An Improved Algorithm For Online Reranking

Marcin Bienkowski\textsuperscript{1} and Marcin Mucha\textsuperscript{2}

\textsuperscript{1}Institute of Computer Science, University of Wroclaw, Poland
\textsuperscript{2}Institute of Informatics, University of Warsaw, Poland

Abstract

We study a fundamental model of online preference aggregation, where an algorithm maintains an ordered list of \( n \) elements. An input is a stream of preferred sets \( R_1, R_2, \ldots, R_t, \ldots \). Upon seeing \( R_t \) and without knowledge of any future sets, an algorithm has to rerank elements (change the list ordering), so that at least one element of \( R_t \) is found near the list front. The incurred cost is a sum of the list update costs (the number of swaps of neighboring list elements) and access cost (the position of the first element of \( R_t \) on the list). This scenario occurs naturally in applications such as ordering items in an online shop using aggregated preferences of shop customers. The theoretical underpinning of this problem is known as Min-Sum Set Cover.

Unlike previous work (Fotakis et al., ICALP 2020, NIPS 2020) that mostly studied the performance of an online algorithm \( \text{Alg} \) in comparison to the static optimal solution (a single optimal list ordering), in this paper, we study an arguably harder variant where the benchmark is the provably stronger optimal dynamic solution \( \text{Opt} \) (that may also modify the list ordering). In terms of an online shop, this means that the aggregated preferences of its user base evolve with time. We construct a computationally efficient randomized algorithm whose competitive ratio (\( \text{Alg} \)-to-\( \text{Opt} \) cost ratio) is \( O(r^2) \) and prove the existence of a deterministic \( O(r^4) \)-competitive algorithm. Here, \( r \) is the maximum cardinality of sets \( R_t \). This is the first algorithm whose ratio does not depend on \( n \): the previously best algorithm for this problem was \( O(r^{3/2} \cdot \sqrt{n}) \)-competitive and \( \Omega(r) \) is a lower bound on the performance of any deterministic online algorithm.

1 Introduction

We focus on the problem of maintaining an ordered (ranked) list of elements and updating the order to better reflect the preferences of users. This problem occurs naturally in an online shop that has to present a list of items in some order to users. Items that a user is interested in should be placed sufficiently close to the list beginning; otherwise, a user has to scroll down, which could degrade the overall experience and reduce customer retention. Similar phenomena occur not only in online shopping [DGMM20], but also in ordering results from a web search for a given keyword [DKNS01, ABD18], or ordering news and advertisements.

In the Min-Sum Set Cover (MSSC) problem, which serves as a theoretical model for this problem, there is a universe \( U \) of \( n \) elements, and a set of \( m \) users, where the \( i \)-th user has a
set of preferred elements \( R_t \subseteq U \). The goal is to find a fixed permutation of \( n \) elements, which minimizes the sum of users’ dissatisfaction, where the dissatisfaction of user \( t \) is the position of the first element from \( R_t \) in the permutation. This measures (in a perhaps simplistic way) how far a user has to scroll till an interesting item is found. This problem and its variants have been thoroughly studied in the approximation algorithms community: the best polynomial-time solution is a 4-approximation \([\text{BBH}^+98]\) and this approximation factor is tight unless \( P = \text{NP} \) \([\text{FLT04}]\).

In this paper, we study an online variant of the MSSC problem, where an algorithm has to maintain a permutation of \( n \) elements, and preferred sets appear in an online manner. Upon seeing a set \( R_t \), an algorithm (i) first has to pay an access cost equal to the position of the first element from \( R_t \); (ii) may reorder the list arbitrarily, paying the Kendall tau distance between old and new permutation (minimal number of swapped adjacent elements).\(^1\) This setting captures a frequent case where the service (e.g., a shop) is learning user preferences on the fly and has to react accordingly, without knowing the preferences of future users.

### 1.1 Competitive ratio

To measure the effectiveness of online algorithms we use one of the standard yardsticks, namely competitive analysis \([\text{BE98}]\), and we compare the overall cost of an online algorithm \( \text{Alg} \) to an optimal (offline) solution \( \text{Opt} \) on the same input instance \( \mathcal{I} \). We emphasize that we compare our algorithm to an optimal solution that can also change the permutation dynamically.

An algorithm \( \text{Alg} \) is \( c \)-competitive if there exists \( \xi \) such that for any input instance \( \mathcal{I} \), it holds that \( \text{Alg}(\mathcal{I}) \leq c \cdot \text{Opt}(\mathcal{I}) + \xi \). The competitive ratio of \( \text{Alg} \) is the infimum of values of \( c \), for which \( \text{Alg} \) is \( c \)-competitive. For randomized algorithms, we replace the cost \( \text{Alg}(\mathcal{I}) \) with its expected value \( E[\text{Alg}(\mathcal{I})] \), where the expectation is taken over random choices of \( \text{Alg} \).

### 1.2 Previous and our results

Fotakis et al. \([\text{FKK}^+20]\) studied the online MSSC problem and constructed an online \( O(r^{3/2} \cdot \sqrt{n}) \)-competitive algorithm \textsc{Move-All-Equally} (\text{Mae}), where \( r \) is the maximum cardinality of requested sets \( R_t \). Their solution, for the requested set \( R_t \), computes the position \( d \) of the first element from \( R_t \) in the current permutation and moves all elements \( d - 1 \) positions towards the list front. They showed that the competitive ratio of any deterministic algorithm cannot be lower than \( \Omega(r) \) and proved that many natural algorithms based on the \text{move-to-front} heuristic have a competitive ratio of \( \Omega(n) \). They also asked whether attaining a competitive ratio which is only a function of \( r \) is possible. (Note that \( r \ll n \) for most practical applications).

