn independently and uniformly from $[0, \frac{1}{2\ln N}]$.
      iii. Assign $\tau_i = t$.
   b. If there exists a pending request q_j such that $t \geq t_j$, buy S_{i_j}.

We refer to the buying of sets at Step 2a as "type a", and to the buying of sets at Step 2b as "type b".

The intuition for the randomized rounding scheme is that we would like the probability of buying a set in a certain interval of time to be proportional to the fraction of that set bought by the fractional algorithm in that interval, independently of the other sets. This is achieved by the "type a" buying. However, using "type a" alone is problematic. Consider, for example, a request on an element in k sets, such that the fractional algorithm buys $\frac{1}{k}$ of each of the sets to cover the request. Since the probability of buying a set is independent of other sets, there exists a probability that the randomized algorithm would not buy any of the k sets, leaving the request unserved. This bears possibly infinite delay cost for the rounding algorithm, which is not incurred by the underlying fractional algorithm.

The "type b" buying solves this problem, by serving a pending request *deterministically* when it is covered in the underlying fractional algorithm, through buying the cheapest set containing that request. This special service for the request might be expensive, but its probability is low, yielding low expected cost. This is ensured by the $2\log N$ "speedup" given to the "type a" buying, through choosing the thresholds Λ_i from $[0, \frac{1}{2\ln N}]$ (rather than $[0, 1]$).

▶ **Theorem 7.** *The randomized algorithm for SCD described above is $O(\log k \cdot \log N)$-competitive.*

The proof of Theorem 7 is given in the full version of this paper.

Improved $O(\log k \cdot \log n)$-Competitive Rounding

By modifying the $O(\log k \cdot \log N)$-competitive randomized rounding, we prove the following theorem.

▶ **Theorem 8.** *There exists a non-clairvoyant, randomized $O(\log k \cdot \log n)$-competitive algorithm for SCD.*

The modified rounding algorithm and its analysis appear in the full version of this paper.

5 Lower Bounds for Clairvoyant SCD

In this section, we show $\Omega(\sqrt{\log k})$ and $\Omega(\sqrt{\log n})$ lower bounds on competitiveness for any randomized, clairvoyant algorithm for SCD or fractional SCD. While the lower bounds use instances in which different sets can have different costs, these instances can be modified to obtain instances with identical set costs. This implies that the lower bounds also apply to the unweighted setting. This modification is shown in Subsection 5.2.

This section shows the following theorem.

▶ **Theorem 9.** *Any randomized algorithm for SCD or fractional SCD is both $\Omega(\sqrt{\log k})$-competitive and $\Omega(\sqrt{\log n})$-competitive.*

In proving Theorem 9, we show a lower bound on competitiveness of a deterministic fractional algorithm against an integral optimum. Showing this is enough to prove the theorem, since any randomized online algorithm (fractional or integral) can be converted to a deterministic fractional online algorithm with identical (or lesser) cost. This follows from setting the momentary buying function of a set at a given time to be the expectation of that value in the randomized algorithm. Since the optimum is integral, the bound also holds for integral SCD, as the theorem states. Therefore, we only consider deterministic fractional online algorithms henceforth.

We show our lower bounds by constructing a set of SCD instances, $\{I_i\}_{i=0}^{\infty}$. For each $i \geq 0$, the SCD instance I_i contains 2^i sets and 3^i elements. We show that any algorithm must be $\Omega(\sqrt{i})$-competitive for I_i, which is both $\Omega(\sqrt{\log m})$ and $\Omega(\sqrt{\log n})$. Noting that $k \leq m$, we also have $\Omega(\sqrt{\log k})$ as required.

The instance I_i exists within the time interval $[0, 3^i)$. That is, no request of I_i is released before time 0, and at time 3^i the optimum has served all requests in I_i, and the algorithm has incurred a high enough cost.

We define the sequence $(c_i)_{i=0}^{\infty}$, which is used in the construction of I_i. The sequence is defined recursively, such that $c_0 = 1$ and for any $i \geq 1$, we have that

$$c_i = c_{i-1} + \frac{1}{12 c_{i-1}}.$$

We now describe the recursive construction of the instance I_i. We first describe the universe of I_i, which consists of its sets and elements. We then describe the requests of I_i.

