Multiplicative Approximation
Algorithms for Generalized Covering
and Packing Problems

Jonathan A. Wagner
Multiplicative Approximation Algorithms for Generalized Covering and Packing Problems

Research Thesis

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science

Jonathan A. Wagner

Submitted to the Senate of
the Technion — Israel Institute of Technology
Tammuz 5776 Haifa July 2016
The research thesis was done under the supervision of Prof. Elad Hazan in the Computer Science Department.

I would like to thank with all my heart my advisor Prof. Elad Hazan for giving me the opportunity of working under his supervision and allowing me to witness and to participate in academic research done by world class computer scientists. Thank you for the guidance, for the one-on-one sessions, for the fun, and for caring for my own academic success from the beginning.

In addition I want to thank all the people who supported me and helped me during the different stages of the studies: I want to thank Prof. Yuval Yishay for all the times he did his best to help me, I want to thank Prof. Erez Petrank for his help, Prof. Roi Reichart for the interesting talks during my studies, and Prof. Shmuel Zaks for his support, flexibility and care. I want to thank Prof. Nati Srebro for his in-depth consideration of my work and for all his remarks, and also I want to thank all the other Technion faculty that have helped me during my studies. I want to thank the administrative staff of the faculty of the Technion for their help. I want to thank the students from Prof. Hazan’s group who helped me and advised me - Oren Anava, Kfir Levi, Tomer Koren and Dan Garber. In addition I want to thank my Computer Science Officemates Yuri Meshman and Itay Feirewerker, and also Gil eizinger, Ido Cohen, and Rami Band. In addition I want to thank the Rabbies of the Technion - Rabbi Zinni and Rabbi Elad Dukov and the religious comunity of the Technion for all their support. I want to thank Jonathan Fromm for proofreading parts of this thesis. I want to thank Hadas Sofer for pushing me to finish this thesis. I want to thank my sister Noa and brother Itamar for being there and for their support. Finally I want to thank my Parents for being loving, caring and supporting parents.

The generous financial support of the Technion is gratefully acknowledged.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>1</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td>2</td>
</tr>
<tr>
<td>1.1 Preliminaries</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Previous Work</td>
<td>8</td>
</tr>
<tr>
<td>1.3 Statement of Results</td>
<td>9</td>
</tr>
<tr>
<td>1.3.1 Comparison with the 'Sublinear Perceptron' approach</td>
<td>9</td>
</tr>
<tr>
<td>1.3.2 Runtime comparison</td>
<td>10</td>
</tr>
<tr>
<td><strong>2 Basic Frameworks for multiplicative approximation</strong></td>
<td>12</td>
</tr>
<tr>
<td>2.1 Online Scaling for Generalized Covering and Generalized Packing Problems</td>
<td>12</td>
</tr>
<tr>
<td>2.1.1 Packing</td>
<td>12</td>
</tr>
<tr>
<td>2.1.2 Covering</td>
<td>14</td>
</tr>
<tr>
<td>2.2 Online Scaling for general Min-Max Problems</td>
<td>16</td>
</tr>
<tr>
<td>2.2.1 Packing Dual (or Covering Primal) Online Scaling</td>
<td>16</td>
</tr>
<tr>
<td>2.2.2 Packing Primal (or Covering Dual) Online Scaling</td>
<td>18</td>
</tr>
<tr>
<td>2.2.3 Primal - Dual Online Scaling</td>
<td>19</td>
</tr>
<tr>
<td>2.3 Domain Shrinkage for Packing Problems</td>
<td>20</td>
</tr>
<tr>
<td><strong>3 Speeding-up the Framework</strong></td>
<td>24</td>
</tr>
<tr>
<td>3.1 Recursive $l_1$-sampling</td>
<td>26</td>
</tr>
<tr>
<td>3.2 Amortized Complexity</td>
<td>29</td>
</tr>
<tr>
<td>3.3 Analysis of the Sped-up Framework</td>
<td>29</td>
</tr>
<tr>
<td><strong>4 Applications</strong></td>
<td>31</td>
</tr>
<tr>
<td>4.1 Normalized Covering SDP</td>
<td>31</td>
</tr>
<tr>
<td>4.2 Non-negative Linear Classifier</td>
<td>32</td>
</tr>
<tr>
<td><strong>5 Lower Bounds</strong></td>
<td>33</td>
</tr>
<tr>
<td>5.1 Lower Bounds for Zero Sum Games</td>
<td>33</td>
</tr>
<tr>
<td>5.2 Lower Bounds for Non-Negative Linear Classifier</td>
<td>34</td>
</tr>
</tbody>
</table>
## Appendix

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.0.1 Regret of the MW(_U) Algorithm</td>
<td>35</td>
</tr>
<tr>
<td>6.0.2 A More Detailed Analysis of the Recursive (\ell_1)-Sampling</td>
<td>38</td>
</tr>
<tr>
<td>6.0.3 (\ell_1)-Sampling and Amortised Complexity</td>
<td>41</td>
</tr>
<tr>
<td>6.0.4 Analysis of the Sped-up Framework for Covering</td>
<td>43</td>
</tr>
<tr>
<td>6.0.5 Auxillary Lemma for Multiplicative Approximation</td>
<td>46</td>
</tr>
</tbody>
</table>
List of Figures
Abstract

Approximation of min-max problems is a common task in convex optimization which has many important uses in machine learning and combinatorial optimization.

Approximations of problems can be divided into two main categories - additive approximations and multiplicative approximations. Additive approximations are usually relevant to general settings, but have runtime that in many cases depends significantly on the magnitude of natural parameters of the problem. Multiplicative approximations on the other hand, are natural for non-negative settings, and unlike the additive case, allow in many cases algorithms that are independent of the magnitude, or width, of the input parameters. This property is also known as width-free running time. Multiplicative approximation can also be useful if the optimum value of the problem is very small. In this case a multiplicative approximation of even 1/2, gives a very small additive approximation which may require much larger running time by additive approximation methods.

Recently, for the case of additive approximation, the use of sampling methods together with low-regret algorithms enabled the development of a general method for approximating an important class of min-max problems. This led to remarkably fast algorithms for several important problems in machine learning. This approach, however, did not address the task of multiplicative approximation.

In this work we present simple schemes based on low regret algorithms that give width-independent multiplicative approximation algorithms for two important classes of non-negative min-max problems - generalized covering and generalized packing. Our main contribution is a novel sampling and speed-up technique that in certain cases can be incorporated into the schemes and lead to very fast algorithms. As an application, we describe the first near-linear time, width-free multiplicative approximation algorithms for Normalized Covering Semi-definite Programming, and for Non-negative Linear Classifier.
Chapter 1

Introduction

The min-max problem of a non-negative convex-concave function is of finding $x \in \mathcal{X}$ and $y \in \mathcal{Y}$ that achieve:

$$\min_{x \in \mathcal{X}} \max_{y \in \mathcal{Y}} g(x, y) \quad (1.1)$$

where $g : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}_+$ is a convex-concave non-negative function, and $\mathcal{X} \subset \mathbb{R}^n$, $\mathcal{Y} \subset \mathbb{R}^m$ are convex sets. Two important subclasses of the above formulation are the generalized packing and generalized covering problems. These formulations include non-negative linear programming and positive semi-definite programming and have varied applications in machine learning and combinatorial optimization. Examples include the problems of K-Nearest Neighbor Classification [11], MAXCUT [8], Undirected Sparsest Cut [11], minimum-cost multicommodity flow [18], network embeddings [18], Held and Karp bound for the traveling salesman problem [18] and many more.

In general convex optimization approximations can be divided into two main categories - additive approximations and multiplicative approximations. Additive approximations are usually relevant to less restricted settings, and have runtime that may depend on the magnitude of the gradients or other natural parameters of the problem. Multiplicative approximations on the other hand, are natural for non-negative settings, and allow in many cases algorithms that are independent of the magnitude, or width, of the input parameters. This property is also known as width-free running time.

This latter width-free property can be extremely useful in two settings of interest. First, if the range of the input parameters is very large - many additive approximation methods will run in time polynomially (usually quadratically) proportional to this range. In contrast, width-free methods are invariant to the magnitude of the input parameters. Second - if the optimum value of the entire formulation is very small , a multiplicative approximation of $1/2$, say, gives a very small additive approximation which may require much larger running time by additive approximation methods.

Recently, [5] developed a general method for additively approximating an important class of min-max problems. The key idea behind the method was coupling of low-regret algorithms from online convex optimization and sampling techniques. This approach proved to be very effective as special cases and subsequent developments gave remarkably fast algorithms for several important
problems in machine learning such as training a linear classifier, SDP, and linear SVM. This approach, however, could not be directly applied to obtain width-free multiplicative approximations.

In this work we present general frameworks for multiplicative approximation algorithms for covering and packing problems based on variations of the multiplicative weights method.

Our main contribution is a novel sampling and speed-up technique that in certain cases can be coupled with the frameworks and lead to very fast algorithms. As an application, we describe the first near-linear time, width-free multiplicative approximation algorithms for Normalized Covering Semi-definite Programming, and for Non-negative Linear Classifier.

1.1 Preliminaries

Generalized Covering and Packing Problems

This work deals with several problems which belong to the general form of solving the min-max problem of a non-negative convex-conave function as follows:

$$\min_{x \in \mathcal{X}} \max_{y \in \mathcal{Y}} g(x, y)$$

(1.2)

where $g : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}_+$ is a convex-concave non-negative function, $\mathcal{X} \subset \mathbb{R}^n$ and $\mathcal{Y} \subset \mathbb{R}^m$.

For this problem we will focus on finding a multiplicative approximation: Given in addition an approximation parameter $\epsilon \in (0, 1)$,

Find $\bar{x} \in \mathcal{X}, \bar{y} \in \mathcal{Y}$

s.t. $(1 - \epsilon) \max_{y \in \mathcal{Y}} g(\bar{x}, y) \leq \lambda^* \leq \frac{1}{1 - \epsilon} \min_{x \in \mathcal{X}} g(x, \bar{y})$  

(1.3)

We note that in this work we always assume that $\min_{x \in \mathcal{X}} \max_{y \in \mathcal{Y}} g(x, y) = \max_{y \in \mathcal{Y}} \min_{x \in \mathcal{X}} g(x, y)$.

Two important special cases of the above problem are the generalized packing and generalized covering problems. The generalized packing problem, which is sometimes referred to as the min-max resource sharing problem (e.g. in [13]) is the following problem:

$$\min_{x \in \mathcal{K}} \max_{i \in [m]} f_i(x)$$

(1.4)

where $\mathcal{K} \subset \mathbb{R}^n$ is a convex set and $f_i$ is a non-negative convex function over $\mathcal{K}$ for all $i \in [m]$ (we denote $[m] := \{1, \ldots, m\}$).

and the corresponding approximation problem is:

Find $x \in \mathcal{K}$

s.t. $\max_{i \in [m]} f_i(x) \leq (1 + \epsilon) \min_{x' \in \mathcal{K}} \max_{i \in [m]} f_i(x')$  

(1.5)

where $\epsilon$ is a non-negative approximation parameter.
Let us denote by $f(x)$ the vector $(f_i(x))_{i=1}^m$, and for any $q \in \mathbb{R}^m$ denote $q^T f(x) = \sum_{i=1}^m q_i f_i(x)$. At first glance problem 1.4 might not seem a special case of the above non-negative min-max problem, but replacing $\max_{i \in [m]} f_i(x)$ with $\max_{p \in \Delta_m} p^T f(x)$ where $\Delta_m = \{ p \in \mathbb{R}^m | \sum_{i=1}^m p_i = 1, \forall i \in [m]: p_i \geq 0 \}$, we get an equivalent problem which is a special case of the generalised packing problem. The equivalent formulation is therefore:

$$\min_{x \in \mathcal{K}} \max_{p \in \Delta_m} p^T f(x)$$

and the approximation version:

Find $x \in \mathcal{K}$

$$\text{s.t. } \max_{p \in \Delta_m} p^T f(x) \leq (1 + \epsilon) \min_{x' \in \mathcal{K}} \max_{p \in \Delta_m} p^T f(x')$$

The generalised covering problem, which is sometimes referred to as the max-min resource sharing problem (e.g. in [13]) is the following problem:

$$\max_{x \in \mathcal{K}} \min_{i \in [m]} f_i(x)$$

where $\mathcal{K} \subset \mathbb{R}^n$ is a convex set and $f_i$ is a non-negative concave function over $\mathcal{K}$ for all $i \in [m]$.

The corresponding approximation problem is:

Find $x \in \mathcal{K}$

$$\text{s.t. } \min_{i \in [m]} f_i(x) \geq (1 - \epsilon) \max_{x' \in \mathcal{K}} \min_{i \in [m]} f_i(x')$$

where $\epsilon$ is a non-negative approximation parameter. Again by replacing $\min_{i \in [m]} f_i(x)$ with $\min_{p \in \Delta_m} p^T f(x)$ we get an equivalent problem which is a special case of the general non-negative min-max problem. The equivalent formulation is therefore:

$$\max_{x \in \mathcal{K}} \min_{p \in \Delta_m} p^T f(x)$$

and the approximation version:

Find $x \in \mathcal{K}$

$$\text{s.t. } \min_{p \in \Delta_m} p^T f(x) \geq (1 - \epsilon) \max_{x' \in \mathcal{K}} \min_{p \in \Delta_m} p^T f(x')$$

We note that in this work we may switch seamlessly between optimizing over the discrete set $[m]$ and optimizing over the convex set $\Delta_m$, when applicable, as done in this section. We also note that in this work we always assume that $\max_{x \in \mathcal{K}} \min_{p \in \Delta_m} p^T f(x) = \min_{p \in \Delta_m} \max_{x \in \mathcal{K}} p^T f(x)$ and may use this equality when needed. It is important to note though that it is not always possible to switch between optimizing over $[m]$ and optimizing over $\Delta_m$, for example $\min_{p \in \Delta_m} \max_{x \in \mathcal{K}} p^T f(x)$ is not necessarily equal to $\min_{i \in [m]} \max_{x \in \mathcal{K}} p^T f(x)$.
Some Other Notations

Some other notations we use are the following: for \( v \in \mathbb{R}^m \) and \( E \subseteq [m] \) we denote \( v_E^0 \) to be a vector in \( \mathbb{R}^m \) such that \( v_E^0(i) = v(i) \) for \( i \in E \), and \( v_E^0(i) = 0 \) for \( i \not\in E \). For a function \( f \) over some domain \( \mathcal{K} \) we will denote \( \arg\max_{x' \in \mathcal{K}} f(x') \) to be some \( x \in \mathcal{K} \), for which \( f(x) \geq (1-\epsilon) \max_{x' \in \mathcal{K}} f(x') \).

We will now present the special cases of generalised covering and generalised packing that we will consider.

Covering SDP Problems

For Semidefinite Programming (SDP) we will use the following notations. We will denote \( S^n_+ = \{ X \in \mathbb{R}^{n \times n} | X \succeq 0 \} \) and for two matrices \( A, B \in \mathbb{R}^{n \times n} \) we will denote \( A \bullet B := \text{Tr}(A^T B) \) where \( \text{Tr} \) stands for the trace of a matrix.

We will define a Normalized Covering SDP problem as follows:

\[
\min_{X \in S_+^n} \text{Tr}(X) \\
\text{s. t. } A_i \bullet X \geq 1 \quad \forall i = 1, ..., m
\]

(1.12)

where \( \forall i : 0_{n \times n} \neq A_i \succeq 0 \).

In [12] and [17] the algorithms proposed for general covering SDP are first converted to this formulation and then solved. Notice this formulation may be referred to in the literature as Normalized Positive SDP (as in [17]).

The following problem is a special case of the generalised covering problem which we will call Maxmin Positive SDP:

\[
\max_{X \in \mathcal{K}} \min_{i \in [m]} A_i \bullet X
\]

(1.13)

\[
\mathcal{K} = \{ X \in S_+^n | \text{Tr}(X) \leq 1 \}
\]

where \( A_i \) for \( i = 1, ..., m \) are positive semidefinite matrices in \( \mathbb{R}^{n \times n} \).

The Normalized Covering SDP can be reduced to the Maxmin Positive SDP as stated in the following claim:

Claim 1.1.1. Denote \( \lambda^*_NC \) the optimum of the Normalized Covering SDP and \( \lambda^*_MP \) the optimum of the Maxmin Positive SDP. If \( \lambda^*_MP = 0 \) then the Normalized Covering SDP has no feasible solution (\( \lambda^*_NC = \infty \)), otherwise \( \lambda^*_NC = \frac{1}{\lambda^*_MP} \). In the case \( \lambda^*_MP > 0 \), if \( X_{MP} \in \mathcal{K} \) is a \( 1 - \epsilon \)-multiplicative approximation to the Maxmin Positive SDP, that is \( \min_{i \in [m]} A_i \bullet X_{MP} = \alpha \geq (1-\epsilon) \lambda^*_MP > 0 \), then \( \frac{1}{\alpha} X_{MP} \) is a \( \frac{1}{1-\epsilon} \)-multiplicative approximation to the Normalized Covering SDP problem.
Non-negative linear classifier

In this problem we are given points in the positive orthant, \( \{a_i\}_{i=1}^m \) and we wish to find a hyperplane that passes through the origin which maximizes the minimal distance from the points. Formally, we have the problem:

\[
\max_{X \in \mathbb{B}_n} \min_{i \in [m]} a_i^T x
\]  

where \( a_i \in \mathbb{R}_n^+ \) for \( i = 1, ..., m \).

It will be convenient for us to think of \( a_i^T \) as rows of a matrix \( A \) and formulate our problem in the following way:

\[
\max_{X \in \mathbb{B}_n} \min_{p \in \Delta_m} p^T Ax
\]

where \( A \in \mathbb{R}_+^{m \times n} \) for \( i = 1, ..., m \).

We finish this section by presenting auxiliary techniques and lemmas that we will use in this work.

The Multiplicative Weights Update Method

We now present the Multiplicative Weights algorithm, a review of this method could be found in [3]. Notice that this algorithm is an Online Convex Optimization algorithm, which also means that it does not obtain all its input at once but rather iteratively receives input and performs calculations.