We answer this question affirmatively, providing a novel approach, which allows us to construct a randomized \( O(r^2) \)-competitive algorithm. Moreover, our result holds even if each set \( R_t \) is chosen by an adversary based on the current state of the algorithm’s list, i.e., holds also against

\(^1\)A careful reader may observe that the same cost measure is applied both to accessing the first element and to swapping two adjacent list elements. This choice is made to make the model coherent with the previous papers. However, one can easily set the swapping cost to an arbitrary constant: by the expense of small constant factors, these variants are reducible to our problem using standard rent-or-buy approaches \([\text{KMMO94}]\).
Table 1: Competitive ratios of old algorithms and algorithm Lma presented in this paper, against dynamic and static Opt. Asterisked entries are trivially implied by other ones.

|       | dynamic Opt | static Opt |
|-------|-------------|------------|
| Lma   | det         | $O(r^4)$   | * $O(r^4)$ |
| Lma   | rand        | $O(r^2)$   | * $O(r^2)$ |
| MAE   | det         | $O(r^{3/2} \cdot \sqrt{n})$ | $2^{O(\sqrt{\log n \cdot \log r})}$ |
| Mwu derand. | det       | $\Omega(r)$ | $\Omega(r)$ |
| Lower bound | det       | * $\Omega(r)$ | $\Omega(r)$ |
| Mwu   | rand        | $\Omega(r)$ | $O(1)$ |

On the technical level, to solve the problem, in Section 2 we introduce the Exponential Caching (EC) problem and show that the MSSC problem reduces to EC with the loss of constant factors. Glossing over details, EC treats the list as being split into chunks of geometrically growing sizes and captures the intuition that — neglecting constant factors — the costs depend only on the chunk index. In Section 3, we provide an algorithm for the EC problem. To circumvent the lower bound for the algorithm MAE, we move only the element that is closest to the list front, and we increase the budgets of the remaining elements in the requested set instead. Once the budgets become sufficient to pay for the element movement, the respective elements are moved to the first chunk.

1.3 Related work: static optimality

The online variant of the MSSC problem has been also considered in an easier setting where an online algorithm is compared to a static optimal solution that has to stick to a single permutation for the whole runtime [FKK+20]. We emphasize that this variant differs from the online learning setting; that is, we assume that an online algorithm is still charged for changing its permutation.

The static model forfeits optimization possibilities that occur when the preferences of the user base are evolving (e.g., due to influences from advertisements or because of seasonality). It is also worth mentioning that the costs of static and dynamic optimal solutions can differ by a factor of $O(n)$ [FKK+20].

The randomized $O(1)$-competitive solution follows by combining multiplicative weight updates (Mwu) [LW94, AHK12] with the techniques of Blum and Burch [BB00] designed for the metrical task systems. This approach has been derandomized by Fotakis et al. [FKK+20], who gave a deterministic solution with an asymptotically optimal ratio of $\Theta(r)$. 
1.4 Other related work

The variant of the problem where all sets $R_t$ are singletons, known as the list update problem, has been studied in a long line of work and admits $O(1)$-competitive solutions, see [Kam16] and references therein.

Another line of work studied a generalization of the MSSC problem where each set $R_t$ comes with a covering requirement $k_t$ and an algorithm is charged for the positions of the first $k_t$ elements from $R_t$ on the list. (The original MSSC corresponds to $k_t = 1$ for any $t$). Known solutions include $O(1)$-approximation (offline) algorithms [AGY09, BGK10, SW11, ISvdZ14, BBFT21] and $O(1)$-competitive polynomial-time solution against static optimum without reordering costs [FLPS20].

Finally, a large amount of research was devoted to efficiently learning a permutation with limited feedback, see, e.g., [HW09, YHK + 11, YHTT12, SRG13, Ail14]. While the general aim is similar to ours, the specific objectives and cost measures make these results incomparable to ours.

1.5 Problem definition and notation

In the Min-Sum Set Cover (MSSC) problem, we are given a universe $U$ of $n$ elements. For the sake of notation, we assume that a permutation of $U$ is given as a bijective mapping $U \rightarrow \{1, \ldots, n\}$, returning for any element $x \in U$ its position on the ordered list.

An input $I$ consists of an initial permutation $\pi_0$ of $U$ and a sequence of $m$ sets $R_1, R_2, \ldots, R_m$. Upon seeing set $R_t$, an online algorithm $A$ is first charged the access cost $\min_{x \in R_t} \pi(x)$. Then $A$ chooses a new permutation $\pi_t$ (possibly $\pi_t = \pi_{t-1}$) paying reordering cost $d(\pi_{t-1}, \pi_t)$, defined as the number of inversions between $\pi_{t-1}$ and $\pi_t$. Note that $d(\pi_{t-1}, \pi_t)$ is also the minimum number of swaps of adjacent elements necessary to change permutation $\pi_{t-1}$ into $\pi_t$. We emphasize that the choice of $\pi_t$ made by $A$ has to be performed without the knowledge of future sets $R_{t+1}, R_{t+2}, \ldots$ and also without the knowledge of the sequence length $m$.

In the following, $\text{Opt}$ denotes an optimal offline algorithm. For an input $I$ and an algorithm $A$, we use $A(I)$ to denote the total cost of $A$ on $I$, and $A(I, t)$ to denote the cost of $A$ in response to set $R_t$. For an integer $j$, we use $[j] = \{0, \ldots, j-1\}$.

2 Exponential Caching Problem

Without loss of generality, in the MSSC problem, we may assume that the universe cardinality is $n = 2^w - 1$, where $w \geq 1$ is an integer. To see this, observe that it is always possible to add dummy elements that are never in any requested set so that $n$ is of this form; these dummy elements are kept by $\text{Opt}$ at its list end, and thus they do not increase its cost. After such modification, the number of elements remains asymptotically the same.

2.1 Problem definition

We now define an Exponential Caching (EC) problem, whose solution will imply the solution to the MSSC problem of the asymptotically same ratio. In the EC problem, an algorithm has to
maintain a time-varying partition of elements into $w$ sets, henceforth called \textit{chunks}, whose sizes are powers of 2. That is, an algorithm has to maintain a partitioning $p : U \to [w]$. We call $p(x)$ the \textit{chunk index} of element $x$ and we say that partitioning $p$ is valid if $|p^{-1}(i)| = 2^i$.

We define chunks $S_0^p, S_1^p, \ldots, S_{w-1}^p$ where $S_i^p = p^{-1}(i)$. We usually skip $p$ in superscript if it does not lead to ambiguity. For any valid partitioning $p$ and element $x$, we use

$$\text{SIZE}(p, x) = 2^{p(x)}$$

to denote the cardinality of the chunk containing $x$ in the partitioning $p$.

An input to the EC problem is an initial partitioning $p_0$ and an online sequence of sets $R_1, R_2, \ldots, R_m$. Time is split into $m$ steps, and when set $R_t$ arrives in step $t$:

- \text{ALG} pays an access cost $\min_{x \in R_t} \text{SIZE}(p_{t-1}, x)$.

- \text{ALG} chooses a valid partitioning $p_t : U \to [w]$. For each element $x$ with $p_t(x) \neq p_{t-1}(x)$, \text{ALG} pays a movement cost equal to $\max\{\text{SIZE}(p_{t-1}, x), \text{SIZE}(p_t, x)\}$.

