On the Online Min-Sum Set Cover Problem

(Wersja online problemu pokrycia zbiorami o minimalnej sumie)

Mateusz Basiak Agnieszka Tatarczuk

Praca magisterska

Promotor: dr hab. Marcin Bieńkowski

Uniwersytet Wrocławski
Wydział Matematyki i Informatyki
Instytut Informatyki

7 czerwca 2023

Abstract

We consider the Online Min-Sum Set Cover (MSSC) problem in different settings. In MSSC, the algorithm has a list S of elements and for every requested set $R \subseteq S$ it pays the cost equal to the position of the first element from R in S. Then the algorithm can permute the list and it pays the cost equal to the Kendall tau distance between the permutations.
We consider the problem in three models depending on whether the algorithm pays for permuting its list (Static OPT, Dynamic OPT) or not (Online Learning), and whether we compare the algorithm to the best permutation (Static OPT, Online Learning) or to the best algorithm which can also permute its list after every request (Dynamic OPT).
We present the DETERMINISTIC-LAZY-MOVE-ALL-TO-FRONT (DLMA) algorithm which achieves several previously undiscovered bounds. In Static OPT setting, DLMA achieves $O(|R|)$-competitive ratio and to our knowledge it is the best result among both deterministic and randomized polynomial-time algorithms. In Dynamic OPT setting, DLMA achieves $O(|R|^2)$-competitive ratio. We also prove that the class of algorithms similar to DLMA cannot achieve a better result.

Rozważamy problem pokrycia zbiorami online o minimalnej sumie w różnych wariantach. W tym problemie algorytm utrzymuje listę S i dla każdego zapytania $R \subseteq S$ o jej podzbiór płaci koszt równy pozycji elementu R najbliższego początkowi listy. Algorytm może następnie dokonać permutacji elementów listy płacąc za to koszt równy minimalnej liczbie transpozycji potrzebnej do przejścia pomiędzy permutacjami (odległość tau Kendalla).
Rozważamy zadany problem w trzech różnych wariantach, różniących się tym, czy algorytm płaci za permutowanie listy (model statyczny, model dynamiczny), czy też nie (online learning) jak również tym, czy porównujemy wyniki osiągane przez algorytm do najlepszej permutacji (model statyczny, online learning), czy też do algorytmu optymalnie zmieniającego swoją permutację po każdym zapytaniu (model dynamiczny).
Jako nasz główny wynik prezentujemy deterministyczny algorytm DLMA, który w leniwy sposób przesuwa wszystkie elementy zapytania na początek listy, co pozwala uzyskać wiele lepszych niż dotychczas znane współczynników konkurencyjności. W modelu statycznym nasz algorytm jest $O(|R|)$-konkurencyjny, co poprawia dotychczas znane wyniki dla algorytmów działających w czasie wielomianowym, zarówno deterministycznych jak i randomizowanych. W modelu dynamicznym nasz algorytm jest $O(|R|^2)$-konkurencyjny. Definiujemy również klasę algorytmów podobnych do naszego i udowadniamy, że nie mogą one osiągać lepszych współczynników konkurencyjności.

Contents

Chapter 1

Introduction

1.1 Problem description

Min-Sum Set Cover (Mssc**)** is a problem where we maintain a list of items and serve requests in the form of sets of those items. For each request, we want at least one of its elements to be close to the front of our list. Mssc was first introduced by Feige et al. [5] as an offline problem. Their notation comes from a related and well known Set Cover problem. We define the problem with terminology more consistent with the one used in online version, as it will be our main area of interest.

Definition 1.1. ***Min-Sum Set Cover problem, offline version***

- *Input $\mathcal{I}$: number of elements n, sets (often called requests) $R_1, R_2, \ldots, R_m$ where $R_i \subseteq \{1, \ldots, n\}$ for all $1 \leq i \leq m$.*
- *Output: permutation π of elements. The goal is to minimize the total cost defined as: $\sum_{i=1}^{m} \min\{\pi(a) \ : \ a \in R_i\}$ where $\pi(x)$ is the position of element x in π.*

As we can see, the order of requests R_i is not important. That is not the case in the online version of this problem, introduced by Fotakis et al. [6].

Definition 1.2. ***Min-Sum Set Cover problem, online version***

- *Input $\mathcal{I}$: number of elements n, initial permutation π_0 requests $R_1, R_2, \ldots$ where $R_t \subseteq \{1, \ldots, n\}$ for all t.*
- *Output: after each request R_t, the output is a permutation π_t.*
- *Cost: total cost is the sum of two types of cost. **Access cost** is defined similarly to the offline version, as $\sum_t \min\{\pi_{t-1}(a) : a \in R_t\}$. **Moving cost** is defined as $\sum_t d_{KT}(\pi_{t-1}, \pi_t)$ where $d_{KT}(\pi_{t-1}, \pi_t)$ is the number of inversions between π_{t-1} and π_t also known as the Kendall tau distance.*

We denote $\text{POS}_\pi(x)$ as the position of element x in permutation π. By extension, for any subset $X \in [n]$ of elements, by $\text{POS}_\pi(X)$ we will denote the smallest position in π among the elements of X. We will omit the subscript if the permutation used is clear from the context. We will assume that all sets R_t have r elements for some fixed $r \leq n$. Many competitive bounds will depend on r, especially in Chapters 4 and 5.

A special case of Online MSSC where $r = 1$ is called List Update and was studied earlier by Sleator and Tarjan [10]. Their problem definition differs slightly from Definition 1.2, as in their version algorithms can move only the requested element and do not incur any cost for that movement. They gave a 2-competitive algorithm called Move-To-Front (MTF). Using our definitions, MTF turns out to be 4-competitive for the Dynamic OPT setting. We present this result in Section 1.6.

1.2 Competitive analysis

We measure the effectiveness of our algorithms using competitive analysis [4]. In that, we compare a given algorithm ALG to the optimal solution OPT on some given input $\mathcal{I}$. By $\text{ALG}(\mathcal{I})$ we denote total cost of algorithm ALG on input $\mathcal{I}$. It is the sum of access cost $\text{ALG}^{\text{ACC}}(\mathcal{I})$ and moving cost $\text{ALG}^{\text{MOV}}(\mathcal{I})$ on that input. We use $\text{ALG}_t(\mathcal{I})$, $\text{ALG}_t^{\text{ACC}}(\mathcal{I})$ and $\text{ALG}_t^{\text{MOV}}(\mathcal{I})$, respectively, to denote these costs associated only with request R_t. For randomized algorithms, by $\mathbf{E}[\text{ALG}(\mathcal{I})]$ we denote the expected value of the cost, where the expectation is taken over random choices of ALG.

Definition 1.3. *[2] We say that* ALG *is* c***-competitive*** *if there exists* ξ *such that for every input instance* $\mathcal{I}$ *it holds that* $\text{ALG}(\mathcal{I}) \leq c \cdot \text{OPT}(\mathcal{I}) + \xi$*. The* ***competitive ratio*** *of* ALG *is the infimum of values of* c *for which* ALG *is* c*-competitive.*

Within the Online MSSC problem there are many different settings that restrict the actions of OPT and the cost of ALG. Those restrictions often allow us to achieve better competitive ratios by exploiting those limitations. In this work we will discuss algorithms for three of those settings.

In the **Online Learning** model, we assume that OPT chooses a fixed permutation π. Therefore, OPT is optimal in the sense that it chooses the permutation that achieves the lowest access cost among all the permutations. As OPT does not change its permutation, its moving cost is always 0. Moreover in this model we ignore the moving cost of ALG, counting its access cost as total cost. That allows ALG to change its permutation freely after each request.

In the **Static OPT** model, OPT chooses a fixed permutation, just as in the previous one. The difference is that here ALG pays its moving cost, which hinders its ability to move freely. Nevertheless, as we will see in Chapter 3, many algorithms from Online Learning transfer well to this setting.

In the **Dynamic OPT** model, OPT can change its permutation after each request, just as ALG. We call such OPT the **dynamic adversary**. This setting is

the most "equal" in the sense that both OPT and ALG are allowed to do the same operations and pay the same cost for them.

The Dynamic OPT model is the hardest to obtain a low competitive ratio, as OPT can make moves and therefore improve their cost. Of the other two, the Online Learning setting is easier to obtain a low competitive ratio, as algorithms do not pay their movement cost there.

1.3 Previous results

Feige et al. [5] gave a 4-approximation algorithm for Offline MSSC and proved that the existence of $(4-\varepsilon)$-approximation for any $\varepsilon > 0$ would imply that P=NP. We discuss these results in Section 1.5. The lower bound is especially interesting to us, as all polynomial-time online algorithms inherit that bound.

Online version of MSSC was first studied by Fotakis et al. [6]. They gave a series of results, first obtaining a $\Omega(r)$ lower bound for deterministic algorithms in all online models. We present that proof together with a classic expert-based algorithm for the Online Learning model in Chapter 2. Then they proceeded to algorithms for the Static OPT model, most importantly establishing tight upper bounds both for exponential-time randomized and exponential-time deterministic versions. We present them, along their other results, in Chapter 3.

Their paper was followed by Fotakis et al. [7], which builds on expert-based approach. They were able to simulate expert advice in polynomial time, therefore establishing current-best upper bounds for polynomial time algorithms in the Online Learning model (both randomized and deterministic).

For the Dynamic OPT model, in their original paper Fotakis et al. [6] described a deterministic Move-All-Equally algorithm that achieves a $O(r^{3/2} \cdot \sqrt{n})$ competitive ratio. Their result was improved by Bieńkowski and Mucha [2], who gave a $O(r^2)$-competitive randomized algorithm, which was the first algorithm for that model with a competitive ratio independent of n. We present all those results in Chapter 4.

All of the above results are summarized in the table below.

1.4 Our contributions

In Chapter 5, we present a deterministic, polynomial algorithm DETERMINISTIC-LAZY-MOVE-ALL-TO-FRONT (DLMA). It is based on the algorithm created by Bieńkowski and Mucha [2]. We simplify their idea, as we do not use the intermediate problem of Exponential Caching, which treats the list as being split into chunks of geometrically growing sizes.

We prove that DLMA is $O(r)$-competitive against Static OPT (Theorem 5.9) and $O(r^2)$-competitive against Dynamic OPT (Theorem 5.13). This improves on the previous best, non-constructive $O(r^4)$ upper bounds for polynomial deterministic

		randomized		deterministic	
		LB	UB	LB	UB
Online Learning	exp.	1	$1+\varepsilon$	$\Omega(r)$	$O(r)$
	poly.	4 [5]	11.713 [7]	$\Omega(r)$ [6]	$O(r)$ [7]
Static OPT	exp.	1	$1+\varepsilon$ [3]	$\Omega(r)$	$O(r)$ [6]
	poly.	4	$O(r^2)$ **O(r) (this thesis)**	$\Omega(r)$	$2^{O(\sqrt{\log n \cdot \log r})}$ [6] **O(r) (this thesis)**
Dynamic OPT	exp.	1	$O(r^2)$	$\Omega(r)$	$O(r^4)$ **O($\mathbf{r^2}$) (this thesis)**
	poly.	4	$O(r^2)$ [2]	$\Omega(r)$	$O(r^{3/2} \cdot \sqrt{n})$ [6] $O(r^4)$ [2] **O($\mathbf{r^2}$) (this thesis)**

Table 1.1: Known results for Online Mssc. Results without references are trivially implied by others. Our own results are **in bold**.

algorithms in those settings. The Static OPT result is tight with the corresponding lower bound. To our knowledge, this competitive ratio is also better than any known polynomial randomized algorithm for the Static OPT model. We also note, that although $O(r)$-competitive polynomial algorithm was previously shown for the Online Learning setting, our algorithm presents a vastly different, combinatorial approach and is much faster in terms of asymptotic computational complexity.

1.5 Offline scenario

In this section, we focus on the Offline Mssc problem. Feige et al. [5] presented a following lower bound for this setting.

Theorem 1.4. *[5] For every $\varepsilon > 0$ it is NP-hard to approximate* Mssc *within the ratio of $(4-\varepsilon)$.*

Proof of this theorem is very technical and we will omit it. Instead, we will focus on the Greedy algorithm that achieves the tight approximation ratio of 4.