The usage of this algorithm (and its variations) as a building block in subsequent algorithms in this work is a bit uncommon. Every time the algorithm is being called, it is actually performing another iteration with the new input. To understand the way algorithm [4] and its variations, are used by other algorithms it may be helpful to think of it as an object with internal constants (e.g. in Alg. [4] the numbers \( \eta, m \)) and internal variables (e.g. the vectors \( p \) and \( w \)), which has an initialization method, and a "step" method. When called for the first time (without any input) the "step" method just outputs variables (e.g. \( p \)) obtained by the initialization (e.g. line 2). Following this, every time the "step" method is being called with an input \( v \in [-1, 1]^m \), it updates its internal variables (e.g. lines 5 and 6) and outputs some variable (e.g. \( p \)). For the case of Alg. [4] the internal variable’s \( p \) value when being outputted for the \( t \)-th time could be thought of \( p_t \). Notice that the \( t \)-th output is generated before observing the \( t \)-th input \( v_t \).
Algorithm 1 Multiplicative Weights Algorithm (with costs vectors)

1: Parameters: $m$ - number of experts, $\eta < \frac{1}{2}$
2: Initialization ($t = 0$): $w_1 \leftarrow 1$, $p_1 \leftarrow \frac{1}{mm}$.
3: for $t = 1, 2, ..., T$ do
4: Obtain costs vector $v_t \in [-1, 1]^m$
5: $\forall i \in [m]: w_{t+1}(i) \leftarrow w_t(i)(1 - \eta v_t(i))$
6: $p_{t+1} \leftarrow \frac{w_t}{\|w_t\|_1}$
7: end for

For this algorithm the following regret bound can be shown:

**Lemma 1.1.1.** (Corollary 2.2 in [3]) For any $p \in \Delta_m$ it holds that

$$\sum_{t=1}^{T} p_t^Tv_t \leq \sum_{t=1}^{T} p_t^T(v_t + \eta |v_t|) + \frac{\ln m}{\eta}$$

where $|v_t|$ is the vector obtained by taking the coordinate-wise absolute value of $v_t$

From the above lemma we can directly obtain the following:

**Lemma 1.1.2.** Assume that all $v_t$ are non-negative cost vectors. Then

$$\sum_{t=1}^{T} p_t^Tv_t \leq (1 + \eta) \min_{p \in \Delta_m} \sum_{t=1}^{T} p_t^Tv_t + \frac{\ln m}{\eta}$$

In case the vectors $v_t$ represent gains which we wish to maximise, we can run algorithm 1 with cost vectors $-v_t$. For this version of the algorithm, which we will refer to as the Multiplicative Weights algorithm with gains vectors, we get the following regret bound:

**Lemma 1.1.3.** (Corollary 2.6 in [3]) For any $p \in \Delta_m$ it holds that

$$\sum_{t=1}^{T} p_t^Tv_t \geq \sum_{t=1}^{T} p_t^T(v_t - \eta |v_t|) - \frac{\ln m}{\eta}$$

where $|v_t|$ is the vector obtained by taking the coordinate-wise absolute value of $v_t$

From the above lemma we can directly obtain the following:

**Lemma 1.1.4.** Assume that all $v_t$ are non-negative gain vectors. Then

$$(1 - \eta) \max_{p \in \Delta_m} \sum_{t=1}^{T} p_t^Tv_t \leq \sum_{t=1}^{T} p_t^Tv_t + \frac{\ln m}{\eta}$$
Combining Multiplicative Approximations

The following lemma is an algebraic lemma which is useful in combining multiplicative approximations:

**Lemma 1.1.5.** Assume that:

\[ \forall i \in [n] : (1 - \epsilon_i)A_{i-1} \leq (1 + \eta_i)A_i + \alpha_i \]

Where \( \forall i \in [n] : \alpha_i \geq 0, \epsilon_i \in [0, 1), \eta_i \in [0, 1), \epsilon_i + \eta_i < 1 \) and \( \forall i \in [n] \cup \{0\} : A_i > 0. \)

Then for any \( k \in [n] \cup \{0\} : \)

\[ \frac{A_n}{A_0} \geq 1 - \left( \sum_{i=1}^{n} \epsilon_i + \sum_{i=1}^{n} \eta_i + \frac{\sum_{i=1}^{n} \alpha_i}{A_k} \right) \]

And if also \( \sum_{i=1}^{n} \epsilon_i + \sum_{i=1}^{n} \eta_i + \frac{\sum_{i=1}^{n} \alpha_i}{A_k} < \frac{1}{2} \) then:

\[ \frac{A_0}{A_n} \leq 1 + 2 \left( \sum_{i=1}^{n} \epsilon_i + \sum_{i=1}^{n} \eta_i + \frac{\sum_{i=1}^{n} \alpha_i}{A_k} \right) \]

### 1.2 Previous Work

Our work takes inspiration from the additive approximation algorithms that use speed-up techniques borrowed from the realm of machine learning, prominently as in [5].

The problems of generalized covering and generalized packing are sometimes referred to as Max-Min Resource Sharing and Min-Max Resource Sharing (see [13]). Frameworks for multiplicative approximation for these problems were suggested in the past (For example [9], [14], [16], [10]), and usually gave bounds on the number of iterations of \( \tilde{O} \left( \frac{m}{\epsilon^2} \right) \) which is similar to the bounds in this work (see section 2.1).

The frameworks and templates we present are based either on a dual method, a primal method or a primal-dual method. In order to obtain the multiplicative approximation the frameworks use techniques similar to those in [15] and [7]. The speed-up methods we use are novel in this context (to the best of our knowledge).

Normalized Covering SDP is an important special case of general Covering SDP. In many cases the general covering SDP is first reduced to the normalised case and only then solved (as in [17], [12]). In the past few years several approaches were developed for addressing the problem of general Covering SDPs which applied also to the normalised case (see [2], [4], [17], [12], [11], [6]). For the normalised case (and for the general case) these approaches had at least one of the following shortcomings. They either gave additive approximation (as in [6]), had width-dependent running
time, or had bad dependence on the input size if no further assumptions on the input were made (as in [17], [11]). In some cases algorithms with the latter shortcoming were beneficial for particular cases in which, for example, factorisation of the matrices was known in advance, or some other assumption helped in reducing the runtime. However, they did not give a near-linear time for the normalized case in which no assumptions are made regarding the matrices, whereas we do.

1.3 Statement of Results

For the approximation problems of generalised packing (1.5) and generalised covering (1.9) we present general frameworks for which we propose a speed-up technique named recursive $\ell_1$-sampling. The basic and sped-up frameworks run at most $O \left( \frac{m \log m}{\epsilon^2} \right)$ iterations, and with high probability return a multiplicative approximation to the optimal solution. The speed-up technique can be applied to certain cases, and we show that for the case where $f_i$ are explicit linear functions over a general convex set $K$, the sped-up frameworks can be implemented to run in $O \left( \frac{mn \log^2 m}{\epsilon^2} \right)$ operations plus $O \left( \frac{m \log m}{\epsilon^2} \right)$ multiplicative approximate optimizations of a linear function over $K$. As a special case we obtain a first ever $(1 - O(\epsilon))$-multiplicative approximation algorithm with the running time of $\tilde{O} \left( \frac{mn^2}{\epsilon^2} \right)$ for the Normalized Covering SDP problem (problem 1.12). Previous running times for width-free multiplicative approximation of Normalized Covering SDP were $\tilde{O} \left( \frac{4}{\epsilon^2} \right)$ by [17] under the assumption that all $m$ matrices were given in factorized form, and $q$ is total number of non-zeros in the factorized form and in general can be as large as $O(mn^2)$. We mention that factorizing all matrices may cost $O(mn^3)$ which is much higher than our running time. We will also note that another work - [12], gave a parallel algorithm using poly($N$) processors with $\tilde{O} \left( \frac{1}{\epsilon^2} \right)$ iterations, where each iterations uses polylog parallel time where $N$ is the size of the SDP program (which could be $O(mn^2)$).

Another special case we consider is of the problem of non-negative linear classifier (problem 1.15) for which we develop a $(1 - O(\epsilon))$-multiplicative approximation algorithm with running time of $\tilde{O} \left( \frac{mn}{\epsilon^2} \right)$.

Finally, we give lower bounds of linear running time for some special cases of generalised covering and generalised packing. We give a lower bound of $\Omega(mn)$ for multiplicative approximation of covering and packing Zero-Sum Games, and for multiplicative approximation of non-negative linear classifier. In both cases this matches up to logarithmic factors the upper bounds, which in the latter case is obtained by the algorithm presented in this work.

We would also like to point out one smaller contribution of this work, which is proposing general templates for obtaining a multiplicative approximation for the general non-negative min-max problem (as formulated in 1.3). These generalise the basic frameworks of Online Scaling for the generalised covering and packing problems mentioned above.

1.3.1 Comparison with the ‘Sublinear Perceptron’ approach

This work was motivated and inspired by the work of Clarkson et al. ([5]) and subsequent work (e.g. [6]) which proposed additive approximation algorithms with sublinear running times for
some important max-min problems. In this subsection we compare between the approach used in [5] and the approach used in this work. The fundamental difference between this work and [5] which allows width-free algorithms is this work’s usage of ”online scaling”. A second technique which is important for obtaining multiplicative approximation for the covering case which is used in our work is the technique of ”expert dismissal”. We were inspired to use both these techniques by the work of [15] which refers to them as ”non-uniform increments” and ”deletion of covering constraints once they get satisfied”. [15] themselves used these ideas based on works by Garg and Koenemann ( e.g. in [7])

A third technique used in this work which is important for obtaining the speed-up of the running time is the Recursive $\ell_1$-Sampling technique. While sampling techniques are used in the approach described in [5] as well, the usage of recursive $\ell_1$-Sampling technique is novel in the context of min-max approximation algorithms using Online Approximation algorithms (as far as we know, of course it is very probable that similar binary-recursive techniques for sampling were used in many cases in other contexts before).

Another substantial difference between our work and the papers [5] and [6] is that the best performing algorithms (e.g. Alg. 1 (SublinearSDP) in [6], and Alg. 1. in [5]) are primal-dual, in which speed up is achieved by two sampling steps which take place each iteration (usually one for estimating the dual variable and one for estimating the primal variable). In contrast to this, in this work, the best running time algorithms use only a dual algorithm and use only one sampling step each iteration (for estimating the dual variable). Additional speed up in this work is obtained by the usage of the Recursive $\ell_1$ Sampling, and the fact that it allows small changes in the dual variable between iterations. Generally speaking primal dual frameworks allow algorithms with $\tilde{O}(1/\varepsilon^2)$ iterations, while the two sampling steps enable a reduction of the running time of each iteration from $O(mn)$ to $O(m + n)$. In our work, we use a dual algorithm which forces us to have $\tilde{O}(m/\varepsilon^2)$ iterations. However the recursive $\ell_1$ sampling technique helps us to reduce the running time of each iteration to $O(n \log m)$.

### 1.3.2 Runtime comparison

This subsection presents a partial comparison of the running times presented in this work and running times achieved or derived by previous work.

Some of the previous work considered in the table proposes only additive approximation algorithms. In order to use the running times of additive approximation algorithms to get the running time for multiplicative approximation, the additive approximation parameter $\varepsilon_{\text{additive}}$ is replaced with $\frac{\lambda^*}{G}\varepsilon_{\text{multiplicative}}$. Here, $\lambda^*$ is the value of the optimum and $G$ is the bound on he norm of the input - i.e. the width of the problem (for example in the work of [6] this is the maximal Frobenius norm of a matrix in the SDP instance). The explanation for this is that the additive approximation algorithms referred to in this section assume that the width of the input is 1. In order to obtain a multiplicative approximation the input is first normalized, and then the additive approximation algorithm is used with a normalized approximation parameter.

In this comparison we also bring results for explicitly given packing and covering linear programs which are very important special cases of the general formulations 1.5 and 1.9 respectively in which
the domain $\mathcal{K}$ is the simplex: $\Delta_n$. This work does not improve running times for these problems and does not explicitly address these problems but running time for these problems can be derived from theorems 3.0.2 and 3.0.3 using the fact that minimizing or maximizing the inner product of $x \in \Delta_n$ with some given vector could be done in $O(n)$.

Notes: The result by [17] assumes that all $m$ matrices were given in factorized form, and $q$ is total number of non-zeros in all matrices in the factorized form. In general this can be as large as $O(mn^2)$. Factorizing all matrices may cost $O(mn^3)$ unlike this work that does not assume any assumption about the form of the matrices. In the result by [6] $G$ is a bound on the Frobenius norm of the matrices. In the result by [5] $G$ is a bound on the $\ell_2$ norm of the matrix rows. In the results by [15] and [1], $N$ is the number of non-zero elements in the input matrix.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Running Time</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normalized Covering SDP 1.12</td>
<td>$\tilde{O}\left(\frac{mn^2}{\epsilon^2}\right)$</td>
<td>This work</td>
</tr>
<tr>
<td>Normalized Covering SDP 1.12</td>
<td>$\tilde{O}\left(\frac{q}{\epsilon^2}\right)$</td>
<td>[17]</td>
</tr>
<tr>
<td>Normalized Covering SDP 1.12</td>
<td>$\tilde{O}\left(\left(\frac{G}{\lambda}\right)^2 \frac{m}{\epsilon^2} + \left(\frac{G}{\lambda}\right)^5 \frac{n^2}{\epsilon^5}\right)$</td>
<td>[6] Using Alg. 1</td>
</tr>
<tr>
<td>Non-negative Linear Classifier 1.15</td>
<td>$\tilde{O}\left(\frac{m}{\epsilon^2}\right)$</td>
<td>This work</td>
</tr>
<tr>
<td>Non-negative Linear Classifier 1.15</td>
<td>$\tilde{O}\left(\left(\frac{G}{\lambda}\right)^2 \frac{1}{\epsilon^2} (m+n)\right)$</td>
<td>[5] Using Alg. 1</td>
</tr>
<tr>
<td>Explicitly Given Packing/Covering LP</td>
<td>$\tilde{O}\left(\frac{mn}{\epsilon^2}\right)$</td>
<td>Derived from this work</td>
</tr>
<tr>
<td>Explicitly Given Packing/Covering LP</td>
<td>$O\left(N + \frac{(n+m)\log(N)}{\epsilon^2}\right)$</td>
<td>[15]</td>
</tr>
<tr>
<td>Explicitly Given Packing/Covering LP</td>
<td>$\tilde{O}\left(\frac{N}{\epsilon^2}\right)$ for packing, $\tilde{O}\left(\frac{N}{\epsilon^{1.5}}\right)$ for covering</td>
<td>[1]</td>
</tr>
</tbody>
</table>
Chapter 2

Basic Frameworks for multiplicative approximation

In this chapter we describe basic approaches for achieving multiplicative approximation for generalised covering and packing. The main approach we consider is Online Scaling for which we develop frameworks for both generalised packing and generalised covering. We then generalise the Online Scaling approach for finding a multiplicative approximation to a saddle-point of a general non-negative convex-concave function. Finally, we present another approach which we call Domain Shrinkage which is applicable in the case of generalised packing. This approach has similar runtime (with a slightly inferior dependence on $\frac{1}{\epsilon}$). We bring it here for its elegance and simplicity. This approach demonstrates further principles in the context of multiplicative approximation that may be of interest, and the speed-up techniques in the next chapter can be applied to this framework as well.

2.1 Online Scaling for Generalized Covering and Generalized Packing Problems

2.1.1 Packing

We consider the following framework for multiplicatively approximating covering constrained optimisation.
Algorithm 2 Basic Online Scaling Framework for Packing

1: Input: \( m \) non-negative convex functions \( f_i \), over a convex domain \( K \). Approximation parameter \( \epsilon \in [0, 1) \).
2: \( T \leftarrow \lceil \frac{m \log m}{\epsilon^2} \rceil \), \( \eta_{MW} \leftarrow \epsilon \)
3: for \( t = 0 \ldots T - 1 \) do
4: \( p_{t+1} \leftarrow MW\text{-step}\_\text{gain}\_t(w_t \cdot f(x_t)) \)
5: \( x_{t+1} \leftarrow \arg \min_{x \in K} \frac{1}{1+\epsilon} p_{t+1}^T f(x) \)
6: \( w_{t+1} \leftarrow \frac{1}{\|f(x_{t+1})\|_\infty} \)
7: end for
8: Return \( \bar{x} = \sum_{t} \frac{w_t x_t}{\sum_t w_t} \).

The following theorem states the correctness of the framework and gives a bound on the number of iterations.

**Theorem 2.1.1.** The Basic Online Scaling Framework for Packing returns a \( 1+O(\epsilon) \)-multiplicative approximate solution in \( O\left(\frac{m \log m}{\epsilon^2} \right) \) iterations.