\textbf{Theorem 1.} If there exists a $c$-competitive (deterministic or randomized) algorithm for $\text{ALG}^E$ for the EC problem, then there exists an $O(c)$-competitive (deterministic or randomized) algorithm $\text{ALG}^S$ for the MSSC problem.

\subsection{Canonic partitioning}

To prove Theorem 1, note that MSSC and EC problems are closely related by a natural transformation from any permutation $\pi$ in the MSSC problem to a partitioning $p$ in the EC problem: Assume that the elements are ordered in a list according to $\pi$. Then, $S_0^p$ contains the first list element, $S_1^p$ the next $2^1$ elements, $S_2^p$ the next $2^2$ elements, and so on, with $S_{w-1}^p$ containing the last $2^{w-1}$ elements.

Formally, any permutation $\pi$ of the MSSC induces a \textit{canonic partitioning} $\text{cp}(\pi)$ of the EC problem, defined as

$$\text{cp}(\pi)(x) = \lfloor \log_2 \pi(x) \rfloor$$

for any $x \in U$.

Note that for any permutation $\pi$ and element $x$ it holds that

$$\text{SIZE}(\text{cp}(\pi), x) \leq \pi(x) \leq 2 \cdot \text{SIZE}(\text{cp}(\pi), x) - 1. \quad (1)$$

\subsection{Constructing online algorithm}

To show Theorem 1, we need to construct an algorithm $\text{ALG}^S$ for the MSSC problem on the basis of an existing algorithm $\text{ALG}^E$ for the EC problem.

Observe that any input $\mathcal{I}^S = (\pi_0, R_1, R_2, \ldots, R_m)$ to the MSSC problem has a corresponding input $\mathcal{I}^E = (p_0 = \text{cp}(\pi_0), R_1, R_2, \ldots, R_m)$ to the EC problem. To provide a solution to an input $\mathcal{I}^S$, our algorithm $\text{ALG}^S$ internally executes an algorithm $\text{ALG}^E$ on input $\mathcal{I}^E$. Once $\text{ALG}^E$ responds to $R_t$ by changing its partitioning from $p_{t-1}$ to $p_t$, $\text{ALG}^S$ mimics these changes by modifying its permutation $\pi_{t-1}$ into $\pi_t$, so that $\text{cp}(\pi_t) = p_t$. 

As we show below, such a definition together with (1) guarantees that the access costs of \( \text{ALG}^S \) and \( \text{ALG}^E \) are equal up to a factor of 2. Note that there are multiple ways of obtaining a permutation \( \pi_t \) satisfying \( \text{cp}(\pi_t) = p_t \). The crux is to show that it is possible to choose \( \pi_t \), so that the reordering cost of \( \text{ALG}^S \) is at most constant times higher than the movement cost of \( \text{ALG}^E \).

**Lemma 2.** For any step \( t \), it is possible to choose \( \pi_t \), such that \( \text{cp}(\pi_t) = p_t \) and \( \text{ALG}^S(I^S,t) \leq 4 \cdot \text{ALG}^E(I^E,t) \).

**Proof.** First, we observe that \( \text{ALG}^S \) may swap two elements on positions \( a \neq b \) using \( 2 \cdot |b - a| - 1 \) swaps of adjacent elements, paying \( 2 \cdot |b - a| - 1 < 2 \cdot \max\{a, b\} \). We use \( \text{swap}(a, b) \) to denote both such operation and its cost.

The movements chosen by \( \text{ALG}^E \) in step \( t \) can be expressed by a directed graph, whose vertices are chunks \( S_0, \ldots, S_{w-1} \). Each element \( x \) that changes its chunk (from \( S_{p_t^{-1}(x)} \) to \( S_{p_t(x)} \)) is encoded as a directed edge from chunk \( S_{p_t^{-1}(x)} \) to \( S_{p_t(x)} \). As both partitionings \( p_{t-1} \) and \( p_t \) are valid, the in-degree and out-degree of each vertex are equal. Thus, the graph can be partitioned into a union of edge-disjoint cycles. We treat each such cycle separately. For simplicity of the description, we assume that there is only one such cycle \( S_{i_0} \rightarrow S_{i_1} \rightarrow S_{i_2} \rightarrow \cdots \rightarrow S_{i_{k-1}} \rightarrow S_{i_0} \) (where \( i_k = i_0 \)). The general case follows by simply summing over all cycles.

For any \( j \in [k] \), let \( x_j \) be the element that is moved from chunk \( S_{i_j} \) to \( S_{i_{j+1}} \); let \( v_j = \pi_{t-1}(x_j) \). To mimic the choices of \( \text{ALG}^E \), \( \text{ALG}^S \) executes a sequence of \( k - 1 \) swaps: \( \text{swap}(v_{k-1}, v_{k-2}), \text{swap}(v_{k-2}, v_{k-3}), \ldots, \text{swap}(v_2, v_1), \text{swap}(v_1, v_0) \). It is easy to verify that once these \( \text{swap} \) operations are executed, the position of element \( x_j \) becomes equal to \( v_{j+1} \) for any \( j \in [k - 1] \), and the position of \( x_{k-1} \) becomes equal to \( v_0 \). Thus, the resulting permutation \( \pi_t \) satisfies the property \( \text{cp}(\pi_t) = p_t \).

To estimate the cost of \( \text{ALG}^S \), fix any \( j \in [k - 1] \). By \( p_{t-1} = \text{cp}(\pi_{t-1}) \), \( p_t = \text{cp}(\pi_t) \), and (1), we have

\[
\text{swap}(v_j, v_{j+1}) = \text{swap}(\pi_{t-1}(x_j), \pi_t(x_j)) \\
< 2 \cdot \max\{\pi_{t-1}(x_j), \pi_t(x_j)\} \\
< 4 \cdot \max\{\text{size}(p_{t-1}, x_j), \text{size}(p_t, x_j)\} \\
= 4 \cdot \text{ALG}^E(I^E, t, x_j),
\]

where \( \text{ALG}^E(I^E, t, x_j) \) is the movement cost of \( x_j \) in step \( t \).