Algorithm 1: Greedy

```
S ← {R_1, ..., R_m}, N ← {1, ..., n}, i ← 1
while S is not empty do
    choose x ∈ N that occurs the most times in S
    π(i) ← x
    remove x from N and all sets containing x from S
    i ← i + 1
fill remaining positions in π with elements from N in any order
return π
```

Theorem 1.5. *[5] The* Greedy *algorithm approximates* Mssc *within the ratio no worse than 4.*

Proof. Let us take any input $\mathcal{I} = (n, R_1, R_2, \ldots, R_m)$. By Opt we will denote the optimal solution for that input. Let X_i denote the set of requests first covered in the i-th step of the loop in the Greedy algorithm. Let $Y_i = \bigcup_{j=i}^{n} X_j$ be the set of requests not covered before the i-th step. Note that

$$\textsc{Greedy}(\mathcal{I}) = \sum_{i=1}^{n} i|X_i| = \sum_{i=1}^{n}\sum_{j=i}^{n} |X_j| = \sum_{i=1}^{n} |Y_i|.$$

Now for every request $r \in X_i$ define its price $P_r = |Y_i|/|X_i|$. As Greedy chooses x that maximizes the size of X_i, it also minimizes P_r. We define the total price as $\textsc{Price} = \sum_r P_r$. That yields

$$\textsc{Price} = \sum_r P_r = \sum_{i=1}^{n} |X_i| \cdot \frac{|Y_i|}{|X_i|} = \textsc{Greedy}(\mathcal{I}).$$

Now all we need is to prove the following lemma.

Lemma 1.6. $\textsc{Price} \leq 4 \cdot \textsc{Opt}(\mathcal{I})$

Proof. Let π^* be the permutation chosen by the algorithm Opt for input $\mathcal{I}$ and let π be the permutation chosen by Greedy for that input. Let $R_{i_1}, R_{i_2}, \ldots, R_{i_m}$ be the ordering of requests from the most expensive for Opt (the one that has its first element furthest from the list front) to the cheapest one. Furthermore, let a_k be the index of the first element of R_{i_k} in π^*. Therefore trivially $a_1 \geq a_2 \geq \ldots \geq a_m$ and $\textsc{Opt}(\mathcal{I}) = \sum_{i=1}^{m} a_i$.

Let $R_{j_1}, R_{j_2}, \ldots, R_{j_m}$ be the requests sorted in the same way, but with regards to the permutation π of Greedy. Let $b_k = P_{j_k}$ be the price of R_{j_k}. We will show that

$$\frac{b_k}{2} \leq a_{\lceil k/2 \rceil} \qquad \text{for every } k \in \{1, \ldots, m\}. \tag{1.1}$$

That immediately yields:

$$\begin{aligned}
\textsc{Price} &= \sum_{k=1}^{m} b_k \\
&= \sum_{\substack{1 \leq k \leq m \\ k \text{ odd}}} b_k + b_{k+1} \\
&\leq \sum_{1 \leq k \leq \lceil m/2 \rceil} 4 \cdot a_k \leq 4 \cdot \textsc{Opt}(\mathcal{I})
\end{aligned}$$

Now we need to show inequality (1.1). Let i denote the step in which request R_{j_k} was covered, meaning $j_k \in X_i$. Then by definition $b_k = |Y_i|/|X_i|$. As after $i-1$ steps $|Y_i|$ requests were not covered, we also have $k \leq |Y_i|$.

The step number $a_{\lceil k/2 \rceil}$ is the step of Opt in which the $\lceil k/2 \rceil$-th last request is covered, thus we want to show that at least $\lceil k/2 \rceil$ requests are not covered in Opt after $\lfloor b_k/2 \rfloor$ steps. Let us consider only requests from Y_i. By line 3 of Greedy, $\pi(i)$ is chosen to maximize $|X_i|$ among all elements, as elements chosen previously do not appear in any request from Y_i. Therefore, $|X_i|$ is the maximum number of sets from Y_i one element can cover, so in $\lfloor b_k/2 \rfloor$ steps Opt can cover at most

$$\left\lfloor \frac{b_k}{2} \right\rfloor \cdot |X_i| = \left\lfloor \frac{|Y_i|}{2|X_i|} \right\rfloor \cdot |X_i| \leq \left\lfloor \frac{|Y_i|}{2} \right\rfloor$$

requests from Y_i. Therefore there are at least $\lceil |Y_i|/2 \rceil \geq \lceil k/2 \rceil$ requests not covered after $\lfloor b_k/2 \rfloor$ steps and hence $a_{\lceil k/2 \rceil} \geq b_k/2$. □

That concludes the proof of Theorem 1.5. □

1.6 List Update problem

Before we move on to the general Online Mssc, we present a simplified version of the problem called List Update, where each request contains only one element. In this section, we will refer to each request as r_t instead of R_t, indicating that we refer to the requested element.

The classic algorithm for List Update is called Move-To-Front (Mtf). In each step, it moves the requested element to the front of the list and it does not perform any other actions in that step.

Theorem 1.7. Mtf *is* 4*-competitive for the List Update problem against dynamic adversary.*

The following proof is based on the classic proof given by Sleator and Tarjan [10], but is modified to work in the Dynamic OPT setting (where algorithms have to pay for their movement as well).

Proof. Let us denote the input as $\mathcal{I}$ and its length by m. First, as Mtf only moves r_t to the front, its number of swaps is equal to the number of elements before r_t, therefore $\text{Mtf}_t^{\text{MOV}}(\mathcal{I}) = \text{Mtf}_t^{\text{ACC}}(\mathcal{I}) - 1$. We will therefore concentrate on the access cost only.

In our proof, we will use amortized analysis and the potential function Φ. We define its value after t-th request as

$$\Phi_t = |\{(x, y) : x \text{ is before } y \text{ in Mtf and } y \text{ is before } x \text{ in Opt after } t \text{ steps}\}|.$$

In our analysis, we split each step into two parts. In the first part, Mtf moves r_t and pays its cost, while Opt pays the access cost only. In the second part, Mtf does nothing, while Opt moves its elements and pays for these movements.

Let $\mathcal{A}_t$ denote the set of elements that are closer to the front of the list than r_t before t-th request in permutations of both MTF and OPT and let $\mathcal{B}_t$ denote a similar set, but with the elements that are closer only in the permutation of MTF. Then,

$$\text{MTF}_t^{\text{ACC}}(\mathcal{I}) = |\mathcal{A}_t| + |\mathcal{B}_t| + 1, \qquad \text{OPT}_t^{\text{ACC}}(\mathcal{I}) \geq |\mathcal{A}_t| + 1. \tag{1.2}$$

Now, let us examine change in potential $\Delta\Phi_t$ that is caused by moving r_t to the front of the list by MTF. Only pairs with one of the elements being r_t were affected as the order of other elements did not change. All the elements from $\mathcal{B}_t$ were counted in Φ_{t-1} as they were before r_t in MTF and after it in OPT, and they are not counted in Φ_t as r_t moved in front of them in MTF. On the other hand, elements from $\mathcal{A}_t$ were not in Φ_{t-1}, but as r_t moved in front of them in MTF, they are counted in Φ_t. As r_t swapped places only with elements from $\mathcal{A}_t$ and $\mathcal{B}_t$, no other pairs were affected, and therefore

$$\Delta\Phi_t = |\mathcal{A}_t| - |\mathcal{B}_t|. \tag{1.3}$$

Now we combine (1.2) with (1.3).

$$\begin{aligned}\text{MTF}_t^{\text{ACC}}(\mathcal{I}) + \Delta\Phi_t &= |\mathcal{A}_t| + |\mathcal{B}_t| + 1 + |\mathcal{A}_t| - |\mathcal{B}_t| \\ &= 2 \cdot |\mathcal{A}_t| + 1 \\ &\leq 2 \cdot \text{OPT}_t^{\text{ACC}}(\mathcal{I})\end{aligned}$$

Now we examine the part when OPT moves its elements. Let $\Delta\Phi_t^{\text{OPT}}$ denote change of potential caused by those moves. Every swap made by OPT creates at most one new pair to be added to Φ, therefore $\Delta\Phi_t^{\text{OPT}} \leq \text{OPT}_t^{\text{MOV}}(\mathcal{I})$. That yields

$$\begin{aligned}\text{MTF}_t^{\text{ACC}}(\mathcal{I}) + \Delta\Phi_t + \Delta\Phi_t^{\text{OPT}} &\leq 2 \cdot \text{OPT}_t^{\text{ACC}}(\mathcal{I}) + \text{OPT}_t^{\text{MOV}}(\mathcal{I}) \\ &\leq 2 \cdot \text{OPT}_t(\mathcal{I}).\end{aligned}$$

When we sum over all requests, we obtain that $\text{MTF}^{\text{ACC}}(\mathcal{I}) + \Phi_m - \Phi_0 \leq 2 \cdot \text{OPT}(\mathcal{I})$. As both algorithms start from the same permutation, $\Phi_0 = 0$. Therefore,

$$\begin{aligned}\text{MTF}(\mathcal{I}) &= \text{MTF}^{\text{ACC}}(\mathcal{I}) + \text{MTF}^{\text{MOV}}(\mathcal{I}) \\ &\leq 2 \cdot \text{MTF}^{\text{ACC}}(\mathcal{I}) \\ &\leq 2 \cdot (\text{MTF}^{\text{ACC}}(\mathcal{I}) + \Phi_m) \\ &\leq 4 \cdot \text{OPT}(\mathcal{I}).\end{aligned}$$

□

Chapter 2

Online Learning

Now we move on to the online version of MSSC. We will start with the Online Learning model. In that model OPT is static and algorithms do not pay for their moves, and therefore we are able to obtain the lowest upper bounds on the competitive ratio. Known results are presented in Table 2.1 below.

Table 2.1: Known results for the Online Learning model

Online Learning	randomized		deterministic	
	LB	UB	LB	UB
exponential	1	$1+\varepsilon$	$\Omega(r)$	$O(r)$
polynomial	4 [5]	11.713 [7]	$\Omega(r)$ [6]	$O(r)$ [7]

Note that all presented results are tight up to a constant factor. Key result in this section is proving $(1+\varepsilon)$ exponential randomized upper bound by introducing the WMR algorithm. The randomized polynomial-time algorithm was introduced by Fotakis et al. [7], but is not discussed in detail here. Lower bound for randomized polynomial-time algorithms is implied by the offline setting, namely Theorem 1.4.

We prove the deterministic lower bound of $\Omega(r)$ in Theorem 2.5. There exists a deterministic polynomial-time algorithm that matches this lower bound. It was presented by Fotakis et al. [7] and it is a derandomization of the randomized polynomial-time algorithm mentioned above. The deterministic exponential upper bound is implied by the polynomial-time algorithm.

First, we want to describe a classic algorithm for the Online Learning model. It is widely used in many fields of computer science and therefore known by many names, such as the Randomized Weighted Majority algorithm (WMR, Littlestone & Warmuth [9]), the Hedge algorithm (Freund & Schapire [8]) or the Multiplicative Weight Update algorithm (MWU, survey by Arora et al. [1]).

2.1 Experts setting

The idea of the algorithm WMR is based on a very generic setting where, in every round t, we choose a solution to a given problem from a given finite set of possible solutions U. Each solution is represented by an "expert" that tries to convince us to use it. In the randomized version of the problem, an algorithm produces a probability vector p^t that assigns every expert a probability that we are going to choose their solution. Then, a penalty vector ℓ^t of size $|U|$ is presented to the algorithm and its expected cost for the round is $(p^t)^\top \cdot \ell^t$.

In our case, we want to choose the order of elements of our list. Therefore, each expert corresponds to a permutation π of length n and in round t the algorithm assigns it probability p^t_π. Then, the request $R_t = \{y_1, \ldots, y_r\}$ is made. The loss for each permutation π is defined as $\ell^t_\pi = \text{POS}_\pi(R_t)$.

2.2 WMR algorithm

To be able to calculate the probability vectors, WMR assigns every expert π a weight $w_\pi \in [0, 1]$. It is the measure of how much the algorithm trusts that expert and is directly proportional to the probability assigned later to that π, which is simply $w_\pi \;/\; \sum_{j=1}^{n!} w_j$. Then, after each round, weights are multiplied by β^{ℓ^t}, where $\beta \in [0, 1]$ is a parameter called the learning rate. This follows the basic intuition that the permutation with the highest penalty should have the highest decrease in probability. Learning rate allows us to control how fast the algorithm changes its weights. WMR is presented in Algorithm 2.

