Set Cover and Vertex Cover with Delay

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

Ashish Chiplunkar  
ashish.chiplunkar@gmail.com  
École Polytechnique Fédérale de Lausanne

Shay Kutten  
kutten@ie.technion.ac.il  
Technion - Israel Institute of Technology

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

Abstract

The set cover problem is one of the most fundamental problems in computer science. We present the problem of online set cover with delay (SCD). A family of sets with costs and a universe of elements are known in advance. Requests then arrive over time on the elements, and each request accumulates delay cost until served by the algorithm through buying a containing set. A request can only be served by sets that are bought after the request’s arrival, and thus a set may be bought an unbounded number of times over the course of the algorithm. This property sets SCD apart from previous considerations of set cover in the online setting, in which there are no delays, elements are covered immediately, and sets stay bought permanently. This allows SCD to describe an unbounded process, with an unlimited number of requests for any given universe.

For the SCD problem, we show an $O(\log k \cdot \log n)$-competitive randomized algorithm, where $n$ is the number of elements and $k$ is the maximum number of sets containing any single element. We also show a lower bound of $\Omega(\sqrt{\log k})$ and $\Omega(\sqrt{\log n})$ on the competitiveness of any algorithm for SCD. For the special case of Vertex Cover with Delay (VCD), we show a simple 3-competitive deterministic algorithm. The $O(\log k \cdot \log n)$-competitive algorithm is based on exponential weights combined with the max operator (in contrast to most algorithms employing exponential weights, which use summation). The lower bound is described by a recursive construction.
1 Introduction

The set cover problem is one of the most fundamental problems in computer science. Given a family of sets and a universe of elements, the objective is to find the smallest subfamily of sets such that their union equals the universe (the smallest cover). In the weighted version, each set has some positive cost, with the objective of minimizing the total cost of sets in the cover.

In the classic online version of set cover, considered in [2], an online algorithm is given a family of sets and a universe of elements in advance. Some subset of the elements of the universe then arrives, one element after the other. If an arriving element is not in any set already bought by the algorithm, the algorithm must immediately buy a set containing that element. The objective is to minimize the total cost of the sets bought. This version of online set cover is a terminating process; once all elements are covered, the instance is over. In particular, buying a set more than once or releasing an element more than once is meaningless. Therefore, this problem cannot describe an unbounded process.

We present the problem of online set cover with delay (SCD). A family of sets and a universe of elements are again 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. Buying a set only serves pending requests released prior to buying the set, and thus future requests must be served separately. For that reason, a set may be bought an unbounded number of times over the course of the algorithm.

As a possible motivation for the problem, 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.

In the course of designing an online 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 may be unbounded (certainly not \( \leq 1 \)).

While introduced as a tool in solving the integer problem, the fractional version may be useful in its own right for some applications. For example, the elements may correspond to wireless devices, and a set may correspond to a broadcasting station, such that some of the wireless devices are in range of the station. A request on a wireless device is for a news item, and requires turning on the device’s transceiver until the news item is received, and then turning it off. Buying the set may mean that the station transmits the recent news to every member now listening. A fractional buying of the set may correspond to the set transmitting a part of the news. There may be a probability that this part already contains the news item needed by the member, so it can already turn its transceiver off. The fraction of the service then corresponds to the probability the needed item is received.

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

For an instance of SCD (set cover with delay), we denote the number of given sets by $m$ and the number of elements in the universe by $n$. We let $k \leq m$ be the maximum number of sets to which a specific element may belong.

In this paper, we present:

- An $O(\log k)$-competitive algorithm for the fractional version of SCD.
- An $O(\log k \cdot \log n)$-competitive randomized algorithm for the integer version of SCD, based on randomized rounding of the algorithm in the previous bullet.
- A lower bound of $\Omega(\sqrt{\log k})$ and $\Omega(\sqrt{\log n})$ on the competitiveness of any algorithm (deterministic or randomized) for both the fractional and integer versions of SCD. Note that as a lower bound for the online model, this lower bound applies to algorithms with unbounded computational power.
- For VCD (vertex cover with delay) we design a simple and natural 3-competitive deterministic algorithm. Note that the generalization of our 3-competitive algorithm to general SCD is as bad as $(k + 1)$-competitive, and thus our former algorithms are needed.

All of our algorithms work for arbitrary continuous accumulated delay functions (not only for linear functions). Note that it can also be used for other functions, such as deadline functions or step functions, through approximating them by continuous functions.

In addition to those results, we show an integrality gap between the fractional and integer versions of SCD, which shows that our rounding scheme for the $O(\log k \cdot \log n)$-competitive algorithm is nearly optimal.

In the process of obtaining our $\Omega(\sqrt{\log k})$ and $\Omega(\sqrt{\log n})$ lower bound, we also obtain $\Omega(\sqrt{\log m})$ (which immediately implies $\Omega(\sqrt{\log k})$ since $k \leq m$).

For VCD, note that there is a lower bound of 2 for a graph with a single edge; this graph corresponds to the TCP acknowledgement problem, analyzed in [16].

1.2 Our Techniques

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 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, as an intermediary step, we construct a randomized $O(\log k \cdot \log N)$-competitive algorithm for the integer version in subsection [4.1], with $N$ the number of requests. This algorithm is based on randomized rounding of the fractional algorithm. The 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
of starvation. 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 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 in subsection 4.2 by modifying the rounding process. The main modi-
fication is giving each element a partition of time into phases, and aggregating requests on that element
that are released in the same phase.

