The Online Min-Sum Set Cover Problem

Dimitris Fotakis 🆔
National Technical University of Athens, Greece
fotakis@cs.ntua.gr

Loukas Kavouras
National Technical University of Athens, Greece
lukaskavouras@gmail.com

Grigorios Koumoutsos
Université libre de Bruxelles, Belgium
gregkoumoutsos@gmail.com

Stratis Skoulakis
Singapore University of Technology and Design, Singapore
efstratios@sutd.edu.sg

Manolis Vardas
ETH Zurich, Switzerland
evardas@student.ethz.ch

──── **Abstract** ────

We consider the online Min-Sum Set Cover (MSSC), a natural and intriguing generalization of the classical list update problem. In Online MSSC, the algorithm maintains a permutation on n elements based on subsets $S_1, S_2, \ldots$ arriving online. The algorithm serves each set S_t upon arrival, using its current permutation π_t, incurring an access cost equal to the position of the first element of S_t in π_t. Then, the algorithm may update its permutation to π_{t+1}, incurring a moving cost equal to the Kendall tau distance of π_t to π_{t+1}. The objective is to minimize the total access and moving cost for serving the entire sequence. We consider the r-uniform version, where each S_t has cardinality r. List update is the special case where $r = 1$.

We obtain tight bounds on the competitive ratio of deterministic online algorithms for MSSC against a static adversary, that serves the entire sequence by a single permutation. First, we show a lower bound of $(r + 1)(1 - \frac{r}{n+1})$ on the competitive ratio. Then, we consider several natural generalizations of successful list update algorithms and show that they fail to achieve any interesting competitive guarantee. On the positive side, we obtain a $O(r)$-competitive deterministic algorithm using ideas from online learning and the multiplicative weight updates (MWU) algorithm.

Furthermore, we consider efficient algorithms. We propose a memoryless online algorithm, called *Move-All-Equally*, which is inspired by the Double Coverage algorithm for the k-server problem. We show that its competitive ratio is $\Omega(r^2)$ and $2^{O(\sqrt{\log n \cdot \log r})}$, and conjecture that it is $f(r)$-competitive. We also compare Move-All-Equally against the dynamic optimal solution and obtain (almost) tight bounds by showing that it is $\Omega(r\sqrt{n})$ and $O(r^{3/2}\sqrt{n})$-competitive.

47th International Colloquium on Automata, Languages, and Programming (ICALP 2020).
Editors: Artur Czumaj, Anuj Dawar, and Emanuela Merelli; Article No. 51; pp. 51:1–51:16

Leibniz International Proceedings in Informatics
LIPIcs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In Min-Sum Set Cover (MSSC), we are given a universe U on n elements and a collection of subsets $\mathcal{S} = \{S_1, \ldots, S_m\}$, with $S_t \subseteq U$, and the task is to construct a permutation (or list) π of elements of U. The cost $\pi(S_t)$ of covering a set S_t (a.k.a. the cover time of S_t) with a permutation π is the position of the first element of S_t in π, i.e., $\pi(S_t) = \min\{i \,|\, \pi(i) \in S_t\}$. The goal is to minimize the overall cost $\sum_t \pi(S_t)$ of covering all subsets of $\mathcal{S}$.

The MSSC problem generalizes various NP-hard problems such as Min-Sum Vertex Cover and Min-Sum Coloring and it is well-studied. Feige, Lovasz and Tetali [25] showed that the greedy algorithm, which picks in each position the element that covers the most uncovered sets, is a 4-approximation (this was also implicit in [11]) and that no $(4 - \epsilon)$-approximation is possible, unless P = NP. Several generalizations have been considered over the years with applications in various areas (we discuss some of those problems and results in Section 1.2).

Online Min-Sum Set Cover. In this paper, we study the online version of Min-Sum Set Cover. Here, the sets arrive online; at time step t, the set S_t is revealed. An online algorithm is charged the *access cost* of its current permutation $\pi_t(S_t)$; then, it is allowed to change its permutation to π_{t+1} at a *moving cost* equal to the number of inversions between π_t and π_{t+1}, known as the Kendall tau distance $d_{\mathrm{KT}}(\pi_t, \pi_{t+1})$. The goal is to minimize the total cost, i.e., $\sum_t \big(\pi_t(S_t) + d_{\mathrm{KT}}(\pi_t, \pi_{t+1}) \big)$. This is a significant generalization of the classic list update problem, which corresponds to the special case where $|S_t| = 1$ for all sets $S_t \in \mathcal{S}$.

Motivation. Consider a web search engine, such as Google. Each query asked might have many different meanings depending on the user. For example, the query "Python" might refer to an animal, a programming language or a movie. Given the pages related to "Python", a goal of the search engine algorithm is to rank them such that for each user, the pages of interest appear as high as possible in the ranking (see e.g., [23]). Similarly, news streams include articles covering different reader interests each. We want to rank the articles so that every reader finds an article of interest as high as possible. The MSSC problem serves as a theoretical model for practical problems of this type, where we want to aggregate disjunctive binary preferences (expressed by the input sets) into a total order. E.g., for a news stream, the universe U corresponds to the available articles and the sets S_t correspond to different user types. The cost of a ranking (i.e., permutation on U) for a user type is the location of the first article of interest. Clearly, in such applications, users arrive online and the algorithm might need to re-rank the stream (i.e., change the permutation) based on user preferences.

Benchmarks. For the most part, we evaluate the performance of online algorithms by comparing their cost against the cost of an optimal offline solution that knows the input in advance and chooses an optimal permutation π. Note that this solution is *static*, in the

sense that it does not change permutations over time. This type of analysis, called *static optimality*, is typical in online optimization and online learning. It was initiated in the context of adaptive data structures by the landmark result of Sleator and Tarjan [44], who showed that *splay trees* are asymptotically as fast as any *static* tree. Since then, it has been an established benchmark for various problems in this area (see e.g. [13, 30]); it is also a standard benchmark for several other problems in online optimization (e.g., online facility location [26, 37], minimum metric matching [28, 33, 39], Steiner tree [38], etc.).

A much more general benchmark is the *dynamic Min-Sum Set Cover* problem, where the algorithm is compared against an optimal solution allowed to change permutations over time. This problem has not been studied even in the offline case. In this work, we define the problem formally and obtain first results for the online case.

We remark that the online dynamic MSSC problem belongs to a rich class of problems called *Metrical Task Systems* (MTS) [15]. MTS is a far-reaching generalization of several fundamental online problems and provides a unified framework for studying online problems (we discuss this in more detail in Section 1.2). Indeed, our results suggest that solving the online dynamic MSSC requires the development of powerful generic techniques for online problems, which might have further implications for the broader setting of MTS.

Throughout this paper, whenever we refer to online problems, like Min-Sum Set Cover or list update, we assume the static case, unless stated otherwise.