Universe of I_i

For the base instance I_0, the universe consists of a single element e and a single set $S = \{e\}$. We have that $c(S) = 1$.

For $i \geq 1$, the recursive construction of I_i using I_{i-1} is as follows. Denote by E_{i-1} the elements in the universe of I_{i-1}, and by H_{i-1} the family of sets in the universe of I_{i-1}. For the construction of I_i, consider three disjoint copies of E_{i-1} and H_{i-1}. For $l \in \{1, 2, 3\}$, we

This figure shows the universes of I_0, I_1 and I_2. In the figure, each element is a point and the sets are the bodies containing them, where each set has a distinct color. The costs of the sets are also shown in the figure. The figure shows how three copies of the set of elements E_{i-1} (of the instance I_{i-1}) appear in I_i – the copy E^1_{i-1} appears at the top of I_i's visualization, the copy E^2_{i-1} appears at the bottom-left, and the copy E^3_{i-1} appears at the bottom-right.

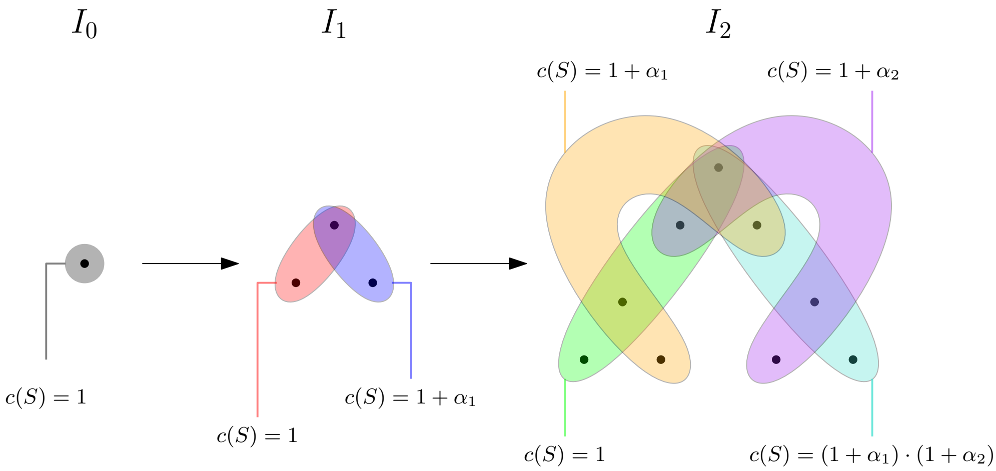

Figure 5.1 The Universes of I_0, I_1 and I_2.

denote by E^l_{i-1} and H^l_{i-1} the l'th copy of E_{i-1} and H_{i-1}, respectively. We denote by S^l the copy of the set $S \in H_{i-1}$ in H^l_{i-1}. Similarly, we denote by e^l the copy of an element $e \in E_{i-1}$ in E^l_{i-1}.

The universe of I_i consists of:

- The elements $E_i = E^1_{i-1} \cup E^2_{i-1} \cup E^3_{i-1}$.
- The family of sets $H_i = \mathcal{T}_1 \cup \mathcal{T}_2$, where $\mathcal{T}_1$ and $\mathcal{T}_2$ are defined below.

We define:

- The family of sets $\mathcal{T}_1 = \{S^1 \cup S^2 | S \in H_{i-1}\}$. A set $T \in \mathcal{T}_1$ formed from $S \in H_{i-1}$ has cost $c(T) = c(S)$.
- The family of sets $\mathcal{T}_2 = \{S^1 \cup S^3 | S \in H_{i-1}\}$. A set $T \in \mathcal{T}_2$ formed from $S \in H_{i-1}$ has cost $c(T) = (1 + \alpha_i) \cdot c(S)$, with $\alpha_i = \frac{1}{2c_{i-1}}$.

Requests of I_i

We first describe a type of request used in our construction. Let S be a set such that there exists an element $e \in S$ such that e is in no other set besides S (we call e unique to S). For times a, b such that $a < b$, we define a request $q^b_a(S)$ that can be released at any time $r \leq a$ on an element unique to S, and satisfies:

1. $\int_r^a d_j(t)\,\mathrm{d}t = 0$
2. $\int_r^b d_j(t)\,\mathrm{d}t \geq c(S)$.