**Proof.** Assume \( \epsilon \leq \frac{1}{6} \). First, from the entry-wise convexity of \( f \) we know that \( \sum_{t=1}^{T} w_t f(x_t) \geq \left( \sum_{t=1}^{T} w_t \right) f(\bar{x}) \), and so:

\[
\left( \sum_{t=1}^{T} w_t \right) \max_{p \in \Delta_m} p^T f(\bar{x}) \leq \max_{p \in \Delta_m} \sum_{t=1}^{T} w_t p^T f(x_t) \tag{2.1}
\]

Second from the regret of the Multiplicative Weights algorithm (lemma [1.1.4]) with \( \eta = \epsilon \) we get that:

\[
(1 - \epsilon) \max_{p \in \Delta_m} \sum_{t=1}^{T} w_t p^T f(x_t) \leq \sum_{t=1}^{T} w_t p_t^T f(x_t) + \frac{\log m}{\epsilon} \tag{2.2}
\]

Third, from the choice of \( x_t \) each iteration we get that \( \forall t \in [T] : p_t^T f(x_t) \leq (1+\epsilon) \min_{x \in K} p_t^T f(x) \leq (1 + \epsilon) \min_{x \in K} \max_{p \in \Delta_m} p^T f(x) = (1 + \epsilon) \lambda^* \), and so,

\[
\sum_{t=1}^{T} w_t p_t^T f(x_t) \leq (1 + \epsilon) \left( \sum_{t=1}^{T} w_t \right) \lambda^* \tag{2.3}
\]

Last, from the choice of \( w_t \) we get a lower bound on \( \max_{p \in \Delta_m} \sum_{t=1}^{T} w_t p^T f(x_t) \) (A more detailed proof appears after the proof of the theorem):

**Claim 2.1.1.** \( \max_{p \in \Delta_m} \sum_{t=1}^{T} w_t p^T f(x_t) \geq \frac{T}{m} \)
Combining the inequalities 2.1, 2.2 and 2.3 according to the auxiliary algebraic lemma 1.1.5 gives us:

\[
\max_{p \in \Delta_m} p^T f(\bar{x}) = \left( \frac{\sum_{t=1}^{T} w_t}{\sum_{t=1}^{T} w_t} \right) \max_{p \in \Delta_m} p^T f(\bar{x}) \lambda^* \\
\leq 1 + 2\left( 2\epsilon + \frac{\log m}{\epsilon \max_{p \in \Delta_m} \sum_{t=1}^{T} w_t p^T f(x_t)} \right) \\
\leq 1 + 2\left( 2\epsilon + \frac{m \log m}{\epsilon T} \right) \\
\leq 1 + 6\epsilon
\]

Combining 2.1, 2.2 and 2.3 according to lemma 1.1.5

Claim 2.1.1

Choose $T$.

Notice we assumed that $3\epsilon \leq \frac{1}{2}$ (Due to the condition of the auxiliary lemma 1.1.5), but got a $1 + 6\epsilon$ multiplicative approximation, and so for any $0 < \epsilon' < 1$ we can obtain a $1 + \epsilon'$ multiplicative approximation with the same asymptotic running time, using $\epsilon = \epsilon' \frac{\epsilon}{6}$.

Proof of claim 2.1.1

Proof. From the non-negativity of $f$ and the definition of $w_t = \frac{1}{\|f(x_t)\|_\infty}$ we get that for all $t$, $w_t \sum_{i=1}^{m} f(x_t)(i) = w_t \cdot \|f(x_t)\|_1 \geq w_t \cdot \|f(x_t)\|_\infty = 1$. Summing on all $t$, and using non-negativity of $f$ we get: $T \leq \sum_{t=1}^{T} \sum_{i=1}^{m} w_t f(x_t)(i) = \sum_{i=1}^{m} \sum_{t=1}^{T} w_t f(x_t)(i) = \| \sum_{t=1}^{T} w_t f(x_t) \|_1$, and so $\max_{p \in \Delta_m} \sum_{t=1}^{T} w_t p^T f(x_t) = \max_{p \in \Delta_m} p^T \left( \sum_{t=1}^{T} w_t f(x_t) \right) \geq \frac{T}{m}$.

2.1.2 Covering

We consider the following framework for multiplicatively approximating covering constrained optimisation.
Algorithm 3 Basic Online Scaling Framework for Covering

1: Input: $m$ non-negative concave functions $f_i$, over domain $K$. Approximation parameter $\epsilon$.
2: \( K \leftarrow \frac{9 \log m}{\epsilon^2}, t \leftarrow 0, \text{MW}_U \text{ parameters: } m, U \leftarrow 2K, \eta \leftarrow \epsilon \)
3: while \( \min_{i} \sum_{s=1}^{t} \omega_{s} f(x_s)(i) < K \) do
4: \( p_{t+1}, E_{t+1} \leftarrow \text{MW}_U\text{-step}(v_t) \)
5: \( x_{t+1} \leftarrow \arg \max_{x \in K} \frac{1-\epsilon}{x \in K} P_{t+1}^T f(x) \)
6: \( \omega_{t+1} \leftarrow \frac{1}{\|f(x_{t+1})\|_{E_{t+1}}^{0}} \)
7: \( v_{t+1} \leftarrow \omega_{t+1} \cdot f(x_{t+1}) \)
8: \( t \leftarrow t + 1 \)
9: end while
10: Return \( \bar{x} = \sum_{s=1}^{t} \omega_{s} x_s \sum_{s=1}^{t} \omega_{s} \).

Notice that in contrast with the generalized packing case, in order to formulate a framework for generalised covering we use a modified version of the Multiplicative Weights (MW) algorithm which we denote \( \text{MW}_U \).

The original MW algorithm could be thought of as an iterative game in which at each round we choose a distribution $p_t$ over a set of $m$ experts after which the cost of each expert for the round is revealed. The cost we pay in the round is the weighted average of the experts’ costs according to the distribution we chose (in this formulation costs can also be negative). The MW algorithm is a strategy for minimizing regret in this scenario, and has a proven regret bound guarantee.

In the modified version, when the cumulative cost of some expert has reached the bound $U$ then the expert is ”dismissed” and from now on it will be given weight 0. Notice that every round the MW$_U$ algorithm calculates both $p_t$ and $E_t$, where $p_t$ is the distribution at round $t$ (as before) and $E_t \subset [m]$ is the indices of the experts that are still ”in the game” at the beginning of the $t$-th round. Despite the modification the MW$_U$ algorithm can still have the same regret bound guarantee as the original MW algorithm in the case relevant to our framework in which the costs are always non-negative. This is stated in the following lemma.

Algorithm 4 MW$_U$

1: Parameters: $m$ - number of experts, $U > 0$, $\eta < \frac{1}{2}$
2: Initialization ($t = 0$): $V_0 \leftarrow 0_m, E_1 \leftarrow [m], w_1 \leftarrow 1_m, p_1 \leftarrow \frac{1}{m}$.
3: for $t = 1, 2, \ldots, T$ do
4: Obtain non-negative costs vector $v_t \in [0, 1]^m$
5: \( V_t \leftarrow V_{t-1} + v_t \)
6: \( E_{t+1} \leftarrow \{i \in [m] : V_t(i) < U\} \)
7: \( \forall i \in [m] : w_{t+1}(i) \leftarrow w_t(i)(1 - \eta v_t(i)) \)
8: \( p_{t+1} \leftarrow \frac{w_{t+1}^{0}_{E_{t+1}}}{\|w_{t+1}^{0}_{E_{t+1}}\|_1} \)
9: end for
Lemma 2.1.1. Assume that \( E_{T+1} \neq \emptyset \), then the following holds: 
\[
\sum_{t=1}^{T} p_t T v_t \leq (1+\eta) \min_{x \in \Delta_m} \sum_{t=1}^{T} p^T v_t + \frac{\log m}{\eta} \text{ (the regret bound is exactly the same as in the standard MW algorithm).}
\]

The proof of the above lemma could be found in the appendix.

The following theorem guarantees correctness of the general framework and gives a bound on the number of iterations. Its proof follows the same lines as the proof for the fast version (theorem 3.0.3).

Theorem 2.1.2. The Online Scaling Algorithm for Generalized Covering returns a \( 1 - O(\epsilon) \)-multiplicative approximate solution in \( O\left(\frac{m \log m}{\epsilon^2}\right) \) iterations.

### 2.2 Online Scaling for general Min-Max Problems

In this section we present templates for multiplicative approximation of general non-negative min-max problems.

Before presenting the templates we mention that in this section we will refer to the regret of OCO algorithms in a slightly modified form, which will be easier to deal with in the context of multiplicative approximations.

We will say that \( R(T), \epsilon_{\text{regret}} \) are the regret parameters of an OCO Online-Alg\_costs if:

\[
(1 - \epsilon_{\text{regret}}) \sum_{t=1}^{T} f_t(x_t) \leq \min_{x \in X} \sum_{t=1}^{T} f_t(x) + R(T)
\]

We will say that \( R(T), \epsilon_{\text{regret}} \) are the regret parameters of an OCO Online-Alg\_gains if:

\[
(1 - \epsilon_{\text{regret}}) \max_{x \in X} \sum_{t=1}^{T} f_t(x) \leq \sum_{t=1}^{T} f_t(x_t) + R(T)
\]

Notice that for all previous OCO bounds we can just choose \( R(T) \) to be the original bound and \( \epsilon_{\text{regret}} \) to be zero. On the other hand, for the MW algorithm we can choose \( \epsilon_{\text{regret}} = \eta \) and \( R(T) = \frac{\log n}{\eta} \).

#### 2.2.1 Packing Dual (or Covering Primal) Online Scaling

We will begin by introducing the general Packing Dual Online Scaling template.
Algorithm 5 Packing Dual Online Scaling

1: Input: non-negative convex-concave function \( g : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}_+ \), where \( \mathcal{X} \subseteq \mathbb{R}^n \) and \( \mathcal{Y} \subseteq \mathbb{R}^m \) are convex. Approximation parameter \( \epsilon \in [0, 1) \).

2: for \( t = 0...T - 1 \) do
3: Let \( y_{t+1} \leftarrow \text{Online-Alg}_{gains}(w_t \cdot g(x_t, y)) \)
4: Let \( x_{t+1} \leftarrow \arg \min_{x \in \mathcal{X}} g(x, y_{t+1}) \)
5: Set \( w_{t+1} > 0 \)
6: end for
7: Return \( \bar{x} = \frac{\sum_t w_t x_t}{\sum_t w_t} \) and \( \bar{y} = \frac{\sum_t w_t y_t}{\sum_t w_t} \)

Note: Notice that \( w_t \) must be chosen so that \( w_t \cdot g(x_t, y) \) is a legal input for the Online-Alg\(_{gains}\).

We now present a theorem which can be used to guarantee correctness and number of iterations.

Theorem 2.2.1. Let \( R(T) \) and \( \epsilon_{\text{regret}} \in [0, 1) \) be the regret parameters of Online-Alg\(_{gains}\). Then

\[
\min_{x \in \mathcal{X}} g(x, \bar{y}) \geq \max_{y \in \mathcal{Y}} g(\bar{x}, y) \left( 1 - \epsilon - \epsilon_{\text{regret}} - \frac{R(T)}{\max_{y \in \mathcal{Y}} \sum_{t=1}^T w_t \cdot g(x_t, y)} \right)
\]

Notice that in order to guarantee a multiplicative approximation we need to complete the template such that \( \epsilon_{\text{regret}} + \frac{R(T)}{\max_{y \in \mathcal{Y}} \sum_{t=1}^T w_t \cdot g(x_t, y)} \) is smaller than \( O(\epsilon) \). This will also typically imply the number of iterations.

Proof. First, for any \( y' \in \mathcal{Y} \) the function \( g(x, y') \) is convex in \( x \) and so \( \sum_{t=1}^T w_t g(x_t, y') \geq \left( \sum_{t=1}^T w_t \right) g \left( \frac{\sum_{t=1}^T w_t x_t}{\sum_{t=1}^T w_t}, y' \right) = \left( \sum_{t=1}^T w_t \right) g(\bar{x}, y') \). Since this holds for all \( y' \) we get:

\[
\left( \sum_{t=1}^T w_t \right) \max_{y \in \mathcal{Y}} g(\bar{x}, y) \leq \max_{y \in \mathcal{Y}} \sum_{t=1}^T w_t g(x_t, y) \quad (2.4)
\]

Second from the regret of the Online-Alg\(_{gains}\) algorithm we get that:

\[
(1 - \epsilon_{\text{regret}}) \sum_{t=1}^T w_t g(x_t, y) \leq \sum_{t=1}^T w_t g(x_t, y_t) + R(T) \quad (2.5)
\]

Third, from the choice of \( x_t \) each iteration we get that \( \forall t \in [T] : g(x_t, y_t) \leq (1+\epsilon) \min_{x \in \mathcal{X}} g(x, y_t) \), and so,

\[
\sum_{t=1}^T w_t g(x_t, y_t) \leq (1 + \epsilon) \sum_{t=1}^T w_t \min_{x \in \mathcal{X}} g(x, y_t) \quad (2.6)
\]
and from basic algebra laws and positivity of \( w_t \):

\[
\sum_{t=1}^{T} w_t \min_{x \in \mathcal{X}} g(x, y_t) \leq \min_{x \in \mathcal{X}} \sum_{t=1}^{T} w_t g(x, y_t) \tag{2.7}
\]

Last, since for any \( x' \in \mathcal{X} \) the function \( g(x', y) \) is concave in \( y \), we get in a similar manner to the way we obtained 2.4 that:

\[
\min_{x \in \mathcal{X}} \sum_{t=1}^{T} w_t g(x, y_t) \leq \left( \sum_{t=1}^{T} w_t \right) \min_{x \in \mathcal{X}} \left( 1 - \epsilon \right) g(x, \bar{y}) \tag{2.8}
\]

Combining all inequalities (2.4, 2.5, 2.6, 2.7, 2.8) according to the auxiliary algebraic lemma gives us:

\[
\min_{x \in \mathcal{X}} g(x, \bar{y}) \geq \max_{y \in \mathcal{Y}} g(x, \bar{y}) \left( 1 - \epsilon - \epsilon_{\text{regret}} - \frac{R(T)}{\max_{y \in \mathcal{Y}} \sum_{t=1}^{T} w_t \cdot g(x_t, y)} \right)
\]

\[
\square
\]

### 2.2.2 Packing Primal (or Covering Dual) Online Scaling

The following is the general Packing Primal (or Covering Dual) Online Scaling template.

\[\text{Algorithm 6 Packing Primal Online Scaling}\]

1: \textbf{for} \( t = 0 \ldots T - 1 \) \textbf{do}
2: \hspace{0.5cm} Let \( x_{t+1} \leftarrow \text{Online-Alg}_{\text{costs}}(w_t \cdot g(x, y_t)) \)
3: \hspace{0.5cm} Let \( y_{t+1} \leftarrow \arg \max_{y \in \mathcal{Y}} g(x_{t+1}, y) \)
4: \hspace{0.5cm} Set \( w_{t+1} > 0 \)
5: \textbf{end for}
6: \text{Return } \bar{x} = \frac{\sum_{t} w_t x_t}{\sum_{t} w_t} \text{ and } \bar{y} = \frac{\sum_{t} w_t y_t}{\sum_{t} w_t}

Note: Notice that \( w_t \) must be chosen so that \( w_t \cdot g(x, y_t) \) is a legal input for the Online-Alg_{\text{costs}}. We now present a theorem which can be used to guarantee correctness and number of iterations.

**Theorem 2.2.2.** Let \( R(T), \epsilon_{\text{regret}} \) be the regret parameters of Online-Alg_{\text{costs}}. Then

\[
\min_{x \in \mathcal{X}} g(x, \bar{y}) \geq \lambda^* \left( 1 - \epsilon - \epsilon_{\text{regret}} - \frac{R(T)}{\min_{x \in \mathcal{X}} \sum_{t=1}^{T} w_t \cdot g(x, y_t)} \right)
\]

Notice that in order to guarantee a multiplicative approximation we need to complete the
template such that $\epsilon_{\text{regret}} + \frac{R(T)}{\min_{x \in X} \sum_{t=1}^{T} w_t \cdot g(x, y_t)}$ is smaller than $O(\epsilon)$. This will also typically imply the number of iterations. Notice that instead of predetermining $T$ sometimes it may be useful to continue the iterations until indeed this condition holds (as done for example in algorithm 3).

The proof for this theorem resembles the proof of 2.2.1, it is omitted and could be found in the appendix.

2.2.3 Primal - Dual Online Scaling

The Online Scaling primal-dual template for non-negative min-max problems is the following:

\begin{algorithm}
\begin{algorithmic}[1]
\State for $t = 0 \ldots T - 1$ do
\State Let $y_{t+1} \leftarrow \text{Online-Alg}_{\text{gains}}(w_t \cdot g(x_t, y))$
\State Let $x_{t+1} \leftarrow \text{Online-Alg}_{\text{costs}}(w_t \cdot g(x, y_t))$
\State Set $w_{t+1} > 0$
\End for
\State Return $\bar{x} = \frac{\sum_{t=1}^{T} w_t x_t}{\sum_{t=1}^{T} w_t}$ and $\bar{y} = \frac{\sum_{t=1}^{T} w_t y_t}{\sum_{t=1}^{T} w_t}$
\end{algorithmic}
\end{algorithm}

Note: Notice that $w_t$ must be chosen so that $w_t \cdot g(x_t, y)$ is a legal input for the Online-Alg\text{gains} and also $w_t \cdot g(x, y_t)$ is a legal input for the Online-Alg\text{costs}. The following theorem can be used to guarantee correctness and number of iterations.

The proof can be found in the appendix.