Algorithm 2: WMR

```
1 w_i^1 ← 1 for each i ∈ [n!]
2 for t = 1, ..., T do
3 |   p_i^t ← w_i^t / Σ_{j=1}^{n!} w_j^t for each i ∈ [n!]
4 |   return p^t
5 |   read loss ℓ^t
6 |   w_i^{t+1} ← w_i^t · β^{ℓ_i^t} for each i ∈ [n!]
```

To prove an upper bound on the competitiveness of WMR, we will use the following theorem (based on Freund & Schapire [8]).

Theorem 2.1. *Let p^t, ℓ^t for $t = 1, ..., T$ be probability and loss vectors of* WMR *described above and $\beta \in [0, 1]$. Then, for any expert $k \in [n!]$:*

$$\sum_{t=1}^{T} (p^t)^\top \ell^t \leq \frac{\ln(1/\beta) \cdot \sum_{t=1}^{T} \ell^t_k + \ln(n!)}{1 - \beta}.$$

Before we prove the theorem, let us look at its implications. The left side of the inequality in the theorem is, as we established earlier, the cost of WMR. On

the other hand, as the theorem works for any expert k, it also works for the expert representing the optimal permutation OPT, whose total cost is $\sum_{t=1}^{T} \ell^t_{\text{OPT}}$.

Therefore, the cost of WMR is upper-bounded by $\frac{\ln(1/\beta)}{1-\beta} \cdot \text{OPT}$ plus some additive cost depending only on n and β. We set $\beta = e^{-\varepsilon}$ for some $\varepsilon > 0$ and obtain:

$$\begin{aligned}
\frac{\ln(1/\beta)}{1-\beta} \cdot \text{OPT} &= \frac{\varepsilon}{1-e^{-\varepsilon}} \cdot \text{OPT} \\
&= \frac{(1+\varepsilon)\varepsilon}{(1+\varepsilon)-(1+\varepsilon)e^{-\varepsilon}} \cdot \text{OPT} \\
&\le \frac{(1+\varepsilon)\varepsilon}{(1+\varepsilon)-1} \cdot \text{OPT} \qquad \text{(as } e^{-x}(1+x) \le 1 \text{ for all } x\text{)} \\
&= (1+\varepsilon) \cdot \text{OPT}.
\end{aligned}$$

Corollary 2.2. *The* WMR *algorithm is* $(1+\varepsilon)$*-competitive for any* $\varepsilon > 0$.

Now we proceed to prove Theorem 2.1.

Proof of Theorem 2.1. The proof will be based on the potential function Φ. For any round t we define Φ_t as $\Phi_t = \sum_{i=1}^{n!} w_i^t$. First, we expand the left side of the inequality in the theorem for any given round t:

$$(p^t)^\top \ell^t = \sum_{i=1}^{n!} p_i^t \ell_i^t = \sum_{i=1}^{n!} \frac{w_i^t \ell_i^t}{\sum_{j=1}^{n!} w_j^t} = \frac{1}{\Phi_t} \sum_{i=1}^{n!} w_i^t \ell_i^t.$$

Now we analyze the change of the potential in round t.

$$\begin{aligned}
\Phi_{t+1} &= \sum_{i=1}^{n!} w_i^{t+1} = \sum_{i=1}^{n!} w_i^t \cdot \beta^{\ell_i^t} \\
&\le \sum_{i=1}^{n!} w_i^t (1-(1-\beta)\ell_i^t) \\
&= \sum_{i=1}^{n!} w_i^t - (1-\beta) \sum_{i=1}^{n!} w_i^t \ell_i^t \\
&= \Phi_t - (1-\beta)\Phi_t \cdot (p^t)^\top \ell^t \\
&\le \Phi_t \cdot \exp(-(1-\beta)(p^t)^\top \ell^t)
\end{aligned}$$

The first inequality follows from the fact that $a^r \le 1-(1-a)r$ for $a \ge 0$ and $r \in [0,1]$ by the Bernoulli inequality. The second one follows from $1-x \le e^{-x}$ for all x. Applying the above to every t from 1 to T, we obtain the inequality

$$\Phi_{T+1} \le \Phi_1 \cdot \exp\left(-(1-\beta)\sum_{t=1}^{T} (p^t)^\top \ell^t\right).$$

Now fix any expert $k \in [n!]$. As $w_k^{T+1} = 1 \cdot \beta^{\sum_{t=1}^{T} \ell_k^t}$,

$$\exp\left(\ln(\beta) \cdot \sum_{t=1}^{T} \ell_k^t\right) = \beta^{\sum_{t=1}^{T} \ell_k^t} = w_k^{T+1} \le \Phi^{T+1} \le \Phi^1 \cdot \exp\left(-(1-\beta)\sum_{t=1}^{T} (p^t)^\top \ell^t\right).$$

Reorganizing the terms and using $\Phi^1 = n!$, we get

$$\exp\left((1-\beta)\sum_{t=1}^{T}(p^t)^\top \ell^t\right) \leq \exp\left(-\ln(\beta)\cdot\sum_{t=1}^{T}\ell_k^t\right)\cdot n!.$$

Taking logarithms from both sides and dividing by $1-\beta$,

$$\sum_{t=1}^{T}(p^t)^\top \ell^t \leq \frac{\ln(1/\beta)\cdot\sum_{t=1}^{T}\ell_k^t + \ln(n!)}{1-\beta},$$

which concludes the proof. □

Obviously, WMR algorithm has exponential running time, as it changes $n!$ weights in every round. One may ask whether we need exponentially many operations. As was proved by Fotakis et al. [7], limiting ourselves only to polynomial-time algorithms still allows to get constant upper-bound on competitive ratio.

Theorem 2.3. *[7, Theorem 4] There exists a polynomial-time, randomized algorithm for* MSSC *in Online Learning model that achieves a competitive ratio of* $11.713(1+\varepsilon)$.

2.3 Generalized MSSC

Definition 2.4. ***Generalized Min-Sum Set Cover*** *(*GMSSC*) is a problem where an algorithm maintains a permutation π of n elements. In each round t, it receives a request of the form (R_t, k_t) where $R_t \subseteq \{1,\ldots,n\}$ is a subset of elements of π and $k_t \in \{1,\ldots,|R_t|\}$. Cost ℓ_π^t of the algorithm is defined as the position of the element that is k_t-th nearest from the beginning of π among elements of R_t.*

Analogously to MSSC, in the randomized version of GMSSC, in each round the algorithm returns a probability vector p on permutation space and its cost is defined as $p^\top \ell$. MSSC is a special case of GMSSC where $k_t = 1$ for every t.

The advantage of algorithms that use experts is that they are independent of exact characteristic of the problem. For example both WMR itself and its analysis do not rely on the way ℓ^t is calculated. Therefore, it is $(1+\varepsilon)$-competitive also for the GMSSC problem.

The algorithm described in Theorem 2.3 also works for GMSSC problem, although it achieves a worse competitive ratio of $28(1+\varepsilon)$ [7].

2.4 Lower bound for deterministic algorithms

Now we turn our attention to deterministic algorithms. It turns out that allowing randomization is more important than exponential time in this case, as for deterministic algorithms we can actually prove non-constant lower bound even for exponential-time algorithms. The following proof is due to Fotakis et al. [6].

Theorem 2.5. *Any deterministic online algorithm for the* MSSC *problem has competitive ratio at least* $(r+1)\cdot(1-\frac{r}{n+1})$.

Proof. Let ALG be any deterministic online algorithm. We can choose a sequence $\mathcal{I}$ of requests $R_1, R_2, \ldots, R_T$ so that R_t is the set of the last r elements in the list of ALG after $t-1$ rounds. Thus,

$$\text{ALG}(\mathcal{I}) = T\cdot(n-r+1). \tag{2.1}$$

Now we proceed to estimate the overall cost of OPT. We will calculate the average cost of all $n!$ permutations, which is an upper bound on OPT. For every request R_t, where $t\in\{1,\ldots,T\}$, we will calculate the number of permutations π that have cost exactly i, for every $i\in\{1,\ldots,n-r+1\}$.

Permutation π has to have an element of R_t on the i-th position and no elements of R_t before it. Therefore, other $r-1$ elements of R_t must be distributed among the last $n-i$ positions of π. There are $\binom{n-i}{r-1}$ ways to do it. Once the positions of elements of R_t are fixed, there are $r!$ possible assignments of those elements to positions, and $(n-r)!$ assignments of other elements to their positions. They all incur the same cost i. To sum up, there are exactly

$$\binom{n-i}{r-1}r!(n-r)!$$

permutations with cost i, which means that

$$\sum_{i=1}^{n-r+1}\binom{n-i}{r-1}r!(n-r)! = n!. \tag{2.2}$$

Now we calculate total cost of R_t over all permutations.

$$\begin{aligned}
\text{OPT}_t(\mathcal{I}) &= \sum_{i=1}^{n-r+1} i\binom{n-i}{r-1}r!(n-r)! \\
&= \sum_{i=1}^{n-r+1}\sum_{j=i}^{n-r+1}\binom{n-j}{r-1}r!(n-r)! \\
&= \sum_{i=1}^{n-r+1}\binom{n-i+1}{r}r!(n-r)! \\
&= \frac{(n+1)!}{r+1} \qquad \text{(by (2.2) for } n'=n+1 \text{ and } r'=r+1)
\end{aligned}$$

The third equality holds because the inner sum is the number of permutations that have cost greater or equal to i. Those permutations can be otherwise represented as all the ways to place r elements of R_t among the last $n-i+1$ elements of the permutation. Total cost of OPT can be upper-bounded by:

$$\text{OPT}(\mathcal{I}) \le T\cdot\frac{1}{n!}\cdot\frac{(n+1)!}{r+1} = T\cdot\frac{n+1}{r+1} \tag{2.3}$$

Combining (2.1) with (2.3) yields:

$$\frac{\text{ALG}(\mathcal{I})}{\text{OPT}(\mathcal{I})} \ge \frac{T(n-r+1)\cdot(r+1)}{T(n+1)} = (r+1)\cdot\left(1-\frac{r}{n+1}\right). \qquad \square$$

As the Online Learning model places the most restrictions on Opt and Alg does not pay for its moves, this lower bound applies also to Static OPT and Dynamic OPT models. Therefore, all deterministic algorithms in this thesis will have $\Omega(r)$ competitive ratio.

Chapter 3

Static Opt

In this section, we present known results for the Online Mssc problem against Static OPT both in randomized and deterministic setting in exponential time (Theorem 3.1 and Theorem 3.9). The polynomial-time Algorithm Mae ([6]) (which is $2^{O(\sqrt{\log n \cdot \log r})}$-competitive in this setting) will be presented in the next chapter. All of described results are based on work of Fotakis et al. [6] and Blum & Burch [3].

In this model, online algorithms compete against a static optimal algorithm. More formally, for input $\mathcal{I} = R_1, R_2 \ldots$ there is a set of offline algorithms, each of whom sets its permutation before the first request and does not change it later. The best such algorithm (the one with the lowest serving cost) is called static optimal algorithm — here referred as $\textsc{Opt}(\mathcal{I})$.

As in previous tables our results are **in bold**. The randomized polynomial upper-bound is implied by the deterministic polynomial algorithm. We will prove polynomial-time upper-bounds in Section 5.3.

		randomized		deterministic	
		LB	UB	LB	UB
Static OPT	exp.	1	$1+\varepsilon$ [3]	$\Omega(r)$	$O(r)$ [6]
	poly.	4 [5]	$O(r^2)$ [2] **O(r) (implied by deterministic setting)**	$\Omega(r)$[6]	$2^{O(\sqrt{\log n \cdot \log r})}$ [6] $O(r^4)$ [2] **O(r) (this thesis)**

Table 3.1: Known results for Online Mssc in the Static OPT setting.

3.1 Wmr-based algorithms

The Wmr algorithm has been presented in Section 2.2. Recall that the loss generated by the k-th request equals the position of the first element of the requested set in our permutation

$$\ell_\pi = \textsc{pos}_\pi(R_k).$$

Having those definitions, we can proceed to presenting an exponential randomized $(1+\delta)$-competitive algorithm for any constant $\delta > 0$.

3.1.1 $(1+\delta)$–competitive randomized algorithm

Presented solution comes with combining WMR with the results from the work of Blum & Burch [3]. It is also described in [6].

The difference between the Online Learning and Static OPT models is the movement cost paid by the algorithm. Results shown in [3] allow us to bound it.