▶ Remark 10. For the degenerate case of set cover with deadlines, when observing a request with deadline at time b, it can be said to accumulate 0 delay until any time before b, and infinite delay until time b. Therefore, deadline requests can function as $q^b_a(S)$ requests. Since all requests used in our construction are $q^b_a(S)$ requests for some a, b, S, our lower bound applies for set cover with deadlines as well.

To use those $q^b_a(S)$ requests, we require the following proposition, which states that a $q^b_a(S)$ request can be released on every S.

▶ **Proposition 11.** *For every set $T \in H_i$, there exists an element $e \in E_i$ unique to T.*

Proof. By induction on i. For the base case, this holds since there is only a single set with a single element. Assuming the proposition holds for I_{i-1}, we show that it holds for I_i by observing that there exists $S \in H_{i-1}$ such that $T = S^1 \cup S^l$ for $l \in \{2,3\}$. Via induction, there exists an element $e \in E_{i-1}$ such that $e \in S$ and $e \notin S'$ for every $S' \in H_{i-1}$ such that $S' \neq S$. Choosing the element e^l yields the proposition. ◀

Base case of I_0 – at time 0, the request $q^1_0(S)$ is released on the single element e.

Recursive construction of I_i using I_{i-1} – we define $C(I_i)$ to be $\sum_{S\in H_i} c(S)$. We now define the instance I_i:

1. At time 0:
   a. Release $q^{3^i}_{2\cdot 3^{i-1}}(T)$ for every $T \in \mathcal{T}_2$.
   b. Release I_{i-1} on the elements E^1_{i-1} (see Remark (a)).
2. At time 3^{i-1}:
   a. If the algorithm has bought sets of $\mathcal{T}_2$ at a total cost of at least $\frac{1}{2} \cdot (1+\alpha_i) \cdot C(I_{i-1})$, release $(1+\alpha_i)I_{i-1}$ on the elements E^3_{i-1} (see Remark (c)).
   b. Otherwise, release I_{i-1} on the elements of E^2_{i-1} (see Remark (b)).

The construction of I_i includes releasing copies of I_{i-1} on the elements E^l_{i-1}, for $l \in \{1,2,3\}$. The following remarks make this well-defined.

▶ Remark (a). **The $\boldsymbol{I_{i-1}}$ on $\boldsymbol{E^1_{i-1}}$:** every set $S \in H_{i-1}$ forms two sets in H_i, which are $T_1 = S^1 \cup S^2 \in \mathcal{T}_1$ and $T_2 = S^1 \cup S^3 \in \mathcal{T}_2$. The I_{i-1} construction on E^1_{i-1} treats buying either of these sets as buying the set S. That is, it treats the sum of the momentary buying of T_1 and of T_2 as the momentary buying of S.

▶ Remark (b). **The $\boldsymbol{I_{i-1}}$ on $\boldsymbol{E^2_{i-1}}$:** in this instance, for every set $S \in H_{i-1}$, the I_{i-1} construction treats buying $T_1 = S^1 \cup S^2 \in \mathcal{T}_1$ as buying S.

▶ Remark (c). **The scaled $\boldsymbol{(1+\alpha_i)I_{i-1}}$ on $\boldsymbol{E^3_{i-1}}$:** similarly to Remark 5, in this instance, for every set $S \in H_{i-1}$, the I_{i-1} construction treats buying $T_2 = S^1 \cup S^3 \in \mathcal{T}_2$ as buying S. In addition, since the sets of $\mathcal{T}_2$ are $(1+\alpha_i)$-times more expensive than the original sets of H_{i-1}, the delays of the jobs in I_{i-1} are also scaled by $1+\alpha_i$ in order to maintain the I_{i-1} instance. We denote this scaled instance by $(1+\alpha_i)I_{i-1}$.

5.1 Analysis of Lower Bounds

We show that any online fractional algorithm is at least c_i competitive on I_i with respect to the integral optimum.