Theorem 2.2.3. Let $R_{\text{costs}}(T), \epsilon_{\text{regret-costs}}$ be the regret parameters of Online-Alg\text{costs}, and let $R_{\text{gains}}(T), \epsilon_{\text{regret-gains}}$ be the regret parameters of Online-Alg\text{gains}. Then:

$$\min_{x \in X} g(x, \bar{y}) \geq \max_{y \in Y} g(\bar{x}, y) \left(1 - \epsilon_{\text{regret-costs}} - \epsilon_{\text{regret-gains}} - \frac{R_{\text{costs}}(T) + R_{\text{gains}}(T)}{\max_{y \in Y} \sum_{t=1}^{T} w_t \cdot g(x_t, y)} \right)$$

Moreover, the inequality remains correct when we replace $\max_{y \in Y} \sum_{t=1}^{T} w_t \cdot g(x_t, y)$ with $\min_{x \in X} \sum_{t=1}^{T} w_t \cdot g(x, y_t)$ or with $\sum_{t=1}^{T} w_t \cdot g(x_t, y_t)$

As in the previous templates in order to guarantee a multiplicative approximation we need to complete the template such that $\epsilon_{\text{regret-costs}} + \epsilon_{\text{regret-gains}} + \frac{R_{\text{costs}}(T) + R_{\text{gains}}(T)}{\max_{y \in Y} \sum_{t=1}^{T} w_t \cdot g(x, y)}$ (or the expression that is chosen to replace this as stated in the theorem) is smaller than $O(\epsilon)$. This will also typically imply the number of iterations. Notice that instead of predetermining $T$ sometimes it may be useful to continue the iterations until indeed this condition holds (as done for example in algorithm 3).
2.3 Domain Shrinkage for Packing Problems

In this section we describe a different approach for obtaining a multiplicative approximation. In this approach the algorithm is exactly a simple dual algorithm for obtaining the additive approximation with one change - we shrink the domain of the dual function from the simplex $\Delta_m$ to the simplex-alpha $\Delta_m^\alpha := \{ x \in \Delta_m \mid x_i \geq \frac{\alpha}{m} \}$. The multiplicative nature of the approximation in this approach is based on two principles: First, roughly speaking, shrinking the domain of a concave function affects the maximum only in a multiplicative manner (this claim is stated more formally in claim 2.3.1). Second, limiting the dual function ($f_{\text{Dual}}(p) = \min_{x \in K} p^T f(x)$) to the simplex-alpha bounds the norm of the gradients of the dual function by a constant that is a multiplicative the optimum $\lambda^*$. We now present the Domain Shrinkage framework for generalised packing:

Algorithm 8 Domain Shrinkage Framework for Generalized Packing

1: Input: $m$ non-negative convex functions $f_i$, over a convex domain $K$. Approximation parameter $\epsilon \in (0, \frac{1}{2})$.
2: Set $T \left\lceil \frac{4m \log m}{\epsilon^2} \right\rceil$, $\alpha \leftarrow \epsilon$, $\eta_{MW} \leftarrow \epsilon$
3: for $t = 0, ..., T - 1$ do
4: Let $p_{t+1} \leftarrow \text{MW-step}(p_t, x_t)$ over $\Delta_m^\alpha$
5: Let $x_{t+1} \leftarrow \arg\min_{x \in K} g(p_{t+1}, x)$.
6: end for
7: Return $\bar{x} = \frac{1}{T} \sum_t x_t$.

Note: The primal minimisation, $\arg\min_{x \in K} g(p_{t+1}, x)$, can be replaced with a multiplicative approximated minimisation, and the new framework can obtain similar results, using slight modifications of the proofs.

Notice that since we changed the domain for which the MW is applied to, we had to modify the MW algorithm to fit the new domain. The modified MW algorithm is presented next in Algorithm 9.

Notice the algorithm is formulated to already handle gain vectors with an arbitrary bound $G$ on their norm rather than the usual bound of 1. We will note that the Domain Shrinkage framework assumes knowledge of $\lambda^*$ since $\text{MW}_{\Delta_m^\alpha}$ uses $G$ which depends on $\lambda^*$. 
Algorithm 9 MW algorithm over $\Delta^\alpha_m$ for gains with bounded $\ell_\infty$ norm

1: Input: $m$ - number of experts, $G$ - bound on the $\ell_\infty$ norm of the gain vectors, $\alpha \in (0, \frac{1}{m}]$, $\eta < \frac{1}{2}$
2: Set $p_1 = \tilde{p}_1 = \frac{1}{m}$, $w_1 = 1_m$,
3: for $t = 1..T$ do
4: Guess $p_t$
5: Incur gains vector $f_t$
6: $w_{t+1} \leftarrow w_t e^{\eta \frac{G}{2}}$ (entry-wise)
7: $\tilde{p}_{t+1} \leftarrow \frac{w_{t+1}}{|w_{t+1}|}$
8: $p_{t+1} \leftarrow (1 - \alpha)\tilde{p}_{t+1} + \alpha \frac{1}{m}$
9: end for

The following lemma states that despite the modification of the algorithm, when the gains are non-negative the regret bound is the same.

Lemma 2.3.1. For non-negative gain vectors with bound $G$ on the infinity norm, the MW algorithm over $\Delta^\alpha_m$ achieves:

$$\text{Regret}(T) = \max_{p \in \Delta^\alpha_m} \frac{\sum_{t=1}^T p^T f_t - \sum_{t=1}^T p_t^T f_t}{\eta \frac{G}{2} \log m}$$

We now state the main theorem regarding the correctness and runtime of the Domain Shrinkage framework:

Theorem 2.3.1. The Domain Shrinkage Framework for Packing returns a $1 + O(\epsilon)$-multiplicative approximate solution in $O \left( \frac{m \log m}{\epsilon^3} \right)$ iterations.

In order to prove this theorem we will need the following two lemmas.

Lemma 2.3.2. Consider the modified problem in which $p \in \Delta^\alpha_m = \{ x \in \Delta_m : x_i \geq \frac{\alpha}{m} \}$, i.e.

$$\min_{x \in K} \max_{p \in \Delta^\alpha_m} p^T f(x) = \mu^* = \max_{p \in \Delta^\alpha_m} \min_{x \in K} p^T f(x)$$

Then:

$$(1 - \alpha)\lambda^* \leq \mu^* \leq \lambda^*$$

Definition 2.3.1. For $p \in \Delta_m$ we define $x_p = \arg \min_{x \in K} p^T f(x)$.

We will note that $f(x_p) \in \nabla_p f^{\text{Dual}}$ where $f^{\text{Dual}}(p) = \min_{x \in K} p^T f(x)$.

Lemma 2.3.3. Let $p \in \Delta^\alpha_m$. Then $\forall i \in [m] : f_i(x_p) \leq \|f(x_p)\|_1 \leq \frac{\lambda^* m}{\alpha}$.

The proofs of these lemmas appear after the proof of the theorem.

Proof. (Theorem 2.3.1)

Let $G_\infty$ be a bound on the $\ell_\infty$ norm of the gradients given to the MW$_{\Delta^\alpha_m}$ algorithm, and notice that from lemma 2.3.3 we can define $G_\infty = \frac{\lambda^* m}{\alpha}$. Now,
\[
\max_i f_i(\bar{x}) = \max_{p \in \Delta_m} p^T f(\bar{x}) \\
\leq \frac{1}{1-\alpha} \max_{p \in \Delta_m^n} p^T f(\bar{x}) \quad \text{claim 2.3.1}
\]
\[
\leq \frac{1}{1-\alpha} \frac{1}{1-\eta} \left( \sum_{t=1}^T p_t^T f(x_t) + G_\infty \log \frac{m}{\eta} \right) \quad \text{regret (lemma 2.3.1)}
\]
\[
\leq \frac{1}{1-\alpha} \frac{1}{1-\eta} \left( T \lambda^* + G_\infty \log \frac{m}{\eta} \right) \quad \text{lemma 2.3.2}
\]
\[
\leq \frac{1}{1-\alpha} \frac{1}{1-\eta} \lambda^* + \frac{1}{1-\alpha} \frac{1}{1-\eta} \frac{\lambda^* m \log m}{\epsilon \eta T} \quad \text{lemma 2.3.3}
\]
\[
= \frac{1}{\epsilon} \lambda^* + \frac{1}{\epsilon^2 T} \frac{\lambda^* m \log m}{\epsilon^2 T} \quad \text{choice of } \alpha \text{ and } \eta
\]
\[
\leq \lambda^* \left( 1 + \frac{6\epsilon}{\epsilon} + \frac{4 \epsilon m \log m}{\epsilon^2 T} \right) \quad \epsilon < \frac{1}{2}
\]
\[
\leq \lambda^* \left( 1 + 7\epsilon \right) \quad \text{Choice of } T
\]

Lemma 2.3.2 is a special case of the following more general claim. This claim, roughly speaking, bounds the effect of shrinkage of the domain, on the maximum of a concave function.

**Claim 2.3.1.** Let \( \mathcal{L} \) be a convex domain and let \( g : \mathcal{L} \to \mathbb{R}_+ \) be a concave function. Let \( z_0 \in \mathcal{L} \) such that \( g(z_0) \geq 0 \) and let \( \alpha \in [0,1) \). Define \( \mathcal{L}' = (1-\alpha)\mathcal{L} + \alpha z_0 = \{(1-\alpha)z + \alpha z_0 | z \in \mathcal{L} \} \) (Notice that \( \mathcal{L}' \subset \mathcal{L} \)). Denote \( \lambda^* = \max_{z \in \mathcal{L}} g(z) \) and \( \mu^* = \max_{z \in \mathcal{L}'} g(z) \).

Then:
\[
(1-\alpha)\lambda^* \leq \mu^* \leq \lambda^*
\]

**Proof.** Notice that \( \mu^* \leq \lambda^* \), since \( \Delta^\alpha \subset \Delta \) and a maximum of a function over a subset is never larger than the maximum over the original set. For the other direction, let \( z^* \) be a point in \( \mathcal{L} \) for which the maximum is achieved. then:

\[
(1-\alpha)\lambda^* = (1-\alpha)g(z^*)
\]
\[
\leq (1-\alpha)g(z^*) + \alpha g(z_0) \quad \text{non-negativity of } g(z_0)
\]
\[
\leq g((1-\alpha)z^* + \alpha z_0) \quad \text{concavity of } g
\]
\[
\leq \max_{z \in \mathcal{L}'} g(z) = \mu^*
\]

**Proof.** (Lemma 2.3.2)

Lemma 2.3.2 is obtained by using claim 2.3.1 and taking \( \mathcal{L} = \Delta_m, z_0 = \frac{1}{m}, g(z) = \min_{x \in \mathcal{K}} z^T f(x) \).
We now turn to prove lemma 2.3.3 in order to do so we use the following claim, which is useful also in the case we are using a multiplicative approximated minimisation instead of exact minimisation for in the primal step.

Claim 2.3.2. Let \( p \in \Delta_m^\alpha \) and \( x \in \mathcal{K} \) such that \( p^T f(x) \leq \gamma \). Then

\[
\sum_{i=1}^{m} f_i(x) = |f(x)|_1 \leq \frac{\gamma m}{\alpha}
\]

and in particular

\[
\forall i \in [m], \ f_i(x) \leq \frac{\gamma m}{\alpha}
\]

Proof.

\[
\frac{\alpha}{m} |f(x)|_1 = \sum_{i=1}^{m} \frac{\alpha}{m} f(x)(i) \quad \text{non-negativity of } f(x)
\]
\[
\leq \sum_{i=1}^{m} p(i) f(x)(i) \quad p \in \Delta_m^\alpha
\]
\[
= p^T f(x)
\]
\[
\leq \gamma \quad \text{assumption on } p^T f(x)
\]

Proof. (Lemma 2.3.3) Let \( p \in \Delta_m^\alpha \). From the definition of \( x_p \) we know that

\[
p^T f(x_p) = \min_{x \in \mathcal{K}} p^T f(x) \leq \max_{p \in \Delta_m} \min_{x \in \mathcal{K}} p^T f(x) = \lambda^*
\]

Applying claim 2.3.2 with \( \gamma = \lambda^* \) we get

\[
\forall i \in [m], \ f_i(x_p) \leq |f(x_p)|_1 \leq \frac{\lambda^* m}{\alpha}
\]

\[
\square
\]
Chapter 3

Speeding-up the Framework

In this chapter we introduce our main contribution - a sampling and speed-up technique which we incorporate into the basic frameworks from the previous chapter. The technique is based on $\ell_1$-sampling of non-negative vectors which is executed in an efficient manner using a binary-recursive approach. The speed-up obtained by this technique stems from two directions. The $\ell_1$ sampling generates gradient estimators that are sparse, and so the change between iterations is small and this allows us to reduce the complexity of calculating variable values using variable values from the previous iteration. The other direction is the recursive manner in which we perform the sampling which enables us to generate the gradient estimators efficiently.

The organization of this chapter is as follows, we first present the sped-up frameworks and the main theorems regarding correctness and running time. We then introduce the recursive $\ell_1$ sampling method and analyse it. This is followed by a discussion regarding the amortised complexity. Finally we give the proofs for the theorems which were stated at the beginning of the chapter.

We now introduce the sped-up general frameworks and state the main theorems regarding correctness and running times. For the simplicity of analysis we will assume in this section that $f_i$ are linear functions. We would like to note though that many of the claims in this section can be proved for some other cases as well. An intuition for why this is possible is that in many cases it is enough for the set of functions $f_i$ to be linear in the parameters defining them but not necessarily linear with respect to their input, e.g. functions of the form $\sum_{j=1}^d a_{i,j}g_j(x)$ for concave scalar functions $g_j$. 
Algorithm 10 Online Scaling for Packing with Recursive $\ell_1$ Sampling

1: Input: $m$ non-negative convex functions $f_i$, over a convex domain $K$. Approximation parameter $\epsilon > 0$.
2: Perform pre-processing for the recursive $\ell_1$-sampling
3: $T \leftarrow \lceil \frac{7m \log m}{\epsilon^2} \rceil$, MW (gains) Parameters: $m, \eta \leftarrow \epsilon$
4: for $t = 0 \ldots T - 1$ do
5: Let $p_{t+1} \leftarrow$ MW-step ($v_t$)
6: Let $x_{t+1} \leftarrow \arg \min_{x \in K} \left( \frac{1}{\|f(x_{t+1})\|_1} \right) p_{t+1}^T f(x)$
7: $\omega_{t+1} \leftarrow \frac{1}{\|f(x_{t+1})\|_1}$
8: $v_t \leftarrow \omega_t \cdot$ recursive $\ell_1$-sample $(f(x_t))$
9: end for
10: Return $\bar{x} = \frac{\sum_{t} \omega_t x_t}{\sum_{t} \omega_t}$.

For this framework we have the following result:

**Theorem 3.0.2.** The Online Scaling Framework for Packing with recursive $\ell_1$ sampling (Algorithm 10) runs $O \left( \frac{m \log m}{\epsilon^2} \right)$ iterations, and with probability at least $3/4$ returns a $(1 + O(\epsilon))$-multiplicative approximation solution. In the case $f_i$ are linear functions, then algorithm 10 can be implemented to run in $O \left( \frac{mn \log^2 m}{\epsilon^2} \right)$ operations plus $O \left( \frac{m \log m}{\epsilon^2} \right)$ minimizations of a linear function over $K$.

Algorithm 11 Online Scaling for Covering with Recursive $\ell_1$ Sampling

1: Input: $m$ non-negative concave functions $f_i$, over domain $K$. Approximation parameter $\epsilon$.
2: Perform pre-processing for the recursive $\ell_1$-sampling
3: $K \leftarrow \frac{9 \log m}{\epsilon^2}, t \leftarrow 0$, MW Parameters: $m, U \leftarrow 2K, \eta \leftarrow \epsilon$.
4: while $\min_i \sum_{s=1}^{t} v_s(i) < K$ do
5: $p_{t+1}, E_{t+1} \leftarrow$ MW-step($v_t$)
6: $x_{t+1} \leftarrow \arg \max_{x \in K} \left( \frac{1}{\|f(x_{t+1})\|_1} \right) p_{t+1}^T f(x)$
7: Update the recursive $\ell_1$-sampling data-structure with $\Delta E$ equal $E_t \setminus E_{t+1}$
8: $\omega_{t+1} \leftarrow \frac{1}{\|f(x_{t+1})\|_1}$
9: $v_{t+1} \leftarrow \omega_{t+1} \cdot$ recursive $\ell_1$-sample $(f(x_{t+1})|_{E_{t+1}})$
10: $t \leftarrow t + 1$
11: end while
12: Return $\bar{x} = \frac{\sum_{s \leq t} \omega_s x_s}{\sum_{s \leq t} \omega_s}$.

For this framework we have the following result:

**Theorem 3.0.3.** Algorithm 11 always ends after at most $O \left( \frac{m \log m}{\epsilon^2} \right)$ iterations, and with probability at least $3/4$ returns a $(1 - O(\epsilon))$-multiplicative approximation solution. In the case $f_i$ are
linear functions, then algorithm \[1\] can be implemented to run in \(O\left(\frac{mn \log^2 m}{\epsilon^2}\right)\) operations plus \(O\left(\frac{m \log m}{\epsilon^2}\right)\) maximizations of a linear function over \(\mathcal{K}\).

We now introduce the speed-up techniques used in these frameworks. The proofs of theorems 3.0.2 and 3.0.3 is discussed in section 3.3.

3.1 Recursive \(\ell_1\)-sampling

We define the \(\ell_1\)-sampling of a non-negative vector \(u \in \mathbb{R}^m\) to be the random variable \(v \in \mathbb{R}^m\) which has the following distribution:

\[
\forall i \in [m] : v = e_i \cdot \|u\|_1 \quad \text{w.p.} \quad \frac{u_i}{\|u\|_1}.
\]

Suppose we are given some explicit linear function \(h(x) : \mathcal{K} \rightarrow \mathbb{R}^m_{+}\), a series of inputs \(x_1, x_2, \ldots, x_T\), and we would like to generate a corresponding series of \(\ell_1\)-samplings of \(h(x_t)\) for \(t = 1, 2, \ldots, T\). Naively this would take \(O(Tnm)\) operations since for each \(x_t\) we need to calculate \(h(x_t)\) in order to generate its \(\ell_1\)-sampling, and this costs \(O(nm)\) operations per input. In this section we will show that by using preprocessing we can generate the desired series of \(\ell_1\)-samplings using only a total of \(O(nm + Tn \log m)\) operations.

In order to describe the algorithm we first introduce a few definitions.

Definitions:

- For a range of \(s\) numbers \(I = \{k, k+1, \ldots, k+s-1\}\), we will define \(\text{Split}(I)\) to be the two ranges of numbers obtained by splitting \(I\) "in the middle" - \(\{k, \ldots, k + \lceil \frac{s}{2} \rceil - 1\}\) and \(\{k + \lceil \frac{s}{2} \rceil, \ldots, k+s-1\}\) (When \(I\) consists of only one number \(\text{Split}(I) = I\)).

- We will define \(\mathcal{I}\) to be all the ranges we get by applying the \(\text{Split}\) function on \([m]\) and continuing recursively. Formally, denote the set \(U_m = \{\{1\}, \{2\}, \ldots, \{m\}\}\), and construct \(\mathcal{I}\) in the following way:

1. Initialize: \(\mathcal{I}_1 = \{[m]\}, i = 1\).
2. While \(\mathcal{I}_i \neq U_m\) do: \(\mathcal{I}_{i+1} \leftarrow \{\text{Split}(I) | I \in \mathcal{I}_i\}, i \leftarrow i + 1\)
3. \(\mathcal{I} \leftarrow \bigcup_{j=1}^{i} \mathcal{I}_j\)

Notice that the number of elements in \(\mathcal{I}\) is bounded by \(2m\).