Theorem 3.1. *There exists an $(1+\delta)$-competitive algorithm for the Online* MSSC *problem in the Static* OPT *setting based on* WMR *algorithm.*

First, we show a bound on the access cost.
Recall that w_π^t is the value of w_π just before t-th request. For the requested set R_t and probabilities p_t maintained by WMR, the expected access cost is:

$$\mathbf{E}[\mathrm{WMR}_t^{\mathrm{ACC}}] = \sum_{\pi \in [n!]} p_\pi^t \cdot \mathrm{POS}_\pi(R_t).$$

Recall that by Theorem 2.1, we have

$$\mathbf{E}[\mathrm{WMR}^{\mathrm{ACC}}(\mathcal{I})] \leq \frac{\ln(1/\beta) \cdot \mathrm{OPT}(\mathcal{I}) + \ln(n!)}{1-\beta}. \tag{3.1}$$

Assuming that any change between permutations costs 1 (we will justify this assumption in Lemma 3.4), we can bound the movement cost of WMR as well. We start with analyzing the total variation distance d between two consecutive distributions p^t and p^{t+1} of WMR, defined as:

$$d(p^t, p^{t+1}) = \sum_{i : p_i^t > p_i^{t+1}} p_i^t - p_i^{t+1}.$$

Lemma 3.2. *Let p^t denote a distribution vector over all permutations used by* WMR*, and ℓ^t a vector of losses over all permutations. Then,*

$$d(p^t, p^{t+1}) \leq \ln\left(\frac{1}{\beta}\right) p^t \cdot \ell^t.$$

Proof. Let $W^t = \sum_{\pi \in [n!]} w_\pi^t$. Then,

$$\begin{aligned}
d(p^t, p^{t+1}) &= \sum_{i:p_i^t > p_i^{t+1}} p_i^t - p_i^{t+1} \\
&= \sum_{i:p_i^t > p_i^{t+1}} \left(\frac{w_i^t}{W^t} - \frac{w_i^{t+1}}{W^{t+1}} \right) \\
&\leq \sum_{i:p_i^t > p_i^{t+1}} \left(\frac{w_i^t}{W^t} - \frac{w_i^{t+1}}{W^t} \right) \\
&\leq \sum_{i \in [n!]} \left(\frac{w_i^t}{W^t} - \frac{w_i^{t+1}}{W^t} \right) \\
&= \sum_{i \in [n!]} \frac{w_i^t}{W^t} \cdot \left(1 - \beta^{\ell_i^t} \right) \\
&\leq \sum_{i \in [n!]} p_i^t \left(\ell_i^t \ln \frac{1}{\beta} \right) \\
&= \ln \left(\frac{1}{\beta} \right) p^t \cdot \ell^t
\end{aligned}$$

The last inequality follows by $1 - e^x \leq -x$. □

Having (3.1) and Lemma 3.2 we can prove the following lemma:

Lemma 3.3. *Expected movement cost of* WMR *is at most*

$$\ln \left(\frac{1}{\beta} \right) \cdot \mathbf{E}[\text{WMR}^{\text{ACC}}].$$

Proof. In considered MSSC problem, we first serve a request, and only then we can change the state. This implies that our algorithm pays $p^t \cdot \ell^t$. Our expected movement cost equals total variation distance from Lemma 3.2. Thus,

$$\mathbf{E}[\text{WMR}_t^{\text{MOV}}] = d(p^t, p^{t+1}) \leq \ln \left(\frac{1}{\beta} \right) \cdot p^t \ell^t = \ln \left(\frac{1}{\beta} \right) \cdot \mathbf{E}[\text{WMR}_t^{\text{ACC}}]. \qquad \square$$

Combining (3.1) and Lemma 3.3 we can write:

$$\mathbf{E}[\text{WMR}(\mathcal{I})] \leq \frac{(1 + \ln(1/\beta)) \cdot (\ln(1/\beta)\text{OPT}(\mathcal{I}) + \ln(n!))}{1 - \beta},$$

therefore WMR is $f(\beta)$-competitive, where

$$f(\beta) = \frac{(1 + \ln(1/\beta)) \cdot (\ln(1/\beta))}{1 - \beta}.$$

Using $\ln(x) \geq 1 - \frac{1}{x}$, we obtain

$$f(\beta) \leq \frac{\frac{1}{\beta} \cdot \frac{1-\beta}{\beta}}{1 - \beta} = \frac{1}{\beta^2},$$

which is arbitrarily close to 1 for β tending to 1.

The last thing before proving Theorem 3.1 is to check what happens when we are not in a uniform space, but costs of the state changes are bounded by a constant D (in our problem $D \leq n^2$).

Lemma 3.4. *If the distance between states in our problem is bounded by a constant D, then* WMR *is $(1+\delta)$-competitive for any $\delta > 0$.*

Proof. Let $\widetilde{\text{WMR}}$ be an instance of WMR algorithm which receives a loss vector scaled by 1/D ($\tilde{\ell} = \frac{\ell}{D}$). The $\widetilde{\text{WMR}}$ algorithm will produce probability distribution $\tilde{p}$.
By Theorem 2.1,

$$\mathbf{E}[\widetilde{\text{WMR}}^{\text{ACC}}(\mathcal{I})] \leq \frac{\ln(1/\beta) \cdot \text{OPT}(\mathcal{I}) + D\ln(n!)}{D \cdot (1-\beta)}. \tag{3.2}$$

Combining Lemma 3.3 and the fact that a change of permutation costs at most D (instead of 1) we obtain

$$\begin{aligned}\mathbf{E}[\widetilde{\text{WMR}}^{\text{MOV}}] &\leq D \cdot \ln\left(\frac{1}{\beta}\right) \cdot \mathbf{E}[\widetilde{\text{WMR}}^{\text{ACC}}] \\ &\leq \ln\left(\frac{1}{\beta}\right) \cdot \frac{\ln(1/\beta) \cdot \text{OPT}(\mathcal{I}) + D\ln(n!)}{1-\beta}\end{aligned}$$

As the algorithm (despite using $\tilde{\ell}$ intentionally for calculating p^t) has to pay ℓ for access,

$$\mathbf{E}[\widetilde{\text{WMR}}(\mathcal{I})] \leq \frac{(1+\ln(1/\beta)) \cdot (\ln(1/\beta)\text{OPT}(\mathcal{I}) + D\ln(n!))}{1-\beta}, \tag{3.3}$$

which concludes that WMR is $f(\beta)$-competitive even in our nonuniform space. □

This also concludes proof of Theorem 3.1.

3.2 Lazy-Rounding

The Lazy-Rounding algorithm proposed by Fotakis et al. [6] is $(5r+2)$-competitive, and it is the best known deterministic non-polynomial algorithm in the Static OPT setting. The algorithm is also based on WMR.

3.2.1 WMR base

This time the algorithm will use derandomized WMR with a specific $\beta = e^{-1/n^3}$ parameter. Thus, by Theorem 2.1, using $\ln(x) \geq 1 - \frac{1}{x}$, for $n \geq 2$:

$$\mathbf{E}[\text{WMR}^{\text{ACC}}(\mathcal{I})] \leq \frac{5}{4}\text{OPT}(\mathcal{I}) + 2n^4 \ln n. \tag{3.4}$$

By Lemma 3.3, we obtain the following observation.

Observation 3.5. *Fix any steps $t_1 < t_2$. If $d(p^{t_1}, p^{t_2}) \geq \frac{1}{n}$ and $\beta = e^{\frac{-1}{n^3}}$ then*

$$\sum_{t=t_1}^{t_2-1} \mathbf{E}[\mathrm{WMR}_t^{\mathrm{ACC}}] \geq n^2.$$

The facts above will be used in the analysis of Lazy-Rounding.

3.2.2 Greedy-Rounding

Lazy-Rounding algorithm uses rounding schema called Greedy Rounding (Algorithm 3). It transforms the probability distribution μ over permutations (μ produced by WMR algorithm) and returns a permutation. However, Greedy Rounding may produce unbounded moving cost.

Algorithm 3: Greedy Rounding [6]

```
R ← [n]
for i = 1, ..., ⌈n/r⌉ do
    S^i ← argmin_{S⊆R:|S|=r} E_{π∼μ}[POS_π(S)]
    Place the elements of S^i from positions (i − 1) · r + 1 to i · r of ρ.
    R ← R \ S^i
return ρ
```

Now we show that the access cost of Greedy Rounding is at most $O(r)$ times $\mathbf{E}[\mathrm{WMR}^{\mathrm{ACC}}]$. To prove it, the following lemma will be useful.

Lemma 3.6. *Let μ be a probability distribution over permutations and $1 \leq k \leq \lceil \frac{n}{r} \rceil$. Let $S_1, \ldots S_k$ be disjoint subsets of $[n]$ of cardinality r. If $\mathbf{E}_{\pi\sim\mu}[\mathrm{POS}_\pi(S_1)] \leq \cdots \leq \mathbf{E}_{\pi\sim\mu}[\mathrm{POS}_\pi(S_k)]$ then for every $j \in \{1, 2, \ldots k\}$:*

$$\mathbf{E}_{\pi\sim\mu}[\mathrm{POS}_\pi(S_j)] \geq \frac{j+1}{2}.$$

Proof. We have

$$\begin{aligned}
&\mathbf{E}_{\pi\sim\mu}[\mathrm{POS}_\pi(S_j)] \\
&\quad \geq \frac{1}{j} \cdot \sum_{l=1}^{j} \mathbf{E}_{\pi\sim\mu}[\mathrm{POS}_\pi(S_l)] \qquad (\text{by } \mathbf{E}_{\pi\sim\mu}[\mathrm{POS}_\pi(S_1)] \leq \cdots \leq \mathbf{E}_{\pi\sim\mu}[\mathrm{POS}_\pi(S_k)]) \\
&\quad = \frac{1}{j} \sum_{l=1}^{j} \sum_{\pi} \Pr_{\mu}[\pi] \cdot \mathrm{POS}_\pi(S_l) \\
&\quad = \frac{1}{j} \sum_{\pi} \Pr_{\mu}[\pi] \cdot \sum_{l=1}^{j} \mathrm{POS}_\pi(S_l) \qquad (\mathrm{POS}_\pi(S_l) \text{ take } j \text{ different positive values}) \\
&\quad \geq \frac{1}{j} \sum_{\pi} \Pr_{\mu}[\pi] \cdot \frac{j(j+1)}{2} \\
&\quad = \frac{j+1}{2}.
\end{aligned}$$

□

Theorem 3.7. *Let μ be a distribution over permutations of* WMR *at time t and let ρ be the output of Algorithm 3 on μ. Then for any requested set R,*

$$\text{POS}_\rho(R) \leq 2r \cdot \mathbf{E}_{\pi\sim\mu}[\text{POS}_\pi(R)].$$

Proof. Let $S_1, S_2 \dots S_{\lceil \frac{n}{r} \rceil}$ be the sets from Algorithm 3. Let $e \in S_k$ be the element used by ρ to serve R. There are two cases:

1. If $R = S_k$, then by Lemma 3.6, $\mathbf{E}_{\pi\sim\mu}[\text{POS}_\pi(S)] \geq \frac{k+1}{2}$.
2. If $R \neq S_k$, then $(\bigcup_{\ell=1}^{k-1} S_\ell) \cap R = \emptyset$. Hence,

$$\mathbf{E}_{\pi\sim\mu}[\text{POS}_\pi(R)] \geq \mathbf{E}_{\pi\sim\mu}[\text{POS}_\pi(S_k)] \geq \frac{k+1}{2}$$

Since $\text{POS}_\rho(R) \leq k \cdot r$, we finally get

$$\frac{\text{POS}_\rho(R)}{\mathbf{E}_{\pi\sim\mu}[\text{POS}_\pi(R)]} \leq \frac{2 \cdot kr}{k+1} \leq 2 \cdot r. \qquad \square$$

3.2.3 Lazy-Rounding

Lazy-Rounding (given as Algorithm 4) serves the request first, and then sometimes it changes its permutation. It simulates variables from WMR and changes permutation only when $d(\pi_{\text{ALG}}, \pi_{\text{WMR}}) \geq \frac{1}{n}$. The time between two consecutive permutation changes is called a phase. During a phase the moving cost of the algorithm is 0.