Let \( s \) be the element from \( R_t \) which is the earliest on the list for permutation \( \pi_{t-1} \). By (1) and \( p_{t-1} = \text{cp}(\pi_{t-1}) \), we have \( \pi_{t-1}(s) < 2 \cdot \text{size}(p_{t-1}, s) \). Summing up,

\[
\text{ALG}^S(I^S, t) = \pi_{t-1}(s) + \sum_{j \in [k-1]} \text{swap}(v_j, v_{j+1}) \\
< 2 \cdot \text{size}(p_{t-1}, s) + \sum_{j \in [k-1]} 4 \cdot \text{ALG}^E(I^E, t, x_j) \\
< 4 \cdot \text{ALG}^E(I^E, t).
\]

\[\square\]

### 2.4 Proof of Theorem 1

Let \( \text{Opt}^S \) and \( \text{Opt}^E \) be the optimal solutions for inputs \( I^S \) and \( I^E \), respectively. To show the competitive ratio of \( \text{ALG}^S \), it remains to relate the costs of these optimal solutions.
We say that an algorithm is move-to-front based (MTF-based) if, in response to $R_i$, it chooses exactly one of the elements from $R_i$ and brings it to the list front; furthermore, it does not perform any further list reordering.

**Lemma 3.** For any input $I^S$ for the MSSC problem, there exists an (offline) MTF-based solution $\text{Mtf}^S$, such that $\text{Mtf}^S(I^S) \leq 2 \cdot \text{Opt}^S(I^S)$.

**Proof.** Based on the actions of $\text{Off}^S$ on $I^S = (\pi_0, R_1, \ldots, R_m)$, we may create an input $J^S = (\pi_0, R'_1, \ldots, R'_m)$, where $R'_i$ is a singleton set containing exactly the element from $R_i$ that is closest to the front on the list of $\text{Off}^S$.

Clearly, $\text{Off}^S(J^S) = \text{Off}^S(I^S)$. Furthermore, $J^S$ is an instance of the list update problem, for which it is known that moving the requested element to the list front is a 2-approximation [ST85]. Thus, $\text{Mtf}^S(J^S) \leq 2 \cdot \text{Off}^S(J^S)$. Finally, we observe that reordering actions of $\text{Mtf}^S(J^S)$ can be also applied to input $I^S$. While the movement cost remains then the same, the access cost can be only smaller, i.e., $\text{Mtf}^S(I^S) \leq \text{Mtf}^S(J^S)$. The lemma follows by combining the shown inequalities. \hfill \Box

**Lemma 4.** For any input $I^S$ for the MSSC problem, and the associated input $I^E$ for the EC problem, $\text{Off}^E(I^E) \leq 6 \cdot \text{Mtf}^S(I^S)$.

**Proof.** To show the lemma, it suffices to show that there exists an offline algorithm $\text{Off}^E$ satisfying $\text{Off}^E(I^E, t) \leq 6 \cdot \text{Mtf}^S(I^S, t)$ for any step $t$.

Let $\text{Off}^E$ be an (offline) algorithm for $I^E$ that in step $t$ takes the permutation $\pi_t^{MTF}$ of $\text{Mtf}^S$ and changes its partitioning to $\text{cp}(\pi_t^{MTF})$. We now compare the costs of $\text{Off}^E$ to $\text{Mtf}^S$ in step $t$, separately for access costs and movement/reordering costs.

Let $s$ be the element from $R_i$ that $\text{Mtf}^S$ has closest to the list front. The access cost of $\text{Off}^E$ is $\text{size}(\text{cp}(\pi_{t-1}^{MTF}), s)$ which by (1) is at most $\pi_{t-1}^{MTF}(s)$, the access cost of $\text{Mtf}^S$.

Let $x$ be the element that $\text{Mtf}^S$ moves to the list front and let $v = \pi_{t-1}^{MTF}(x)$. If $v = 1$, then neither $\text{Mtf}^S$ nor $\text{Off}^E$ perform any reordering/movement, and the lemma follows. Thus, we assume that $v \geq 2$. To move $x$ to the list front, $\text{Mtf}^S$ executes $v - 1$ swaps; this reordering increments the positions of all elements that originally preceded $x$.

To estimate the cost of $\text{Off}^E$, we analyze the movement costs associated with changing partitioning from $\text{cp}(\pi_{t-1}^{MTF})$ to $\text{cp}(\pi_t^{MTF})$. Let $\ell = \lfloor \log_2 v \rfloor \geq 1$. When $x$ is moved to the front, its chunk changes from $S_\ell$ to $S_0$. To describe the remaining changes, we assume that the list is ordered from left to right with the list front on the left. Then, the rightmost element of any chunk $S_0, S_1, \ldots, S_{\ell-1}$ changes its chunk to the next one. Hence, the movement cost of $\text{Off}^E$ is

$$\max\{2^\ell, 2^0\} + \sum_{j \in [\ell]} \max\{2^j, 2^{i+1}\} < 3 \cdot 2^\ell \leq 3 \cdot v \leq 6 \cdot (v - 1),$$

which is 6 times the reordering cost of $\text{Mtf}^S$. \hfill \Box

**Proof of Theorem 1.** Let $\text{Alg}^S$ be defined as in Lemma 2. Then,

$$\text{Alg}^S(I^S) \leq 4 \cdot \text{Alg}^E(I^E) \leq 4 \cdot c \cdot \text{Opt}^E(I^E) \leq 24 \cdot c \cdot \text{Mtf}^S(I^S) \leq 48 \cdot c \cdot \text{Opt}^S(I^S).$$

The inequalities follow by summing Lemma 2 over all steps, $c$-competitiveness of $\text{Alg}^E$, Lemma 4, and finally by Lemma 3. \hfill \Box
Routine 1: fetch(z), where z is any element

1 if \( p(z) > 0 \) then
2 \( \ell \leftarrow p(z) \)
3 for \( i = 0, 1, \ldots, \ell - 1 \) do
4 \( a_i \leftarrow \text{random element of } S_i \)
5 move z from \( S_\ell \) to \( S_0 \)
6 for \( i = 0, 1, \ldots, \ell - 1 \) do
7 move \( a_i \) from \( S_i \) to \( S_{i+1} \)
8 \( b(z) \leftarrow 0 \)

Algorithm 2: Lazy-Move-All-To-Front
Input: Set \( R = \{x, y_0, y_1, \ldots, y_{q-2}\} \), where \( q \leq r \) and \( p(x) \leq p(y_i) \) for \( i \in [r - 1] \)

1 pay access cost \( \text{size}(p, x) = 2^{p(x)} \)
2 execute fetch(x)
3 for \( i = 0, 1, \ldots, q - 2 \) do
4 \( b(y_i) \leftarrow b(y_i) + 2^{p(x)} \)
5 while exists \( z \) such that \( b(z) \geq 2^{p(z)} \) do
6 execute fetch(z)

3 Solving Exponential Caching

In this section, we provide an \( O(r^2) \)-competitive randomized algorithm for the Exponential Caching problem, where \( r \) is the maximum cardinality of requested sets. By Theorem 1, this will yield an \( O(r^2) \)-competitive algorithm for the Min-Sum Set Cover problem. We note that our algorithms do not require prior knowledge about \( r \).