- For a set of numbers \(I \subset [m]\) we will define the function \(h_I(x) = \sum_{i \in I} h_i(x)\)

We will note that the total number of iterations in the construction of \(\mathcal{I}\) is bounded by \(2 \log m\). In addition, for every iteration in the construction \(|\mathcal{I}_j| \leq 2^{j-1}\).

We now describe the recursive \(\ell_1\) sampling algorithm.

The following lemmas state the correctness and running time of the Recursive \(\ell_1\)-Sampling. The first addresses the important case in which the functions \(h_i\) are explicitly given linear functions. The second lemma states the correctness and running time for a more general setting, although the technique may be adapted for other settings as well.
Algorithm 12 Recursive $\ell_1$ Sampling

1: Preprocessed: $h_I$ for all $I \in \mathcal{I}$.
2: Input: $x_t$
3: Initialize: $I = [m]$
4: while $|I| > 1$ do
5: \hspace{1em} $I_1, I_2 \leftarrow$ Split($I$)
6: \hspace{1em} $\alpha_{I_1} \leftarrow h_{I_1}(x_t), \alpha_{I_2} \leftarrow h_{I_2}(x_t), \alpha_I \leftarrow \alpha_{I_1} + \alpha_{I_2}$
7: \hspace{1em} Randomly update $I$ in the following way: $I \leftarrow \begin{cases} I_1 & \text{w.p. } \frac{\alpha_{I_1}}{\alpha_I} \\ I_2 & \text{w.p. } \frac{\alpha_{I_2}}{\alpha_I} \end{cases}$
8: end while
9: Return $v_t = e_i|h_{[m]}(x_t)| = e_i|h(x_t)|_1$, for $i$ such that $I = \{i\}$.

Lemma 3.1.1. (Correctness and running time in the linear case)

Suppose $h_i$ are explicit linear functions. Then,

(Running time) The preprocessing for the recursive $\ell_1$-sampling can be carried out in $O(nm)$ operations, and given the preprocessing, every time the recursive $\ell_1$-sampling is carried out, it runs $O(\log m)$ iterations and takes $O(n \log m)$ operations. (Correctness) Let $u = \text{recursive } \ell_1 \text{- sampling}(h(x))$ according to algorithm 12. Then the distribution of $u$ is equal to the distribution obtained by $\ell_1$-sampling of $h(x)$, that is, $\forall i \in [m] : u = e_i|h(x)|_1$ w.p. $\frac{h_i(x)}{|h(x)|_1}$.

Lemma 3.1.2. (Correctness and running time in a more general setting)

Let $\mathcal{G}$ be a set of convex (concave) functions over $K \subset \mathbb{R}^n$ which is parametrised by a vector $a \in A \subset \mathbb{R}^d$, that is: $\mathcal{G} = \{g_a(x) : K \rightarrow \mathbb{R}\}_{a \in A}$. Assume that $A$ is closed under addition ( $\forall a_1, a_2 \in A : a_1 + a_2 \in A$) and $\mathcal{G}$ is linear in its parameter: $\forall a_1, a_2 \in A, x \in K : g_{a_1}(x) + g_{a_2}(x) = g_{a_1 + a_2}(x)$. Suppose that for any $a \in A$ and any $x \in K$ the evaluation of $g_a(x)$ can be done in at most $T_G$ operations. Finally assume that for all $i \in [m]$ it holds that $h_i(x) = g_a(x)$ for some given $a_i \in A$, and for all $i \in [m]$ these are non-negative functions. In this case we claim the following:

(Running time) The preprocessing for the recursive $\ell_1$-sampling can be carried out in $O(dm)$ operations, and given the preprocessing, every time the recursive $\ell_1$-sampling is carried out, it runs $O(\log m)$ iterations and takes $O(T_G \log m)$ operations. (Correctness) Let $u = \text{recursive } \ell_1 \text{- sampling}(h(x))$ according to algorithm 12. Then the distribution of $u$ is equal to the distribution obtained by $\ell_1$-sampling of $h(x)$, that is, $\forall i \in [m] : u = e_i|h(x)|_1$ w.p. $\frac{h_i(x)}{|h(x)|_1}$.

We will present the proof of the correctness which is the same for both lemmas and an outline of the proof for the running time.

Proof. (Correctness) Let $i \in [m]$. Notice that the probability that $v_t = e_i|h(x_t)|_1$ equals to the probability that at the end of the algorithm $I = \{i\}$. Consider the decision tree of all the possible choices done by the algorithm, and notice that there exists exactly one leaf that is $\{i\}$ and so there exists one path from the root to this leaf. Denote the different values of $I$ in this path: $I_1 = [m], I_2, \ldots, I_k = \{i\}$. Since there is only one such path, the probability that at the end of the
algorithm $I = \{i\}$, is the probability that the algorithm has chosen this path which equals to

$$\Pr \left( \bigwedge_{i=1}^{k-1} \text{The algorithm chose to update } I \text{ to be } I_{i+1} \text{ at the } i\text{-th iteration of the algorithm} \right) = \prod_{i=1}^{k-1} \Pr \left( \text{The algorithm chose to update } I \text{ to be } I_{i+1} \text{ at the } i\text{-th iteration of the algorithm} \mid I_i \right) = \prod_{i=1}^{k-1} \frac{h_{I_{i+1}}(x_t)}{h_{I_i}(x_t)} = \frac{h_{I_k}(x_t)}{h_{I_1}(x_t)} = \frac{h_i(x_t)}{|h(x_t)|_1}$$

(Running Time) For a given $x_t$, algorithm 12 runs $O(\log m)$ iterations. Using the preprocessing each iteration takes $O(n)$ operations in the linear case and $T_G$ operations in the more general case. This gives a total running time of $O(n \log m)$ for the linear case and $O(T_G \log m)$ in the more general case.

### Recursive $\ell_1$ sampling for time-varying functions: updating the recursive $\ell_1$ sampling data structure

Notice that in the case of generalised covering, algorithm 11 performs $\ell_1$-sampling on the set of functions $f|_t^{E_t}$ which changes as $E_t$ changes. This implies that we should change the data structure created by the preprocessing when the set of functions changes. Updating the data structure to correspond to $f|_{E_{t+1}}^0$ based on the current state of the data structure which corresponds to $f|_{E_t}^0$ is done using the following algorithm.

**Algorithm 13** Recursive $\ell_1$ Sampling Data Structure Update

1: Input: $(f|_{E_t}^0)_I$ for all $I \in \mathcal{I}$, $\Delta E = E_t \setminus E_{t+1}$
2: for $j \in \Delta E$ do
3: for $I \in \mathcal{I}$ s.t. $j \in I$ do
4: $(f|_{E_{t+1}}^0)_I \leftarrow (f|_{E_t}^0)_I - f_j$
5: end for
6: end for

The following lemma states the correctness and running time of the recursive $\ell_1$ sampling data structure update in the sped-up framework. The running time in this case relies on the fact that for all $0 \leq t < T : |E_t \setminus E_{t+1}| \leq 1$. The lemma refers only to the case in which $f_i$ are explicit linear functions. Similar lemmas can be proven for some other setting as well.

**Lemma 3.1.3.** (Recursive $\ell_1$-sampling data structure update correctness and running time)

Suppose $f_i$ are explicit linear functions. In the fast framework for covering (algorithm 11) if for all $0 \leq t$, the data structure is updated with $\Delta E = E_t \setminus E_{t+1}$, then after this update the data structure contains the data for the functions $f|_{E_{t+1}}^0$. In this case, every time the update is carried out it takes $O(n \log m)$ operations.
3.2 Amortized Complexity

When using the $\ell_1$ sampling, the actual gradients we are using, $v_t$, contain only one non-zero entry, (which is an index of a “live” expert in the case of generalised coverings). The main advantage of this is that the changes between iterations are small, and by remembering variables from previous iterations, the runtime complexity of an iteration can be reduced. In particular, algorithm 11 can be implemented such that for each iteration: a. Calculating the function $p^T_{t+1}f$ takes $O(n)$ operations. b. The recursive $\ell_1$-sampling data structure update takes $O(n \log m)$ operations. c. An iteration in the MWU algorithm takes $O(1)$ operations. d. Checking whether the while loop condition is satisfied or not can be carried out using $O(1)$ operations.

From the above we get the following:

**Lemma 3.2.1.** Suppose $f_i$ are explicit linear functions. Algorithm 11 can be implemented such that each iteration takes $O(n \log m)$ operations plus one maximisation of a linear function over the domain $\mathcal{K}$.

For the detailed description of the efficient implementation the reader is referred to the appendix. However, we will mention that the implementation is done without much sophistication, and requires mainly looking carefully into the details of the algorithm and carrying it out wisely.

3.3 Analysis of the Sped-up Framework

We now develop the proof for theorem 3.0.3 - the theorem regarding the running time and correctness of the generalised covering framework. A discussion about the proof for theorem 3.0.2 for the generalised packing case which is similar in many ways, can be found in the appendix.

**Proof.** (Theorem 3.0.3)

Note: The proofs for all the lemmas stated in this proof could be found in the appendix.

Denote by $T$ the value of $t$ at the end of the algorithm. First we notice that the number of iterations $T$, is bounded.

**Lemma 3.3.1.** $T \leq 2Km = O \left(\frac{m \log m}{\epsilon^2}\right)$.

The following lemmas give a bound on the distortion created by the sampling, in a multiplicative sense, and their proofs rely on a variation of a Multiplicative-Azuma-like inequality stated in 6.0.17 in the appendix (based entirely on lemma 10 from [15]).

**Lemma 3.3.2.** $(1 - \epsilon) \sum_{t=1}^{T} p^T_t \omega_t(f(x_t)|_{E_t}) \leq \sum_{t=1}^{T} p^T_t v_t + \frac{3}{\epsilon} \quad w.p \ at \ least \ 1 - e^{-3}$

**Lemma 3.3.3.** $(1 - \epsilon) \min_{p \in \Delta_m} \sum_{t=1}^{T} p^T v_t \leq \min_{p \in \Delta_m} \sum_{t=1}^{T} p^T \omega_t(f(x_t)|_{E_t}) + \frac{3 \log m}{\epsilon} \quad w.p \ at \ least \ 1 - e^{-3}$

Notice that the probability that the events in both lemmas hold is larger than $\frac{2}{3}$. From the regret bound of the MWU algorithm (theorem 2.1.1) we get:
Lemma 3.3.4. \((1 - \eta) \sum_{t=1}^{T} p_t^T v_t \leq \min_{p \in \Delta_m} \sum_{t=1}^{T} p^T v_t + \frac{\log m}{\eta} \).

Rearranging 3.3.3 we get:

\[
\min_{p \in \Delta_m} \frac{\sum_{t=1}^{T} p^T \omega_t f(x_t)|_{E_t}}{\sum_{t=1}^{T} p^T v_t} \geq 1 - \epsilon - \frac{3 \log m}{\epsilon \sum_{t=1}^{T} p^T v_t} \quad (3.1)
\]

Using 3.3.2 and 3.3.4 we get:

\[
\min_{p \in \Delta_m} \frac{\sum_{t=1}^{T} p^T \omega_t f(x_t)|_{E_t}}{\sum_{t=1}^{T} p^T v_t} \geq 1 - 2\epsilon - \frac{\log m + (1 - \epsilon)3}{\epsilon \sum_{t=1}^{T} p^T v_t} \quad (3.2)
\]

Now,

\[
\frac{\min_{p \in \Delta_m} p^T f(\bar{x})}{\lambda^*} \geq \frac{\min_{p \in \Delta_m} \sum_{t=1}^{T} p^T \omega_t f(x_t)}{\lambda^* \sum_{t=1}^{\lambda^*} \omega_t} \quad \text{Entry-wise concavity of } f(x)
\]

\[
\geq \frac{(1 - \epsilon) \min_{p \in \Delta_m} \sum_{t=1}^{T} p^T \omega_t f(x_t)}{\sum_{t=1}^{T} p_t^T \omega_t f(x_t)} \quad \text{Choice of } x_t
\]

\[
\geq \frac{(1 - \epsilon) \min_{p \in \Delta_m} \sum_{t=1}^{T} p^T \omega_t (f(x_t)|_{E_t})}{\sum_{t=1}^{T} p_t^T \omega_t (f(x_t)|_{E_t})} \quad \text{Positiveness of } f \text{ and } \forall t \forall i \notin E_t : p_t(i) = 0
\]

\[
\geq 1 - 4\epsilon - \frac{9 \log m}{\epsilon \sum_{t=1}^{T} p^T v_t} \quad \text{Using the multiplication of 3.1 and 3.2}
\]

\[
\geq 1 - 4\epsilon - \frac{9 \log m}{\epsilon K} \geq 1 - 5\epsilon \quad \text{Stopping condition of algorithm 11}
\]

All we have left to show, is the running time of the sped-up framework in the case \(f_i\) are linear functions. Since we already bounded the number of iterations, using lemma 3.2.1 we get the desired result.
Chapter 4

Applications

4.1 Normalized Covering SDP

In this section we will show how to approximate the Normalized Covering SDP problem (1.12) in near-linear time. Suppose we are given a Normalized Covering SDP defined by $A_i \in \mathbb{S}^n_+$ for $i = 1, ..., m$ and $\epsilon \in (0, \frac{1}{2})$. Claim 1.1.1 implies that finding an approximate solution to the corresponding Maxmin Positive SDP problem (1.13) is sufficient for finding an approximate solution to the original problem (using linear runtime operations for the reduction itself).

We will apply the sped-up framework for generalised covering (Algorithm 11) for the corresponding Maxmin Positive SDP problem. This problem is a special case of the generalized covering problem in which $K = \{X \in \mathbb{S}^n_+ | \text{Tr}(X) \leq 1\}$ and $\forall i \in [m]: f_i (X) := A_i \bullet X$ where $A_i \in \mathbb{S}^n_+$ for all $i$. Notice that $f_i$ are linear functions.

The optimization step of our algorithm (line 6 in algorithm 11) for the case of Maxmin Positive SDP amounts to solving the maximum eigenvector problem. The following lemma summarises the running time for this problem, which can be achieved using the Lanczos method (for proof see appendix).

Lemma 4.1.1. Let $A \in \mathbb{S}^n_+$ be a matrix with $N$ non-zero entries. Denote $K = \{x \in \mathbb{S}^n_+ | \text{Tr}(X) \leq 1\}$. Then there is an algorithm that with high probability returns in total time of $\tilde{O}\left(\frac{N}{\epsilon} \right)$ a matrix $X \in K$ such that $A \bullet X \geq (1 - \epsilon) \max_{X' \in K} A \bullet X'$.

We can now apply the sped-up framework for this case.

Corollary 4.1.1. Let $K = \{X \in \mathbb{S}^n_+ | \text{Tr}(X) \leq 1\}$. Then, Algorithm 11 with input $\{A_i\}_{i=1}^m$, $K$, $\epsilon$ which uses the algorithm from lemma 4.1.1 as a multiplicative approximation oracle, returns $\bar{X} \in K$ such that with probability at least $\frac{3}{4}$:

$$\min_{i \in [m]} A_i \bullet (\bar{X}) \geq (1 - O(\epsilon)) \max_{X' \in K} \min_{i \in [m]} A_i \bullet X$$

The algorithm can be implemented such that the running time of the algorithm in this case is $\tilde{O}\left(\frac{mn^2}{\epsilon^2} \right)$.
The matrix \( \frac{1}{\min_{i \in [m]} A_i \cdot \bar{X}} \bar{X} \) is a \( 1 + O(\epsilon) \)-multiplicative approximation to the corresponding Normalized Covering SDP problem.

4.2 Non-negative Linear Classifier

We will apply the sped-up framework for generalised covering (Algorithm 11) for the Non-negative linear classifier (problem 1.15). This problem is a special case of the generalized covering problem in which \( \mathcal{K} = \mathbb{B}_n \) and \( f(x) = Ax \) where \( A \in \mathbb{R}^{m \times n}_+ \), and so again \( f_i \) are linear functions.

The optimization step of our algorithm (line 6 in Algorithm 11) for the case of Non-negative linear classifier amounts to the problem of normalising a vector in \( \mathbb{R}^n \), which takes \( O(n) \) operations. Using this and Theorem 3.0.3 we get the following result.

**Corollary 4.2.1.** Let \( A \in \mathbb{R}^{m \times n}_+ \) and let \( \epsilon < \frac{1}{2} \). Algorithm 11 with input \( A \), \( \mathbb{B}_n \), \( \epsilon \) returns \( \bar{x} \in \mathcal{K} \) such that with probability at least \( \frac{3}{4} \):

\[
\min_{i \in [m]} A \bar{x} \geq (1 - O(\epsilon)) \max_{x \in \mathbb{B}_n} \min_{i \in [m]} A x
\]

The algorithm can be implemented such that the running time of the algorithm in this case is \( \tilde{O}(\frac{mn}{\epsilon^2}) \).
Chapter 5

Lower Bounds

In this chapter we give some lower bounds on the running times of multiplicative approximation algorithms for several problems, all of which are special cases of the generalised packing or generalised covering problems with linear functions. Note that the lower bounds are only in terms of the input size, i.e. $m$ and $n$, but not in terms of the approximation parameter, $\epsilon$.

The bounds we give are based on the following conjecture:

**Conjecture 5.0.1.** Consider a matrix of dimensions $n \times d$ such that with probability $1/2$ it is a NO instance - each row of the matrix is a random unit vector (i.e. all zeros except for one entry of which the value is 1) chosen uniformly and independently between different rows, and with the remaining $1/2$ probability it is a YES instance - one row is chosen randomly to be zero, and the remaining rows are randomly chosen unit vectors (in a similar manner to the first case the first case). Then, any algorithm that with probability at least $2/3$ determines whether the matrix is a YES instance or a NO instance (i.e. it contains a row of zeros or not), must read $\Omega(nd)$ entries of the matrix.