Algorithm 4: Lazy-Rounding [6]

start-phase $\leftarrow 1$
for $\pi \in [n!]$ **do**
  $w_\pi^1 \leftarrow 1$
$p^1 \leftarrow$distribution from WMR algorithm $(p_\pi^1 = \frac{w_\pi^1}{\sum_{\pi\in[n!]} w_\pi^1})$
for $t = 1, 2, 3 \dots$ **do**
  if $d(p^t, p^{\text{start-phase}}) \leq \frac{1}{n}$ **then**
    $\pi_t \leftarrow \pi_{t-1}$
  else
    $\pi_t \leftarrow$ GREEDY ROUNDING(p^t)
    start-phase $\leftarrow t$
  Serve S_t using π_t
  for $\pi \in [n!]$ **do**
    $w_\pi^{t+1} \leftarrow w_\pi^t \cdot e^{-\frac{\pi^t(S_t)}{n^3}}$
  $p^{t+1} \leftarrow$distribution from WMR algorithm $(p_\pi^{t+1} = \frac{w_\pi^t}{\sum_{\pi\in[n!]} w_\pi^t})$

The analysis of Lazy-Rounding cost starts with the following:

Lemma 3.8. *Let p, p' be two probability distributions over permutations. If $d(p, p') \leq \frac{1}{n}$, then for any R:*

$$\mathbf{E}_{\pi\sim p'}[\textsc{pos}_\pi(R)] \leq 2 \cdot \mathbf{E}_{\pi\sim p}[\textsc{pos}_\pi(R)].$$

Proof.

$$\begin{aligned}
\mathbf{E}_{\pi\sim p'}[\textsc{pos}_\pi(R)] - \mathbf{E}_{\pi\sim p}[\textsc{pos}_\pi(R)] &= \sum_\pi p'(\pi) \cdot \textsc{pos}_\pi(R) - \sum_\pi p(\pi) \cdot \textsc{pos}_\pi(R) \\
&\leq n \cdot \sum_{\pi : p'(\pi) > p(\pi)} \big(p'(\pi) - p(\pi)\big) \\
&= n \cdot d(p, p') \\
&\leq 1
\end{aligned}$$

Thus,

$$\mathbf{E}_{\pi\sim p'}[\textsc{pos}_\pi(R)] \leq 2 \cdot \mathbf{E}_{\pi\sim p}[\textsc{pos}_\pi(R)]. \qquad \square$$

Now we can prove the following theorem:

Theorem 3.9. *Lazy-Rounding algorithm is $(5r + 2)$-competitive in the Static OPT model.*

Proof. First, we bound the moving cost. By t_1, t_2 we denote the initial steps of two consecutive phases. We know that $d(p^{t_1}, p^{t_2}) > \frac{1}{n}$ (by the definition of algorithm). From Observation 3.5 we get that between steps t_1 and t_2 it holds that $\mathbf{E}[\textsc{Wmr}^{\textsc{acc}}] \geq n^2$. Thus, by (3.4),

$$\textsc{Lr}(\mathcal{I})^{\textsc{mov}} \leq n^2 \leq \mathbf{E}[\textsc{Wmr}^{\textsc{acc}}(\mathcal{I})] \leq \frac{5}{4}\textsc{Opt}(\mathcal{I}) + 2n^4 \ln n.$$

Now we can bound the access cost of a single phase $[t_1, t_2)$:

$$\begin{aligned}
\textsc{Lr}^{\textsc{acc}}(\mathcal{I}) &= \sum_{t=t_1}^{t_2-1} \pi_{t_1}(S_t) \\
&\leq \sum_{t=t_1}^{t_2-1} 2r \cdot \mathbf{E}_{\pi_{t_1}\sim p^{t_1}}(S_t) && \text{(by Theorem 3.7)} \\
&\leq \sum_{t=t_1}^{t_2-1} 2 \cdot 2r \cdot \mathbf{E}_{\pi_t\sim p^t}(S_t) && \text{(by Lemma 3.8)} \\
&= 4r \cdot \mathbf{E}[\textsc{Wmr}^{\textsc{acc}}(\mathcal{I})] \\
&= 4r \cdot \left(\frac{5}{4}\textsc{Opt}(\mathcal{I}) + 2n^4 \ln(n)\right) && \text{(by (3.4))} \\
&= 5r \cdot \textsc{Opt}(\mathcal{I}) + 8r \cdot n^4 \ln(n)
\end{aligned}$$

Hence, the total cost of Lazy-Rounding is

$$\textsc{Lr}(\mathcal{I}) = \textsc{Lr}^{\textsc{acc}}(\mathcal{I}) + \textsc{Lr}^{\textsc{mov}}(\mathcal{I}) \leq \left(5r + \frac{5}{4}\right) \cdot \textsc{Opt}(\mathcal{I}) + (8r + 2) \cdot n^4 \ln n,$$

which concludes the proof. $\square$

Chapter 4

Dynamic OPT

This chapter contains known results for the Dynamic OPT setting. This time, for every set of requests $\{R_t\}$, both Opt and Alg after serving a request can change their permutation paying the Kendall tau distance between the old and the new permutation.

Table 4.1 presents known results in this setting. All lower-bounds are implied by previous settings. The randomized polynomial-time algorithm was proposed by Bieńkowski and Mucha [2]. The $O(r^4)$ result was existential only. Deterministic algorithm constitutes our contribution.

Table 4.1: Dynamic Opt known results

Dynamic OPT	randomized		deterministic	
	LB	UB	LB	UB
exponential	1	$O(r^2)$ [2]	$\Omega(r)$ [6]	$\mathbf{O(r^2)}$ **(this thesis)**
polynomial	4 [5]	$O(r^2)$ [2]	$\Omega(r)$ [6]	$O(r^{3/2} \cdot \sqrt{n})$ [6] $O(r^4)$ [2] $\mathbf{O(r^2)}$ **(this thesis)**

4.1 Algorithm Move-All-Equally

As a first example of a competitive algorithm we present Move-All-Equally. Mae (Algorithm 5) is a deterministic polynomial-time algorithm proposed by Fotakis et al. [6]. As an input, it takes a request sequence and the initial permutation. After every request, the algorithm serves it and changes the permutation.

4.1.1 MAE against Static OPT

MAE is $\Omega(r^2)$-competitive and $2^{O\left(\sqrt{\log n \cdot \log r}\right)}$-competitive against Static OPT.

Algorithm 5: Move-All-Equally [6]

```
for t = 1, 2, 3 ... do
    k_t ← min{i | π_{t-1}(i) ∈ R_t}
    Decrease the index of all elements of R_t by k_t − 1
```

Lemma 4.1. MAE *is* $\Omega(r^2)$*-competitive in Static OPT setting.*

Proof. In order to get cost r^2 times greater that optimal, an adversary always asks about the last r elements of a MAE permutation. For simplicity we will assume that $\frac{n}{r}$ is an integer. After $\frac{n}{r}$ requests the permutation of MAE is the same as at the beginning. Thus, OPT can use $\frac{n}{r}$ elements to serve all the requests and its permutation will have them at the beginning. The OPT cost for every $\frac{n}{r}$ requests is:

$$\textsc{Opt}(\mathcal{I}) = \sum_{i=1}^{\frac{n}{r}} i = \left(\frac{n}{r} + 1\right) \frac{n}{2r} \leq \frac{n}{r} \cdot \frac{n}{r},$$

while

$$\begin{aligned}\textsc{Mae}(\mathcal{I}) &= \sum_{i=1}^{\frac{n}{r}} \left(n - r + 1 + \sum_{j=n-r+1}^{n} (n - r + 1) \right) \\ &\geq \frac{n}{r}(r+1)(n-r+1).\end{aligned}$$

Thus,

$$\frac{\textsc{Mae}(\mathcal{I})}{\textsc{Opt}(\mathcal{I})} = \frac{\frac{n}{r}(r+1)(n-r+1)}{\frac{n}{r} \cdot \frac{n}{r}} = \frac{(r+1)(n-r+1)}{\frac{n}{r}} = \Omega(r^2). \qquad \square$$

4.1.2 MAE against Dynamic OPT

It is shown in [6] that MAE is $O\left(r^{3/2}\sqrt{n}\right)$-competitive against Dynamic OPT and for $r \geq 3$ it is $\Omega\left(r\sqrt{n}\right)$-competitive. In this section, we only prove the first result.

First, we define algorithm $\textsc{Mtf}_{OPT}$. After request R_t, $\textsc{Mtf}_{OPT}$ moves into the first position element e_t — the one which OPT uses to serve R_t. We will use $\textsc{Mtf}_{OPT}$ instead of OPT in comparison. It is justified by the following lemma:

Lemma 4.2 ([6])**.** *For any m and any requested sequence $\mathcal{I} = \{S_i\}_{i=1}^{m}$,*

$$\textsc{Mtf}_{OPT}(\mathcal{I}) \leq 2 \cdot \textsc{Opt}(\mathcal{I}).$$

Proof. In this proof we will use potential function Ψ equal the Kendall tau distance between $\textsc{Mtf}_{OPT}$ permutation ($\pi_{\textsc{Mtf}_{OPT}}$) and OPT permutation ($\pi_{\textsc{Opt}}$):

$$\Psi_t = d_{KT}(\pi^t_{\textsc{Mtf}_{OPT}}, \pi^t_{\textsc{Opt}}).$$

Let e_t be the element that OPT uses to serve S_t, L_t be the set of elements that are before e_t in $\pi^{t-1}_{\text{MTF}_{OPT}}$ and in π^{t-1}_{OPT}, Q_t be the set of elements that are before e_t in $\pi^{t}_{\text{MTF}_{OPT}}$ and after e_t in π^{t}_{OPT}. We can bound MTF_{OPT} cost and $\Delta\Psi$:

$$\begin{aligned}
&\text{MTF}_{OPT}(S_t) + \Psi_t - \Psi_{t-1} \\
&\qquad = L_t + Q_t + 1 + d_{KT}(\pi^t_{\text{MTF}_{OPT}}, \pi^t_{\text{OPT}}) - d_{KT}(\pi^{t-1}_{\text{MTF}_{OPT}}, \pi^{t-1}_{\text{OPT}}) \\
&\qquad = L_t + Q_t + 1 + d_{KT}(\pi^t_{\text{MTF}_{OPT}}, \pi^t_{\text{OPT}}) - d_{KT}(\pi^{t-1}_{\text{MTF}_{OPT}}, \pi^t_{\text{OPT}}) \\
&\qquad\qquad + d_{KT}(\pi^{t-1}_{\text{MTF}_{OPT}}, \pi^t_{\text{OPT}}) - d_{KT}(\pi^{t-1}_{\text{MTF}_{OPT}}, \pi^{t-1}_{\text{OPT}}) \\
&\qquad \le L_t + Q_t + 1 + (L_t - Q_t) + d_{KT}(\pi^{t-1}_{\text{MTF}_{OPT}}, \pi^t_{\text{OPT}}) \\
&\qquad\qquad - d_{KT}(\pi^{t-1}_{\text{MTF}_{OPT}}, \pi^{t-1}_{\text{OPT}}) \\
&\qquad \le 2 \cdot (L_t + 1) + d_{KT}(\pi^t_{\text{OPT}}, \pi^{t-1}_{\text{OPT}}) \\
&\qquad \le 2 \cdot \text{OPT}(\mathcal{I})
\end{aligned}$$

The first inequality follows by

$$L_t - Q_t \ge d_{KT}(\pi_{(t)\text{MTF}_{OPT}}, \pi_{(t)\text{OPT}}) - d_{KT}(\pi_{(t-1)\text{MTF}_{OPT}}, \pi_{(t-1)\text{OPT}}).$$

The second one follows by the triangle inequality for Kendall tau distance. Moreover, OPT pays at least L_t+1 for access e_t. Thus, summing over all $t \in \{1 \dots m\}$ concludes the proof. □

Using the definition of MTF_{OPT}, we can write the potential function defined for time t:

$$\Phi(t) = \sum_{j=1}^{n} \alpha^t_j \cdot \pi_t(j),$$

where:

- $\alpha^t_j = 1$ if element j is in first $c = \sqrt{n/r}$ positions of MTF_{OPT}, otherwise $\alpha^t_j = 0$
- π_t is the permutation of MAE after request R_t.

Now we can prove the following theorem.