The lower bound of $\Omega(\log k)$ and $\Omega(\log n)$ is by a recursive construction. Given a recursed instance for
which any algorithm has a lower bound on the competitive ratio, we amplify that bound by duplicating any
set in the recursed instance into two sets, one slightly more expensive than the other. Both sets perform
the same function with respect to the recursed 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 recursed instance, servicable 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 oppurtunity to serve
the additional requests and the recursed instance simultaneously, and must serve them separately.

Our lower bound’s recursive description 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 based on simple 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 [4], 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.

Another version of the online set cover problem is in the fully dynamic model, presented in [20], in which
elements arrive and also depart. Here the goal is to maintain a set cover which approximates the current
optimal solution, while being allowed to make only a small number of changes to the maintained cover.
Specifically, they achieve $O(\log n)$ competitive algorithm with amortized constant number of changes
for each arrival or departure on an element. This extends the results of [2], where elements only arrive
and never depart. This version of the problem is very different from our problem, since their model has
no delays (one needs to react immediately after every change). In addition, in their model if an arriving
element is already covered by a set then there is no need to make any change, while in our model it starts
to accumulate delay and would eventually need to be covered again (previously-bought sets that cover this
element are not useful). Furthermore, our model does not allow changes, since buying a set is irrevocable.

The problem of set cover in the online setting has seen much additional work. Some of it can be found in
[19, 9, 15, 22, 1].

There are known inapproximability results for the (offline) set cover and vertex cover problems. In [18] it
is shown that the offline set cover problem cannot be approximated in polynomial time to within a factor
better than $\ln n$. For the offline vertex cover, it is shown in [21] that no approximation to within a factor
better than 2 is possible in polynomial time. 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.

The field of online problems with delay over time has been of interest recently. The problem of min-cost perfect matching with delays was presented in [17]. In this problem, requests arrive on points of a metric space, and must be matched to other requests, accumulating delay while unmatched. Matching a pair of requests costs the distance between the points in the metric space. This problem has been studied in [6, 13, 12, 11, 5].

Another problem is online service with delay, presented in [7], in which a server (or servers) receives requests on points in a metric space, each with an arbitrary delay function, and must move to these points in order to serve the requests. The algorithm aims to minimize the sum of the delay incurred and the total moving distance. This problem generalizes the classic $k$-server problem, and demonstrates the added difficulty of such generalizations; while the 1-server problem is trivially 1-competitive, for online service with delay using a single server only a $O(\log^4 n)$-competitive randomized algorithm is known, with $n$ the number of points in the metric space. The special case where the metric space is a line is studied in [13]. Another special case of this problem called online multi-level aggregation has been studied in [10, 14].

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 $S_i$.

**Integer version:** 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 $\int_{r_j}^t d_j(t)dt$. 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 $\int_{r_j}^{t_j} d_j(t)dt$ (the accumulated delay of $q_j$ at time $t_j$) for every request $q_j$, where $t_j$ 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 existing requests which belong to the set (but does not affect future requests). The **buying cost** of an algorithm $ON$ is $\text{Cost}_{ON}^{p} = \sum_i n_i \cdot c(S_i)$, where $n_i$ is the number of times $S_i$ has been bought by the algorithm. The **delay cost** of $ON$ is $\text{Cost}_{ON}^{d} = \sum_j \int_{t_j}^{t_j'} d_j(t)dt$, where $t_j$ is the time in which $q_j$ is served by the algorithm.

Overall, the cost of $ON$ for the problem is

$$\text{Cost}_{ON} = \text{Cost}_{ON}^{p} + \text{Cost}_{ON}^{d}$$

**Fractional version:** In the fractional relaxation of the standard integer version, a set can be bought in parts. A fractional algorithm determines for each set $S_i$ a continuous momentary buying function $x_i(t)$. The total buying cost a fractional algorithm $ONF$ incurs is $\text{Cost}_{ONF}^{p} = \sum_i c(S_i) \cdot \int_0^{\infty} x_i(t)dt$.

In the fractional version, a request can be partially served. Under a fractional algorithm $ONF$, 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 $\int_{r_j}^{t_j} x_i(t')dt'$. The total amount by which $q_j$ is covered at time $t$ is

$$\gamma_j(t) = \sum_{i | q_j \in S_i} \int_{r_j}^{t_j} x_i(t')dt'$$
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 $ONF$ incurs delay proportional to the uncovered fraction of $q_j$. Formally, at time $t$ the request $q_j$ incurs $d_{ONF}^j(t)$ delay in $ONF$, where

$$d_{ONF}^j(t) = \begin{cases} 
    d_j(t) \cdot (1 - \gamma_j(t)) & \text{if } \gamma_j(t) < 1 \\
    0 & \text{otherwise}
\end{cases}$$  \hspace{1cm} (2.1)

The delay cost of the algorithm is $\text{Cost}_{ONF}^d = \sum_j \int_{r_j}^{\infty} d_{ONF}^j(t) dt$. The total cost of the fractional algorithm is thus

$$\text{Cost}_{ONF} = \text{Cost}_{p_{ONF}} + \text{Cost}_{d_{ONF}}$$

3 The Algorithm for Fractional SCD

In this section, we show an 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 $q_{j_1} \preceq q_{j_2}$ (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 $d_{ONF}^j(t)$ by its definition in Equation 2.1
2. For every set $S_i$ and request $q_j \in S_i$, define $D_i^j(t) = \sum_{j' | q_{j'} \in S_i \land q_{j'} \preceq q_j} d_{ONF}^{j'}(t)$.
3. For every set $S_i$ and request $q_j$, define

$$x_i^j(t) = \frac{1}{k} \cdot \left( \frac{\ln(1 + k)}{c(S_i)} \cdot D_i^j(t) \right) \cdot e^{\frac{\ln(1 + k)}{c(S_i)} \int_{r_j}^{t} D_i^{j'}(t') dt'}$$

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.