Previous Work on List Update. Prior to our work, the only version of online MSSC studied is the special case where $|S_t| = 1$ for all sets; this is the celebrated list update problem and it has been extensively studied (an excellent reference is [14]). It is known that the deterministic competitive ratio it least $2 - \frac{2}{n+1}$ and there are several 2-competitive algorithms known; most notably, the Move-to-Front (MTF) algorithm, which moves the (unique) element of S_t to the first position of the permutation, and the Frequency Count algorithm, which orders the elements in decreasing order according to their frequencies.

The dynamic list update problem has also been extensively studied. MTF is known to be 2-competitive [43] and there are several other 2-competitive algorithms [1, 24].

1.1 Our Results

In this work, we initiate a systematic study of the online Min-Sum Set Cover problem. We consider the *r*-uniform case, where all request sets have the same size $|S_t| = r$. This is without loss of generality, as we explain in Section 1.3.

The first of our main results is a tight bound on the deterministic competitive ratio of Online MSSC. We show that the competitive ratio of deterministic algorithms is $\Omega(r)$.

▶ **Theorem 1.** *Any deterministic online algorithm for the Online Min-Sum Set Cover problem has competitive ratio at least $(r + 1)(1 - \frac{r}{n+1})$.*

Note that for $r = 1$, this bound evaluates to $2 - \frac{2}{n+1}$, which is exactly the best known lower bound for the list update problem.

We complement this result by providing a matching (up to constant factors) upper bound.

▶ **Theorem 2.** *There exists a $(5r + 2)$-competitive deterministic online algorithm for the Online Min-Sum Set Cover problem.*

Interestingly, all prior work on the list update problem (case $r = 1$) does not seem to provide us with the right tools for obtaining an algorithm with such guarantees! As we discuss in Section 2, virtually all natural generalizations of successful list update algorithms

(e.g., Move-to-Front, Frequency Count) end up with a competitive ratio way far from the desired bound. In fact, even for $r = 2$, most of them have a competitive ratio depending on n, such as $\Omega(\sqrt{n})$ or even $\Omega(n)$.

This suggests that online MSSC has a distinctive combinatorial structure, very different from that of list update, whose algorithmic understanding calls for significant new insights. The main reason has to do with the disjunctive nature of the definition of the access cost $\pi(S_t)$. In list update, where $r = 1$, the optimal solution is bound to serve a request S_t by its unique element. The only question is how fast an online algorithm should upgrade it (and the answer is "as fast as possible"). In MSSC, the hard (and crucial) part behind the design of any competitive algorithm is how to ensure that the algorithm learns fast enough about the element e_t used by the optimal solution to serve each request S_t. This is evident in the highly adaptive nature of the deceptively simple greedy algorithm of [25] and in the adversarial request sequences for generalizations of Move-to-Front, in Section 2.

To obtain the asymptotically optimal ratio of Theorem 2, we develop a rounding scheme and use it to derandomize the multiplicative weights update (MWU) algorithm. Our analysis bounds the algorithm's access cost in terms of the optimal cost, but it does not account for the algorithm's moving cost. We then refine our approach, by performing lazy updates to the algorithm's permutation, and obtain a competitive algorithm for online MSSC.

We also observe (in Section 1.3) that based on previous work of Blum and Burch [12], there exists a (computationally inefficient) randomized algorithm with competitive ratio $1 + \epsilon$, for any $\epsilon \in (0, 1/4)$. This implies that no lower bound is possible, if randomization is allowed, and gives a strong separation between deterministic and randomized algorithms.

Memoryless Algorithms.   While the bounds of Theorems 1 and 2 are matching, our algorithm from Theorem 2 is computationally inefficient since it simulates the MWU algorithm, which in turn, maintains a probability distribution over all $n!$ permutations. This motivates the study of trade-offs between the competitive ratio and computational efficiency. To this end, we propose a memoryless algorithm, called *Move-All-Equally* (MAE), which moves all elements of set S_t towards the beginning of the permutation at the same speed until the first reaches the first position. This is inspired by the Double Coverage algorithm from k-server [20, 21]. We believe that MAE achieves the best guarantees among all memoryless algorithms. We show that this algorithm can not match the deterministic competitive ratio.

▶ **Theorem 3.** *The competitive ratio of the Move-All-Equally algorithm is $\Omega(r^2)$.*

Based on Theorem 3, we conjecture that an $O(r)$ guarantee cannot be achieved by a memoryless algorithms. We leave as an open question whether MAE has a competitive ratio $f(r)$, or a dependence on n is necessary. To this end, we show that the competitive ratio of MAE is at most $2^{O(\sqrt{\log n \cdot \log r})}$ (see Section 4 for details).

Dynamic Min-Sum Set Cover.   We also consider the dynamic version of online MSSC. Dynamic MSSC is much more general and the techniques developed for the static case do not seem adequately powerful. This is not surprising, since the MWU algorithm is designed to perform well against the best static solution. We investigate the performance of the MAE algorithm. First, we obtain an upper bound on its competitive ratio.

▶ **Theorem 4.** *The competitive ratio of the Move-All-Equally algorithm for the dynamic online Min-Sum Set Cover problem is $O(r^{3/2}\sqrt{n})$.*

Although this guarantee is not very strong, we show that, rather surprisingly, it is essentially tight and no better guarantees can be shown for this algorithm.

▶ **Theorem 5.** *For any $r \geq 3$, the competitive ratio of the Move-All-Equally algorithm for the dynamic online Min-Sum Set Cover problem is $\Omega(r\sqrt{n})$.*

This lower bound is based on a carefully crafted adversarial instance; this construction reveals the rich structure of this problem and suggests that more powerful generic techniques are required in order to achieve any $f(r)$ guarantees. In fact, we conjecture that the lower bound of Theorem 1 is the best possible (ignoring constant factors) even for the dynamic problem and that using a work-function based approach such a bound can be obtained.

1.2 Further Related Work

Multiple Intents Re-ranking. This is a generalization of MSSC where for each set S_t, there is a *covering requirement* $K(S_t)$, and the cost of covering a set S_t is the position of the $K(S_t)$-th element of S_t in π. The MSSC problem is the special case where $K(S_t) = 1$ for all sets S_t. Another notable special case is the Min-Latency Set Cover problem, which corresponds to the other extreme case where $K(S_t) = |S_t|$ [29]. Multiple Intents Re-ranking was first studied by Azar et. al. [5], who presented a $O(\log r)$-approximation; later $O(1)$-approximation algorithms were obtained [10, 32, 42]. Further generalizations have been considered, such as the Submodular Ranking problem, studied by Azar and Gamzu [4], which generalizes both Set Cover and MSSC, and the Min-Latency Submodular Cover, studied by Im et.al [31].

