Bin packing, number balancing, and rescaling linear programs

Rebecca Hoberg

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2017

Reading Committee:

Thomas Rothvoss, Chair

Rekha Thomas

Sara Billey

Program Authorized to Offer Degree:
Mathematics

University of Washington

Abstract

Bin packing, number balancing, and
rescaling linear programs

Rebecca Hoberg

Chair of the Supervisory Committee:
Assistant Professor Thomas Rothvoss
Mathematics and CSE

This thesis deals with several important algorithmic questions using techniques from diverse areas including discrepancy theory, machine learning and lattice theory.

In Chapter 2, we construct an improved approximation algorithm for a classical NP-complete problem, the *bin packing problem*. In this problem, the goal is to pack items of sizes $s_i \in [0, 1]$ into as few bins as possible, where a set of items fits into a bin provided the sum of the item sizes is at most one. We give a polynomial-time rounding scheme for a standard linear programming relaxation of the problem, yielding a packing that uses at most $OPT + O(\log OPT)$ bins. This makes progress towards one of the "10 open problems in approximation algorithms" stated in the book of Shmoys and Williamson [WS11]. In fact, based on related combinatorial lower bounds, Rothvoss conjectures that $\Theta(\log OPT)$ may be a tight bound on the additive integrality gap of this LP relaxation.

In Chapter 3, we give a new polynomial-time algorithm for linear programming. Our algorithm is based on the *multiplicative weights update* (MWU) method, which is a general framework that is currently of great interest in theoretical computer science. An algorithm for linear programming based on MWU was known previously, but was not polynomial time – we remedy this by alternating between a MWU phase and a rescaling phase. The rescaling methods we introduce improve upon previous methods such as those of [PS16] by reducing

the number of iterations needed until one can rescale[1], and they can be used for any algorithm with a similar rescaling structure. Finally, we note that the MWU phase of the algorithm has a simple interpretation as gradient descent of a particular potential function, and we show we can speed up this phase by walking in a direction that decreases both the potential function and its gradient.

In Chapter 4, we show that an approximate oracle for *Minkowski's Theorem* gives an approximate oracle for the *number balancing problem*, and conversely. Number balancing is the problem of minimizing $|\langle a, x\rangle|$ over $x \in \{-1, 0, 1\}^n \setminus \{\mathbf{0}\}$, given $a \in [0, 1]^n$. While an application of the pigeonhole principle shows that there always exists x with $|\langle a, x\rangle| \leq O(\frac{\sqrt{n}}{2^n})$, the best known algorithm only guarantees $|\langle a, x\rangle| \leq n^{-\Theta(\log n)}$ [KK82a]. We show that an oracle for Minkowski's Theorem with approximation factor ρ would give an algorithm for NBP that guarantees $|\langle a, x\rangle| \leq 2^{-n^{\Theta(1/\rho)}}$. In particular, this would beat the bound of [KK82a] provided $\rho \leq O(\log n / \log\log n)$. In the other direction, we prove that any polynomial time algorithm for NBP that guarantees a solution of difference at most $2^{\sqrt{n}}/2^n$ would give a polynomial approximation for Minkowski as well as a polynomial factor approximation algorithm for the *Shortest Vector Problem*.

[1]This rescaling was discovered in a parallel and independent work by Dadush, Végh and Zambelli [DVZ16].

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I would especially like to thank my adviser, Thomas Rothvoss, who provided me with great projects to work on and was very involved and helpful throughout. In many ways he led by example, and working with him helped shape the ways I approach problems and communicate my work.

I would also like to thank the many others who have taught, encouraged, and inspired me in my mathematical career. In particular, I would like to thank Anna Karlin, Rekha Thomas, Sara Billey, and Eric Bohn.

I would like to thank the Department of Mathematics at the University of Washington and the National Science Foundation for their financial support during this project.

Finally, I would like to thank my fellow students at UW, both in the math department and in the CS theory group, for the collaborative atmosphere, the stimulating discussions, and for making my time here so much fun.

DEDICATION

to my family

Chapter 1
INTRODUCTION

This dissertation deals with three important problems in optimization.

In Chapter 2, we give a new approximation algorithm for bin packing that uses at most $OPT + O(\log OPT)$ bins, where OPT is the actual optimal number of bins. This improves over the algorithm of Rothvoss, which had an additive error of $O(\log OPT \cdot \log\log OPT)$ [Rot13]. Chapter 2 is based on a paper coauthored with Thomas Rothvoss which appeared at SODA 2017 [HR17].