Theorem 4.3 ([6]). *Move-All-Equally is $O\left(r^{3/2}\sqrt{n}\right)$-competitive against Dynamic OPT.*

Proof. First, let us bound the access cost of MAE and potential changes. By k_t we denote the position of the first element from request S_t in π_{MAE}. Let us observe that:

$$\pi_t(j) - \pi_{t-1}(j) \le r$$

Claim 4.4 ([6]).

$$\sum_{j=1}^{n} (\alpha^t_j - \alpha^{t-1}_j) \le \frac{\text{MTF}_{OPT}(t)}{c}.$$

The inequality above follows by the fact that for any $j \neq e_t$, $\alpha_j^t \leq \alpha_j^{t-1}$ and $\alpha_{e_t}^t - \alpha_{e_t}^{t-1} = 1$ only if MTF_{OPT} pays at least c for moving e_t. Thus,

$$\begin{aligned}
\text{MAE}^{\text{ACC}} + \Phi(t) - \Phi(t-1) &= k_t + \Phi(t) - \Phi(t-1) \\
&= k_t + \sum_{j=1}^{n} \alpha_j^t \cdot \text{POS}_t(j) - \sum_{j=1}^{n} \alpha_j^{t-1} \cdot \text{POS}_{t-1}(j) \\
&= k_t - \alpha_{e_t}^t k_t + \sum_{j \neq e_t} \alpha_j^t \cdot (\text{POS}_t(j) - \text{POS}_{t-1}(j)) \\
&\quad + \sum_{j=1}^{n} (\alpha_j^t - \alpha_j^{t-1}) \cdot \text{POS}_{t-1}(j) \\
&\leq r \cdot c + \frac{\text{MTF}_{OPT}(t)}{c} \cdot n \qquad \text{(as } \alpha_{e_t}^t = 1\text{)}.
\end{aligned}$$

Now we can bound the cost of MAE:

$$\begin{aligned}
\text{MAE}_t(\mathcal{I}) &= \text{MAE}_t^{\text{ACC}}(\mathcal{I}) + \text{MAE}_t^{\text{MOV}}(\mathcal{I}) \\
&= k_t + r k_t \\
&\leq (r+1)\left(rc + \text{MTF}_{OPTt}(\mathcal{I}) \cdot \frac{n}{c} + \Phi(t-1) - \Phi(t)\right) \\
&\leq (r+1)\left(\sqrt{nr}(1 + \text{MTF}_{OPTt}(\mathcal{I})) + \Phi(t-1) - \Phi(t)\right)
\end{aligned}$$

Summing over $t \in \{1 \dots m\}$, we get:

$$\text{MAE}(\mathcal{I}) \leq (r+1)\left(\sqrt{nr}(1 + \text{MTF}_{OPT}(\mathcal{I})) + \Phi(0) - \Phi(m)\right)$$

As $\Phi(0) \leq \Phi(m)$,

$$\text{MAE}(\mathcal{I}) = O\left(r^{3/2}\sqrt{n}\right) \cdot \text{MTF}_{OPT}(\mathcal{I}) = O\left(r^{3/2}\sqrt{n}\right) \cdot \text{OPT}(\mathcal{I}). \qquad \square$$

4.2 Exponential Caching

The Exponential Caching problem was defined by Bieńkowski and Mucha [2]. It is similar to MSSC, although this time instead of list of elements there is a list of chunks $S_0, S_1, \dots S_m$. Inside chunk S_i there are 2^i elements ($\text{SIZE}(S_i) = 2^i$). Requests are defined as in MSSC — at time t there comes request R_t. Algorithm has to serve request R_t paying *access* cost depending on the smallest chunk containing an element from R_t, and then can reorganize the elements paying *move* cost. Costs are defined below:

- access cost for R_t is $\min_{\{S_i | x \in R_t \cap S_i \neq \emptyset\}} \text{SIZE}(S_i)$,
- cost of moving element x from S_i to S_j equals $\max\{\text{SIZE}(S_i), \text{SIZE}(S_j)\}$.

Moreover we will use $p(x)$ to denote the index of chunk containing element x.

There is a reduction between EC and MSSC — algorithm for EC can be used

Routine 6: FETCHRAND(z), where z is any element [2]

```
1 if POS(z) > 0 then
2     ℓ ← POS(z)
3     for i = ℓ − 1, ..., 2, 1 do
4         a_i ← random element of S_i
5     move z from S_ℓ to S_0
6     for i = ℓ − 1, ..., 2, 1 do
7         move a_i from S_i to S_{i+1}
8 b(z) ← 0
```

Algorithm 7: 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 $i \in [q-2]$,

```
1 pay access cost 2^{p(x)}
2 execute FETCHRAND(x)
3 for i = 0, 1, ..., q − 2 do
4     b(y_i) ← b(y_i) + 2^{p(x)}
5 while exists z such that b(z) ≥ 2^{p(z)} do
6     execute FETCH(z)
```

to solve MSSC with a loss of a constant factor [2]. The core of the reduction is the observation that any list from MSSC can be trivially transformed into chunks (first element into chunk S_0, the following two into chunk S_1 and so on). In the EC problem, algorithm does not care about the order of the elements inside a chunk, but it pays more for changing the chunk of element.

4.3 Lazy-Move-All-to-Front Algorithm

The $O\left(r^2\right)$-competitive randomized algorithm called Lazy-Move-All-to-Front (Algorithm 7) proposed by Bieńkowski and Mucha [2] solves the Exponential Caching problem. The idea behind the algorithm is to accumulate budgets $b(\cdot)$ for requested elements and move them towards front only when their budgets reach a specific value — for element z this value equals $2^{p(z)}$.

Our algorithm is based on Lazy-Move-All-to-Front — we derandomized it and modified it to solve MSSC directly without an intermediate Exponential Caching problem. Thus, we omit here all the proofs which are similar in our version presented in the next chapter.

Chapter 5

Our contribution

In this chapter, we introduce the polynomial algorithm called Deterministic-Lazy-Move-All-To-Front (Dlma). We prove that it is $O(r)$-competitive against the static adversary and $O(r^2)$-competitive against the dynamic adversary. Both of those bounds are the best known for polynomial, deterministic algorithms in their respective models. The result for the Static OPT is asymptotically optimal and it also implies a randomized $O(r)$-competitive solution, which is better than any previously known randomized algorithm.

Without loss of generality, we assume that the number of elements n is $2^h - 1$, where $h \geq 1$ 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 elements are kept by Opt at its list end, and thus they do not increase its cost. After such modification, the number of elements remains asymptotically the same.

5.1 Definition of Dlma

Our algorithm Deterministic-Lazy-Move-All-To-Front is inspired by the one presented by Bieńkowski and Mucha [2] sketched in the previous chapter. We skip the reduction to Exponential Caching problem, therefore our algorithm does not divide its list into chunks of exponential size. That creates a problem, as in Exponential Caching, when we move an element z to the front, only a fraction of other elements are moved back. Therefore even if the budgets of some of the moving elements could increase significantly, there was a small probability that these elements would be in fact chosen to move. In our case, all elements before z are moved back one spot. Therefore we introduce new components to the potential to amortize the cost of moving these elements back.

As mentioned above, we do not use chunks in the definition of our algorithm. However, we define the set of virtual chunks and use them in our proof. In the following description, we skip t subscripts in the notations and for element x we use $\textsc{pos}(x) = 2^{p(x)} + q(x)$ as the *current* value of the position of element x, and S_i as the *current* elements on positions from $\{2^i, 2^i, \ldots, 2^{i+1} - 1\}$, that is contents of an

Routine 8: FETCH(z), where z is any element

```
1 if POS(z) > 0 then
2     ℓ ← POS(z)
3     for i = ℓ − 1, ..., 2, 1 do
4         SWAP(π_i, π_{i+1})
5 b(z) ← 0
```

Algorithm 9: DETERMINISTIC-LAZY-MOVE-ALL-TO-FRONT
Input: Set $R = \{x, y_0, y_2, \ldots, y_{q-2}\}$, where $q \leq r$ and $\text{POS}(x) \leq \text{POS}(y_i)$ for $i \in [q-2]$, current permutation π of elements

```
1 pay access cost = POS(x)
2 execute FETCH(x)
3 for i = 0, 1, ..., q − 2 do
4     b(y_i) ← b(y_i) + (1/r) · POS(x)
5 while exists z such that b(z) ≥ POS(z) do
6     execute FETCH(z)
```

appropriate virtual chunk. Our algorithm maintains budget $b(z)$ for any element $z \in U$. Initially, all budgets are set to zero.

At certain times, DLMA wants to move an element z to the front. Therefore it needs to push all elements in front of z back one spot. It does so using a procedure FETCH(z) defined in Routine 8. This procedure goes through elements of π from the position of z to the front and swaps z with them one by one. It also resets the budget of z to zero.

In order to serve a request $R = \{x, y_0, y_1, \ldots, y_{r-2}\}$ (where for all y_i, $\text{POS}(x) < \text{POS}(y_i)$), DLMA executes routine FETCH(x). After that, the element x is at the front of the list. It would be natural to move all the other requested elements towards the front of the list, however DLMA does so in lazy manner — similarly to the LMA algorithm. Instead of moving $y_i \in R$, DLMA increases its budget and moves it to the front once its budget reaches certain threshold. The details are given in Algorithm 9.

5.2 Termination

First, we show that Algorithm 9 terminates after every request. If an element z satisfies $b(z) \leq \text{POS}(z)$, we will call its budget *controlled*, otherwise we will call it *uncontrolled*.

Observation 5.1. *Executing* FETCH(z) *makes the budget of z controlled and it does not cause budgets of other elements to become uncontrolled.*

Proof. At the beginning of routine FETCH(z) element z is moved to the front and the first $\text{POS}(z) - 1$ elements have their positions increased without budget change,

therefore their budgets can only become controlled. The budget of element z is set to 0 and also becomes 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 9. Thus, processing a set R_t by algorithm DLMA terminates.

5.3 Competitiveness in the Static OPT model

5.3.1 Potential function

Analyzing the competitive ratio of DLMA, we compare the cost of our algorithm to the one of OPT — optimal offline solution. As mentioned above, we use POS to denote the current position in permutation of DLMA and, respectively, by POS^* we denote the position in current permutation of OPT.

In our analysis, we use five parameters: $\alpha = 13$, $\gamma = 27r$, $\beta = 27r + 54$, $\kappa = \lceil \log \beta \rceil$, $\xi = 2$. Our analysis does not depend on the specific values of these parameters, but we require that they satisfy the following relations.

Fact 5.2. *Parameters α, β, γ, κ and ξ satisfy the following relations: $\alpha \geq 13$, $\gamma \geq r \cdot (14 + \alpha)$, $\beta \geq 14 + \alpha + (1 + 1/r) \cdot \gamma$, $\xi \geq 2$. Furthermore, κ is an integer satisfying $2^\kappa \geq \beta$.*

In competitive analysis, we use amortized cost. To do so we define potential for any element $z \in U$ as

$$\Phi_z = \begin{cases} \alpha \cdot b(z) & \text{if } p(z) \leq p^*(z) + \kappa - 1, \\ \alpha \cdot b(z) + \xi \cdot \beta \cdot q(z) & \text{if } p(z) = p^*(z) + \kappa, \\ \beta \cdot \text{POS}(z) - \gamma \cdot b(z) + \xi \cdot \beta \cdot q(z) & \text{if } p(z) \geq p^*(z) + \kappa + 1. \end{cases}$$

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

We use Δ notation to denote the cost changes referring to currently analyzed action of algorithms DLMA, OPT and potential, e.g., ΔDLMA, ΔOPT, $\Delta\Phi$. We show that, for every step, it holds that $\Delta\text{DLMA} + \Delta\Phi \leq O(r) \cdot \Delta\text{OPT}$. The competitive ratio of $O(r)$ will then follow by summing this relation over all steps of the input.

5.3.2 Budget invariant

First, we upper-bound the budgets.

Observation 5.3. *At any time, for any element $z \in U$, it holds that $b(z) \leq (1 + 1/r) \cdot \text{POS}(z)$.*

Proof. Note that, by Observation 5.1 and the while loop in Lines 5–6 of Algorithm 9, between requests all the budgets are controlled ($b(z) \leq \text{POS}(z)$ for any element z).