Prediction from Expert Advice and Randomized MSSC. In prediction from expert advice, there are N experts and expert i incurs a cost c_i^t in each step. A learning algorithm decides which expert i_t to follow (before the cost vector $\mathbf{c}^t$ is revealed) and incurs a cost of $c_{i_t}^t$. The landmark technique for solving this problem is the multiplicative weights update (MWU - a.k.a. Hedge) algorithm. For an in-depth treatment of MWU, we refer to [3, 27, 35].

In the classic online learning setting, there is no cost for moving probability mass between experts. However, in a breakthrough result, Blum and Burch [12] showed that MWU is $(1 + \epsilon)$-competitive against the best expert, even if there is a cost D for moving probability mass between experts. By adapting this result to online MSSC (regarding permutations as experts), we can get an (inefficient) randomized algorithm with competitive ratio $(1 + \epsilon)$, for any constant $\epsilon \in (0, 1/4)$. A detailed description is deferred to the full version of this paper.

Metrical Task Systems and Online Dynamic MSSC. The online dynamic Min-Sum Set Cover problem belongs to a rich family of problems called Metrical Task Systems (MTS). In MTS, we are given a set of N states and a metric function d specifying the cost of moving between the states. At each step, a task arrives; the cost of serving the task at state i is c_i. An algorithm has to choose a state to process the task. If it switches from state i to state j and processes the task there, it incurs a cost $d(i, j) + c_j$. Given an initial state and a sequence of requests, the goal is to process all tasks at minimum cost.

It is easy to see that the online version of dynamic MSSC problem is a MTS, where the states correspond to permutations, thus $N = n!$, and the distance between two states is their Kendall tau distance. For a request set S_t, the request is a vector specifying the cost $\pi(S_t)$ for every permutation π.

Several other fundamental online problems (e.g., k-server, convex body chasing) are MTS. Although there has been a lot of work on understanding the structure of MTS problems [2, 8, 9, 15, 16, 22, 34, 40, 41], there is not a good grasp on how the structure relates to the hardness of MTS problems. Getting a better understanding on this area is a long-term goal, since it would lead to a systematic framework for solving online problems.

1.3   Preliminaries

Notation.   Given a request sequence $\mathcal{S} = \{S_1, \ldots, S_m\}$, for any algorithm ALG we denote $\mathrm{Cost}(\mathrm{ALG}(\mathcal{S}))$ or simply $\mathrm{Cost}(\mathrm{ALG})$ the total cost of ALG on $\mathcal{S}$. Similarly we denote $\mathrm{AccessCost}(\mathrm{ALG})$ the total access cost of ALG and $\mathrm{MovingCost}(\mathrm{ALG})$ the total movement cost of ALG. For a particular time step t, an algorithm using permutation π_t incurs an access cost $\mathrm{AccessCost}(\mathrm{ALG}(t)) = \pi_t(S_t)$. We denote by $\pi_t[j]$ the position of element $j \in U$ in the permutation π_t.

Online Min-Sum Set Cover.   We focus on the r-uniform case, i.e., when all sets S_t have size $r \ll n$. This is essentially without loss of generality, because we can always let $r = \max_t |S_t|$ and add the $r - |S_t|$ last unrequested elements in the algorithm's permutation to any set S_t with $|S_t| < r$. Assuming that $r \leq n/2$, this modification cannot increase the optimal cost and cannot decrease the online cost by more than a factor of 2.

2   Lower Bounds on the Deterministic Competitive Ratio

We start with a lower bound on the deterministic competitive ratio of online MSSC.

▶ **Theorem 1.** *Any deterministic online algorithm for the Online Min-Sum Set Cover problem has competitive ratio at least $(r + 1)(1 - \frac{r}{n+1})$.*

For the proof, we employ an averaging argument, similar to those in lower bounds for list update and k-server [36, 43]. In each step, the adversary requests the last r elements in the algorithm's permutation. Hence, the algorithm's cost is at least $(n - r + 1)$. Using a counting argument, we show that for any fixed set S_t of size r and any $i \in [n - r + 1]$, the number of permutations π with access cost $\pi(S_t) = i$ is $\binom{n-i}{r-1} r! (n - r)!$. Summing up over all permutations and dividing by $n!$, we get that the average access cost for S_t is $\binom{n+1}{r+1} \frac{r!(n-r)!}{n!} = \frac{n+1}{r+1}$. Therefore, the cost of the optimal permutation is a most $\frac{(n+1)}{r+1}$, and the competitive ratio of the algorithm at least $\frac{(n-r+1)(r+1)}{n+1}$. The details can be found in the full version of this paper.

Lower Bounds for Generalizations of Move-to-Front.   For list update, where $r = 1$, simple algorithms like Move-to-Front (MTF) and Frequency Count achieve an optimal competitive ratio. We next briefly describe several such generalizations of them and show that their competitive ratio depends on n, even for $r = 2$. Missing details can be found in the full version.

MTF$_{\mathsf{first}}$: Move to the first position (of the algorithm's permutation) the element of S_t appearing first in π_t. This algorithm is $\Omega(n)$-competitive when each request S_t consists of the last two elements in π_t. Then, the last element in the algorithm's permutation never changes and is used by the optimal permutation to serve the entire sequence!

MTF$_{\mathsf{last}}$: Move to the first position the element of S_t appearing last in π_t.

MTF$_{\mathsf{all}}$: Move to the first r positions all elements of S_t (in the same order as in π_t).

MTF$_{\mathsf{random}}$: Move to the first position an element of S_t selected uniformly at random.

MTF$_{\mathrm{last}}$, MTF$_{\mathrm{all}}$ and MTF$_{\mathrm{random}}$ have a competitive ratio of $\Omega(n)$ when each request S_t consists of a fixed element e (always the same) and the last element in π_t, because they all incur an (expected for MTF$_{\mathrm{random}}$) moving cost of $\Theta(n)$ per request.

The algorithms seen so far fail for the opposite reasons: MTF$_{\mathrm{first}}$ cares only about the first element and ignores completely the second, and the others are very aggressive on using the second (rth) element. A natural attempt to balance those two extremes is the following.

MTF$_{\text{relative}}$: Let i be the position of the first element of S_t in π_t. Move to the first positions of the algorithm's permutation (keeping their relative order) all elements of S_t appearing up to the position $c \cdot i$ in π_t, for some constant c. The bad instance for this algorithm is when each request S_t consists of the last element and the element at position $\lfloor n/c \rfloor - 1$ in π_t; it never uses the nth element and the adversary serves all requests with it at a cost of 1.

All generalizations of MTF above are memoryless and they all fail to identify the element by which optimal serves S_t. The following algorithm tries to circumvent this by keeping memory and in particular the frequencies of reqested elements.