In the following description, we skip \( t \) subscripts in the notations and use \( p \) as the current value of the partition function, and \( S_i \) as the current contents of an appropriate chunk.

Our algorithm Lazy-Move-All-To-Front (LMA) maintains budget \( b(z) \) for any element \( z \in U \). Initially, all budgets are set to zero.

At certain times, LMA wants to move an element \( z \) to chunk \( S_0 \). However, to preserve the cardinality of \( S_0 \), it needs to make space in \( S_0 \). It does so using a procedure fetch(z) defined in Routine 1. This procedure chooses a random sequence of elements and moves them to chunks of larger indexes. It also moves \( z \) to \( S_0 \) and resets its budget to zero.

To serve a set \( R = \{x, y_0, y_1, \ldots, y_{q-2}\} \), where \( q \leq r \) and \( x \) is an element of \( R \) with the smallest chunk index, LMA executes fetch(x) moving \( x \) to \( S_0 \). A natural strategy would be then to move the remaining elements \( y_i \) towards chunks with smaller indexes. However, such an approach leads to a huge competitive ratio. Instead, LMA performs these movements in a lazy manner: it increases the budgets of the remaining elements and moves the elements to \( S_0 \) once their budgets reach a certain threshold. The details of LMA are given in Algorithm 2.
3.1 Termination

We start by showing that the algorithm Algorithm 2 is well-defined, i.e., it terminates. If an element \( z \) satisfies \( b(z) \leq 2p(z) \), then we say that its budget is controlled, and it is uncontrolled otherwise.

**Observation 5.** Executing \( \text{fetch}(z) \) makes the budget of \( z \) controlled and it does not cause budgets of other elements to become uncontrolled.

**Proof.** For the elements randomly chosen within routine \( \text{fetch}(z) \), their chunk indexes are increased without changing their budgets which can only make their budgets controlled. The only element whose chunk index is decreased is \( z \) itself, but its budget is reset to zero, which trivially makes it controlled.

Note that execution of a single \( \text{fetch} \) operation may make multiple budgets controlled. □

By the observation above, the number of elements with uncontrolled budgets decreases with each iteration of the while loop in Line 5 of Algorithm 2. Thus, processing a set \( R_t \) by algorithm Lma terminates.

3.2 Potential function

We compare the cost of Lma to that of an optimal offline solution Opt. We use \( p \) and \( S_i \) to denote the partitioning function and appropriate chunks in the solution of Alg, and we use \( p^* \) and \( S^*_i \) for the corresponding notions in the solution of Opt.

In our analysis, we use four parameters: \( \alpha = 7 \), \( \gamma = 7r - 6 \), \( \beta = 21r - 11 \), \( \kappa = \lceil \log \beta \rceil \). Our analysis does not depend on the specific values of these parameters, but we require that they satisfy the following relations.

**Fact 6.** Parameters \( \alpha, \beta \) and \( \gamma \) satisfy the following relations: \( \alpha \geq 7 \), \( \gamma \geq \alpha \cdot (r - 2) + 8 \), \( \beta \geq \alpha \cdot (r - 1) + 2\gamma + 8 \). Furthermore, \( \kappa \) is an integer satisfying \( 2^\kappa \geq \beta \).

To compute the competitive ratio of Lma, we use amortized analysis. To this end, for any element \( z \in U \), we define its potential

\[
\Phi_z = \begin{cases} 
    \alpha \cdot b(z) & \text{if } p(z) \leq p^*(x) + \kappa, \\
    \beta \cdot 2^{p(z)} - \gamma \cdot b(z) & \text{if } p(z) \geq p^*(z) + \kappa + 1.
\end{cases}
\]

We also define the total potential as \( \Phi = \sum_{z \in U} \Phi_z \).

To streamline the proof, we split each step \( t \) into two parts. In the first part (studied in Sections 3.4 and 3.5), Lma executes Algorithm 2 (pays the access cost, chooses element movements, and pays for them), while Opt pays the access cost only. In the second part (studied in Section 3.6), Lma does nothing, while Opt moves its elements and pays for these movements.

We use \( \Delta\text{Lma} \), \( \Delta\text{Opt} \), and \( \Delta\Phi \) to denote the increments in costs of Lma and Opt and the total potential, respectively, associated with the currently discussed event. We show that for both step parts, it holds that \( E[\Delta\text{Lma} + \Delta\Phi] \leq O(r^2) \cdot \Delta\text{Opt} \). The competitive ratio of \( O(r^2) \) will then follow by summing this relation for both parts over all steps of the input.
3.3 Budget invariant

We start with a simple bound on the budget values.

**Observation 7.** At any time, for any element \( z \in U \), it holds that \( b(z) \leq 2 \cdot 2^{p(z)} \).

**Proof.** Note that at the beginning of any step, the budgets of all elements are controlled (\( b(z) \leq 2^{p(z)} \) for any element \( z \)). This holds trivially at the beginning as all budgets are then zeros. Moreover, by Observation 5, this property is ensured at the end of a step by the while loop in Lines 5–6 of Algorithm 2.

Within a step, the budget of an element may not be controlled because of budget increases in Line 4. However, because of this action, the budget of \( y_i \) may grow only to \( 2^{p(y_i)} + 2^{p(x)} \), which is at most \( 2 \cdot 2^{p(y_i)} \) as \( p(x) \leq p(y_i) \).

Note that \( \beta \geq 2\gamma \) by Fact 6. Thus, Observation 7 and the potential definition immediately imply the following claim.

**Corollary 8.** At any time, \( \Phi_z \geq 0 \) for any element \( z \).

3.4 Analysis of operation FETCH

To analyze the amortized cost associated with a single operation fetch, we start with calculating the change in the potential due to a movement of a random element.

**Lemma 9.** Fix a chunk index \( i \in [w - 1] \). Let \( a \) be an element chosen uniformly at random from chunk \( S_i \). If \( a \) is moved to chunk \( S_{i+1} \), then \( \mathbb{E}[\Delta \Phi_a] \leq 2^{i+2} \). The result holds even conditioned on the current partitioning of Alg.