▶ **Lemma 12.** *The optimal integral algorithm can serve I_i by time 3^i with no delay cost by buying every set in H_i exactly once, for a total cost of $C(I_i)$.*

Proof. Via induction on i. For the base case of $i = 0$, the optimal algorithm buys the single set S at time 0 and pays $c(S) = C(I_0)$. Now, for $i \geq 1$, suppose the optimum can serve the instance I_{i-1} according to the lemma. We observe the optimum in I_i according to the cases in the release of I_i:

Case 2a: In this case, the optimum could have served I_{i-1} on E^1_{i-1} by time 3^{i-1} by buying each set of $\mathcal{T}_1$ exactly once, with no delay cost. It could then serve $(1+\alpha_i)I_{i-1}$ on E^3_{i-1} by time $2 \cdot 3^{i-1}$ by buying each set of $\mathcal{T}_2$ exactly once, with no delay cost. Since the optimum has bought all of $\mathcal{T}_2$, the requests released on step 1a have also been served before incurring delay. The lemma thus holds for this case.

Case 2b: In this case, the optimum could have served I_{i-1} on E^1_{i-1} by time 3^{i-1} by buying each set of $\mathcal{T}_2$ exactly once, with no delay cost. It could then serve I_{i-1} on E^2_{i-1} by time $2 \cdot 3^{i-1}$ by buying each set of $\mathcal{T}_1$ exactly once, with no delay cost. Since the optimum has bought all of $\mathcal{T}_2$, the requests released on step 1a have again been served before incurring delay. The lemma thus holds for this case as well. ◀

We now analyze the cost of the algorithm.

▶ **Lemma 13.** *Any online algorithm has a cost of at least $c_i \cdot C(I_i)$ on I_i by time 3^i.*

Proof. By induction on i.

For $i = 0$, observe the algorithm at time 1. Denoting by Γ_S the total buying of the single set S by the algorithm by time 1, the algorithm has a cost of at least

$$c(S) \cdot \Gamma_S + (1 - \Gamma_S) \cdot \int_0^1 d_{q_0^1(S)}(t)\,\mathrm{d}t \geq c(S) = C(I_0)$$

where the inequality is due to the definition of $q_0^1(S)$. This finishes the base case of the induction.

For the case that $i \geq 1$, assume that the lemma holds for $i-1$. We show that it holds for i.

Fix any algorithm for I_i. We denote by Γ the total buying cost of the algorithm in the time interval $[0, 3^{i-1})$ for sets of $\mathcal{T}_2$. We again split into cases according to the chosen branch in the construction of I_i.

Case 2a: In this case we have $\Gamma \geq \frac{1}{2} \cdot (1+\alpha_i) \cdot C(I_{i-1})$. From the definition of the first I_{i-1} released, the adversary is oblivious to whether a copy of $S \in H_{i-1}$ came from $\mathcal{T}_1$ or $\mathcal{T}_2$. Using the induction hypothesis, any online algorithm for this instance incurs a cost of at least $c_{i-1} \cdot C(I_{i-1})$ by time 3^{i-1}, including the algorithm in which buying sets from $\mathcal{T}_2$ are replaced with buying the equivalent sets from $\mathcal{T}_1$. Such a modified online algorithm would cost $\frac{\alpha_i}{1+\alpha_i}\Gamma$ less than the current algorithm, which is at least $\frac{\alpha_i}{2} \cdot C(I_{i-1})$. Therefore, the algorithm pays at least $(c_{i-1} + \frac{\alpha_i}{2}) \cdot C(I_{i-1})$ in the interval $[0, 3^{i-1})$.
As for the second instance $(1+\alpha_i)I_{i-1}$, the algorithm must pay at least $(1+\alpha_i) \cdot c_{i-1} \cdot C(I_{i-1})$ by time $2 \cdot 3^{i-1}$ via induction.
Overall, the algorithm pays by time 3^i at least

$$\begin{aligned}
\left(\left(c_{i-1} + \frac{\alpha_i}{2}\right) \cdot C(I_{i-1})\right) &+ ((1+\alpha_i) \cdot c_{i-1} \cdot C(I_{i-1})) \\
&= \left((2+\alpha_i)c_{i-1} + \frac{\alpha_i}{2}\right) \cdot C(I_{i-1}) \\
&= c_{i-1} \cdot C(I_i) + \frac{\alpha_i}{2} \cdot C(I_{i-1}) \\
&\geq \left(c_{i-1} + \frac{\alpha_i}{6}\right) \cdot C(I_i) \\
&= \left(c_{i-1} + \frac{1}{12 c_{i-1}}\right) \cdot C(I_i)
\end{aligned}$$

where the inequality is due to $C(I_i) = (2+\alpha_i)C(I_{i-1}) \leq 3C(I_{i-1})$.