MTF$_{\text{count}}$: Move to the first position the most frequent element of S_t (i.e., the element of S_t appearing in most requested sets so far).

This algorithm behaves better in easy instances, however with some more work we can show a lower bound of $\Omega(\sqrt{n})$ on its competitive ratio. Let $e_1, \ldots, e_n$ be the elements indexed according to the initial permutation π_0 and $b = \sqrt{n}$. The request sequence proceeds in m/n phases of length n each. The first $n - b$ requests of each phase are $\{e_1, e_2\}, \{e_1, e_3\}, \ldots, \{e_1, e_{n-b}\}$, and the last b requests consist of e_{n-b+i} and the element at position $n - b$ at the current algorithm's permutation, for $i = 1, \ldots, b$. An optimal solution can cover all the requests by the elements $e_1, e_{n-b+1}, \ldots, e_n$ with total cost $\Theta(m + n\sqrt{n})$. The elements $e_{n-b+1}, \ldots, e_n$ are never upgraded by MTF$_{\text{count}}$. Hence, the algorithm's cost is $\Theta(m\sqrt{n})$.

3 An Algorithm with Asymptotically Optimal Competitive Ratio

Next, we present algorithm Lazy-Rounding (Algorithm 2) and analyze its competitive ratio. The following is the main result of this section:

▶ **Theorem 2.** *Deterministic online algorithm* Lazy-Rounding, *presented in Algorithm 2, is* $(5r + 2)$-*competitive for the static version of the Online Min-Sum Set Cover problem.*

The remainder of this section is devoted to the proof of Theorem 2. At a high-level, our approach is summarized by the following three steps:

1. We use as black-box the multiplicative weights update (MWU) algorithm with learning rate $1/n^3$. Using standard results from learning theory, we show that its expected access cost is within a factor $5/4$ of OPT, i.e., $\text{AccessCost}(\text{MWU}) \leq \frac{5}{4} \text{Cost}(\text{OPT})$ (Section 3.1).
2. We develop an online rounding scheme, which turns any randomized algorithm $\mathcal{A}$ into a deterministic one, denoted $\text{Derand}(\mathcal{A})$, with access cost at most $2r \cdot \mathbb{E}[\text{AccessCost}(\mathcal{A})]$ (Section 3.2). However, our rounding scheme does not provide any immediate guarantee on the moving cost of $\text{Derand}(\mathcal{A})$.
3. Lazy-Rounding is a lazy version of $\text{Derand}(\text{MWU})$ that updates its permutation only if MWU's distribution has changed a lot. A *phase* corresponds to a time interval that Lazy-Rounding does not change its permutation. We show that during a phase:
   (i) The upper bound on the access cost increases, compared to $\text{Derand}(\text{MWU})$, by a factor of at most 2, i.e., $\text{AccessCost}(\text{Lazy-Rounding}) \leq 4r \cdot \mathbb{E}[\text{AccessCost}(\text{MWU})]$ (Lemma 11).
   (ii) The (expected) access cost of MWU is at least n^2. Since our algorithm moves only once per phase, its movement cost is at most n^2. Thus we get that (Lemma 12):

$$\text{MovingCost}(\text{Lazy-Rounding}) \leq \mathbb{E}[\text{AccessCost}(\text{MWU})] .$$

For the upper bound on the moving cost above, we relate how much MWU's distribution changes during a phase, in terms of the total variation distance, to the cost of MWU and the cost of our algorithm.

Based on the above properties, we compare the access and the moving cost of Lazy-Rounding against the access cost of MWU and to get the desired competitive ratio:

$$\text{Cost(Lazy-Rounding)} \leq (4r+1)\,\mathbb{E}[\text{AccessCost(MWU)}] \leq (5r+2)\,\text{Cost(OPT)}\,.$$

Throughout this section we denote by $d_{\text{TV}}(\delta, \delta')$ the total variation distance of two discrete probability distributions $\delta, \delta' : [N] \to [0,1]$, defined as $d_{\text{TV}}(\delta, \delta') = \sum_{i=1}^{N} \max\{0, \delta(i) - \delta'(i)\}$.

3.1   Using Multiplicative Weights Update in Online Min-Sum Set Cover

In this section, we explain how the well-known MWU algorithm [27, 35] is used in our context.

The MWU Algorithm.   Given $n!$ permutations of elements of U, the algorithm has a parameter $\beta \in [0,1]$ and a weight w_π for each permutation $\pi \in [n!]$, initialized at 1. At each time step the algorithm chooses a permutation according to distribution $\text{P}_\pi^t = w_\pi^t / (\sum_{\pi \in [n!]} w_\pi^t)$. When request S_t arrives, MWU incurs an expected access cost of

$$\mathbb{E}[\text{AccessCost(MWU}(t))] = \sum_{\pi \in [n!]} \text{P}_\pi^t \cdot \pi(S_t)$$

and updates its weights $w_\pi^{t+1} = w_\pi^t \cdot \beta^{\pi(S_t)}$, where $\beta = e^{-1/n^3}$; this is the so-called *learning rate* of our algorithm. Later on, we discuss the reasons behind choosing this value.

On the Access Cost of MWU.   Using standard results from learning theory [27, 35] and adapting them to our setting, we get that the (expected) access cost of MWU is bounded by Cost(OPT). This is formally stated in Lemma 6 (and is proven in the full version).

▶ **Lemma 6.** *For any request sequence $\sigma = (S_1, \ldots, S_m)$ we have that*

$$\mathbb{E}[\text{AccessCost(MWU)}] \leq \frac{5}{4} \cdot \text{Cost(OPT)} + 2n^4 \ln n\,.$$

On the Distribution of MWU.   We now relate the expected access cost of the MWU algorithm to the total variation distance among MWU's distributions. More precisely, we show that if the total variation distance between MWU's distributions at times t_1 and t_2 is large, then MWU has incurred a sufficiently large access cost. The proof of the following makes a careful use of MWU's properties and is deferred to the full version of this paper.

▶ **Lemma 7.** *Let P^t be the probability distribution of the MWU algorithm at time t. Then,*

$$d_{\text{TV}}(\text{P}^t, \text{P}^{t+1}) \leq \frac{1}{n^3} \cdot \mathbb{E}[\text{AccessCost(MWU}(t))].$$

The following is useful for the analysis of Lazy-Rounding. Its proof follows from Lemma 7 and the the triangle inequality and is deferred to the full version of this paper.

▶ **Lemma 8.** *Let t_1 and t_2 two different time steps such that $d_{\text{TV}}(\text{P}^{t_1}, \text{P}^{t_2}) \geq 1/n$. Then,*

$$\sum_{t=t_1}^{t_2-1} \mathbb{E}[\text{AccessCost(MWU}(t))] \geq n^2\,.$$

3.2 Rounding

Next, we present our rounding scheme. Given as input a probability distribution δ over permutations, it outputs a fixed permutation ρ such that for each possible request set S of size r, the cost of ρ on S is within a $O(r)$ factor of the expected cost of the distribution δ on S. For convenience, we assume that n/r is an integer. Otherwise, we use $\lceil n/r \rceil$.