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

$$X_i^j(t) = \frac{1}{k} \cdot \left[ e^{\frac{\ln(1 + k)}{c(S_i)} \int_{r_j}^{t} D_i^{j'}(t') dt'} - 1 \right]$$  \hspace{1cm} (3.1)

In the following subsections, we prove the following theorem.

**Theorem 1.** The algorithm for fractional SCD described above is $O(\log k)$-competitive.

3.1 Charging Buying Cost to Delay

In this subsection we prove the following lemma.
Lemma 2. $\text{Cost}^p_{\text{ONF}} \leq 2 \ln(1 + k) \cdot \text{Cost}^d_{\text{ONF}}$

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 $\text{ONF}$ at $t$. Let $q_j$ be some request released by time $t$. For every $x_i$ such that $q_j \in S_i$, we charge some amount $z^j_i(t)$ to $d^\text{ONF}_j(t)$. Denote by $j_i$ the request in $S_i$ such that

$$x_i(t) = x^j_i(t)$$

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

$$z^j_i(t) = \frac{\ln(1 + k)}{k} \cdot d^\text{ONF}_j(t) \cdot e^{\frac{\ln(1+k)}{c(S_i)} \int_{r_{j_i}}^t D^i_j(t') dt'}$$

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

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

$$\sum_{i \in S_i} z^j_i(t) \leq 2 \ln(1 + k) \cdot d^\text{ONF}_j(t)$$

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

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

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

$$\sum_{i \in T_j} z^j_i(t) = \ln(1 + k) \cdot d^\text{ONF}_j(t) \cdot \sum_{i \in T_j} \frac{1}{k} \cdot e^{\frac{\ln(1+k)}{c(S_i)} \int_{r_{j_i}}^t D^i_j(t') dt'}$$

Now note that

$$\frac{1}{k} \cdot e^{\frac{\ln(1+k)}{c(S_i)} \int_{r_{j_i}}^t D^i_j(t') dt'} = \frac{1}{k} + X^j_i(t)$$

$$\leq \frac{1}{k} + \int_{r_{j_i}}^t x_i(t') dt'$$

$$\leq \frac{1}{k} + \int_{r_j}^t x_i(t') dt'$$

where the equality is due to equation 3.1, the first inequality is due to the definition of $X^j_i(t)$ and since $x_i(t) \geq x^j_i(t)$, and the last inequality is due to $j \preceq j_i$.

Thus

$$\sum_{i \in T_j} z^j_i(t) \leq \ln(1 + k) \cdot d^\text{ONF}_j(t) \cdot \sum_{i \in T_j} \left(\frac{1}{k} + \int_{r_j}^t x_i(t') dt'\right) \leq 2 \ln(1 + k) \cdot d^\text{ONF}_j(t)$$

where the last inequality follows from $|T_j| \leq k$, and from $\sum_{i \in T_j} \int_{r_j}^t x_i(t') dt' \leq 1$ (due to the assumption that $d^\text{ONF}_j(t) > 0$).
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 \( t \), which is the fraction of \( q_j \) not covered by bought sets at time \( t \).

The LP instance is therefore:

Minimize:
\[
\sum_i \int_0^\infty c(S_i)x_i(t)dt + \sum_j \int_0^\infty p_j(t) \cdot d_j(t)dt
\]

under the constraints:
\[
\forall j,t: p_j(t) + \sum_{i|q_j \in S_i} \int_{r_j}^t x_i(t')dt' \geq 1
\]
\[
p_j(t) \geq 0, x_i(t) \geq 0
\]

Dual

Maximize:
\[
\sum_j \int_0^\infty y_j(t)dt
\]

under the constraints:
\[
\forall i,t: \sum_{j|i|q_j \in S_i \land r_j \leq t} \int_t^\infty y_j(t')dt' \leq c(S_i)
\]
\[
\forall j,t: y_j(t) \leq d_j(t)
\]
\[
y_j(t) \geq 0
\]

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 3. \( Cost_{ONF}^d \leq Cost_{OPT} \)
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 $y_j(t) = d_{ONF}^j(t)$. This assignment satisfies that the goal function is the total delay incurred by the algorithm.

Note that the C2 constraints trivially hold, since $d_{ONF}^j(t) \leq d_j(t)$ for any $j, t$. Now observe the C1 constraints. For any time $t$ and a set $S_i$, the resulting C1 constraint is contained in 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:

$$\sum_{j' | q_{j'} \in S_i \land q_{j'} \preceq q_j} \int_{r_j}^{\infty} d_{ONF}^j(t') dt' \leq c(S_i)$$

Using the definition of $D^j_i(t)$, we need to show:

$$\int_{r_j}^{\infty} D^j_i(t) dt \leq c(S_i)$$

Define $t_0$ to be the minimal time (possibly $\infty$) such that for all $t \geq t_0$ we have $D^j_i(t) = 0$. We must have that $\int_{r_j}^{t_0} x_i(t) dt \leq 1$; 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

$$1 \geq \int_{r_j}^{t_0} x_i(t) dt \geq \int_{r_j}^{t_0} \hat{x}_i(t) dt$$

$$\geq \frac{1}{k} \left[ \frac{\text{ln}(1+k)}{c(S_i)} \int_{r_j}^{t_0} D^j_i(t) dt - 1 \right]$$

where the second inequality is due to the definition of $x_i(t)$, and the last inequality is due to equation 3.1. This yields

$$(1 + k) \frac{1}{c(S_i)} \int_{r_j}^{t_0} D^j_i(t) dt \leq 1 + k$$

and thus

$$\int_{r_j}^{\infty} D^j_i(t) dt = \int_{r_j}^{t_0} D^j_i(t) dt \leq c(S_i)$$

as required.