**Proof.** We look at all \( 2^{i} \) elements from \( S_i \) and their chunk indexes in the solution of Opt. Let

\[ \tilde{S}_i = \{ z \in S_i : p^*(z) \leq i - \kappa \}. \]

Note that

\[
|\tilde{S}_i| = \sum_{j \in [i-\kappa+1]} |\{ z \in S_i : p^*(z) = j \}|
= \sum_{j \in [i-\kappa+1]} |S_i \cap S_j^*| \leq \sum_{j \in [i-\kappa+1]} |S_j^*|
\leq \sum_{j \in [i-\kappa+1]} 2^j < 2^{i-\kappa+1}.
\]

Hence, when \( a \) is chosen randomly from \( S_i \),

\[
\Pr[p^*(a) \leq i - \kappa] = \frac{|\tilde{S}_i|}{|S_i|} < \frac{2^{i-\kappa+1}}{2^i} = 2^{-\kappa+1}.
\] (2)

To upper-bound \( \mathbb{E}[\Delta \Phi_a \mid p^*(a) \leq i - \kappa] \), we consider two cases. By the potential definition, if \( p^*(a) \leq i - \kappa - 1 \), then \( \Delta \Phi_a = [\beta \cdot 2^{i+1} - \gamma \cdot b(a)] - [\beta \cdot 2^i - \gamma \cdot b(a)] = \beta \cdot 2^i \). Otherwise, \( p^*(a) = i - \kappa \), and then \( \Delta \Phi_a = [\beta \cdot 2^{i+1} - \gamma \cdot b(a)] - a \cdot b(a) \leq \beta \cdot 2^{i+1} \). Hence, in either case

\[
\mathbb{E}[\Delta \Phi_a \mid p^*(a) \leq i - \kappa] \leq \beta \cdot 2^{i+1}.
\]
Again by the potential definition,

\[ \mathbb{E}[\Delta \Phi_a \mid p^*(a) \geq i - \kappa + 1] = \alpha \cdot b(a) - \alpha \cdot b(a) = 0. \]

Combining these bounds on \( \Delta \Phi_a \) with (2) yields

\[
\begin{align*}
\mathbb{E}[\Delta \Phi_a] &= \mathbb{E}[\Delta \Phi_a \mid p^*(a) \leq i - \kappa] \cdot \Pr[p^*(a) \leq i - \kappa] \\
&+ \mathbb{E}[\Delta \Phi_a \mid p^*(a) \geq i - \kappa + 1] \cdot \Pr[p^*(a) \geq i - \kappa + 1] \\
&< \beta \cdot 2^{i+1} \cdot 2^{-\kappa+1}.
\end{align*}
\]

The lemma follows as \( \beta \leq 2^\kappa \) by Fact 6.

**Lemma 10.** Whenever LMA executes operation \( \text{fetch}(z) \), \( \mathbb{E}[\Delta \text{LMA} + \Delta \Phi] \leq 7 \cdot 2^{p(z)} - g \), where \( g \) is the value of \( \Phi_z \) right before this operation.

**Proof.** First, we estimate \( \Delta \text{LMA} \) itself due to \( \text{fetch}(z) \). Recall that the procedure \( \text{fetch} \) creates a sequence of random elements \( a_0, a_1, \ldots, a_{p(z)-1} \), where \( a_i \in S_i \) and moves each \( a_i \) from chunk \( S_i \) to \( S_{i+1} \). Furthermore, \( z \) is moved from chunk \( S_{p(z)} \) to \( S_0 \). Thus, the associated cost is

\[
\Delta \text{LMA} = \max\{2^{p(z)}, 2^0\} + \sum_{i=0}^{p(z)-1} \max\{2^i, 2^{i+1}\} = 2^{p(z)} + \sum_{i=0}^{p(z)} 2^i < 3 \cdot 2^{p(z)}. \tag{3}
\]

It remains to analyze the potential change for the moved elements: \( a_0, a_1, \ldots, a_{p(z)-1}, \) and \( z \). The potential of \( z \) before the movement is equal to \( g \) by the lemma assumption. By the definition, the potential of \( z \) after the movement is \( \alpha \cdot b(z) \), which is equal to 0 as the budget of \( z \) is reset to 0 within \( \text{fetch}(z) \) operation. Thus,

\[ \Delta \Phi_z = -g. \tag{4} \]

Finally, by Lemma 9, \( \mathbb{E}[\Delta \Phi_{a_i}] \leq 4 \cdot 2^i \) for any \( i \in [p(z)] \). Combining that with (3) and (4), and using linearity of expectation yields

\[
\begin{align*}
\mathbb{E}[\Delta \text{LMA} + \Delta \Phi] &= \Delta \text{LMA} + \mathbb{E}[\Delta \Phi_z] + \sum_{i=0}^{p(z)-1} \mathbb{E}[\Delta \Phi_{a_i}] \\
&< 3 \cdot 2^{p(z)} - g + \sum_{i=0}^{p(z)-1} 4 \cdot 2^i \\
&< 7 \cdot 2^{p(z)} - g.
\end{align*}
\]

**3.5 Amortized cost of LMA**

Now we may upper bound the total amortized cost of LMA in a single step. We split this cost into parts incurred by Lines 1–4 and Lines 5–6.

**Lemma 11.** Whenever LMA executes Lines 5–6 of Algorithm 2, it holds that \( \mathbb{E}[\Delta \text{LMA} + \Delta \Phi] \leq 0. \)
Proof. Let $z$ be the element moved in Line 6. Line 5 ensures that $b(z) \geq 2^{p(z)}$. Furthermore, $b(z) \leq 2 \cdot 2^{p(z)}$ by Observation 7. Let $\Phi_z$ be the value of the potential right before operation $\text{fetch}(z)$ is executed in Line 6. By the potential definition,

$$\Phi_z \geq \min\{\beta \cdot 2^{p(z)} - \gamma \cdot b(z), \alpha \cdot b(z)\}$$

$$\geq \min\{\beta - 2 \cdot \gamma, \alpha\} \cdot 2^{p(z)}$$

$$\geq 7 \cdot 2^{p(z)}$$

(by Fact 6).

By Lemma 10, $E[\Delta \text{LMA} + \Delta \Phi] \leq 7 \cdot 2^{p(z)} - \Phi_z \leq 0$. \hfill \qed

**Lemma 12.** Fix any step and consider its first part, where LMA pays for its access and movement costs, whereas Opt pays for its access cost. Then, $E[\Delta \text{LMA} + \Delta \Phi] \leq (\alpha \cdot (r - 1) + 8) \cdot 2^x \cdot \Delta \text{Opt} = O(r^2) \cdot \Delta \text{Opt}$.