▪ **Algorithm 1** Greedy-Rounding (derandomizing probability distributions over the permutations).

Input: A probability distribution δ over $[n!]$.
Output: A permutation $\rho \in [n!]$.

1: $R \leftarrow U$
2: **for** $i = 1$ to n/r **do**
3:     $S^i \leftarrow \arg\min_{S \in \{R\}^r} \mathbb{E}_{\pi \sim \delta}[\pi(S)]$
4:     Place the elements of S^i (arbitrarily) from positions $(i-1) \cdot r + 1$ to $i \cdot r$ of ρ.
5:     $R \leftarrow R \setminus S^i$
6: **end for**
7: **return** ρ

Our rounding algorithm is described in Algorithm 1. At each step, it finds the request S with minimum expected covering cost under the probability distribution δ and places the elements of S as close to the beginning of the permutation as possible. Then, it removes those elements from set R and iterates. The main claim is that the resulting permutation has the following property: *any request S of size r has covering cost at most $O(r)$ times of its expected covering cost under the probability distribution δ*.

▶ **Theorem 9.** *Let δ be a distribution over permutations and let ρ be the permutation output by Algorithm 1 on δ. Then, for any set S, with $|S| = r$,*

$$\rho(S) \leq 2r \cdot \mathbb{E}_{\pi \sim \delta}[\pi(S)].$$

Proof Sketch. The key step is to show that if the element used by ρ to serve the request S was picked during the kth iteration of the rounding algorithm, then $\mathbb{E}_{\pi \sim \delta}[\pi(S)] \geq k/2$. Clearly, $\rho(S) \leq k \cdot r$ and the theorem follows. Full proof is in the full version. ◀

3.3 The Lazy Rounding Algorithm

Lazy-Rounding, presented in Algorithm 2, is essentially a lazy derandomization of MWU. At each step, it calculates the distribution on permutations maintained by MWU. At the beginning of each *phase*, it sets its permutation to that given by Algorithm 1. Then, it sticks to the same permutation for as long as the total variation distance of MWU's distribution at the beginning of the phase to the current MWU distribution is at most $1/n$. As soon as the total variation distance exceeds $1/n$, Lazy-Rounding starts a new phase.

The main intuition behind the design of our algorithm is the following. In Section 3.2 we showed that Algorithm 1 results in a deterministic algorithm with access cost no larger than $2r \mathbb{E}[\text{AccessCost(MWU)}]$. However, such an algorithm may incur an unbounded moving cost; even small changes in the distribution of MWU could lead to very different permutations after rounding. To deal with that, we update the permutation of Lazy-Rounding only if there are substantial changes in the distribution of MWU. Intuitively, small changes in MWU's distribution should not affect much the access cost (this is formalized in Lemma 10). Moreover, Lazy-Rounding switches to a different permutation only if it is really required, which we use to bounds Lazy-Rounding's moving cost.

Bounding the Access Cost.   We first show that the access cost of Lazy-Rounding is within a factor of $4r$ from the expected access cost of MWU (Lemma 11). To this end, we first show that if the total variation distance between two distributions is small, then sampling from those distributions yields roughly the same expected access cost for any request S. The proof of the following is based on the optimal coupling lemma and can be found in the full version of this paper.

▶ **Lemma 10.** *Let δ and δ' be two probability distributions over permutations. If that $d_{\mathrm{TV}}(\delta, \delta') \leq 1/n$, for any request set S of size r, we have that*

$$\mathop{\mathbb{E}}_{\pi \sim \delta'}[\pi(S)] \leq 2 \cdot \mathop{\mathbb{E}}_{\pi \sim \delta}[\pi(S)].$$

We are now ready to upper bound the access cost of our algorithm.

▶ **Lemma 11.** $\mathrm{AccessCost}(\text{Lazy-Rounding}) \leq 4r \cdot \mathbb{E}[\mathrm{AccessCost}(\text{MWU})].$

Proof. Consider a phase of Lazy-Rounding starting at time t_1. We have that at any round $t \geq t_1$, $\pi_t = \text{Greedy-Rounding}(\mathrm{P}^{t_1})$, as long as $d_{\mathrm{TV}}(\mathrm{P}^t, \mathrm{P}^{t_1}) \leq 1/n$. By Theorem 9 and Lemma 10, we have that,

$$\mathrm{AccessCost}(\text{Lazy-Rounding}(t)) = \pi_t(S_t) \leq 2r \cdot \mathop{\mathbb{E}}_{\pi \sim \mathrm{P}^{t_1}}[\pi(S_t)] \leq 4r \mathop{\mathbb{E}}_{\pi \sim \mathrm{P^t}}[\pi(S_t)].$$

Overall we get, $\mathrm{AccessCost}(\text{Lazy-Rounding}) = \sum_{t=1}^{m} \pi_t(S_t) \leq 4r \, \mathbb{E}[\mathrm{AccessCost}(\text{MWU})].$ ◀

Bounding the Moving Cost.   We now show that the moving cost of Lazy-Rounding is upper bounded by the expected access cost of MWU.

▶ **Lemma 12.** $\mathrm{MovingCost}(\text{Lazy-Rounding}) \leq \mathbb{E}[\mathrm{AccessCost}(\text{MWU})].$

Proof. Lazy-Rounding moves at the end of a phase incurring a cost of at most n^2. Let t_1 and t_2 be the starting times of two consecutive phases. By the definition of Lazy-Rounding, $d_{\mathrm{TV}}(P^{t_1}, P^{t_2}) > 1/n$. By Lemma 8, we have that the access cost of MWU during t_1 and t_2 is at least n^2. We get that

$$\frac{\mathrm{MovingCost}(\text{ALG})}{\mathbb{E}[\mathrm{AccessCost}(\text{MWU})]} \leq \frac{n^2 \# \text{ different phases}}{n^2 \# \text{different phases}} = 1.$$ ◀

Theorem 2 follows from lemmas 11, 12 and 6. The details can be found in the full version.

🟨 **Algorithm 2** Lazy Rounding.

Input: Sequence of requests $(S_1, \ldots, S_m)$ and the initial permutation $\pi_0 \in [n!]$.
Output: A permutation π_t at each round t, which serves request S_t.