We can now prove the main theorem.

Proof. (of Theorem 1) Using Lemmas 2 and 3, we have

$$\text{Cost}_{ONF} = \text{Cost}_{ONF}^p + \text{Cost}_{ONF}^d$$

$$\leq (2 \ln(1 + k) + 1) \cdot \text{Cost}_{ONF}^d$$

$$\leq (2 \ln(1 + k) + 1) \cdot \text{Cost}_{OPT}$$

as required.

Remark 4. For the more difficult delay model in which a partially served request $q_j$ incurs delay $d_{ONF}^j(t) = d_j(t)$ instead of $d_{ONF}^j(t) = d_j(t) \cdot (1 - \gamma_j(t))$ 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 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. In appendix \( A \), we show an integrality gap which shows that our rounding is nearly optimal.

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

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

\[
\gamma_j(t) = \sum_{i \in S_i} \int_{t_j}^{t} x_i(t') dt'
\]

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

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

We now describe a randomized integral algorithm \( ONR \) 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 \( \Lambda_i \) 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 \( \int_{t_j}^{t} x_i(t') dt' \geq \Lambda_i \) then:
      i. Buy \( S_i \).
      ii. Assign to \( \Lambda_i \) 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 \( S_{ij} \).

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".

In this subsection, we prove the following theorem.

**Theorem 5.** The randomized algorithm for SCD described above is \( O(\log k \cdot \log N) \)-competitive.
Lemma 6. \( \mathbb{E}[\text{Cost}_{ONR}^a] \leq 4 \ln N \cdot \text{Cost}_{ONF}. \)

Proof. To show the lemma, fix any set \( S_i \). We observe the values chosen for \( \Lambda_i \) in the algorithm as a sequence \((\Lambda_i^l)_{l=1}^\infty\) of independent random variables, taken uniformly from \([0, \frac{1}{2 \ln N}]\). Whenever the algorithm buys \( S_i \) via "type a", it reveals the next element of the sequence. Denoting by \( s \) the number of times \( S_i \) is "type a" bought, we have that for every \( l \) the indicator variable \( 1_{s+1 \geq l} \) and \( \Lambda_i^l \) are independent (the value of \( \Lambda_i^l \) does not affect whether the algorithm reveals it). Since the elements of the sequence are equidistributed, we can use the general version of Wald’s equation to obtain:

\[
\mathbb{E} \left[ \sum_{l=1}^{s+1} \Lambda_i^l \right] = \mathbb{E}[s+1] \cdot \mathbb{E}[\Lambda_i^1] \geq \frac{\mathbb{E}[s]}{4 \ln N} + \frac{1}{4 \ln N} \quad \text{(4.1)}
\]

Denoting by \( t' \) the last time that \( S_i \) was "type a" bought, we also know that

\[
\sum_{l=1}^{s} \Lambda_i^l = \int_0^{t'} x_i(t) dt \leq \int_0^\infty x_i(t) dt
\]

since all revealed thresholds but \( \Lambda_i^{s+1} \) have been surpassed by \( x_i(t) \). Therefore

\[
\mathbb{E} \left[ \sum_{l=1}^{s+1} \Lambda_i^l \right] = \mathbb{E} \left[ \sum_{l=1}^{s} \Lambda_i^l \right] + \mathbb{E}[\Lambda_i^{s+1}]
\leq \int_0^\infty x_i(t) dt + \frac{1}{4 \ln N}
\]

Combining this with equation (4.1) yields

\[
\mathbb{E}[s] \leq 4 \ln N \cdot \int_0^\infty x_i(t) dt
\]

and thus

\[
\mathbb{E}[c(S_i) \cdot s] \leq 4 \ln N \cdot c(S_i) \cdot \int_0^\infty x_i(t) dt
\]

Note that the total "type a" buying cost of \( S_i \) is \( c(S_i) \cdot s \), while the buying cost of \( S_i \) in ONF is \( c(S_i) \cdot \int_0^\infty x_i(t) dt \). Summing the previous inequality over all \( S_i \) therefore yields the lemma. \( \square \)

Lemma 7. \( \text{Cost}_{ONR}^d \leq 2 \cdot \text{Cost}_{ONF}. \)

Proof. Due to the “type b” buying, if a request \( q_j \) is pending in ONR at time \( t \), we have that \( \gamma_j(t) \leq \frac{1}{2} \).

Thus \( d_{ONF}^j(t) \geq \frac{1}{2} \cdot d_j(t) \), and therefore the ONF always incurs at least half the delay cost of ONR. This yields the lemma. \( \square \)

It remains to bound the total "type b" buying. For any request \( q_j \) and time \( t \geq r_j \), we define the event \( A_j^l \), which is the event that \( q_j \) has not been served in ONR by time \( t \).

Lemma 8. For any request \( q_j \), with \( A_j^l \) as defined above, we have

\[
\Pr(A_j^l) \leq \frac{1}{N}.
\]
Corollary 9. $\mathbb{E}[\text{Cost}_{ONR}^b] \leq \text{Cost}_{ONF}$
Proof. We define \( j^* = \arg \max_j c(S_{ij}) \). We have that

\[
\mathbb{E}[\text{Cost}_{ONR}^b] = \sum_j c(S_{ij}) \cdot Pr(A_{ij}^j) \\
\leq \frac{1}{N} \sum_j c(S_{ij}) \\
\leq \frac{1}{N} \sum_j c(S_{ij^*}) = c(S_{ij^*})
\]

where the first equality is due to linearity of expectation, the first inequality is due to Lemma 8. Now note that since \( ONF \) serves all requests, it also serves \( q_{j^*} \), at a total buying cost of at least \( c(S_{ij^*}) \). Thus \( \text{Cost}_{ONF} \geq c(S_{ij^*}) \), which concludes the proof.