**Proof.** Let $R = \{x, y_0, \ldots, y_{q-2}\}$ be the requested set, where $q \leq r$ and $p(x) \leq p(y_i)$ for any $i \in [q - 1]$. Let $\Phi_x, \Phi_{y_0}, \ldots, \Phi_{y_{q-2}}$ be the potentials of elements from $R$ just before the request.

It suffices to analyze the amortized cost of LMA in Lines 1–4, as the cost in the subsequent lines is at most 0 by Lemma 11. The access cost paid by LMA is $2^{p(x)}$ and by Lemma 10, the amortized cost of $\text{fetch}(x)$ is $7 \cdot 2^{p(x)} - \Phi_x$. Therefore,

$$E[\Delta \text{LMA} + \Delta \Phi] = 8 \cdot 2^{p(x)} - \Phi_x$$

As $b(y_i)$ grows by $2^{p(x)}$ for any $i \in [q - 1]$,

$$\Delta \Phi_{y_i} \leq \alpha \cdot 2^{p(x)}$$

for any $i \in [q - 1]$.

Finally, by Corollary 8,

$$\Phi_x \geq 0.$$  

(7)

Let $w \in R$ be the element with the smallest chunk index in the solution of Opt. That is, $\Delta \text{Opt} = 2^{p(w)}$. Assume first that $p(x) \leq p^*(w) + \kappa$. By (5), (6), and (7), $E[\Delta \text{LMA} + \Delta \Phi] \leq (8 + \alpha \cdot (q - 1)) \cdot 2^{p(x)} \leq (8 + \alpha \cdot (r - 1)) \cdot 2^x \cdot \Delta \text{Opt}$, and thus the lemma follows.

Therefore, in the remaining part of the proof, we assume that $p(x) \geq p^*(w) + \kappa + 1$ and we show that, in such case, $E[\Delta \text{LMA} + \Delta \Phi] \leq 0$. We consider two cases.

- If $w = x$, we may use a stronger lower bound on $\Phi_x$, i.e., $\Phi_x = \beta \cdot 2^{p(x)} - \gamma \cdot b(x) \geq (\beta - 2 \gamma) \cdot 2^{p(x)}$ (cf. Observation 7). Together with (5) and (6), this yields $E[\Delta \text{LMA} + \Delta \Phi] \leq (8 + \alpha \cdot (q - 1) - \beta + 2 \gamma) \cdot 2^{p(x)}$.

- If $w = y_j$ for some $j \in [q - 1]$, then as $p(y_j) \geq p(x) \geq p^*(y_j) + \kappa + 1$, we may use a stronger upper bound on $\Delta \Phi_{y_j}$, namely $\Delta \Phi_{y_j} \leq -\gamma \cdot 2^{p(x)}$. Together with (5), (6) (for $i \neq j$) and (7), this yields $E[\Delta \text{LMA} + \Delta \Phi] \leq (8 + \alpha \cdot (q - 2) - \gamma) \cdot 2^{p(x)}$.

In either case, Fact 6 together with $q \leq r$ ensures that $E[\Delta \text{LMA} + \Delta \Phi] \leq 0$. \hfill \qed

12
3.6 Movement of OPT

Lemma 13. Fix any step and consider its second part, where \( \text{LMA} \) does nothing, whereas \( \text{Opt} \) moves elements and pays for their movement. Then \( \Delta \text{LMA} + \Delta \Phi \leq (2\alpha + 2\gamma + \beta) \cdot 2^\kappa \cdot \Delta \text{Opt} = O(r^2) \cdot \Delta \text{Opt} \).

Proof. We focus on a single element \( z \) moved by \( \text{Opt} \). Assume that \( \text{Opt} \) changes its chunk index \( p^*(z) \) from \( a \) to \( a + d \) (where \( d \) is possibly negative).

The only element whose potential might be affected is \( z \) itself. The definition of \( \Phi_z \) has two cases, depending on whether the relation \( p(z) \leq p^*(z) + k \) holds. If this relation remains untouched by the movement, then \( \Phi_z \) remains constant. On the other hand, the relation changes only in one of the following cases.

- \( d \) is positive, \( a \leq p(z) - \kappa - 1, \) and \( a + d \geq p(z) - \kappa \);
- \( d \) is negative, \( a \geq p(z) - \kappa, \) and \( a + d \leq p(z) - \kappa - 1 \).

In the first case, \( p(z) \leq \kappa + a + d \), while in the second case \( p(z) \leq \kappa + a \). Thus, in either case, \( p(z) \leq \kappa + \max\{a, a + d\} \). We obtain

\[
\Delta \Phi_z \leq |a \cdot b(z) - \beta \cdot 2^{p(z)} + \gamma \cdot b(z)| \\
\leq (2\alpha + 2\gamma + \beta) \cdot 2^{p(z)} \\
\leq (2\alpha + 2\gamma + \beta) \cdot 2^\kappa \cdot \max\{2^a, 2^{a+d}\},
\]

where the second inequality is implied by Observation 7. Note that the cost of \( \text{Opt} \) associated with moving \( z \) is \( \max\{2^a, 2^{a+d}\} \) by the definition of the EC problem. Summing over all elements moved by \( \text{Opt} \) immediately yields \( \Delta \Phi \leq (2\alpha + 2\gamma + \beta) \cdot 2^\kappa \cdot \Delta \text{Opt} \). The lemma follows as \( \Delta \text{LMA} = 0 \) in the second part of a step.

3.7 Competitiveness

Theorem 14. \( \text{LMA} \) is \( O(r^2) \)-competitive for the Exponential Caching problem, even against adaptive-online adversaries.

Proof. Fix any input \( \mathcal{I} \) and consider any step \( t \). Let \( \Phi^t \) denote the potential right after step \( t \), and \( \Phi^0 \) be the initial potential. By Lemmas 12 and 13,

\[
E[\text{LMA}(\mathcal{I}, t) + \Phi^t - \Phi^{t-1}] = O(r^2) \cdot \text{Opt}(\mathcal{I}, t).
\]  

By summing (8) over all \( m \) steps of the input, we obtain that \( E[\text{LMA}(\mathcal{I})] + E[\Phi^m] - \Phi^0 \leq O(r^2) \cdot \text{Opt}(\mathcal{I}) \). As the initial potentials of all elements are 0 and the final potentials are non-negative by Corollary 8, \( E[\text{LMA}(\mathcal{I})] \leq O(r^2) \cdot \text{Opt}(\mathcal{I}) \).