1: start-phase ← 1
2: P^1 ← uniform distribution over permutations
3: **for** each round $t \geq 1$ **do**
4:     **if** $d_{\mathrm{tv}}(\mathrm{P}^t, \mathrm{P}^{\text{start-phase}}) \leq 1/n$ **then**
5:         $\pi_t \leftarrow \pi_{t-1}$
6:     **else**
7:         $\pi_t \leftarrow \text{Greedy-Rounding}(\mathrm{P}^t)$
8:         start-phase ← t
9:     **end if**
10:    Serve request S_t using permutation π_t.
11:    $w_\pi^{t+1} = w_\pi^t \cdot e^{-\pi(S_t)/n^3}$, for all permutations $\pi \in [n!]$.
12:    P^{t+1} ← Distribution on permutations of MWU, $\mathrm{P}_\pi^{t+1} = w_\pi^t / (\sum_{\pi \in [n!]} w_\pi^t)$.
13: **end for**

Input: A request sequence $(S_1, \ldots, S_m)$ and the initial permutation $\pi_0 \in [n!]$

Output: A permutation π_t at each round t.

 1: **for** each round $t \geq 1$ **do**
 2:    $k_t \leftarrow \min\{i \,|\, \pi_{t-1}[i] \in S_t\}$
 3:    Decrease the index of all elements of S_t by $k_t - 1$.
 4: **end for**

Remark. Note that to a large extent, our approach is generic and can be used to provide static optimality for a wide range of online problems. The only requirement is that there is a maximum access cost $C_{\max}$ and a maximum moving cost D; then, we should use MWU with learning rate $1/(D \cdot C_{\max})$ and move when $d_{TV} \geq 1/C_{\max}$. Here we used $D = n^2$ and $C_{\max} = n$. The only problem-specific part is the rounding of Section 3.2. We believe it is an interesting direction to use this technique for generalizations of this problem, like multiple intents re-ranking or interpret known algorithms for other problems like the BST problem using our approach.

4   A Memoryless Algorithm

In this section we focus on memoryless algorithms. We present an algorithm, called Move-All-Equally (MAE), which seems to be the "right" memoryless algorithm for online MSSC. MAE decreases the index of all elements of the request S_t at the same speed until one of them reaches the first position of the permutation (see Algorithm 3). Note that MAE belongs to the *Move-to-Front* family, i.e., it is a generalization of the classic MTF algorithm for the list update problem. MAE admits two key properties that substantially differentiate it from the other algorithms in the Move-to-Front family presented in Section 2.

 (i) Let e_t denote the element used by OPT to cover the request S_t. MAE always moves the element e_t towards the beginning of the permutation.
 (ii) It balances moving and access costs: if the access cost at time t is k_t, then the moving cost of MAE is roughly $r \cdot k_t$ (see Algorithm 3). The basic idea is that the moving cost of MAE can be compensated by the decrease in the position of element e_t. This is why it is crucial all the elements to be moved with the same speed.

Lower Bound. First, we show that this algorithm, besides its nice properties, fails to achieve a tight bound for the online MSSC problem.

▶ **Theorem 3.** *The competitive ratio of the Move-All-Equally algorithm is $\Omega(r^2)$.*

In the lower bound instance, the adversary always requests the last r elements of the algorithm's permutation. Since MAE moves all elements to the beginning of the permutation, we end up in a request sequence where n/r disjoint sets are repeatedly requested. Thus the optimal solution incurs a cost of $\Theta(n/r)$ per request, while MAE incurs a cost of $\Omega(n \cdot r)$ per request (the details are in the full version) . Note that in such a sequence, MAE loses a factor of r by moving all elements, instead of one. However, this extra movement seems to be the reason that MAE outperforms all other memoryless algorithms and avoids poor performance in trivial instances, like other MTF-like algorithms.

Upper Bounds. Let $\mathcal{L}$ denote the set of elements used by the optimal permutation on a request sequence such that $|\mathcal{L}| = \ell$. That means, OPT has those ℓ elements in the beginning of its permutation, and it never uses the remaining $n - \ell$ elements. Consider a potential function $\Phi(t)$ being the number of inversions between elements of $\mathcal{L}$ and $U \setminus \mathcal{L}$ in the permutation of MAE (an inversion occurs when an element of $\mathcal{L}$ is behind an element of $U \setminus \mathcal{L}$). Consider the request S_t at time t and let k_t be the access cost of MAE.

Let e_t be the element used by OPT to serve S_t. Clearly, in the permutation of MAE, e_t passes (i.e., changes relative order w.r.t) $k_t - 1$ elements. Among them, let L be the set of elements of $\mathcal{L}$ and R the elements of $U \setminus \mathcal{L}$. Clearly, $|L| + |R| = k_t - 1$ and $|L| \leq |\mathcal{L}| = \ell$. We get that the move of e_t changes the potential by $-|R|$. The moves of all other elements increase the potential by at most $(r - 1) \cdot \ell$. We get that

$$k_t + \Phi(t) - \Phi(t-1) \quad \leq \quad |L| + |R| - |R| + (r-1) \cdot \ell \leq |L| + (r-1) \cdot \ell \leq r \cdot \ell.$$

Since the cost of MAE at step t is no more than $(r+1) \cdot k_t$, we get that the amortized cost of MAE per request is $O(r^2 \cdot \ell)$. This implies that for all sequences such that OPT uses all elements of $\mathcal{L}$ with same frequencies (i.e, the OPT pays on average $\Omega(\ell)$ per request), MAE incurs a cost within $O(r^2)$ factor from the optimal. Recall that all other MTF-like algorithms are $\Omega(\sqrt{n})$ competitive even in instances where OPT uses only one element!

While this simple potential gives evidence that MAE is $O(r^2)$-competitive, it is not enough to provide satisfactory competitiveness guarantees. We generalize this approach and define the potential function $\Phi(t) = \sum_{j=1}^{n} \alpha_j \cdot \pi_t(j)$, where $\pi_t(j)$ is the position of element j at round t and α_j are some non-negative coefficients. The potential we described before is the special case where $\alpha_j = 1$ for all elements of $\mathcal{L}$ and $\alpha_j = 0$ for elements of $U \setminus \mathcal{L}$.

By refining further this approach and choosing coefficients α_j according to the frequency that OPT uses element j to serve requests (elements of high frequency are "more important" so they should have higher values α_j), we get an improved upper bound.

▶ **Theorem 14.** *The competitive ratio of* MAE *algorithm is at most* $2^{O(\sqrt{\log n \cdot \log r})}$.

Note that this guarantee is $o(n^\epsilon)$ and $\omega(\log n)$. The proof is based on the ideas sketched above but the analysis is quite involved and is deferred to the full version of this paper.