We now prove the main theorem.

**Proof.** (of Theorem 5) Combining Lemmas 6 and 7 with Corollary 9 yields:

\[
\mathbb{E}[\text{Cost}_{ONR}] = \mathbb{E}[\text{Cost}_{ONR}^p + \text{Cost}_{ONR}^d] \\
= \mathbb{E}[\text{Cost}_{ONR}^p] + \mathbb{E}[\text{Cost}_{ONR}^b] + \mathbb{E}[\text{Cost}_{ONR}^d] \\
\leq (4 \ln N + 3) \cdot \text{Cost}_{ONF} = O(\log N) \cdot \text{Cost}_{ONF}
\]

Since \( ONF \) is \( O(\log k) \) competitive with respect to the fractional optimum, we get that \( ONR \) is \( O(\log N \cdot \log k) \) competitive with respect to the fractional optimum, and in particular the integral optimum.

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

In this subsection, we show how to modify the \( O(\log k \cdot \log N) \)-competitive randomized rounding shown in subsection 4.1 to yield a \( O(\log k \cdot \log n) \)-competitive randomized algorithm. The intuition behind the modifications is removing the dependency on the number of requests by aggregating requests on the same element. Specifically, we discretize time into intervals, such that requests on the same element that arrive in the same interval are aggregated. Instead of having a threshold time for “type b” buying for every request, we have a threshold time for every interval.

**Definition 10.** For every element \( e \), we define threshold times, spaced by \( ONF \) buying a constant fraction of sets containing \( e \). Formally, for every element \( e \), we define the threshold time \( t_e^l \) for \( l \in \mathbb{N} \) to be the first time for which \( \int_0^{t_e^l} \left( \sum_{i \in S} x_i(t) \right) dt = \frac{l}{4} \).

Denote by \( s_e \) the index of the last threshold time for \( e \). Using the definition of \( t_e^s \), we have

\[
\int_0^{s_e} \left( \sum_{i \in S} x_i(t) \right) dt \geq \frac{s_e}{4}
\]

(4.2)

For simplicity, we denote \( t_e^0 = 0 \). Define \( R_l^e \) for \( 0 \leq l \leq s_e - 1 \) to be the set of requests released on \( e \) in the interval \( [t_e^l, t_e^{l+1}] \). Note that no request is released outside of some \( R_l^e \) – if a request is released on element \( e \) after \( t_e^{s_e} \), it would require set buying by \( ONF \) which would create three new threshold times, in contradiction to \( s_e \)'s definition. For the same reason, \( R_{s_e-2}, R_{s_e-1} \) are empty.
If at time $t$ all the requests of $R^e_i$ have been served, we say that $R^e_i$ has been served. Otherwise, $R^e_i$ is unserved at time $t$.

We modify the $O(\log k \cdot \log N)$ algorithm shown in subsection 4.1 as follows:

1. The “type a” thresholds $\Lambda_i$ are now drawn from $U \left(0, \frac{1}{2 \ln n}\right)$ (using $n$ instead of $N$).
2. "Type b" buying is changed to the following rule – for every element $e$ and every $l$, if $R^e_l$ remains unserved until $t_{l+3}$, we buy $S_e$.

Note that $t_{l+3}$ in (2) is well defined since $R_{s-1}, R_{s-2}$ are empty.

We prove the following theorem.

**Theorem 11.** The modified randomized algorithm for SCD described above is $O(\log k \cdot \log n)$-competitive.

As in subsection 4.1, we define $\text{Cost}_{ONR}^a$ and $\text{Cost}_{ONR}^b$ to be the “type a” buying cost and the “type b” buying cost of the algorithm, respectively.

**Lemma 12.** $\mathbb{E}[\text{Cost}_{ONR}^a] \leq 4 \ln n \cdot \text{Cost}_{ONF}$

**Proof.** The proof is identical to that of Lemma 6.

For every $R^e_i$, we also define $\Gamma^e_i(t)$ for $t \geq t^e_{i+1}$, which is the fraction of $e$ covered by ONF from time $t^e_{i+1}$:

$$\Gamma^e_i(t) = \sum_{i|e \in S_i} \int_{t^e_{i+1}}^t x_i(t') dt'$$

**Proposition 13.** For $q_j \in R^e_i$, we have that $\gamma_j(t_{l+1}) \leq \frac{1}{4}$.

**Proof.** Otherwise, the fractional algorithm has bought a total fraction of more than $\frac{1}{4}$ of sets containing $e$ in $[t^e_{l+1}, t^e_{l+1}]$, a contradiction to the definition of threshold times. 

**Lemma 14.** If a request $q_j$ is pending in the randomized algorithm at time $t$, then

$$\gamma_j(t) \leq \frac{3}{4}$$

**Proof.** Choose $R^e_i$ such that $q_j \in R^e_i$. If $t \leq t^e_{i+1}$, the lemma results from Proposition 13 and we’re done. Otherwise, $t > t^e_{i+1}$ and therefore $\gamma_j(t) = \gamma_j(t^e_{i+1}) + \Gamma^e_i(t)$. Since $q_j$ is pending at $t$, we have that $R^e_i$ is unserved at $t$. This implies that $t \leq t^e_{i+3}$. From the definition of threshold times, $\sum_{i|e \in S_i} \int_{t^e_{i+1}}^{t^e_{i+3}} x_i(t') dt' \leq \frac{1}{2}$ and thus $\Gamma^e_i(t) \leq \frac{1}{2}$. Therefore

$$\gamma_j(t) = \gamma_j(t^e_{i+1}) + \Gamma^e_i(t) \leq \frac{1}{4} + \frac{1}{2} = \frac{3}{4}$$

where the inequality uses Proposition 13.