Within a step, the budget of an element may be increased in Line 4 by at most $1/r \cdot \textsc{pos}(x)$. The only elements which can have their budgets increased are y_i from request R. Thus for y_i its new budget is bounded by $\textsc{pos}(y_i) + 1/r \cdot \textsc{pos}(x)$, which is at most $(1 + 1/r) \cdot \textsc{pos}(y_i)$ as $\textsc{pos}(x) \leq \textsc{pos}(y_i)$. □

By Fact 5.2, it is true that $\beta \geq (1+1/r)\cdot\gamma$. Combining that with Observation 5.3 and the potential definition yields the following claim.

Corollary 5.4. *At any time, $\Phi_z \geq 0$ for any element z.*

5.3.3 Analysis of operation FETCH

To analyze the amortized cost associated with a single operation FETCH, we start with calculating the changes in the potential due to a movement of elements that move one position further from the start of the list.

Lemma 5.5. *Whenever* DLMA *executes operation* $\textsc{fetch}(z)$*, for any element x, initially satisfying* $\textsc{pos}(x) < \textsc{pos}(z)$*:*

$$\Delta\Phi_x \leq \begin{cases} 0 & \text{if } p(x) \leq p^*(x) + \kappa - 1, \\ \xi \cdot \beta & \text{if } p(x) = p^*(x) + \kappa \text{ and } p'(x) = p^*(x) + \kappa, \\ 2^{p(x)} \cdot (2\beta - \xi\beta) + \xi\beta & \text{if } p(x) = p^*(x) + \kappa \text{ and } p'(x) = p^*(x) + \kappa + 1, \\ (\xi + 1) \cdot \beta & \text{if } p(x) \geq p^*(x) + \kappa + 1, \end{cases}$$

where $\textsc{pos}(x) = 2^{p(x)} + q(x)$ *is the position of x before the move and* $\textsc{pos}(x) + 1 = 2^{p'(x)} + q'(x)$ *is the position of x after the move.*
For $\xi = 2$ the third case is equal to $\xi\beta$.

Proof. Let us examine the cases one by one.

- In the first case, if $p'(x) \leq p^*(x) + \kappa - 1$, then $\Phi_x = \alpha \cdot b(x)$ both before and after the move. As the budget does not change, neither does the potential. The only time when that is not the case is when $p'(x) = p^*(x) + \kappa$, but then $\textsc{pos}(x) = 2^{p(x)+1} - 1$, and therefore $\textsc{pos}(x) + 1 = 2^{p(x)+1}, q'(x) = 0$, so the potential does not change.

- In the second case, the budget of x is unchanged, so $\Delta\Phi_x = \xi\beta(q'(x) - q(x)) = \xi\beta$.

- In the third case, as element x changed chunks after the move, it means that $\textsc{pos}(x) = 2^{p(x)+1} - 1$. Therefore $q'(x) = 0$ and $q(x) = 2^{p(x)} - 1$, so:

$$\begin{aligned} \Delta\Phi_x &= \beta \cdot 2^{p(x)+1} - \gamma \cdot b(x) + \xi\beta \cdot 0 - \alpha \cdot b(x) - \xi\beta(2^{p(x)} - 1) \\ &\leq \beta \cdot 2^{p(x)+1} - \xi\beta(2^{p(x)} - 1) \\ &= 2^{p(x)}(2\beta - \xi\beta) + \xi\beta. \end{aligned}$$

- In the last case, we have:

$$\begin{aligned}\Delta\Phi_x &= \beta \cdot (\textsc{pos}(x) + 1 - \textsc{pos}(x)) - \gamma(b(x) - b(x)) + \xi\beta(q'(x) - q(x)) \\ &\leq \beta + \xi\beta. \qquad \square\end{aligned}$$

Lemma 5.6. *Whenever* DLMA *executes operation* FETCH(z)*, it holds that* $\Delta\textsc{Dlma} + \Delta\Phi \leq 13 \cdot \textsc{pos}(z) - g$*, where* g *is the value of* Φ_z *right before this operation.*

Proof. First, we estimate $\Delta\textsc{Dlma}$ due to FETCH(z). The procedure FETCH moves z to the beginning of the permutation and the elements from before $\textsc{pos}(z)$ towards the end of the list. Thus, the associated cost is

$$\Delta\textsc{Dlma} = \textsc{pos}(z). \tag{5.1}$$

It remains to analyze the potential change for all $\textsc{pos}(z)$ moved elements. The operation FETCH(z) resets the budget of z to 0. By definition, the potential Φ_z after the movement is $\alpha \cdot b(z)$, so it is also equal to 0. Thus, by the lemma assumption on the starting value of Φ_z,

$$\Delta\Phi_z = -g. \tag{5.2}$$

Finally, by Lemma 5.5, $\Delta\Phi_x \leq (\xi+1)\cdot\beta$. The same lemma states that $\Delta\Phi_x \geq 0$ only if

$$p(x) \geq p^*(x) + \kappa. \tag{5.3}$$

We can observe that in chunk S_i there can be at most $2 \cdot 2^{i-\kappa}$ elements that satisfy (5.3). Therefore the total number of such elements is at most $\sum_{i=0}^{p(z)} 2 \cdot 2^{i-\kappa} \leq 4 \cdot 2^{p(z)-\kappa}$. Combining that with (5.1) and (5.2) yields:

$$\begin{aligned}\Delta\textsc{Dlma} + \Delta\Phi &= \Delta\textsc{Dlma} + \sum_{x:\textsc{pos}(x)<\textsc{pos}(z)} \Delta\Phi_x + \Delta\Phi_z \\ &\leq \textsc{pos}(z) + 4 \cdot 2^{p(z)-\kappa} \cdot (\xi+1)\beta - g \\ &\leq \textsc{pos}(z) + 12 \cdot 2^{p(z)-\kappa} \cdot \beta - g \\ &\leq \textsc{pos}(z) + 12 \cdot 2^{p(z)} - g \qquad \text{(by Fact 5.2)} \\ &\leq 13 \cdot \textsc{pos}(z) - g. \qquad \square\end{aligned}$$

5.3.4 Amortized cost of DLMA

Now we may split the cost of single DLMA step into parts incurred by Lines 1–4 and Lines 5–6 and bound them separately.

Lemma 5.7. *Whenever* DLMA *executes Lines 5–6 of Algorithm 9, it holds that* $\Delta\textsc{Dlma} + \Delta\Phi \leq 0$.

Proof. Let z be the element moved in Line 6. We want to use Lemma 5.6, and therefore we need to lower bound the value Φ_z of the potential right before operation FETCH(z) is executed in Line 6. As the element z is moving, Line 5 guarantees that

$b(z) \geq \textsc{pos}(z)$. Furthermore, $b(z) \leq (1 + 1/r) \cdot \textsc{pos}(z)$ by Observation 5.3. By the potential definition,

$$\begin{aligned}\Phi_z &\geq \min\{\alpha \cdot b(z) + \xi \cdot \beta \cdot q(z)\,, \beta \cdot \textsc{pos}(z) - \gamma \cdot b(z) + \xi \cdot \beta \cdot q(z)\} \\ &\geq \min\{\alpha \cdot b(z)\,, \beta \cdot \textsc{pos}(z) - \gamma \cdot b(z)\} \\ &\geq \min\left\{\alpha\,, \beta - \left(1 + \tfrac{1}{r}\right) \cdot \gamma\right\} \cdot \textsc{pos}(z) \\ &\geq 13 \cdot \textsc{pos}(z) \qquad \text{(by Fact 5.2).}\end{aligned}$$

By Lemma 5.6, $\Delta\textsc{Dlma} + \Delta\Phi \leq 13 \cdot \textsc{pos}(z) - \Phi_z \leq 0$. □

Lemma 5.8. *Fix any step and consider its first part, where* Dlma *pays for its access and movement costs, whereas* Opt *pays for its access cost. Then,* $\Delta\textsc{Dlma} + \Delta\Phi \leq (14 + \alpha) \cdot 2^{\kappa} \cdot \Delta\textsc{Opt} = O(r) \cdot \Delta\textsc{Opt}$.

Proof. Let $R = \{x, y_0, \ldots, y_{q-2}\}$ be the requested set, where $q \leq r$ and $\textsc{pos}(x) < \textsc{pos}(y_i)$ 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 Dlma in Lines 1–4, as the cost in the subsequent lines is at most 0 by Lemma 5.7. The access cost paid by Dlma is $\textsc{pos}(x)$ and by Lemma 5.6, the amortized cost of $\textsc{fetch}(x)$ is $13 \cdot \textsc{pos}(x) - \Phi_x$. Therefore,

$$\Delta\textsc{Dlma} + \Delta\Phi \leq 14 \cdot \textsc{pos}(x) - \Phi_x + \textstyle\sum_{i \in [q-1]} \Delta\Phi_{y_i}. \tag{5.4}$$

As $b(y_i)$ grows by $1/r \cdot \textsc{pos}(x)$ for any $i \in [q-1]$, and those elements do not move,

$$\Delta\Phi_{y_i} \leq \alpha \cdot \tfrac{1}{r} \cdot \textsc{pos}(x) \qquad \text{for any } i \in [q-1]. \tag{5.5}$$

Finally, by Corollary 5.4,

$$\Phi_x \geq 0. \tag{5.6}$$

Let $w \in R$ be the element with the smallest index in the solution of Opt. That is, $\Delta\textsc{Opt} = \textsc{pos}^*(w)$.

Assume first that $p(x) \leq p^*(w) + \kappa$. By (5.4), (5.5), and (5.6), $\Delta\textsc{Dlma} + \Delta\Phi \leq \leq (14 + (q-1) \cdot \alpha/r) \cdot \textsc{pos}(x) \leq (14 + \alpha) \cdot 2^{\kappa} \cdot \Delta\textsc{Opt}$, and thus the lemma follows.

Therefore, in the remaining part of the proof, we assume that $p(x) > p^*(w) + \kappa$ and we show that, in such case, $\Delta\textsc{Dlma} + \Delta\Phi \leq 0$. We consider two cases.

- If $w = x$, we may use a stronger lower bound on Φ_x, i.e., $\Phi_x \geq \beta \cdot \textsc{pos}(x) - \gamma \cdot b(x) \geq (\beta - (1 + 1/r)\gamma) \cdot \textsc{pos}(x)$ (cf. Observation 5.3). Together with (5.4) and (5.5), this yields

$$\begin{aligned}\Delta\textsc{Dlma} + \Delta\Phi &\leq (14 + (q-1) \cdot \alpha/r - \beta + (1 + 1/r)\gamma) \cdot \textsc{pos}(x) \\ &\leq (14 + \alpha - \beta + (1 + 1/r)\gamma) \cdot \textsc{pos}(x).\end{aligned}$$

- If $w = y_j$ for some $j \in [q-1]$, then as $p(y_j) \geq p(x) > p^*(y_j) + \kappa$, we may use a stronger upper bound on $\Delta\Phi_{y_j}$, namely $\Delta\Phi_{y_j} \leq -\gamma \cdot 1/r \cdot \textsc{pos}(x)$. Together with (5.4), (5.5) (for $i \neq j$) and (5.6), this yields

$$\Delta\textsc{Dlma} + \Delta\Phi \leq (14 + (q-2) \cdot \alpha/r - \gamma/r) \cdot \textsc{pos}(x).$$

In either case, Fact 5.2 together with $q \leq r$ ensures that $\Delta\text{DLMA} + \Delta\Phi \leq 0$. □

Theorem 5.9. DLMA *is $O(r)$-competitive in the Static OPT model.*

Proof. Fix any input $\mathcal{I}$ and consider any step t. Let Φ^t denote the potential right after step t, and Φ^0 be the initial potential. By Lemma 5.8,

$$\text{DLMA}_t(\mathcal{I}) + \Phi^t - \Phi^{t-1} = O(r) \cdot \text{OPT}_t(\mathcal{I}). \tag{5.7}$$

By summing (5.7) over all m steps of the input, we obtain that $\text{DLMA}(\mathcal{I}) + \Phi^m - \Phi^0 \leq O(r) \cdot \text{OPT}(\mathcal{I})$. As the initial potentials of all elements are 0 and the final potentials are non-negative by Corollary 5.4, $\text{DLMA}(\mathcal{I}) \leq O(r) \cdot \text{OPT}(\mathcal{I})$. □

Theorem 5.10. *There exist a polynomial, deterministic $O(r)$-competitive algorithm for the* MSSC *problem in the Static OPT setting.*

5.4 Competitiveness in the Dynamic OPT model

Now we move on to compare DLMA against dynamic adversary. First, we establish an an offline approximation of OPT that will be easier to analyze.