5 Dynamic Online Min-Sum Set Cover

In this section, we turn our attention to the dynamic version of online MSSC. In online dynamic MSSC, the optimal solution maintains a trajectory of permutations $\pi_0^*, \pi_1^*, \dots, \pi_t^*, \dots$ and use permutation π_t^* to serve each request S_t. The cost of the optimal dynamic solution is $\text{OPT}_{\text{dynamic}} = \sum_t (\pi_t^*(S_t) + d_{\text{KT}}(\pi_{t-1}^*, \pi_t^*))$, where $\{\pi_t^*\}_t$ denotes the optimal permutation trajectory for the request sequence that minimizes the total access and moving cost.

We remark that the ratio between the optimal static solution and the optimal dynamic solution can be as high as $\Omega(n)$. For example, in the sequence of requests $\{1\}^b\{2\}^b \dots \{n\}^b$, the optimal static solution pays $\Theta(n^2 b)$, whereas the optimal dynamic solution pays $\Theta(n^2 + n \cdot b)$ by moving the element that covers the next $n \cdot b$ requests to the first position and then incurring access cost 1. The above example also reveals that although Algorithm 2 is $\Theta(r)$-competitive against the optimal static solution, its worst-case ratio against a dynamic solution can be $\Omega(n)$.

MAE Algorithm. As a first study of the dynamic problem, we investigate the competitive ratio of *Move-All-Equally* (MAE) algorithm from Section 4. We begin with an upper bound.

▶ **Theorem 4.** *The competitive ratio of the Move-All-Equally algorithm for the dynamic online Min-Sum Set Cover problem is* $O(r^{3/2}\sqrt{n})$.

The approach for proving Theorem 4 is generalizing that exhibited in Section 4 for the static case. We use a generalized potential function $\Phi(t) = \sum_{j=1}^{n} \alpha_j^t \cdot \pi_t(j)$; i.e, the multipliers α_j may change over time so as to capture the moves of $\text{OPT}_{\text{dynamic}}$. To select coefficients α_j^t we apply a two-level approach. We observe that there is always a 2-approximate optimal solution that moves an element of S_t to the front (similar to classic MTF in list update). We call this MTF_{OPT}. We compare the permutation of the online algorithm with the permutation maintained by this algorithm; at each time, elements the beginning of the offline permutation are considered to be "important" and have higher coefficients α_j^t. The formal proof is in the full version.

Next, we show an almost matching lower bound.

▶ **Theorem 5.** *For any $r \geq 3$, the competitive ratio of the Move-All-Equally algorithm for the dynamic online Min-Sum Set Cover problem is $\Omega(r\sqrt{n})$.*

Sketch of the Construction.   The lower bound is based on a complicated adversarial request sequence; we sketch the main ideas. Let k be an integer. During a *phase* we ensure that:
  (i) There are $2k$ "important" elements used by OPT; we call them $e_1, \ldots, e_{2k}$. In the beginning of the phase, those elements are ordered in the start of the optimal permutation π^*, i.e., $\pi^*[e_j] = j$. The phase contains k consecutive requests to each of them, in order; thus the total number of requests is $\approx 2k^2$. OPT brings each element e_j at the front and uses it for k consecutive requests; thus the access cost of OPT is $2k^2$ (1 per request) and the total movement cost of OPT of order $\Theta(k^2)$. Over a phase of $2k^2$ requests, OPT incurs an overall cost $\Theta(k^2)$, i.e., an average of $O(1)$ per request.
  (ii) The first $k + r - 2$ positions of the online permutation will be always occupied by the same set of "not important" elements; at each step the $r - 2$ last of them will be part of the request set and MAE will move them to the front. Thus the access cost will always be $k + 1$ and the total cost more than $(r + 1) \cdot k$.
The two properties above are enough to provide a lower bound $\Omega(r \cdot k)$; the optimal cost is $O(1)$ per request and the online cost $\Omega(r \cdot k)$. The goal of an adversary is to construct a request sequence with those two properties for the largest value of k possible.

The surprising part is that although MAE moves all requested elements towards the beginning of the permutation, it never manages to bring any of the "important" elements in a position smaller than $r + k - 2$. While the full instance is complex and described in the full version, at a high-level, we make sure that whenever a subsequence of k consecutive requests including element e_j begins, e_j is at the end of the online permutation, i.e., $\pi_t[e_j] = n$. Thus, even after k consecutive requests where MAE moves it forward by distance k, it moves by k^2 positions; by making sure that $n - k^2 > r + k - 2$ (which is true for some $k = \Omega(\sqrt{n})$), we can make sure that e_j does not reach the first $r + k - 2$ positions of the online permutation.

6   Concluding Remarks

Our work leaves several intriguing open questions. For the (static version of) Online MSSC, it would be interesting to determine the precise competitive ratio of the MAE algorithm; particularly whether it depends only on r or some dependency on n is really necessary. More generally, it would be interesting to determine the best possible performance of memoryless algorithms and investigate trade-offs between competitiveness and computational efficiency.

For the online dynamic MSSC problem, the obvious question is whether a $f(r)$-competitive algorithm is possible. Here, we showed that techniques developed for the list update problem seem to be too problem-specific and are not helpful in this direction. This calls for the

use of more powerful and systematic approaches. For example, the online primal-dual method [18] has been applied successfully for solving various fundamental problems [6,7,17,19]. Unfortunately, we are not aware of a primal-dual algorithm even for the special case of list update; the only attempt we are aware of is in [45], but this analysis basically recovers known (problem-specific) algorithms using dual-fitting. Our work gives further motivation for designing a primal-dual algorithm for list-update: this could be a starting point towards solving the online dynamic MSSC.

In a broader context, the online MSSC is the first among a family of poorly understood online problems such as the multiple intents re-ranking problem described in Section 1.2. In this problem, when a set S_t is requested, we need to cover it using $s \leq r$ elements; MSSC is the special case $s = 1$. It is natural to expect that the lower bound of Theorem 1 can be generalized to $\Omega(r/s)$, i.e., as s grows, we should be able to achieve a better competitive ratio. It will be interesting to investigate this and the applicability of our technique to obtain tight bounds for this problem.

References

1 Susanne Albers. Improved randomized on-line algorithms for the list update problem. *SIAM J. Comput.*, 27(3):682–693, 1998. `doi:10.1137/S0097539794277858`.

2 C. J. Argue, Anupam Gupta, Guru Guruganesh, and Ziye Tang. Chasing convex bodies with linear competitive ratio. In *SODA*, pages 1519–1524, 2020. `doi:10.1137/1.9781611975994.93`.

3 Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012. `doi:10.4086/toc.2012.v008a006`.

4 Yossi Azar and Iftah Gamzu. Ranking with submodular valuations. In *SODA*, pages 1070–1079, 2011. `doi:10.1137/1.9781611973082.81`.

5 Yossi Azar, Iftah Gamzu, and Xiaoxin Yin. Multiple intents re-ranking. In *STOC*, pages 669–678, 2009. `doi:10.1145/1536414.1536505`.