### 5.1 Lower Bounds for Zero Sum Games

**Definition 5.1.1.** We will call an algorithm $A$ a multiplicative approximation algorithm for the decision problem of covering ZSG if for all inputs of the form $A \in \mathbb{R}^{n \times d}_+, \epsilon, \lambda$ the algorithm returns with probability at least $2/3$ $x \in \Delta_d$ such that $Ax \geq \lambda(1-\epsilon)$ (entry-wise) if such exists, and declares failure if there is no such $x \in \Delta_d$.

**Definition 5.1.2.** We will call an algorithm $A$ a multiplicative approximation algorithm for the decision problem of packing ZSG if for all inputs of the form $A \in \mathbb{R}^{n \times d}_+, \epsilon, \lambda$ the algorithm returns with probability at least $2/3$ $x \in \Delta_d$ such that $Ax \leq \lambda(1+\epsilon)$ (entry-wise) if such exists, and declares failure if there is no such $x \in \Delta_d$.

**Lemma 5.1.1.** Any multiplicative approximation algorithm for the decision problem of covering ZSG has running time $\Omega(nd)$
Proof. Let \( \mathcal{A} \) be such an algorithm. Let us create an algorithm for determining the YES/NO instances introduced in 5.0.1. Given a matrix \( A \) run \( \mathcal{A} \) on \( A, \epsilon = 1/2, \lambda = 1/d \). If \( \mathcal{A} \) declared failure return YES (the instance contains a row of zeros), otherwise return NO. Notice that if the matrix is a NO instance then for \( x = \frac{1}{d} \) it holds that \( Ax = \frac{1}{d} \geq \lambda(1-\epsilon) \) but if the matrix is a YES instance \( Ax \) will always contain an entry which equals to zero and so smaller than \( \lambda(1-\epsilon) \) and so could not fulfill the desired inequalities. We get that the algorithm we introduced determines between YES and NO instances, and so according to 5.0.1 its running time is \( \Omega(nd) \).

Lemma 5.1.2. Any multiplicative approximation algorithm for the decision problem of packing ZSG has running time \( \Omega(nd) \)

Proof. Let \( \mathcal{A} \) be such an algorithm. Let us create an algorithm for determining the YES/NO instances introduced in 5.0.1. Given a matrix \( A \), define \( B = A^T \) and run \( \mathcal{A} \) on \( B, \epsilon = 1/2, \lambda = 0 \). If \( \mathcal{A} \) declared failure return NO (the instance \( A \) contains a row of zeros), otherwise return YES. Notice that if \( A \) is a NO instance then for any \( x \in \Delta_n \) it holds that at least one entry in \( Bx \) is not smaller than \( \frac{1}{n} \) and so it could not fulfill all desired inequalities. On the other hand, if \( A \) is a YES instance, with the \( j \)-th row of all zeros, then for \( x = e_j \) it holds that \( Bx = 0_d \leq \lambda(1+\epsilon) \). We get that the algorithm we introduced determines between YES and NO instances, and so according to 5.0.1 its running time is \( \Omega(nd) \).

5.2 Lower Bounds for Non-Negative Linear Classifier

Definition 5.2.1. We will call an algorithm \( \mathcal{A} \) a multiplicative approximation algorithm for the decision problem of Non-negative Linear Classifier if for all inputs of the form \( A \in \mathbb{R}^{n \times d}_+, \epsilon, \lambda \) the algorithm returns with probability at least \( 2/3 \) \( x \in \mathbb{B}_d \) such that \( Ax \geq \lambda(1-\epsilon) \) (entry-wise) if such exists, and declares failure if there is no such \( x \in \mathbb{B}_d \).

Lemma 5.2.1. Any multiplicative approximation algorithm for the decision problem of Non-negative Linear Classifier has running time \( \Omega(nd) \)

Proof. Let \( \mathcal{A} \) be such an algorithm. Let us create an algorithm for determining the YES/NO instances introduced in 5.0.1. Given a matrix \( A \) run \( \mathcal{A} \) on \( A, \epsilon = 1/2, \lambda = 1/\sqrt{d} \). If \( \mathcal{A} \) declared failure return YES (the instance contains a row of zeros), otherwise return NO. Notice that if the matrix is a NO instance then for \( x = \frac{1}{\sqrt{d}} \) it holds that \( Ax = \frac{1}{\sqrt{d}} \geq \lambda(1-\epsilon) \) but if the matrix is a YES instance \( Ax \) will always contain an entry which equals to zero and so smaller than \( \lambda(1-\epsilon) \) and so could not fulfill the desired inequalities. We get that the algorithm we introduced determines between YES and NO instances, and so according to 5.0.1 its running time is \( \Omega(nd) \).
Chapter 6

Appendix

6.0.1 Regret of the MW\(_U\) Algorithm

In this part we prove lemma 2.1.1 showing that despite the modification the MW\(_U\) algorithm (algorithm 4) can still have the same regret bound guarantee as the original MW algorithm in the case relevant to our framework in which the costs are always non-negative.

We will assume from here on that:

- \(\{v_t\}_{t=1}^T\) is a series of non-negative cost vectors for which \(\forall t \forall i \in E_t : v_t(i) \leq 1\).
- \(E_{T+1} \neq \emptyset\)

The next lemma says, intuitively, that two series of cost vectors which always agree on the costs of "live" experts will produce the same series of outputs.

**Lemma 6.0.2.** Let \(\{v'_t\}_{t=1}^T\) be another series of non-negative cost vectors, and denote by \(\{p'_t, w'_t, E'_t\}_{t=1}^T\) the corresponding variables of MW-K when running on this series. If \(\forall t \forall i \in E_t : v'_t(i) = v_t(i)\) then: \(\forall t : E'_t = E_t, \forall i \in E_t : w'_t(i) = w_t(i), p'_t = p_t\).

**Proof.** First let us show that \(\forall t : E'_t = E_t\). Let \(1 \leq t \leq T + 1\). If \(i \in E_t\) then from the monotonicity of \(E_s\) for all \(1 \leq s < t\) it holds that \(i \in E_s\) and so: \(\sum_{1 \leq s < t} v'_s(i) = \sum_{1 \leq s < t} v_s(i) < K\) and so \(i \in E'_t\). If \(i \notin E_t\) then there exists some \(1 \leq s < t\) for which \(i \in E_s\) and \(i \notin E_{s+1}\). From the monotonicity of \(E_s\) for all \(1 \leq s \leq s\) it holds that \(i \in E_s\) and so: \(\sum_{1 \leq s \leq s} v'_s(i) = \sum_{1 \leq s \leq s} v_s(i) \geq K\) (because \(i \notin E_{s+1}\))

and so \(i \notin E'_t\).

Now we will show that \(\forall 1 \leq t \leq T + 1 : \forall i \in E_t : w'_t(i) = w_t(i)\). Let \(1 \leq t \leq T + 1\) and let \(i \in E_t\). From monotonicity of \(E_s\) we get that \(\forall 1 \leq s \leq t : i \in E_s \Rightarrow \forall 1 \leq s \leq t : v'_s(i) = v_s(i)\). Now \(w'_t(i) = \prod_{1 \leq s \leq t-1} (1 - \eta v'_s(i)) = \prod_{1 \leq s \leq t-1} (1 - \eta v_s(i)) = w_t(i)\).

At last we will show that for all \(1 \leq t \leq T + 1\) it holds that \(p'_t = p'_t\). Let \(1 \leq t \leq T + 1\), then from the two previous results we get that \(p'_t = \frac{w'_t / E_t}{\|w'_t / E_t\|_1} = \frac{w'_t / E_t}{\|w'_t / E_t\|_1} = \frac{w'_t / E_t}{\|w'_t / E_t\|_1} = p_t\).
In this part we will prove lemma 2.1.1 (Notice that in the current section we assumed already that $E_{T+1} \neq \emptyset$).

**Lemma 6.0.3.** (lemma 2.1.1)

The following holds: $(1 - \eta) \sum_{t=1}^{T} p_t T v_t \leq \min_{p \in \Delta_m} \sum_{t=1}^{T} p_t T v_t + \frac{\log m}{\eta}$ (the regret bound is exactly the same as in the standard MW algorithm).

**Proof.** The idea of the proof is as follows: we first prove the regret bound for the series $v'$ in which the costs of the "dead" experts are always 1. Then, using the lemma 6.0.2 we show the bounds also holds for the original series.

Define the series $v'_t = \left\{ \begin{array}{ll} v_t(i) & i \in E_t \\
1 & i \notin E_t \end{array} \right.$

Let $E'_t, w'_t, p'_t$ be the corresponding MW variables for the series $v'_t$. Lemma 6.0.2 states that $E'_t = E_t, w'_t|_{E_t} = w_t|_{E_t}, p'_t = p_t$ but notice that $w'_t$ could be different than $w_t$.

Denote by $\bar{p}'_t = \frac{|w'_t|_1}{|w|_1}$ and notice that $\{\bar{p}'_t\}_{t=1}^{T}$ would be the output of the vanilla MW algorithm on $v'_t$. Notice that from the definition of $v'_t$ and remark 6.0.1 (hereafter) $\forall t : \langle \bar{p}'_t, v'_t \rangle \leq \langle \bar{p}'_t, v'_t \rangle$ (6.1)

Also notice that since $\forall t : \forall i \notin E_t : p_t(i) = p'_t(i) = 0$ and $v'_t$ differs from $v_t$ only in indices $i \notin E_t$ we get that $p'_t v'_t = p_t v_t$ (6.2)

Now,

$$\sum_{t=1}^{T} p_t T v_t = \sum_{t=1}^{T} p_t T v'_t$$

$$= \sum_{t=1}^{T} p'_t T v'_t$$

$$\leq \sum_{t=1}^{T} \langle \bar{p}'_t, v'_t \rangle$$

$$\leq (1 + \eta) \min_{p \in \Delta_m} \sum_{t=1}^{T} p T v'_t + \frac{\log m}{\eta}$$

vanilla MW regret bound

$$= (1 + \eta) \min_{p \in \Delta_m} \sum_{t=1}^{T} p T v_t + \frac{\log m}{\eta}$$

remark 6.0.2

\[ \square \]

**Remark 6.0.1.** Let $w \in (0, \infty)^m, v \in [0, 1]^m$ and $\emptyset \neq E \subset [m]$. Assume that $\forall i \in [m] \setminus E : v(i) = 1$ and define $p = \frac{w|_E}{|w|_1}$ and $\bar{p} = \frac{w}{|w|_1}$. Then: $x^T v \leq \bar{p}^T v$. 

36
Proof. Define $\Delta p = p - \bar{p}$ and notice that:
\[
\sum_{i=1}^{m} \Delta p(i) = 0 \quad (6.3)
\]
\[
\forall i \in E : \Delta p(i) \geq 0 \quad (6.4)
\]
Now,
\[
x^Tv = (\bar{p} + \Delta p)^Tv
\]
\[
= \bar{p}^Tv + \sum_{i \in E} \Delta p(i)v(i)
\]
\[
= \bar{p}^Tv + \sum_{i \in E} \Delta p(i)v(i) + \sum_{i \in [m] \setminus E} \Delta p(i)v(i)
\]
\[
= \bar{p}^Tv + \sum_{i \in E} \Delta p(i)v(i) + \sum_{i \in [m] \setminus E} \Delta p(i) \cdot 1 \quad \text{assumption on } v
\]
\[
= \bar{p}^Tv + \sum_{i \in E} \Delta p(i)(v(i) - 1)
\]
\[
\leq \bar{p}^Tv \quad \text{6.4 and the domain of } v
\]
\[\square\]

Remark 6.0.2. Let $\{v'_t\}_{t=1}^{T}$ be another series of non-negative cost vectors, and assume that $\forall \forall i \in E_t : v'_t(i) = v_t(i)$ and that $E_{T+1} \neq \emptyset$. Then:
\[
\min_{p \in \Delta_m} \sum_{t=1}^{T} p^Tv_t = \min_{p \in \Delta_m} \sum_{t=1}^{T} p^Tv'_t
\]

Proof. The idea of the proof is that the minimum is achieved by an expert that is “alive” at the end of the run. First we will show that $\forall i \in [m] \setminus E_{T+1} : \sum_{t=1}^{T} v'_t(i) \geq K$. Let $i \in [m] \setminus E_{T+1}$. There exists some $1 \leq s < T+1$ for which $i \in E_s$ and $i \notin E_{s+1}$. From the monotonicity of $E_s$, for all $1 \leq s \leq s$ it holds that $i \in E_s$ and so by non-negativity of $v'_t$ we get:
\[
\sum_{1 \leq s \leq T} v'_s(i) \geq \sum_{1 \leq s \leq s} v'_s(i) = \sum_{1 \leq s \leq s} v_s(i) \geq K
\]
(because $i \notin E_{s+1}$). Second, by definition of $E_{T+1}$ we get that $\forall i \in [m] \setminus E_{T+1} : \sum_{t=1}^{T} v_t(i) \geq K$.
Third, on the other hand since $E_{T+1} \neq \emptyset$, by definition there exists $i \in E_{T+1}$ for which $\sum_{t=1}^{T} v'_t(i) = \sum_{t=1}^{T} v_t(i) < K$, and so $\min_{i \in E_{T+1}} \sum_{t=1}^{T} v_t(i) < K \leq \min_{i \in [m]} \sum_{t=1}^{T} v_t(i)$ and so $\min_{i \in E_{T+1}} \sum_{t=1}^{T} v_t(i) = \min_{i \in [m]} \sum_{t=1}^{T} v_t(i)$ and in a similar way $\min_{i \in [m]} \sum_{t=1}^{T} v'_t(i) = \min_{i \in E_{T+1}} \sum_{t=1}^{T} v'_t(i)$. Now notice that from the definition of $v'_t$ we get that $\forall i \in E_{T+1} : \sum_{t=1}^{T} v_t(i) = \sum_{t=1}^{T} v'_t(i)$ and so $\min_{i \in E_{T+1}} \sum_{t=1}^{T} v_t(i) = \min_{i \in E_{T+1}} \sum_{t=1}^{T} v'_t(i)$. Combining all the
results we get:

\[
\min_{p \in \Delta_m} \sum_{t=1}^{T} p^T v_t = \min_{i \in [m]} \sum_{t=1}^{T} v_t(i) \\
= \min_{i \in \mathcal{E}_{T+1}} \sum_{t=1}^{T} v_t(i) \\
= \min_{i \in [m]} \sum_{t=1}^{T} v'_t(i) \\
= \min_{p \in \Delta_m} \sum_{t=1}^{T} p^T v_t
\]

6.0.2 A More Detailed Analysis of the Recursive $\ell_1$-Sampling

This section contains a more detailed description and analysis of the recursive $\ell_1$-sampling technique.

We define the $\ell_1$-sampling of a non-negative vector $u \in \mathbb{R}^m$ to be the random variable $v \in \mathbb{R}^m$ which has the following distribution: $\forall i \in [m]: v = e_i \cdot \|u\|_1$ w.p. $\frac{u_i}{\|u\|_1}$.

Suppose we are given some explicit linear function $h(x) : \mathcal{K} \rightarrow \mathbb{R}^m_+$, a series of inputs $x_1, x_2, \ldots, x_T$, and we would like to generate a corresponding series of $\ell_1$-samplings of $h(x_t)$ for $t = 1, 2, \ldots, T$. Naively this would take $O(Tnm)$ operations since for each $x_t$ we need to calculate $h(x_t)$ in order to generate its $\ell_1$-sampling, and this costs $O(nm)$ operations per input. In this section we will show that by using preprocessing we can generate the desired series of $\ell_1$-samplings using only a total of $O(nm + Tn \log m)$ operations.

In order to describe the algorithm we first introduce a few definitions.

Definitions:

- For a range of $s$ numbers $I = \{k, k+1, \ldots, k+s-1\}$, we will define Split$(I)$ to be the two ranges of numbers obtained by splitting $I$ “in the middle” - $\{k, \ldots, k + \lceil \frac{s}{2} \rceil - 1\}$ and $\{k + \lceil \frac{s}{2} \rceil, \ldots, k + s - 1\}$ (When $I$ consists of only one number Split$(I) = I$).

- We will define $\mathcal{I}$ to be all the ranges we get by applying the Split function on $[m]$ and continuing recursively. Formally, denote the set $U_m = \{\{1\}, \{2\}, \ldots, \{m\}\}$, and construct $\mathcal{I}$ in the following way:

  1. Initialize: $\mathcal{I}_1 = \{\{m\}\}, i = 1$.
  2. While $\mathcal{I}_i \neq U_m$ do: $\mathcal{I}_{i+1} \leftarrow \{\text{Split}(I) | I \in \mathcal{I}_i\}, i \leftarrow i + 1$
3. \( I = \bigcup_{j=1}^{i} I_j \)

Notice that the number of elements in \( I \) is bounded by \( 2m \).

- For a set of numbers \( I \subseteq [m] \) we will define the function \( h_I(x) = \sum_{i \in I} h_i(x) \)

The following lemmas are given without proofs.

**Lemma 6.0.4.** Consider the construction of \( I \) given above, then \( i \leq \lceil \log_2 m \rceil + 1 \leq \log_2 4m \).

**Lemma 6.0.5.** Consider the construction of \( I \) given above, then \( \forall 1 \leq j \leq i : |I_j| \leq 2^{j-1} \).

**Algorithm 14** Pre-Processing for Recursive \( \ell_1 \) Sampling

1. Input: \( h_j \) for \( j \in [m], \mathcal{I} \).
2. Calculate functions \( h_I \) for all \( I \in \mathcal{I} \)

We give a reminder of the the Recursive \( \ell_1 \) Sampling algorithm (algorithm 12).