Case 2b: In this case we have $\Gamma < \frac{1}{2} \cdot (1+\alpha_i) \cdot C(I_{i-1})$. For the first I_{i-1} instance, the algorithm pays at least $c_{i-1} \cdot C(I_{i-1}) + \Gamma \cdot \frac{\alpha_i}{1+\alpha_i}$ by time 3^{i-1}, similar to the previous case. For the second I_{i-1} instance, released on E^2_{i-1}, the algorithm must pay via induction at least $c_{i-1} \cdot C(I_{i-1})$ by time $2 \cdot 3^{i-1}$. Since sets of $\mathcal{T}_2$ do not satisfy requests in this instance, this cost is either in buying sets of $\mathcal{T}_1$ or in delay of requests from that I_{i-1} instance.

In addition to the two I_{i-1} instances, due to the $q^{3^i}_{2\cdot 3^{i-1}}(S)$ requests released in step 1a, the algorithm has a cost of at least $\left(\sum_{T\in\mathcal{T}_2} c(T)\right) - \Gamma = (1+\alpha_i)C(I_{i-1}) - \Gamma$ during the interval $[1, 3)$ in either buying sets of $\mathcal{T}_2$ in order to finish these requests, or in delay by those requests (using a similar argument to that in the base case). Overall, the algorithm has a cost of at least

$$\begin{aligned}
\left(c_{i-1} \cdot C(I_{i-1}) + \Gamma \cdot \frac{\alpha_i}{1+\alpha_i}\right) &+ (c_{i-1} \cdot C(I_{i-1})) + ((1+\alpha_i)C(I_{i-1}) - \Gamma) \\
&= (2c_{i-1} + 1 + \alpha_i) \cdot C(I_{i-1}) - \frac{1}{1+\alpha_i}\Gamma \\
&\geq (2c_{i-1} + 1 + \alpha_i) \cdot C(I_{i-1}) - \frac{1}{2}C(I_{i-1}) \\
&= \left(2c_{i-1} + \frac{1}{2} + \alpha_i\right) \cdot C(I_{i-1}) \\
&= \left((2+\alpha_i)c_{i-1} + \frac{1}{2} + (1-c_{i-1})\alpha_i\right) \cdot C(I_{i-1}) \\
&= c_{i-1} \cdot C(I_i) + \left(\frac{1}{2} + \frac{1}{2c_{i-1}} - \frac{1}{2}\right) \cdot C(I_{i-1}) \\
&\geq \left(c_{i-1} + \frac{1}{6c_{i-1}}\right) \cdot C(I_i) \geq c_i \cdot C(I_i)
\end{aligned}$$

where the fourth equality and the second inequality are due to $C(I_i) = (2+\alpha_i)C(I_{i-1}) \leq 3C(I_{i-1})$, and the fourth equality uses the definition of α_i. ◀

Proof of Theorem 9. Lemmas 12 and 13 immediately imply that any deterministic fractional algorithm is at least c_i-competitive on I_i with respect to the integral optimum. Solving the recurrence in the definition of c_i, we have that $c_i = \Omega(\sqrt{i})$. To observe this, note that for every i, the first index $i' \geq i$ such that $c_{i'} \geq c_i + 1$ is at most $O(c_i)$ larger than i. Since $k \leq m = 2^i$ and $n = 3^i$, this provides lower bounds of $\Omega(\sqrt{\log k})$ and $\Omega(\sqrt{\log n})$ for deterministic algorithms for fractional SCD. As stated before, this implies the same lower bound for randomized algorithms for both SCD and fractional SCD. ◀

5.2 Reduction to Unweighted

The lower bound above uses weighted instances, in which sets may have different costs. In this subsection, we describe how to convert a weighted instance to an unweighted instance, in which all set costs are equal. This conversion maintains both the $\Omega(\sqrt{\log k})$ and $\Omega(\sqrt{\log n})$ lower bounds on competitiveness. The conversion consists of creating multiple copies of each element, and converting each original set to multiple sets of cost 1. The cost of the original set affects the cardinality of the new sets, such that a set with higher cost turns into smaller sets of cost 1.