6 Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. A polylogarithmic-competitive algorithm for the k-server problem. *J. ACM*, 62(5):40:1–40:49, 2015. `doi:10.1145/2783434`.

7 Nikhil Bansal, Niv Buchbinder, and Joseph Naor. A primal-dual randomized algorithm for weighted paging. *J. ACM*, 59(4):19:1–19:24, 2012. `doi:10.1145/2339123.2339126`.

8 Nikhil Bansal, Marek Eliáš, and Grigorios Koumoutsos. Weighted k-server bounds via combinatorial dichotomies. In *FOCS*, pages 493–504, 2017. `doi:10.1109/FOCS.2017.52`.

9 Nikhil Bansal, Marek Eliáš, Grigorios Koumoutsos, and Jesper Nederlof. Competitive algorithms for generalized k-server in uniform metrics. In *SODA*, pages 992–1001, 2018. `doi:10.1137/1.9781611975031.64`.

10 Nikhil Bansal, Anupam Gupta, and Ravishankar Krishnaswamy. A constant factor approximation algorithm for generalized min-sum set cover. In *SODA*, pages 1539–1545, 2010. `doi:10.1137/1.9781611973075.125`.

11 Amotz Bar-Noy, Mihir Bellare, Magnús M. Halldórsson, Hadas Shachnai, and Tami Tamir. On chromatic sums and distributed resource allocation. *Inf. Comput.*, 140(2):183–202, 1998. `doi:10.1006/inco.1997.2677`.

12 Avrim Blum and Carl Burch. On-line learning and the metrical task system problem. *Machine Learning*, 39(1):35–58, 2000.

13 Avrim Blum, Shuchi Chawla, and Adam Kalai. Static optimality and dynamic search-optimality in lists and trees. *Algorithmica*, 36(3):249–260, 2003.

14 Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

15   Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4):745–763, 1992. `doi:10.1145/146585.146588`.

16   Sébastien Bubeck, Michael B. Cohen, James R. Lee, and Yin Tat Lee. Metrical task systems on trees via mirror descent and unfair gluing. In *SODA*, pages 89–97. SIAM, 2019.

17   Niv Buchbinder and Joseph Naor. Improved bounds for online routing and packing via a primal-dual approach. In *FOCS*, pages 293–304, 2006. `doi:10.1109/FOCS.2006.39`.

18   Niv Buchbinder and Joseph Naor. The design of competitive online algorithms via a primal-dual approach. *Foundations and Trends in Theoretical Computer Science*, 3(2-3):93–263, 2009. `doi:10.1561/0400000024`.

19   Niv Buchbinder and Joseph Naor. Fair online load balancing. *J. Scheduling*, 16(1):117–127, 2013. `doi:10.1007/s10951-011-0226-0`.

20   Marek Chrobak, Howard J. Karloff, T. H. Payne, and Sundar Vishwanathan. New results on server problems. *SIAM J. Discrete Math.*, 4(2):172–181, 1991. `doi:10.1137/0404017`.

21   Marek Chrobak and Lawrence L. Larmore. An optimal on-line algorithm for k-servers on trees. *SIAM J. Comput.*, 20(1):144–148, 1991. `doi:10.1137/0220008`.

22   Christian Coester and James R. Lee. Pure entropic regularization for metrical task systems. In *COLT*, volume 99 of *Proceedings of Machine Learning Research*, pages 835–848. PMLR, 2019.

23   Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the Web. In *Proceedings of the Tenth International World Wide Web Conference, WWW 10*, pages 613–622. ACM, 2001. `doi:10.1145/371920.372165`.

24   Ran El-Yaniv. *There are infinitely many competitive-optimal online list accessing algorithms*. Hebrew University of Jerusalem, 1996.

25   Uriel Feige, László Lovász, and Prasad Tetali. Approximating min sum set cover. *Algorithmica*, 40(4):219–234, 2004. `doi:10.1007/s00453-004-1110-5`.

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

27   Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, 1997. `doi:10.1006/jcss.1997.1504`.

28   Anupam Gupta, Guru Guruganesh, Binghui Peng, and David Wajc. Stochastic online metric matching. In *ICALP*, pages 67:1–67:14, 2019. `doi:10.4230/LIPIcs.ICALP.2019.67`.

29   Refael Hassin and Asaf Levin. An approximation algorithm for the minimum latency set cover problem. In *ESA*, pages 726–733, 2005. `doi:10.1007/11561071_64`.

30   John Iacono and Wolfgang Mulzer. A static optimality transformation with applications to planar point location. *Int. J. Comput. Geometry Appl.*, 22(4):327–340, 2012. `doi:10.1142/S0218195912600084`.

31   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`.

32   Sungjin Im, Maxim Sviridenko, and Ruben van der Zwaan. Preemptive and non-preemptive generalized min sum set cover. *Math. Program.*, 145(1-2):377–401, 2014. `doi:10.1007/s10107-013-0651-2`.

33   Elias Koutsoupias and Akash Nanavati. The online matching problem on a line. In *WAOA*, pages 179–191, 2003. `doi:10.1007/978-3-540-24592-6_14`.

34   Elias Koutsoupias and Christos H. Papadimitriou. On the k-server conjecture. *Journal of the ACM*, 42(5):971–983, 1995.

35   Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. *Inf. Comput.*, 108(2):212–261, 1994.

36   Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive algorithms for server problems. *J. Algorithms*, 11(2):208–230, 1990. `doi:10.1016/0196-6774(90)90003-W`.

37   Adam Meyerson. Online facility location. In *FOCS*, pages 426–431. IEEE Computer Society, 2001.

38   Joseph Naor, Debmalya Panigrahi, and Mohit Singh. Online node-weighted steiner tree and related problems. In *FOCS*, pages 210–219, 2011. `doi:10.1109/FOCS.2011.65`.

39   Krati Nayyar and Sharath Raghvendra. An input sensitive online algorithm for the metric bipartite matching problem. In *FOCS*, pages 505–515, 2017. `doi:10.1109/FOCS.2017.53`.

40   Mark Sellke. Chasing convex bodies optimally. In *SODA*, pages 1509–1518, 2020. `doi:10.1137/1.9781611975994.92`.

41   René Sitters. The generalized work function algorithm is competitive for the generalized 2-server problem. *SIAM J. Comput.*, 43(1):96–125, 2014. `doi:10.1137/120885309`.

42   Martin Skutella and David P. Williamson. A note on the generalized min-sum set cover problem. *Oper. Res. Lett.*, 39(6):433–436, 2011. `doi:10.1016/j.orl.2011.08.002`.

43   Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985. `doi:10.1145/2786.2793`.

44   Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, 1985. `doi:10.1145/3828.3835`.

45   Erez Timnat. The list update problem, 2016. Master Thesis, Technion- Israel Institute of Technology.