**Corollary 15.** $\text{Cost}_{ONR}^d \leq 4 \cdot \text{Cost}_{ONF}$.

**Proof.** Immediate from the previous lemma.
It remains to bound the expected "type b" buying.

**Proposition 16.** The probability that \( R \) triggers "type b" buying is at most \( \frac{1}{n} \).

**Proof.** It is enough to show that the probability that the algorithm does not perform "type a" buying during \((t_i + 1, t_i + 3)\) is at most \( \frac{1}{n} \). Showing this is identical to the proof of Lemma 8.

**Proposition 17.** The total cost of ONF is at least \( \frac{1}{4} \cdot s_e \cdot c(S_e) \), for any element \( e \).

**Proof.** From Equation 4.2, we have that ONF buys at least \( s_e / 4 \) fraction of sets containing \( e \). Since \( S_e \) is the least expensive set containing \( e \), ONF must have paid a buying cost of at least \( \frac{1}{4} \cdot s_e \cdot c(S_e) \). From the definition of threshold times, and the definition of \( S_e \) as the least expensive set containing \( e \).

**Proposition 18.** For every element \( e \), the total expected "type b" buying cost for that element is at most \( \frac{1}{n} \cdot s_e \cdot c(S_e) \).

**Proof.** Let \( X_i^e \) be the indicator random variable of \( R \) being "type b" bought. The lemma results directly from linearity of expectation and Proposition 16.

**Lemma 19.** \( E[\text{Cost}^b_{ONR}] \leq 4 \cdot \text{Cost}_{ONF} \).

**Proof.** We fix \( e^* = \arg\max_e (s_e \cdot c(S_e)) \). Proposition 18 implies that the expected "type b" buying cost is at most:

\[
\sum_{e} \frac{1}{n} \cdot s_e \cdot c(S_e) \leq \frac{1}{n} \cdot \sum_{e} s_e \cdot c(S_e) = s_e^* \cdot c(S_e^*) \leq 4 \cdot \text{Cost}_{ONF}
\]

where the first inequality is from the definition of \( e^* \), and the last inequality is from Proposition 17. This concludes the proof.

We now prove the main theorem.

**Proof.** (of Theorem 11) Using Lemmas 12, 15, and 19, we get:

\[
E[\text{Cost}^a_{ONR} + \text{Cost}^b_{ONR} + \text{Cost}^d_{ONR}] \leq (4 \ln n + 8) \cdot \text{Cost}_{ONF}
\]

which proves the theorem.

## 5 Lower Bound

In this section, we show \( \Omega(\sqrt{\log k}) \) and \( \Omega(\sqrt{\log n}) \) lower bounds on competitiveness for any randomized online algorithm for SCD or fractional SCD.

We show a \( \Omega(\sqrt{\log m}) \) lower bound with \( m \) the number of sets, noting that \( m \geq k \). For \( i \geq 0 \), we create an SCD instance \( I_i \), that contains \( 2^i \) sets and \( 3^i \) elements. 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.

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 \( q^b_i \) that can be released at any time \( r_j \leq a \) on an element unique to \( S \), and satisfies:
1. \( \int_a^b d_j(t) dt = 0 \)
2. \( \int_a^b d_j(t) dt \geq c(S) \)

We define the sequence \( (c_i)_{i=0}^\infty \) recursively, such that \( c_0 = 1 \) and for any \( i \geq 1 \), we have that

\[ c_i = c_{i-1} + \frac{1}{12c_{i-1}} \]

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** \( I_i \):

**Base case of** \( I_0 \) – for the base instance \( I_0 \), the universe consists of a single element \( e \) and a single set \( S = \{ e \} \). We have that \( c(S) = 1 \).

**Recursive construction of** \( I_i \) **using** \( I_{i-1} \) – denote by \( E_{i-1} \) the elements in the universe of \( I_{i-1} \), and by \( S_{i-1} \) 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 \( S_{i-1} \). For \( l \in \{1, 2, 3\} \), we denote by \( E_i^l \) and \( S_i^l \) the \( l \)’th copy of \( E_{i-1} \) and \( S_{i-1} \), respectively. We denote by \( S_i^l \) the copy of the set \( S \in S_{i-1} \) in \( S_i^l \). Similarly, we denote by \( e_i^l \) the copy of an element \( e \in E_{i-1} \) in \( E_i^l \).

We define:

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

We now fully describe the universe of \( I_i \). The universe of \( I_i \) consists of:

- The elements \( E_i = E_i^1 \cup E_i^2 \cup E_i^3 \).
- The family of sets \( S_i = \mathcal{T}_1 \cup \mathcal{T}_2 \).

**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 \( q_{a}^b(S) \) that can be released at any time \( r_j \leq a \) on an element unique to \( S \), and satisfies:

1. \( \int_a^b d_j(t) dt = 0 \)
2. \( \int_a^b d_j(t) dt \geq c(S) \)

To use those \( q_{a}^b(S) \) requests, we require the following proposition, which states that a \( q_{a}^b(S) \) request can be released on every \( S \).