We suppose that the costs of all sets are integer powers of 2. This can easily be achieved by rounding the costs to powers of 2 (losing a factor of 2 in the lower bound), and then scaling the instance (both delays and buying costs) by the inverse of the lowest cost.

Denote by $C = 2^M$ the largest cost in the instance. The universe of the unweighted instance is the following:

- For each element e in the original instance, we have C elements in the unweighted instance, denoted by $e_0, ..., e_{C-1}$.
- For each set S, we have $c(S)$ sets in the unweighted instance, labeled $S_0, ..., S_{c(S)-1}$.
- We have that $e_i \in S_j$ if and only if both $e \in S$ and $i \equiv j \mod c(S)$.

Whenever a request q_j arrives in the original instance on an element e with delay function $d_j(t)$, C requests $q_{j,0}, ..., q_{j,C-1}$ arrive in the unweighted instance on the elements $e_0, ..., e_{C-1}$ respectively. For each $0 \leq l \leq C-1$, the request $q_{j,l}$ has the delay function $d_{j,l}(t) = \frac{d_j(t)}{C}$.

For the instance I_i described above, we consider its unweighted conversion, denoted by I'_i. Any fractional online algorithm for I'_i can be converted to a fractional online algorithm for I_i with a cost which is at most that of the original algorithm. This is done by setting the buying function of a set S in I_i to the average of the buying functions of $S_0, ..., S_{c(S)-1}$.

In addition, the integral optimum described in the analysis of I_i can be modified to an integral optimum for I'_i with identical cost. This is by converting each buying of the set S in I_i to buying the sets $S_0, ..., S_{c(S)-1}$ in I'_i.

The aforementioned facts imply that any fractional algorithm is $\Omega(\sqrt{i})$ competitive on I'_i. Note that the parameter k is the same for I_i and I'_i, implying $\Omega(\sqrt{\log k})$-competitiveness on I'_i. In addition, denoting by n' the number of elements in I'_i, we have that $n' = C \cdot n$. Observing the construction of I_i, we have that $n = 3^i$ and $C \leq 2^i$ (Using the fact that $(1 + \alpha_j) \leq 2$ for any j). Therefore, $\log n' \leq \log(6^i)$, yielding that $i = \Omega(\log n')$, and a $\Omega(\sqrt{\log n'})$ lower bound on competitiveness for I'_i.

6 Vertex Cover with Delay

In this section, we show a 3-competitive deterministic algorithm for VCD. Recall that VCD is a special case of SCD with $k = 2$, where k is the maximum number of sets to which an element can belong. In fact, we show a $(k + 1)$-competitive deterministic algorithm for SCD, which is therefore 3-competitive for VCD. Recall that since the TCP acknowledgment problem is a special case of VCD with a single edge, the lower bound of 2-competitiveness for any deterministic algorithm on the TCP acknowledgment problem (shown in [19]) applies to VCD as well.

The $(k + 1)$-competitive algorithm for SCD, ON, is as follows.

1. For every set S, maintain a counter $z(S)$ of the total delay incurred by ON over requests on elements in S (all $z(S)$ are initialized to 0).
2. If for any S, we have that $z(S) = c(S)$:
   a. Buy S.
   b. Zero the counter $z(S)$.

We denote by $z(S, t)$ the value of $z(S)$ at time t. We prove the following theorem.

▶ **Theorem 14.** *The algorithm* ON *for SCD has a competitive ratio of $k + 1$. In particular,* ON *is 3-competitive for VCD.*

▶ **Lemma 15.** *The cost of the algorithm is at most $k + 1$ times its delay cost.*

Proof. It is sufficient to bound the buying cost in terms of the delay cost. For each purchase of a set S, $z(S)$ must increase from 0 to $c(S)$. A delay for a request contributes to the increase of at most k counters. Thus, the buying cost is at most k times the delay cost. ◀

We are left to bound the delay cost of the algorithm by the adversary's cost.