5.4.1 MTF-based approximation of OPT

We say that an algorithm is *move-to-front based (*MTF*-based)* if, in response to R_t, it chooses exactly one of the elements from R_t, brings it to the front of the list and does not perform any further actions.

Lemma 5.11. *For any input $\mathcal{I}$ for the* MSSC *problem there exists an (offline)* MTF*-based solution* MTFB *such that* $\text{MTFB}(\mathcal{I}) \leq 4 \cdot \text{OPT}(\mathcal{I})$

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

Clearly, $\text{OPT}(\mathcal{J}) = \text{OPT}(\mathcal{I})$. Furthermore, $\mathcal{J}$ is an instance of the List Update problem, for which we have shown in Section 1.6 that moving the requested element to the list front is a 4-approximation. Thus, $\text{MTFB}(\mathcal{J}) \leq 4 \cdot \text{OPT}(\mathcal{J})$. Finally, we observe that reordering actions of $\text{MTFB}(\mathcal{J})$ can be also applied to input $\mathcal{I}$. While the movement cost remains then the same, the access cost can be only smaller, i.e., $\text{MTFB}(\mathcal{I}) \leq \text{MTFB}(\mathcal{J})$. The lemma follows by combining the shown inequalities. □

5.4.2 Movement of OPT

In our analysis we will divide each step into two parts. In the first part, OPT pays only its access cost and DLMA makes its moves and pays both its access cost and movement cost. As OPT does not move in this part, its cost is the same as the cost

of an entire step in Static OPT setting and was calculated in Lemma 5.8. In the second part of the step, DLMA does nothing while OPT makes its moves and pays its movement cost. We want to upper bound the change of potential incurred by those moves with the movement cost of OPT.

Lemma 5.12. *Fix any step and consider its second part, where* DLMA *does nothing, whereas* OPT *moves elements and pays for their movement. Then* $\Delta\text{DLMA} + \Delta\Phi = O(r^2) \cdot \Delta\text{OPT}$

Proof. As we proved in Lemma 5.11, instead on focusing on OPT, we can focus on its MTF-based approximation that always moves elements directly to the front of the list. We focus on a single element z moved by OPT. Assume that OPT changes its position from $\text{POS}(z)$ to 0.

The definition of Φ_x is divided into three cases, depending on the relation between $p(x)$ and $p^*(x) + \kappa$. If that relation remains untouched by the movement, then Φ_x remains constant. Therefore let us consider cases when that relation changes.

Apart from element z, all elements affected by movement of z move one spot further from the front of the list, which means that their $p^*(x)$ can only increase by one. That yields three possible changes in value of Φ_x. We will consider them one by one.

- From $p(x) = p^*(x) + \kappa$ to $p(x) < p^*(x) + \kappa$.
$$\Delta\Phi_x = \alpha \cdot b(x) - \alpha \cdot b(x) - \xi\beta \cdot \text{POS}(x) \leq 0$$
- From $p(x) > p^*(x) + \kappa$ to $p(x) = p^*(x) + \kappa$.
$$\begin{aligned}\Delta\Phi_x &= \alpha \cdot b(x) + \xi\beta \cdot \text{POS}(x) - \beta \cdot \text{POS}(x) + \gamma \cdot b(x) - \xi\beta \cdot \text{POS}(x) \\ &= (\alpha + \gamma) \cdot b(x) - \beta \cdot \text{POS}(x) \\ &\leq (\alpha + \gamma - \beta) \cdot \text{POS}(x) \leq 0\end{aligned}$$
- From $p(x) > p^*(x) + \kappa$ to $p(x) < p^*(x) + \kappa$.
$$\begin{aligned}\Delta\Phi_x &= \alpha \cdot b(x) - \beta \cdot \text{POS}(x) + \gamma \cdot b(x) - \xi\beta \cdot \text{POS}(x) \\ &\leq (\alpha + \gamma) \cdot b(x) - \beta \cdot \text{POS}(x) \leq 0\end{aligned}$$

All the above inequalities are implied by Fact 5.2 and the fact that $\text{POS}(x) \leq b(x)$ during the move of OPT. Therefore, for all elements possibly except of z the change in potential is non-positive. We now focus only on z.

Let us consider element z that is transported by OPT to the front of its list. As $p^*(z)$ can only decrease, for the potential to change, the condition $p(z) \leq p^*(z) + \kappa$ had to be satisfied prior to the move. Therefore, $\text{POS}(z) \leq 2 \cdot 2^\kappa \cdot \text{POS}^*(z)$, which yields:

$$\begin{aligned}\Delta\Phi_z &\leq |\alpha \cdot b(z) - \beta \cdot \text{POS}(z) + \gamma \cdot b(z) - \xi\beta \cdot q(z)| \\ &\leq (2\alpha + \beta + 2\gamma + \xi\beta) \cdot \text{POS}(z) \\ &\leq (2\alpha + \beta + 2\gamma + \xi\beta) \cdot 2 \cdot 2^\kappa \cdot \text{POS}^*(z) \\ &= O(r^2) \cdot \Delta\text{OPT}.\end{aligned}$$

The last equality follows as the cost of OPT associated with moving z is the number of swaps needed to put it in the front, namely $\text{POS}^*(z)$. The lemma follows as $\Delta\text{DLMA} = 0$ in the second part of a step. □

Theorem 5.13. DLMA *is $O(r^2)$-competitive against dynamic adversary.*

Proof of the above theorem is the same as proof of Theorem 5.9, but instead of only Lemma 5.8 we sum over both Lemmas 5.8 and 5.12, therefore obtaining a competitive ratio of $O(r^2)$ instead of $O(r)$.

Theorem 5.14. *There exist a polynomial, deterministic $O(r^2)$-competitive algorithm for the* MSSC *problem in Dynamic OPT setting.*

5.5 Lower Bound

One of the major changes in algorithm DLMA (Algorithm 9) compared to LMA (Algorithm 7 of [2]) is that in line 4 we increase budgets of elements not by the position of x, $\text{POS}(x)$, but just by $1/r \cdot \text{POS}(x)$. One can wonder whether changing $1/r$ to yet another value can improve the competitive ratio again. We will prove that answer to this question is negative.

Definition 5.15. *We call an algorithm* DLMA*-like if it works the same as Algorithm 9, with line 4 changed to $b(y_i) \leftarrow b(y_i) + 1/c \cdot \text{POS}(x)$ for some integer $c > 0$.*

Lemma 5.16. *Every* DLMA*-like algorithm achieves competitive ratio $\Omega(r^2)$ in the Dynamic OPT setting.*

Proof. Let ALG be any DLMA-like online algorithm with a given value of c. Let $T = c + 1$, and we set the number of elements n as

$$n = r \cdot (T + r - 1). \tag{5.8}$$

This value is chosen so that n is divisible by r and satisfies

$$n \geq rc + r^2. \tag{5.9}$$

Both of those properties will be useful in the proof later.

An adversary creates a request sequence by always requesting the last r elements of the current permutation of ALG. We observe that during each of the first $T - 1$ requests, only the element on position $n - r + 1$ moves. Budgets of all elements on positions from $n-r+2$ to n increase, but remain too small to incur their movement. After the T-th request, all their budgets become uncontrolled, as

$$\begin{aligned} T \cdot \frac{1}{c} \cdot (n - r + 1) &= \left(1 + \frac{1}{c}\right) \cdot (n - r + 1) \\ &\geq n + \frac{n}{c} - r - \frac{r}{c} \\ &\geq n + r + \frac{r^2}{c} - r - \frac{r}{c} && \text{(by (5.9))} \\ &\geq n && \text{(as } r \geq 1\text{).} \end{aligned}$$

Therefore all these $r-1$ elements move. After T requests, the budgets of all elements are zero, and all the elements that occurred in any requests occupy the first $T+r-1$ spots on the list of ALG.

By the above reasoning, we can divide elements of the list into r disjoint sets $X_1, X_2, \ldots, X_r$, each of size $T + r - 1$. Each set corresponds to the T consecutive requests, after which elements of that set occupy the front of the list of ALG. We will call such sequence of T requests a **phase**.

As the Algorithm 9 does not specify the order in which it fetches the elements when multiple of them have uncontrolled budgets at the same time, we assume that it keeps their relative order. Therefore the i-th phase moves the last $r-1$ elements of X_i in front of the other ones and does not permute elements of X_i in any other way. That leads to the following observation.

Observation 5.17. *Let $x_1, x_2, \ldots, x_{T+r-1}$ be the elements from X_i before the start of* ALG*. Then for any phase j there exists an index $k \in \{1, ..., T + r - 1\}$ such that the elements of X_i are ordered as follows in the list of* ALG *after the j-th phase: $x_{k+1}, x_{k+2}, \ldots, x_{T+r-1}, x_1, x_2, \ldots, x_k$.*

By Observation 5.17, if we take every $(r-1)$-th element of X_i, for each phase in which the requests contain elements from X_i there is one element among the ones we chose that is in all of the requests. Therefore OPT can care only about those $\lfloor (T + r - 1)/(r - 1) \rfloor$ elements of X_i.

After r phases, elements from X_1 are again at the end of the list. Therefore by the above reasoning the optimal algorithm can choose $\lfloor (T+r-1)/(r-1) \rfloor$ elements from each X_i and bring these elements to the front of its list at the beginning of the sequence, keeping the rest of the elements in any order. This operation is performed only once, at the beginning, and therefore its cost is negligible with long enough sequences of requests. Then in any phase OPT can bring to the front of the list the element that occurs in all of the requests in that phase. It incurs a movement cost of at most $r \cdot \lfloor (T + r - 1)/(r - 1) \rfloor \leq 2T + r$, and then the access cost of all the requests of the phase is T. Therefore the total cost of m phases for OPT is at most $m \cdot (3T + r)$.

Algorithm ALG pays access cost $n - r + 1$ for every request and additionally moves $r-1$ of its last elements to the front at the end of the phase. Therefore its total cost for a single phase is equal to $T \cdot (n-r+1) + \sum_{i=n-r+2}^{n} i \geq T(n-r+1) + r(n-r+1)$. Summing it up, for the input $\mathcal{I}$ containing of m phases:

$$\begin{aligned} \frac{\text{ALG}(\mathcal{I})}{\text{OPT}(\mathcal{I})} &\geq \frac{m \cdot (T + r) \cdot (n - r + 1)}{m \cdot (3T + r)} \\ &\geq \min \left\{ \frac{(T + r) \cdot (n - r + 1)}{4T}, \frac{(T + r) \cdot (n - r + 1)}{4r} \right\} \end{aligned}$$

First element in the above minimum is equal at least $\Omega(n)$ which, by (5.9) is

$\Omega(r^2)$. For the second element:

$$\begin{aligned}
\frac{(T+r)\cdot(n-r+1)}{4r} &\geq \frac{(T+r)\cdot n}{8r} && \text{(by (5.9))}\\
&\geq \frac{(T+r)\cdot(T+r-1)}{8} && \text{(by (5.8))}\\
&= \Omega(r^2),
\end{aligned}$$

which shows that the competitive ratio of ALG is at least $\Omega(r^2)$. □

Chapter 6

Conclusions

This paper presents new results for two settings of Mssc. Our competitive ratio does not depend on the list size and is, as far as we know, the best proven so far.

Although our work presents known results for many settings of Online Mssc, there are still many open problems. The majority of lower bounds from Table 1.3 are implied by easier settings or even the offline version. Even though in the Online Learning setting and the deterministic setting of Static OPT the gap between lower and upper bounds is relatively small, there is still room for improvement.

In the Dynamic OPT setting there are still significant gaps between lower and upper bounds. The best known randomized algorithm is not much better than its deterministic version, and known inefficient exponential algorithms also do not produce better competitive ratios despite mentioned gap.

Our study of algorithms based on bringing some of the requested elements to the front suggest that any improvement of Dynamic OPT upper bound requires a fresh point of view — manipulating chunks, budgets and thresholds for movement do not seem to be useful.

Bibliography

[1] 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.

[2] Marcin Bieńkowski and Marcin Mucha. An improved algorithm for online reranking. *CoRR*, abs/2209.04870, 2022.

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

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

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

[6] Dimitris Fotakis, Loukas Kavouras, Grigorios Koumoutsos, Stratis Skoulakis, and Manolis Vardas. The online min-sum set cover problem. In *47th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 51:1–51:16, 2020.

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

[8] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

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

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