**Proposition 20.** For every set \( T \in S_i \), there exists an element \( e \in E_i \) unique to \( T \).
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.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure5.1.png}
\caption{The Universes of $I_0$, $I_1$ and $I_2$}
\end{figure}

**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 $S \in S_{i-1}$ 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 $S' \in S_{i-1}$ such that $S' \neq S$. Choosing the element $e^l$ yields the proposition. \hfill \blackqed

**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 $\sum_{S \in S_i} c(S)$. We now define the instance $I_i$:

1. At time 0:
   1.1 Release $q^3_{2,3i-1}(T)$ for every $T \in T_2$.
   1.2 Release $I_{i-1}$ on the elements $E^1_{i-1}$ (see Remark (a)).

2. At time $3^{i-1}$:
   2.1 If the algorithm has bought sets of $T_2$ 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)).
   2.2 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 $I_{i-1}$ on $E^1_{i-1}$: every set $S \in S_{i-1}$ forms two sets in $S_i$, which are $T_1 = S^1 \cup S^2 \in T_1$ and $T_2 = S^1 \cup S^3 \in T_2$. 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 $S$. 

17
Remark (b). The $I_{i-1}$ on $E^{2}_{i-1}$: in this instance, for every set $S \in S_{i-1}$, the $I_{i-1}$ construction treats buying $T_1 = S^1 \cup S^2 \in T_1$ as buying $S$.

Remark (c). The scaled $(1 + \alpha_i)I_{i-1}$ on $E^{3}_{i-1}$: similarly to Remark 5 in this instance, for every set $S \in S_{i-1}$, the $I_{i-1}$ construction treats buying $T_2 = S^1 \cup S^3 \in T_2$ as buying $S$. In addition, since the sets of $T_2$ are $(1 + \alpha_i)$-times more expensive than the original sets of $S_{i-1}$, 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 $(1 + \alpha_i)I_{i-1}$.

Theorem 21. Any randomized algorithm for SCD or fractional SCD is both $\Omega(\sqrt{\log k})$-competitive and $\Omega(\sqrt{\log n})$-competitive.

We prove Theorem 21 in the following subsection.

5.1 Analysis

In proving Theorem 21 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 cost, by replacing probabilities with their expectations. 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 that any online fractional algorithm is at least $c_i$ competitive on $I_i$ with respect to the integral optimum.

Lemma 22. The optimal integral algorithm can serve $I_i$ by time $3^i$ with no delay cost by buying every set in $S_i$ 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 2.1

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 $T_1$ 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 $T_2$ exactly once, with no delay cost. Since the optimum has bought all of $T_2$, the requests released on step 1 have also been served before incurring delay. The lemma thus holds for this case.

Case 2.2

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 $T_2$ 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 $T_1$ exactly once, with no delay cost. Since the optimum has bought all of $T_2$, the requests released on step 1 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 23. Any online algorithm has a cost of at least $c_i \cdot C(I_i)$ on $I_i$ by time $3^i$. 

18
Proof. By induction on \( i \).

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

\[
c(S) \cdot \Gamma_S + (1 - \Gamma_S) \cdot \int_0^1 d_{q_0}(S)(t)\,dt \geq c(S) = C(I_0)
\]

where the inequality is due to the definition of \( q_0(S) \). 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 \( \Gamma \) the total buying cost of the algorithm in the time interval \([0, 3^{i-1}]\) for sets of \( T_2 \). We again split into cases according to the chosen branch in the construction of \( I_i \).

Case 2.1

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 \( S \in S_{i-1} \) came from \( T_1 \) or \( T_2 \). 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 \( T_2 \) are replaced with buying the equivalent sets from \( T_1 \). Such a modified online algorithm would cost \( \frac{\alpha_i}{1+\alpha_i} \cdot \Gamma \) 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

\[
\left( \left( c_{i-1} + \frac{\alpha_i}{2} \right) \cdot C(I_{i-1}) \right) + ((1 + \alpha_i) \cdot c_{i-1} \cdot C(I_{i-1}))
\]

\[
= \left( (2 + \alpha_i)c_{i-1} + \frac{\alpha_i}{2} \right) \cdot C(I_{i-1})
\]

\[
= c_{i-1} \cdot C(I_i) + \frac{\alpha_i}{2} \cdot C(I_{i-1})
\]

\[
\geq \left( c_{i-1} + \frac{\alpha_i}{6} \right) \cdot C(I_i)
\]

\[
= \left( c_{i-1} + \frac{1}{12c_{i-1}} \right) \cdot C(I_i)
\]

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

Case 2.2

In this case we have \( \Gamma \leq \frac{1}{2} \cdot (1 + \alpha_i) \cdot C(I_{i-1}) \). 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 \( E_{i-1}^2 \), 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 \( T_2 \) do not satisfy requests in this instance, this cost is either in buying sets of \( T_1 \) or in delay of requests from that \( I_{i-1} \) instance.

In addition to the two \( I_{i-1} \) instances, due to the \( q_2^{3^{i-1}}(S) \) requests released in step 1.1, the algorithm has a cost of at least \( (\sum_{T \in T_2} c(T)) - \Gamma = (1 + \alpha_i)C(I_{i-1}) - \Gamma \) during the interval \([1, 3]\) in either buying sets of \( T_2 \) 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
\[
\left( c_{i-1} \cdot C(I_{i-1}) + \Gamma \cdot \frac{\alpha_i}{1 + \alpha_i} \right) + (c_{i-1} \cdot C(I_{i-1})) + ((1 + \alpha_i)C(I_{i-1}) - \Gamma)
\]
\[
= \left( 2c_{i-1} + 1 + \alpha_i \right) \cdot C(I_{i-1}) - \frac{1}{1 + \alpha_i} \Gamma
\]
\[
\geq \left( 2c_{i-1} + 1 + \alpha_i \right) \cdot C(I_{i-1}) - \frac{1}{2} C(I_{i-1})
\]
\[
= \left( 2 + \alpha_i \right) c_{i-1} + \left( \frac{1}{2} + (1 - c_{i-1})\alpha_i \right) \cdot C(I_{i-1})
\]
\[
= c_{i-1} \cdot C(I_{i}) + \left( \frac{1}{2} + \frac{1}{2c_{i-1}} - \frac{1}{2} \right) \cdot C(I_{i-1})
\]
\[
\geq \left( c_{i-1} + \frac{1}{6c_{i-1}} \right) \cdot C(I_{i}) \geq c_i \cdot C(I_{i})
\]