▶ **Lemma 16.** *For any set S, let T be a subset of the requests on elements of S such that all requests of T are unserved at time t. Then we have $\sum_{j|q_j \in T} \int_t^\infty d_j^{\mathrm{ON}}(t')\,\mathrm{d}t' \leq c(S)$.*

Proof. Denote by $\hat{t}$ the first time in which all requests in T are served. We have that

$$\sum_{j|q_j\in T}\int_t^\infty d_j^{\mathrm{ON}}(t')\,\mathrm{d}t' = \sum_{j|q_j\in T}\int_t^{\hat{t}} d_j^{\mathrm{ON}}(t')\,\mathrm{d}t'.$$

At time t, we have $z(S,t)\geq 0$. Observe that the algorithm never bought S in the time interval $[t,\hat{t})$. Thus, at any time $t''\in[t,\hat{t})$ we have that

$$z(S,t'') = z(S) + \sum_{j|q_j\in T}\int_t^{t''} d_j^{\mathrm{ON}}(t')\,\mathrm{d}t'.$$

Observe that $z(S,t'')<c(S)$, otherwise the algorithm would have bought S at t', serving all requests in T, in contradiction to the definition of $\hat{t}$. Therefore $\sum_{j|q_j\in T}\int_t^{t''} d_j^{\mathrm{ON}}(t')\,\mathrm{d}t' < c(S)$. The claim follows as t'' approaches $\hat{t}$. ◀

▶ **Lemma 17.** *The delay cost of the algorithm is at most the adversary's cost.*

Proof. We construct a solution to the dual LP from section 3, with a goal function which is the delay cost of the algorithm. This charges the delay cost of the algorithm to the fractional optimum, and thus to the integer optimum as well.

Specifically, we set $y_j(t)=d_j^{\mathrm{ON}}(t)$ for every j,t. Obviously, the C2 constraints hold. In order to show that the C1 constraint for a set S_i and a time t holds, observe that any request $q_j\in S_i$ served in ON before time t has $d_j^{\mathrm{ON}}(t')=0$ for all $t'\geq t$. Using Lemma 16 for the requests unserved at t concludes the proof. ◀

Proof of theorem 14. The proof of the theorem results directly from lemmas 16 and 17. ◀

Note that this algorithm's competitive ratio is indeed as bad as $k+1$. Consider, for example, a single request in k sets with unit costs, which the optimum solves with cost 1 and the algorithm has cost $k+1$.

References

1 Sebastian Abshoff, Christine Markarian, and Friedhelm Meyer auf der Heide. Randomized online algorithms for set cover leasing problems. In *8th COCOA*, pages 25–34, 2014.

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 *Proceedings of the APPROX/RANDOM*, pages 1:1–1:20, 2017. `doi:10.4230/LIPIcs.APPROX-RANDOM.2017.1`.

3 Baruch Awerbuch, Yossi Azar, Amos Fiat, and Frank Thomson Leighton. Making commitments in the face of uncertainty: How to pick a winner almost every time. In *Proceedings of the Twenty-Eighth STOC*, pages 519–530, 1996. `doi:10.1145/237814.238000`.

4 Y. Azar and A. Jacob-Fanani. Deterministic Min-Cost Matching with Delays. *ArXiv e-prints*, June 2018. `arXiv:1806.03708`.

5 Yossi Azar, Ashish Chiplunkar, and Haim Kaplan. Polylogarithmic bounds on the competitiveness of min-cost perfect matching with delays. In *28th SODA*, pages 1051–1061, 2017. `doi:10.1137/1.9781611974782.67`.

6 Yossi Azar, Arun Ganesh, Rong Ge, and Debmalya Panigrahi. Online service with delay. In *49th STOC*, pages 551–563, 2017. `doi:10.1145/3055399.3055475`.

7 Yossi Azar and Noam Touitou. General framework for metric optimization problems with delay or with deadlines. In *Proceedings of the 60th IEEE FOCS*, pages 60–71, 2019. `doi:10.1109/FOCS.2019.00013`.

8 Nikhil Bansal and Ho-Leung Chan. Weighted flow time does not admit o(1)-competitive algorithms. In *Proceedings of the Twentieth SODA*, pages 1238–1244, 2009. URL: `http://dl.acm.org/citation.cfm?id=1496770.1496904`.

9 Kshipra Bhawalkar, Sreenivas Gollapudi, and Debmalya Panigrahi. Online set cover with set requests. In *Proceedings of the APPROX/RANDOM*, pages 64–79, 2014.

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 *Proceedings of the 24th ESA*, pages 12:1–12:17, 2016. `doi:10.4230/LIPIcs.ESA.2016.12`.