**Algorithm 15** Recursive \( \ell_1 \) Sampling

1. Preprocessed: \( h_I \) for all \( I \in \mathcal{I} \).
2. Input: \( x_t \)
3. Initialize: \( I = [m] \)
4. while \( |I| > 1 \) do
5. \( I_1, I_2 \leftarrow \text{Split}(I) \)
6. \( \alpha_{I_1} \leftarrow h_{I_1}(x_t), \alpha_{I_2} \leftarrow h_{I_2}(x_t), \alpha_I \leftarrow \alpha_{I_1} + \alpha_{I_2} \)
7. Randomly update \( I \) in the following way: \( I \leftarrow \begin{cases} I_1 & \text{w.p. } \frac{\alpha_{I_1}}{\alpha_I} \\ I_2 & \text{w.p. } \frac{\alpha_{I_2}}{\alpha_I} \end{cases} \)
8. end while
9. Return \( v_t = e_i h_{[m]}(x_t) = e_i |h(x_t)|_1 \), for \( i \) such that \( I = \{i\} \).

Notice that since \( h_{I_1}(x_t) + h_{I_2}(x_t) = h_I(x_t) \) we get that each iteration \( \alpha_I = h_I(x_t) \).

**Lemma 6.0.6.** (Number of Iterations in the Recursive \( \ell_1 \)-Sampling) For a given \( x_t \), algorithm 12 runs \( O(\log m) \) iterations.

**Proof.** Notice that for all \( 1 \leq j \), at the beginning of the \( j \)-th iteration \( I \) is one of the sets in \( I_j \). According to lemma 6.0.4 \( j \) is at most \( \log_2 4m \) when \( I_j = U_m \). This means that after at most \( \log_2 4m \) iterations \( I \) will have exactly one element and the algorithm will end.

**Lemma 6.0.7.** (Preprocessing Running Time) The Preprocessing can be implemented using \( O(m) \) summations of functions of the form \( \sum_{i \in I} h_i \) for some \( I \subseteq [m] \).

39
Proof. Consider the construction of $I$ described above. We would like to calculate $\{h_I | \forall I \in I\}$ in the following order: first we will calculate $\{h_I | \forall I \in I_t\}$, then calculate $\{h_I | \forall I \in I_{t-1}\}$ and so on, until we calculate $\{h_I | \forall I \in I_1\}$. Since $I_t = U_m$, the set $\{h_I | \forall I \in I_t\}$ is exactly the input given. Now, using Lemma 6.0.5 we have that for all $2 \leq j \leq i$ on the one hand $|I_j - I_{j-1}| \leq 2^{i-2}$, and on the other hand each function in $\{h_I | \forall I \in I_j\}$ is the sum of at most two functions from $\{h_I | \forall I \in I_{j-1}\}$. And so we will need to perform at most $\sum_{j=1}^{i} 2^{j-1} \leq 2^i \leq 2\log_2 4m = 4m$ summations of functions.

\[\text{Lemma 6.0.8. (Running Time) Suppose } K \subset R^n, \text{ and } h_i \text{ are linear functions given as vectors of size } l, \text{ then the preprocessing in Algorithm 14 can be carried out in } O(lm) \text{ operations, and given the preprocessing, every time Algorithm 12 is carried out, it takes } O(l \log m) \text{ operations.}\]

Proof. Since each linear function is represented as an $l$-dimensional vector, a summation of two linear functions takes $O(l)$ operations, and so according to Lemma 6.0.7 the preprocessing time can be implemented in $O(lm)$ operations. Given the preprocessing, and given some input vector, every iteration of Algorithm 12 will take $O(l)$ operations, and since we have from Lemma 6.0.6 that the number of iterations is $O(\log m)$, we get that the total running time of Algorithm 15 for this input is $O(l \log m)$. 

\[\text{Recursive } \ell_1 \text{ sampling for time-varying functions: updating the recursive } \ell_1 \text{ data structure}\]

Notice that in the case of generalised covering, Algorithm 11 performs $\ell_1$-sampling on the set of functions $f^0_{E_t}$ which changes as $E_t$ changes. This implies that we should change the data structure created by the preprocessing when the set of functions changes. Updating the data structure to correspond to $f^0_{E_{t+1}}$ based on the current state of the data structure which corresponds to $f^0_{E_t}$ is done using the following algorithm.

\[\text{Algorithm 16 Recursive } \ell_1 \text{ Sampling Data Structure Update}\]

1: Input: $(f^0_{E_t})_I$ for all $I \in I, \Delta E = E_t \setminus E_{t+1}$
2: for $j \in \Delta E$ do
3: for $I \in I$ s.t. $j \in I$ do
4: $(f^0_{E_{t+1}})_I \leftarrow (f^0_{E_t})_I - f_j$
5: end for
6: end for

\[\text{Lemma 6.0.9. (Running time of Recursive } \ell_1 \text{ -sampling data structure update) Suppose } K \subset R^n, \text{ and } f_i \text{ are linear functions given as vectors of size } l, \text{ then the Recursive } \ell_1 \text{-sampling data structure update can be implemented using } O(|\Delta E| \cdot n \log m) \text{ operations.}\]
Proof. Notice that according to the way \( \mathcal{I} \) was built that for any \( j \in [m] \) there can be at most \( i \) (the same \( i \) obtained after the construction of \( \mathcal{I} \)) different sets \( I \) such that \( j \in I \). From lemma 6.0.4 we get that \( i \leq \log_2 4m \) and so the number of iterations in the for loop is bounded by \( \log_2 4m \).

Lemma 6.0.10. (Recursive \( \ell_1 \)-sampling data structure update correctness of one step) Assume that for the input of the Update Algorithm \( 16 \) it holds that \( \forall I \in \mathcal{I} : f_I = (f|_{E_t})_I \) and also that \( \Delta E = E_t \setminus E_{t+1} \). Then at the end of the algorithm \( \forall I \in \mathcal{I} : f_I = (f|_{E_{t+1}})_I \).

Remark 6.0.3. (Recursive \( \ell_1 \)-sampling data structure update correctness) For the sped-up framework for covering (Algorithm 11), for all \( 0 \leq t < T \), after updating the data structure according to \( E_t \setminus E_{t+1} \), the data structure contains the data for the functions \( f|_{E_{t+1}} \).

Proof. This can be proven by induction using Lemma 6.0.10.

6.0.3 \( \ell_1 \)-Sampling and Amortised Complexity

This section addresses only the covering algorithm, since the packing case is simpler. When using the \( \ell_1 \) sampling, the actual gradients we are using, \( v_t \), contain only one non-zero entry (in the covering case this entry is also a “live” expert). The main advantage of this is that the changes between iterations are small, and by remembering variables from previous iterations, the run time complexity of an iteration can be reduced. In this section we describe how to do this, and in particular how to compute the function \( p^T f \) using \( O(1) \) summations of functions, how to implement each MW\(_{2^K}\) step in \( O(1) \) operations, and how to check the while condition in \( O(1) \) operations.

Remark 6.0.4. \( \forall t : |J_t \setminus J_{t+1}| \leq 1 \)

Efficient computation of \( p^T f(x) \) using information from the previous iteration

Notations

- Let \( w_t \) and \( p_t \) be as defined in the MW\(_K\) Algorithm (algorithm 4).
- For all \( t \) define \( i_t \) to be the index for which \( v_t|_{J_t} = e_{i_t} \) (according to remark 6.0.9).
- For a distribution \( q \in \Delta_n \) let us denote \( f_q(x) = q^T f(x) \).
- For ease of notation let us denote \( w'_t = w_t|_{J_t} \).

Remark 6.0.5. \( \forall t \forall i : i_t \neq i_t : w'_t(i) = w'_{t+1}(i) \).

\[
\begin{align*}
    w'_t(i_t) - w'_t(i_t) &= \|w'_{t+1}\|_1 - \|w'_t\|_1. \\
w'_{t+1} &= w'_t + (\|w'_{t+1}\|_1 - \|w'_t\|_1) e_{i_t}.
\end{align*}
\]

Lemma 6.0.11. (Correctness of Efficient-\( f_p \)) \( u_t \) returned by algorithm 17 equals to \( f_p \).
**Algorithm 17** Efficient calculation of $p_t^T f(x)$ for MW$_K$ given $p_{t-1}^T f(x)$

1: Input: $f_{p_{t-1}}(x), f_{i_{t-1}}(x), \|w_{t-1}'\|_1, \|w_t'\|_1$
2: $\beta_t \leftarrow \frac{\|w_{t-1}'\|_1}{\|w_t'\|_1}$
3: $u_t \leftarrow \beta_t f_{p_{t-1}} + (1 - \beta_t) f_{i_{t-1}}$
4: Return $u_t$.

**Proof.**

$$u_t = \beta_t f_{p_{t-1}} + (1 - \beta_t) f_{i_{t-1}} \quad \text{by definition of } u_t \tag{6.5}$$
$$= (\beta_t p_{t-1} + (1 - \beta_t) e_{i_{t-1}})^T f \quad \text{definitions of } f_{p_{t-1}}, f_{i_{t-1}} \tag{6.6}$$
$$= \left(\frac{w_{t-1}' \cdot w_{t-1}'}{\|w_t'\|_1} + (1 - \|w_{t-1}'\|_1) e_{i_{t-1}}\right)^T f \quad \text{definitions of } \beta_t \text{ and definition of } p_{t-1} \text{ according to MW}_K \tag{6.7}$$
$$= \left(\frac{w_{t-1}' + (\|w_{t-1}'\|_1 - \|w_t'\|_1) e_{i_{t-1}}}{\|w_t'\|_1}\right)^T f \quad \text{rearranging} \tag{6.8}$$
$$= \frac{w_t'}{\|w_t'\|_1}^T f \quad \text{remark 6.0.5} \tag{6.9}$$
$$= p_t^T f = f_{p_t} \quad \text{definition of } p_t \text{ according to MW}_K \tag{6.10}$$

Notice that Algorithm 17 can be carried out using 2 multiplications of functions of the form $f_q$, (for some $q \in \Delta_n$) by a scalar, one summation of functions of the form $\alpha f_q$, (for some $q \in \Delta_n, \alpha \in \mathbb{R}$), and $O(1)$ other operations.

**Remark 6.0.6.** (Running time of Efficient-$f_{p_t}$ when $f_i$ are explicit linear functions) Suppose $K \subset \mathbb{R}^l$, and $f_i$ are linear functions given as vectors of size $l$, then algorithm 17 can be carried out in $O(l)$ operations.

**Efficient Implementation of the MW$_K$ Step**

The following efficient implementation of MW$_K$ relies on the fact that the costs vectors are actually unit vectors (remark 6.0.9), and also on the observation that in order to carry out an iteration all that is needed from the MW$_K$ are the values of $\|w_t'\|_1$ for the Efficient-$f_{p_t}$ (algorithm 17), and $\Delta J_t$ for updating the data structure for the recursive $\ell_1$-sampling (algorithm 16).
Algorithm 18 Efficient-MW$_K$

1: Input: $\eta$
2: Set $M = 0_n, w' = 1_n, |w'_1|_1 = n, \Delta J_1 = \emptyset$
3: for $t = 1, 2, ..., T$ do
4:   Inform of $|w'_t|_1, \Delta J_t$
5:   Incur gains vector $v_t = e_{i_t}$ in the form $(i_t)$.
6:   $M(i_t) \leftarrow M(i_{t-1}) + 1$
7:   if $M(i_t) \geq K$ then
8:      $\|w'_{t+1}\|_1 \leftarrow \|w'_t\|_1 - w'(i_t)$
9:      $w'(i_t) \leftarrow 0$
10:     $\Delta J_{t+1} \leftarrow \{i_t\}$
11:   else
12:      $\|w'_{t+1}\|_1 \leftarrow \|w'_t\|_1 - w'(i_t)\eta$
13:      $w'(i_t) \leftarrow w'(i_t)(1 - \eta)$
14:     $\Delta J_{t+1} \leftarrow \emptyset$
15:   end if
16: end for

Remark 6.0.7. (Correctness of Efficient implementation of MW$_K$) Assume that $\forall t \in [T]: v_t = e_{i_t}$ for some $i_t \in J_t$, then for all $t \in [T]$, $w', |w'_1|_1$ and $\Delta J_t$ at the beginning of the $t$-th iteration in algorithm 18 equal $w'_t, |w'_1|_1$ and $J_{t-1}\setminus J_t$ obtained by the original MW$_K$ algorithm with input $\{v_t\}_t^{T=1}$.

The above remark can be proven by induction on $t$.

Remark 6.0.8. (Running time of Efficient implementation of MW$_K$) Each iteration in algorithm 18 takes $O(1)$ operations.

Efficient Checking of the While Condition

Lemma 6.0.12. Checking whether the loop condition $(\min_i \sum_{s=1}^t v_s(i) < K)$ is satisfied or not can be carried out using $O(1)$ operations each iteration.

Proof. Notice that $\forall t: v_t = e_{i_t}$ The checking can be done using algorithm 19

$\square$

6.0.4 Analysis of the Sped-up Framework for Covering

Denote by $T$ the value of $t$ at the end of the algorithm. Notice that $T$ is a random variable.

Remark 6.0.9. For all $t$: $v_t = v^0_{E_t} = e_{i_t}$ for some $i_t \in E_t$.

The following remark’s correctness stems from the “expert throwing” in the MW$_U$ algorithm.

Remark 6.0.10. $\forall i \in [m]: \sum_{t=1}^T v^0_{E_t}(i) \leq 2K$. 

43
Algorithm 19 Efficient Loop Condition Check

1: Input: \( n, K \).
2: \( V \leftarrow 0_n \), counter \( \leftarrow 0 \).
3: while 1 do
4: \( \text{Incur} \; v_t = e_i \).
5: \( V(i) \leftarrow V(i) + 1 \).
6: if \( V(i) = K \) then
7: counter \( \leftarrow \) counter + 1
8: if counter \( = n \) then
9: Announce FALSE, end.
10: end if
11: end if
12: Announce TRUE
13: end while

Lemma 6.0.13. \( T \leq 2Km \).

Proof.

\[
T = \sum_{t=1}^{T} \sum_{i=1}^{m} v_t^0 E_t(i) \quad \text{(6.0.9)}
\]

\[
= \sum_{i=1}^{m} \sum_{t=1}^{T} v_t^0 E_t(i) \leq 2Km \quad \text{(6.0.10)}
\]

The following lemmas give a bound on the distortion created by the sampling, in a multiplicative sense, and their proofs rely on a variation of a Multiplicative-Azuma-like inequality given stated in \( \text{(6.0.17)} \) (based entirely on Lemma 10 from [15]).

Lemma 6.0.14. \( (1 - \epsilon) \sum_{t=1}^{T} p_t^T \omega_t(f(x_t))_E^0 \leq \sum_{t=1}^{T} p_t^T v_t + \frac{3}{\epsilon} \) w.p at least \( 1 - e^{-3} \)

Proof. Define the two sequences of non-negative random variables \( z_t = p_t^T \omega_t(f(x_t))_E^0 \) and \( y_t = p_t^T v_t \) for \( t = 1, 2, ..., T \), and notice that using Lemma 6.0.13 we get that \( T \) has finite expectation. From the definition of \( v_t \) and \( \omega_t \) we get that \( \forall t : |z_t - y_t| \leq 1 \) and also that \( E \left[ z_t - y_t \sum_{s < t} z_s, \sum_{s < t} y_s \right] \leq 0 \).
(Notice that \( \omega_t \) does not depend on the sampling result for \( v_t \)). Applying Lemma 6.0.17 with \( \gamma = \epsilon \), \( K = 1 \), and \( A = k/\epsilon \) we get the result.

Lemma 6.0.15. \( (1 - \epsilon) \min_{p \in \Delta_m} \sum_{t=1}^{T} p_t^T v_t \leq \min_{p \in \Delta_m} \sum_{t=1}^{T} p_t^T \omega_t(f(x_t))_E^0 \) + \( \frac{3 \log m}{\epsilon} \) w.p at least \( 1 - e^{-3} \)
Proof. The idea of the proof is to apply Lemma \ref{lem:multi} for each coordinate $i$ separately and then use the union bound to achieve the result. Let $i \in [m]$, and define the two sequences of non-negative random variables $z_t = v_t(i)$, $y_t = (\omega_t f(x_t)|_{E_t})(i)$ for $t = 1, 2, \ldots, T$, and notice that using \ref{lem:finite_avg} we get that $T$ has finite expectation. From the definition of $\omega_t$ and $v_t$ we get that $\forall t: |z_t - y_t| \leq 1$ and also that $E \left[ z_t - y_t \sum_{s < t} z_s, \sum_{s < t} y_s \right] \leq 0$. Applying \ref{lem:multi} with $\gamma = \epsilon$, $K = 1$, and $A = k \log (m)/\epsilon$ we get that

\[(1 - \epsilon) \sum_{t=1}^{T} v_t(i) \leq \sum_{t=1}^{T} (\omega_t f(x_t)|_{E_t})(i) + \frac{k \log (m)}{\epsilon} \quad (6.11)\]

with probability at least $1 - \frac{1}{m} e^{-k}$. Applying the union bound we get that with probability at least $1 - e^{-k}$, \ref{6.11} holds for all $i$. Assume this happens and let $p^* = \arg \min_{p \in \Delta_m} \sum_{t=1}^{T} p^T \omega_t(f(x_t)|_{E_t})$.