We note that the only place where \( \text{LMA} \) uses randomness is in choosing a sequence of random elements in the procedure \( \text{fetch} \). As noted in its analysis (cf. Lemma 9), the bound on the expected amortized cost of \( \text{LMA} \) holds also conditioned on its current state, and thus it holds even if the adversary chooses the requested set \( R_t \) on the basis of the random bits of \( \text{LMA} \) used till step \( t - 1 \). 

The result of Ben-David et al. [BBK94] shows that the existence of a randomized algorithm that is $c$-competitive against adaptive-online adversaries implies the existence of a $c^2$-competitive deterministic algorithm.

**Corollary 15.** There exists an $O(r^4)$-competitive deterministic algorithm for the Exponential Caching problem.

Finally combining the results for Exponential Caching with Theorem 1 immediately gives improved guarantees for the online Min-Sum Set Cover problem.

**Theorem 16.** There exist a randomized $O(r^2)$-competitive algorithm and a deterministic $O(r^4)$-competitive algorithm for the Min-Sum Set Cover problem.

### 4 Conclusions

In this paper, we studied the online Min-Sum Set Cover problem on a universe of $n$ elements with requested sets of cardinalities at most $r$. We gave a first (randomized) algorithm whose competitive ratio does not depend on $n$: our algorithm is $O(r^2)$-competitive. While our construction implies also the existence of $O(r^4)$-competitive deterministic solution, it is unknown how to make it constructive and efficient.

Closing the gaps between our results and lower bounds is an intriguing open question: while the deterministic lower bound is $\Omega(r)$, no super-constant randomized bounds are known.

### Acknowledgements

This paper was supported by Polish National Science Centre grant 2016/22/E/ST6/00499 and by the ERC CoG grant TUgbOAT no 772346.

### References

[ABD18] Eugene Agichtein, Eric Brill, and Susan T. Dumais. Improving web search ranking by incorporating user behavior information. *SIGIR Forum*, 52(2):11–18, 2018.

[AGY09] Yossi Azar, Iftah Gamzu, and Xiaoxin Yin. Multiple intents re-ranking. In *Proc. 41st ACM Symp. on Theory of Computing (STOC)*, pages 669–678, 2009.

[AHK12] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing Systems*, 8(1):121–164, 2012.

[Ail14] Nir Ailon. Improved bounds for online learning over the permutahedron and other ranking polytopes. In *Proc. 17th Int. Conf. on Artificial Intelligence and Statistics (AISTATS)*, pages 29–37, 2014.
[BB00] Avrim Blum and Carl Burch. On-line learning and the metrical task system problem. *Machine Learning*, 39(1):35–58, 2000.

[BBFT21] Nikhil Bansal, Jatin Batra, Majid Farhadi, and Prasad Tetali. Improved approximations for min sum vertex cover and generalized min sum set cover. In *Proc. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 998–1005, 2021.

[BBH+98] Amotz Bar-Noy, Mihir Bellare, Magnús M. Halldórsson, Hadas Shachnai, and Tami Tamir. On chromatic sums and distributed resource allocation. *Information and Computation*, 140(2):183–202, 1998.

[BBK+94] Shai Ben-David, Allan Borodin, Richard M. Karp, Gabor Tardos, and Avi Wigderson. On the power of randomization in online algorithms. *Algorithmica*, 11(1):2–14, 1994.

[BE98] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[BGK10] Nikhil Bansal, Anupam Gupta, and Ravishankar Krishnaswamy. A constant factor approximation algorithm for generalized min-sum set cover. In *Proc. 21st ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 1539–1545, 2010.

[DGMM20] Mahsa Derakhshan, Negin Golrezaei, Vahideh H. Manshadi, and Vahab S. Mirrokni. Product ranking on online platforms. In *Proc. 21st ACM Conference on Economics and Computation (EC)*, page 459, 2020.

[DKNS01] Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the web. In *Proc. 10th Int. World Wide Web Conference (WWW)*, pages 613–622, 2001.

[FKK+20] Dimitris Fotakis, Loukas Kavouras, Grigoris Koumoutsos, Stratis Skoulakis, and Manolis Vardas. The online min-sum set cover problem. In *Proc. 47th Int. Colloq. on Automata, Languages and Programming (ICALP)*, pages 51:1–51:16, 2020.

[FLPS20] Dimitris Fotakis, Thanasis Lianeas, Georgios Piliouras, and Stratis Skoulakis. Efficient online learning of optimal rankings: Dimensionality reduction via gradient descent. In *Proc. 33rd Annual Conf. on Neural Information Processing Systems (NeurIPS)*, pages 7816–7827, 2020.

[FLT04] Uriel Feige, László Lovász, and Prasad Tetali. Approximating min sum set cover. *Algorithmica*, 40(4):219–234, 2004.

[HW09] David P. Helmbold and Manfred K. Warmuth. Learning permutations with exponential weights. *Journal of Machine Learning Research (JMLR)*, 10:1705–1736, 2009.

[ISvdZ14] Sungjin Im, Maxim Sviridenko, and Ruben van der Zwaan. Preemptive and non-preemptive generalized min sum set cover. *Mathematical Programming*, 145(1-2):377–401, 2014.
[Kam16] Shahin Kamali. Online list update. In *Encyclopedia of Algorithms*, pages 1448–1451. Springer, 2016.

[KMMO94] Anna R. Karlin, Mark S. Manasse, Lyle A. McGeoch, and Susan Owicki. Competitive randomized algorithms for non-uniform problems. *Algorithmica*, 11(6):542–571, 1994.

[LW94] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.

[SRG13] Aleksandrs Slivkins, Filip Radlinski, and Sreenivas Gollapudi. Ranked bandits in metric spaces: learning diverse rankings over large document collections. *Journal of Machine Learning Research (JMLR)*, 14(1):399–436, 2013.

[ST85] Daniel D. Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[SW11] Martin Skutella and David P. Williamson. A note on the generalized min-sum set cover problem. *Operations Research Letters*, 39(6):433–436, 2011.

[YHK+11] Shota Yasutake, Kohei Hatano, Shuji Kijima, Eiji Takimoto, and Masayuki Takeda. Online linear optimization over permutations. In *Proc. 22nd Int. Symp. on Algorithms and Computation (ISAAC)*, pages 534–543, 2011.

[YHTT12] Shota Yasutake, Kohei Hatano, Eiji Takimoto, and Masayuki Takeda. Online rank aggregation. In *Proc. 4th Asian Conference on Machine Learning (ACML)*, volume 25 of *JMLR Proceedings*, pages 539–553, 2012.