11 Marcin Bienkowski, Artur Kraska, Hsiang-Hsuan Liu, and Pawel Schmidt. A primal-dual online deterministic algorithm for matching with delays. *CoRR*, abs/1804.08097, 2018. `arXiv:1804.08097`.

12 Marcin Bienkowski, Artur Kraska, and Pawel Schmidt. A match in time saves nine: Deterministic online matching with delays. In *15th WAOA*, pages 132–146, 2017. `doi:10.1007/978-3-319-89441-6_11`.

13 Marcin Bienkowski, Artur Kraska, and Pawel Schmidt. Online service with delay on a line. In *24th SIROCCO*, 2018.

14 Niv Buchbinder, Moran Feldman, Joseph (Seffi) Naor, and Ohad Talmon. O(depth)-competitive algorithm for online multi-level aggregation. In *Twenty-Eighth SODA*, pages 1235–1244, 2017. `doi:10.1137/1.9781611974782.80`.

15 Rodrigo A. Carrasco, Kirk Pruhs, Cliff Stein, and José Verschae. The online set aggregation problem. In *Proceedings of the LATIN 2018:*, pages 245–259, 2018. `doi:10.1007/978-3-319-77404-6_19`.

16 Amit Chakrabarti and Anthony Wirth. Incidence geometries and the pass complexity of semi-streaming set cover. In *Proceedings of the Twenty-Seventh SODA*, pages 1365–1373, 2016. `doi:10.1137/1.9781611974331.ch94`.

17 Irit Dinur and David Steurer. Analytical approach to parallel repetition. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 624–633. ACM, 2014. `doi:10.1145/2591796.2591884`.

18 Stefan Dobrev, Jeff Edmonds, Dennis Komm, Rastislav Královic, Richard Královic, Sacha Krug, and Tobias Mömke. Improved analysis of the online set cover problem with advice. *Theor. Comput. Sci.*, 689:96–107, 2017.

19 Daniel R. Dooly, Sally A. Goldman, and Stephen D. Scott. TCP dynamic acknowledgment delay: Theory and practice. In *Proceedings of the Thirtieth STOC*, pages 389–398, 1998. `doi:10.1145/276698.276792`.

20 Yuval Emek, Shay Kutten, and Roger Wattenhofer. Online matching: haste makes waste! In *Proceedings of the 48th STOC*, pages 333–344, 2016. `doi:10.1145/2897518.2897557`.

21 Yuval Emek and Adi Rosén. Semi-streaming set cover - (extended abstract). In *Proceedings of the 41st ICALP*, pages 453–464, 2014. `doi:10.1007/978-3-662-43948-7_38`.

22 Fabrizio Grandoni, Anupam Gupta, Stefano Leonardi, Pauli Miettinen, Piotr Sankowski, and Mohit Singh. Set covering with our eyes closed. *SIAM J. Comput.*, 42(3):808–830, 2013.

23 Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Debmalya Panigrahi. Online and dynamic algorithms for set cover. In *Proceedings of the 49th STOC*, pages 537–550, 2017. `doi:10.1145/3055399.3055493`.

24 Sungjin Im, Viswanath Nagarajan, and Ruben van der Zwaan. Minimum latency submodular cover. *ACM Trans. Algorithms*, 13(1):13:1–13:28, 2016. `doi:10.1145/2987751`.

25 Lujun Jia, Guolong Lin, Guevara Noubir, Rajmohan Rajaraman, and Ravi Sundaram. Universal approximations for tsp, steiner tree, and set cover. In *37th STOC*, pages 386–395, 2005. `doi:10.1145/1060590.1060649`.

26 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2-epsilon. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008. `doi:10.1016/j.jcss.2007.06.019`.

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

28 Noam Nisan. The communication complexity of approximate set packing and covering. In *Proceedings of the ICALP*, pages 868–875, 2002. doi:10.1007/3-540-45465-9_74.

29 Ashwin Pananjady, Vivek Kumar Bagaria, and Rahul Vaze. The online disjoint set cover problem and its applications. In *2015 IEEE INFOCOM*, pages 1221–1229, 2015.

30 Thomas W. Reiland. Optimality conditions and duality in continuous programming ii. the linear problem revisited. *Journal of Mathematical Analysis and Applications*, 1980.