Then

\[(1 - \epsilon) \min_{p \in \Delta_m} \sum_{t=1}^{T} p^T v_t \leq (1 - \epsilon) \sum_{t=1}^{T} p^* T v_t \leq \sum_{t=1}^{T} p^* T \omega_t(f(x_t)|_{E_t}) + \frac{k \log (m)}{\epsilon} \quad \text{we assume} \ref{6.11} \text{holds for all } i \in [m] \]

\[= \min_{p \in \Delta_m} \sum_{t=1}^{T} p^T \omega_t(f(x_t)|_{E_t}) + \frac{k \log (m)}{\epsilon} \quad \text{definition of } p^* \]

From the regret bound of the MWU algorithm (theorem \ref{thm:regret_bound}) we get:

\[\text{Lemma 6.0.16.} \quad (1 - \eta) \sum_{t=1}^{T} p_t^T v_t \leq \min_{p \in \Delta_m} \sum_{t=1}^{T} p^T v_t + \frac{\log m}{\eta}. \]

\[\text{Theorem 6.0.1.} \quad \text{Algorithm 11 always ends after at most } O \left( \frac{m \log m}{\epsilon^2} \right) \text{ iterations, and with probability at least } 3/4 \text{ returns a } (1 - O(\epsilon))-\text{multiplicative approximation solution.} \]

Proof. First notice that due to Lemma \ref{lem:finite_avg} and the values of the parameters, the algorithm always ends after at most $18m \log m / \epsilon^2$ iterations. Using the union bound, the events in Lemmas \ref{lem:uniform} and \ref{lem:epsilon} both hold with probability at least $1 - 2e^{-3} \geq 3/4$. In this case:

Rearranging the inequality from Lemma \ref{lem:epsilon} we get:

\[
\frac{\min_{p \in \Delta_m} \sum_{t=1}^{T} p^T \omega_t(f(x_t)|_{E_t})}{\min_{p \in \Delta_m} \sum_{t=1}^{T} p^T v_t} \geq 1 - \epsilon - \frac{k \log m}{\epsilon \cdot \min_{p \in \Delta_m} \sum_{t=1}^{T} p^T v_t} \quad (6.12)
\]
Combining lemmas 6.0.14 and 6.0.16 we get (Notice we use positivity of both sides of the inequality in Lemma 6.0.14):

\[(1 - \eta)(1 - \epsilon) \sum_{t=1}^{T} p_t^T \omega_t (f(x_t)|_{E_t}) \leq (1 - \eta) \sum_{t=1}^{T} p_t^T v_t + (1 - \eta) \frac{k}{\epsilon} \leq \min_{p \in \Delta_m} \sum_{t=1}^{T} p^T v_t + \frac{\log m}{\eta} + (1 - \eta) \frac{k}{\epsilon} \]

(6.13)

Rearranging the two ends of this and plugging the value of \(\eta\) we get (again we assume positivity in order to apply the algebraic manipulations):

\[
\min_{p \in \Delta_m} \sum_{t=1}^{T} p^T v_t \geq \frac{(1 - \epsilon)^2}{1 + \frac{\log m + (1 - \epsilon)k}{\epsilon \cdot \min_{p \in \Delta_m} \sum_{t=1}^{T} p^T v_t}} \geq 1 - 2\epsilon - \frac{\log m + (1 - \epsilon)k}{\epsilon \cdot \min_{p \in \Delta_m} \sum_{t=1}^{T} p^T v_t} \]  

(6.14)

where the right inequality is due to: \(\forall x : \frac{1}{1+x} \geq 1 - x\) and \(\forall x : (1 - x)^2 \geq 1 - 2x\)

Now,

\[
\min_{p \in \Delta_m} \frac{p^T f(\bar{x})}{\lambda^*} \geq \min_{p \in \Delta_m} \frac{T \sum_{t=1}^{T} p_t^T \omega_t f(x_t)}{\lambda^* \sum_{t=1}^{T} \omega_t} \geq \frac{(1 - \epsilon) \min_{p \in \Delta_m} \sum_{t=1}^{T} p^T \omega_t f(x_t)}{\sum_{t=1}^{T} p_t^T \omega_t f(x_t)} \geq \frac{(1 - \epsilon) \min_{p \in \Delta_m} \sum_{t=1}^{T} p^T \omega_t (f(x_t)|_{E_t})}{\sum_{t=1}^{T} p_t^T \omega_t (f(x_t)|_{E_t})} \geq 1 - 4\epsilon - \frac{3k \log m}{\epsilon \cdot \min_{p \in \Delta_m} \sum_{t=1}^{T} p^T v_t} \geq 1 - 5\epsilon \]

using the multiplication of 6.12 and 6.14

choice of \(x_t\)

entry-wise concavity of \(f(x)\)

positiveness of \(f\) and \(\forall t \forall i \notin E_t : p_t(i) = 0\)

stopping condition of algorithm 11

value of \(K\)

\[\Box\]

### 6.0.5 Auxillary Lemma for Multiplicative Approximation

The following lemma is based almost entirely on Lemma 10 from [15]

**Lemma 6.0.17.** Multiplicative Azuma with bound on the variables: Let \(Z = \sum_{t=1}^{T} z_t\), \(Y = \sum_{t=1}^{T} y_t\)
be sums of non-negative random variables where $T$ is a random stopping time with finite expectation, and let $K$ be a positive constant such that for all $t \in [T]$, $|z_t - y_t| \leq K$, and also $E \left[ z_t - y_t \mid \sum_{s < t} z_s, \sum_{s < t} y_s \right] \leq 0$. Let $\gamma \in (0, 1], A \in \mathbb{R}$, then:

$$Pr[(1 - \gamma)Z \geq Y + KA] \leq \exp(-\gamma A)$$

**Proof.** Define $\tilde{Z} = Z/K$, $\tilde{Y} = Y/K$, and for all $t$: $\tilde{z}_t = z_t/K$, $\tilde{y}_t = y_t/K$, and notice that for the new variables we can apply Lemma 10 from [15]. Using the lemma we get that $\exp(-\gamma A) \geq Pr[(1 - \gamma)\tilde{Z} \geq \tilde{Y} + A] = Pr[(1 - \gamma)Z/K \geq Y/K + A] = Pr[(1 - \gamma)Z \geq Y + KA]$. \qed
Bibliography


אלגוריתמים לקבלת קירוב כפל יעד
בעיות כיסוי ואריזה מוכללות

יונתן א. וגנר
אלגוריתמים לקבלת קירוב כפלי עבור בעיות כיסוי ואירוטה מוכללות

יובל א. וגנר

הוגש לסנט הטכניון - מכון טכנולוגי לישראל
תמוז תשע"ו, חיפה, יולי 2016
המחקר נועש בהנחייתו של פרופ' אלעד חזן הפקולטה למדעי המחשב

אני מודה לטכניון על התמיכה הכספית הנדיבה בהשתלמות

אני מודד לטכניון על התמיכה הכספית הנדיבה בהשתלמות

7
In the non-negative Min-Max Problem (Non-negative Min-Max Problem), we are given a function $f$ that depends on two variables, where one variable is convex and the other is concave. The goal of the problem is to find the value of the first variable that minimizes the maximum of the function with respect to the second variable.

Several important classes of the problem mentioned above are Generalized Packing (Generalized Packing) and Generalized Covering (Generalized Covering). These classes include other problems in combinatorial optimization (combinatorial optimization) and machine learning (Machine Learning), including various problems such as $k$-nearest neighbor classification, $k$-nearest neighbor classification (K-nearest neighbor classification), MAXCUT (Maximum Cut), minimum cost multicommodity flow (minimum cost multicommodity flow), and Held-Karp bound for the Traveling Salesman Problem (Held and Karp Bound for the Travelling Salesman Problem).

In general, we can divide combinatorial optimization problems into two main categories: complete solutions and approximate solutions. Complete solutions are usually relevant to problems with fewer constraints but the runtime may be dependent on the magnitude of input parameters, such as gradients. In contrast, approximate solutions are more suitable for problems with non-negative constraints, but in many cases allow algorithms that are independent of the magnitude of input parameters. This property is known as width-free running time. This property can be particularly useful in two cases. First, in the case of large ranges of input parameter values, complete solutions may run in polynomial (usually quadratic) time. In contrast, algorithms that are independent of width are not affected by the width of input parameters. Secondly, if the optimal solution of the approximation is very small, then an approximate solution of 0.5 can provide a much better solution and thus improve the runtime.

Recently, [4] developed a general algorithm for finding complete solutions for a wide range of problems, which is based on the multiplicative weights method (Multiplicative Weights Method). The main contribution of our work is a fast sampling method that can be used to combine with the general frameworks and thus produce very fast algorithms. We illustrate the application of these methods in various fields.

In this work, we present general frameworks for approximate algorithms for Generalized Packing (Generalized Packing) and Generalized Covering (Generalized Covering) problems, which are based on the multiplicative weights method (Multiplicative Weights Method). The main contribution of our work is a fast sampling method that can be used to combine with the general frameworks and thus produce very fast algorithms. We illustrate the application of these methods in various fields.
Normalized Covering Semi-Definite Programming
(Covering Semi-Definite Programming)

For the problem of covering, the normalized covering semi-definite problem (Normalized Covering Semi-Definite Programming) and the non-negative linear classifier (Non-negative Linear Classifier) are sometimes referred to as the maximum-minimum resource sharing problem (Max-min Resource Sharing) or the minimum-maximum resource sharing problem (Min-max Resource Sharing). These problems are sometimes referred to as the max-min resource sharing problem and the min-max resource sharing problem, respectively.

In recent years, various approaches have been developed for solving the covering semi-definite problem. These approaches also have practical applications in covering semi-definite programming (as shown in [1], [3], [16], [11], [10], [5]).

In general, the methods for improving runtime in this work are new as far as we know.

The normalized covering semi-definite programming is a special case of the covering semi-definite programming.

In many cases, when attempting to solve the covering semi-definite problem, it is first transformed into a normalized version and then solved (as shown in [16], [11]).

Several approaches have been developed in recent years for solving the covering semi-definite problem, which also have practical applications in covering semi-definite programming (as shown in [1], [3], [16], [11], [10], [5]).

In general, the methods for improving runtime in this work are new as far as we know.

The normalized covering semi-definite programming is a special case of the covering semi-definite programming.

In many cases, when attempting to solve the covering semi-definite problem, it is first transformed into a normalized version and then solved (as shown in [16], [11]).

Several approaches have been developed in recent years for solving the covering semi-definite problem, which also have practical applications in covering semi-definite programming (as shown in [1], [3], [16], [11], [10], [5]).

In general, the methods for improving runtime in this work are new as far as we know.

The normalized covering semi-definite programming is a special case of the covering semi-definite programming.

In many cases, when attempting to solve the covering semi-definite problem, it is first transformed into a normalized version and then solved (as shown in [16], [11]).

Several approaches have been developed in recent years for solving the covering semi-definite problem, which also have practical applications in covering semi-definite programming (as shown in [1], [3], [16], [11], [10], [5]).

In general, the methods for improving runtime in this work are new as far as we know.

The normalized covering semi-definite programming is a special case of the covering semi-definite programming.

In many cases, when attempting to solve the covering semi-definite problem, it is first transformed into a normalized version and then solved (as shown in [16], [11]).

Several approaches have been developed in recent years for solving the covering semi-definite problem, which also have practical applications in covering semi-definite programming (as shown in [1], [3], [16], [11], [10], [5]).

In general, the methods for improving runtime in this work are new as far as we know.

The normalized covering semi-definite programming is a special case of the covering semi-definite programming.

In many cases, when attempting to solve the covering semi-definite problem, it is first transformed into a normalized version and then solved (as shown in [16], [11]).

Several approaches have been developed in recent years for solving the covering semi-definite problem, which also have practical applications in covering semi-definite programming (as shown in [1], [3], [16], [11], [10], [5]).

In general, the methods for improving runtime in this work are new as far as we know.

The normalized covering semi-definite programming is a special case of the covering semi-definite programming.

In many cases, when attempting to solve the covering semi-definite problem, it is first transformed into a normalized version and then solved (as shown in [16], [11]).

Several approaches have been developed in recent years for solving the covering semi-definite problem, which also have practical applications in covering semi-definite programming (as shown in [1], [3], [16], [11], [10], [5]).

In general, the methods for improving runtime in this work are new as far as we know.

The normalized covering semi-definite programming is a special case of the covering semi-definite programming.

In many cases, when attempting to solve the covering semi-definite problem, it is first transformed into a normalized version and then solved (as shown in [16], [11]).

Several approaches have been developed in recent years for solving the covering semi-definite problem, which also have practical applications in covering semi-definite programming (as shown in [1], [3], [16], [11], [10], [5]).

In general, the methods for improving runtime in this work are new as far as we know.

The normalized covering semi-definite programming is a special case of the covering semi-definite programming.

In many cases, when attempting to solve the covering semi-definite problem, it is first transformed into a normalized version and then solved (as shown in [16], [11]).

Several approaches have been developed in recent years for solving the covering semi-definite problem, which also have practical applications in covering semi-definite programming (as shown in [1], [3], [16], [11], [10], [5]).

In general, the methods for improving runtime in this work are new as far as we know.

The normalized covering semi-definite programming is a special case of the covering semi-definite programming.

In many cases, when attempting to solve the covering semi-definite problem, it is first transformed into a normalized version and then solved (as shown in [16], [11]).

Several approaches have been developed in recent years for solving the covering semi-definite problem, which also have practical applications in covering semi-definite programming (as shown in [1], [3], [16], [11], [10], [5]).

In general, the methods for improving runtime in this work are new as far as we know.

The normalized covering semi-definite programming is a special case of the covering semi-definite programming.

In many cases, when attempting to solve the covering semi-definite problem, it is first transformed into a normalized version and then solved (as shown in [16], [11]).

Several approaches have been developed in recent years for solving the covering semi-definite problem, which also have practical applications in covering semi-definite programming (as shown in [1], [3], [16], [11], [10], [5]).

In general, the methods for improving runtime in this work are new as far as we know.

The normalized covering semi-definite programming is a special case of the covering semi-definite programming.

In many cases, when attempting to solve the covering semi-definite problem, it is first transformed into a normalized version and then solved (as shown in [16], [11]).

Several approaches have been developed in recent years for solving the covering semi-definite problem, which also have practical applications in covering semi-definite programming (as shown in [1], [3], [16], [11], [10], [5]).

In general, the methods for improving runtime in this work are new as far as we know.
The elegant and simple. On the other hand, in the first perspective, the running time of this framework depends on the size of the input in the perspective of the matching size, but slightly less as a function of the approximation.

Moreover, we note that for our proposed framework for the multiubication problem, we use a new variation of the multiplicative weights algorithm (Multiplicative Weights algorithm) in which the "expert" whose cumulative loss reaches some fixed threshold is "released" and from then on, the weight assigned to him in the next iterations will be 0.

We show in our work in which relevant to us, in which all losses are always negative, the algorithm with the change remains within the same complexity bound as the original algorithm.

After presenting the basic frameworks, we present a method to improve the running time of the frameworks for the multiubication and the multiubication problem, for some cases, and especially when the functions in the definition of the problem are linear. The method is based on sampling an l1 vector of the gradient in a negative that is computed efficiently by a recursive-binary approach. The improvement of the running time comes from two directions.

First, the sampling is done with few values of the gradients that are small, and therefore the change between iterations is small, which allows to reduce the complexity of computing the values of the variables by using the values of the variables in the previous iteration. The second direction is the recursive-binary form in which the random sampling of vectors that allow us to compute the gradient values efficiently. The main idea behind the method is that in the case of a collection of vectors and matrices, and we want to get for each vector the l1 norm of its product with the matrix, you can precompute the matrix and use a recursive-binary multiplication, and the running time of the multiplication will depend on the number of rows of the matrix, instead of depending linearly on the naive implementation.

After present the methods to improve the running time of the frameworks, we implement frameworks and methods that we described in order to obtain an algorithm with a nearly linear running time (near-linear running time) for the multiubication problem, and for the non-negative linear classifier.

For these two problems, we use the framework for the multiubication problem with the improved running time for the case where the functions are linear. In the case of the multiubication problem, the domain of the functions is a set of matrices defined as non-negative matrices with a trace of -1, in the case of the non-negative linear classifier, the domain of the functions is a single dimension ball.

For the multiubication problem, we show that we can get an algorithm with a running time nearly linear O~(mn^2/epsilon^2.5), when the size of the input is in general up to mn^2. (The meaning of this expression is that the running time includes terms that are logarithmic in m,n and 1/epsilon).

For the non-negative linear classifier, we show that we can get an algorithm with a running time nearly linear O~(mn/epsilon^2), when the size of the input can reach up to mn.

Finally, we present lower bounds on the algorithms for finding a two-sided approximation for the special cases of the multiubication problem and the multiubication problem with linear functions.

The lower bounds are based on the conjecture that we need a linear running time in the size of the input to determine whether a matrix is such that its rows are random unit vectors or a matrix where all rows except one are random unit vectors and the remaining row is zero.

Technion - Computer Science Department - M.Sc. Thesis 2016-09 - 2016

Technion - Computer Science Department - M.Sc. Thesis 2016-09 - 2016
לסיום, בנובודה זו אנו מציגים מסגרות למジーיק קירוב כפלי לבעיות מינימקס לכלול החלוקה האריתית המוכללת ובפרט. מסגרות אלו מתאימות לאלגוריתמים אופטימליים ידועים המחולקים.

אותות המסגרות מתאימות לברכה חיה של אלגוריתמים המсколькоים הכפליים.

לאחר מכן, אנו מציגים שיטה לשיפור זמן עיבוד המסגרות הכפליים, המתחברת ב

השראת שיטת דגימה אקראית משלהי דגם 1-1Қ-רורסיבב. המקרה העניין שבשיטה זו מועילה הוא המקרה שבו הפונקציות המדורגות זהות את העבירה הוא פונקציה ישרה של התווך או קבוק. על основе לע מתייחסים גם אם השיטות הנפרדות של מסגרת בתחום אישיות המשוואה ואית שים שיטות בעיות

אותות ומגון אלה יש לייצר את אלגוריתמי קירוב ביעילות התכונות הכפליים המוגדים -ϋל ※ יעיבה, המסגרון הליניארי או-שליל. את השיטה של האלגוריתמים המנסים לע מים-לייזר בוונד הקול Почט

שובר ביעילות התכונת הכפליים המוגדים -ϋל אלגוריתמים של זה לאلاقات הקירוב

הכפלי advantageous שוה ביעילות המקרה עליה יאירי-อันตรายוי רוחב. שובר הקירוב הכפלי ביעילות

וב אל מניהан שיאו הנהנהengeance לאטריות שבקול. לזכויות של מרשים ממסגרת התוכן עקר

תמי ריצת של אלגוריתמים קירוב כפליים למקים פרוימ של ביעילות התוכן המוכללת ביעילות המראיה

המוכללת.