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 \( \alpha_i \).

\[\square\]

**Proof.** (of Theorem 2) Lemmas 22 and 23 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.

\[\square\]

**References**

[1] Sebastian Abshoff, Christine Markarian, and Friedhelm Meyer auf der Heide. Randomized online algorithms for set cover leasing problems. In *Combinatorial Optimization and Applications - 8th International Conference, COCOA*, pages 25–34, 2014.

[2] Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. The online set cover problem. *SIAM J. Comput.*, 39(2):361–370, 2009.

[3] 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*, pages 1:1–1:20, 2017.

[4] 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 (extended abstract). In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 519–530, 1996.

[5] Y. Azar and A. Jacob-Fanani. Deterministic Min-Cost Matching with Delays. *ArXiv e-prints*, June 2018.
[6] Yossi Azar, Ashish Chiplunkar, and Haim Kaplan. Polylogarithmic bounds on the competitiveness of min-cost perfect matching with delays. In Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA, pages 1051–1061, 2017.

[7] 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, pages 551–563, 2017.

[8] Nikhil Bansal and Ho-Leung Chan. Weighted flow time does not admit $o(1)$-competitive algorithms. In Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA, pages 1238–1244, 2009.

[9] Kshipra Bhawalkar, Sreenivas Gollapudi, and Debmalya Panigrahi. Online set cover with set requests. In Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, 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 24th Annual European Symposium on Algorithms, ESA, pages 12:1–12:17, 2016.

[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.

[12] 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, pages 132–146, 2017.

[13] Marcin Bienkowski, Artur Kraska, and Pawel Schmidt. Online service with delay on a line. In Structural Information and Communication Complexity - 24th International Colloquium, SIROCCO, 2018.

[14] Niv Buchbinder, Moran Feldman, Joseph (Seffi) Naor, and Ohad Talmon. $O(\text{depth})$-competitive algorithm for online multi-level aggregation. In Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA, pages 1235–1244, 2017.

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

[16] 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, pages 389–398, 1998.

[17] 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, pages 333–344, 2016.

[18] Uriel Feige. A threshold of $\ln n$ for approximating set cover. J. ACM, 45(4):634–652, 1998.

[19] 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.

[20] Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Debmalya Panigrahi. Online and dynamic algorithms for set cover. In Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC, pages 537–550, 2017.
Appendix

A Integrality Gap

In section 4, we have seen a randomized rounding algorithm, where the rounding comes at a competitiveness cost of $\log n$. This section shows that this rounding scheme is nearly optimal. Specifically, we show a $\Omega(\log n)$ integrality gap, in an instance in which $k = O(\log n)$. Since the fractional algorithm is $O(\log k)$-competitive, it is $O(\log \log n)$-competitive for this instance. This implies that any rounding scheme must lose a factor of $\Omega(\frac{\log n}{\log \log n})$ in competitiveness with respect to the fractional algorithm.

We now describe the instance for the integrality gap. The instance includes $2k - 1$ sets of cost 1, where at time 0 every subset of $k$ sets receives a request. In this instance, $n = \binom{2k - 1}{k}$, and thus $k = \Theta(\log n)$. Any integral algorithm has to buy at least $k$ sets to complete all requests, for a total cost of $\Omega(\log n)$. However, a possible fractional solution would be to buy $\frac{1}{k}$ of every set, completing all requests at a total cost of less than 2. This yields the $\Omega(\log n)$ integrality gap.

B 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 acknowledgement problem is a special case of VCD with a single edge, the lower bound of 2-competitiveness for any deterministic algorithm on the TCP acknowledgement problem (shown in [14]) 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 24.** The algorithm $ON$ for SCD has a competitive ratio of $k + 1$. In particular, $ON$ is 3-competitive for VCD.

**Lemma 25.** 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 26.** 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 $\sum_{j|q_j \in T} \int_t^\infty d_{j}^{ON}(t')dt' \leq c(S)$.

**Proof.** Denote by $\hat{t}$ the first time in which all requests in $T$ are served. We have that $\sum_{j|q_j \in T} \int_t^\infty d_{j}^{ON}(t')dt' = \sum_{j|q_j \in T} \int_t^{\hat{t}} d_{j}^{ON}(t')dt'$

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 $z(S, t'') = z(S) + \sum_{j|q_j \in T} \int_t^{t''} d_{j}^{ON}(t')dt'$

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 $\sum_{j|q_j \in T} \int_t^{t''} d_{j}^{ON}(t')dt' < c(S)$. The claim follows as $t''$ approaches $\hat{t}$. 

**Lemma 27.** 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 $y_j(t) = d_{j}^{ON}(t)$ 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 $d_{j}^{ON}(t') = 0$ for all $t' \geq t$. Using Lemma 26 for the requests unserved at $t$ concludes the proof.

**Proof.** (of theorem 24) The proof of the theorem results directly from lemmas 26 and 27.

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 $k + 1$. 

23