In Chapter 3, we show that a classical algorithm for linear programming based on the multiplicative weights update method can be made polynomial-time by using a rescaling such as that used for the *perceptron algorithm*. In addition, we provide more efficient rescaling methods that can be used for linear programming algorithms with a similar structure. Chapter 3 is based on a paper coauthored with Thomas Rothvoss which will appear at IPCO 2017 [HR16].

In Chapter 4, we give polynomial reductions between approximate versions of the *number balancing problem* and *Minkowski's Theorem*. Chapter 4 is based on a paper coauthored with Harish Ramadas, Thomas Rothvoss, and Xin Yang that will appear at IPCO 2017 [HRRY16].

1.1 Bin Packing Approximation

One of the classical combinatorial optimization problems that is studied in computer science is the *bin packing problem*. Bin packing has been studied since the 1950's [Eis57] and was listed as one of the prototypical NP hard problems in [GJ79]. In this problem, we are given b_i items of size $s_i \in [0, 1]$ for $i = 1, \ldots, n$. The goal is to pack the items into as few bins as possible, where a set of items fits into a bin provided the sum of the item sizes is at most one.

Finding an optimal packing is NP-hard in general, but there have been many approximation algorithms over the years. The most recent approaches use an LP rounding procedure. The *Gilmore-Gomory LP relaxation* [Eis57, GG61] is given by

$$\min\left\{\mathbf{1}^T x : Ax \geq b, x \geq \mathbf{0}\right\}, \tag{1.1}$$

where the constraint matrix $A \in \mathbb{Z}_{\geq 0}^{n \times \mathcal{P}}$ has rows indexed by item size, and has a column for each *feasible pattern*. That is, for any column $p \in \mathcal{P}$ we have $\sum_{i=1}^{n} s_i A_{ip} \leq 1$. Notice that x_p gives the (possibly fractional) number of times that solution x uses pattern p. The constraint $Ax \geq b$ says that the number of total slots in x for item i must be at least b_i.

In 1982, Karmarkar and Karp [KK82b] showed that (1.1) could be solved in polynomial time up to an additive error despite there being exponentially many feasible patterns. In addition, they devised a rounding scheme that found an integral packing using at most $OPT + O(\log^2(OPT))$ bins, where OPT is the actual optimal number of bins.

In 2013, Rothvoss was able to reduce this to $OPT + O(\log(OPT) \cdot \log\log(OPT))$ bins [Rot13]. The rounding technique of Rothvoss uses a result from discrepancy theory which allows one to find a more integral point that is not too far away from the initial point.

Theorem 1 (Lovett-Meka '12)**.** *Suppose $x_{start} \in [0,1]^n$ and let $v_1, \ldots, v_m \in \mathbb{R}^n$ be vectors with parameters $\lambda_1, \ldots, \lambda_m \geq 0$ satisfying $\sum_{j=1}^{m} e^{-\lambda_j^2/16} \leq \frac{n}{16}$. Then in randomized polynomial time one can find a vector $x_{end} \in [0,1]^n$ so that $|\langle x_{end} - x_{start}, v_j \rangle| \leq \lambda_j \cdot \|v_j\|_2$ for all $j \in \{1, \ldots, m\}$ and at least half of the entries of x_{end} are in $\{0,1\}$.*

A good choice for the vectors v_j turns out to be sums of rows of the constraint matrix A. To ensure that x does not change too much, we want $\|v_j\|_2$ to not be too large, and so the algorithm of Rothvoss begins each iteration with a preprocessing phase on A. The basic structure of the algorithm is given in Algorithm 1.

Algorithm 1

1. Compute a fractional solution x with $\mathbf{1}^T x \leq OPT$ and $|\text{supp}(x)| \leq n$.

2. For $O(\log n)$ iterations:

   (a) **Preprocessing phase**: Update x so that the Euclidean norm of the rows of A_x are small, where A_x denotes the restriction of A to the columns in $\text{supp}(x)$.

   (b) **Rounding phase**: Define $v_1, ..., v_m$ to be sums of rows of A_x, and apply Lovett-Meka to sparsify x.

3. Use the updated integral x to find a feasible packing.

1.1.1 Our Contribution

We design an approximation algorithm for bin packing that incurs only a constant loss in each of logarithmically many iterations, giving us a total loss of at most $O(\log n)$. By a reduction due to Karmarkar and Karp, this is enough to obtain the desired gap:

Theorem 2. *There exists a randomized polynomial-time algorithm which finds a packing of items into at most* $OPT + O(\log(OPT))$ *bins.*

More precisely, as we go through the algorithm, we update two vectors x and y. The vector y represents how our items are arranged into *containers*, where a container is a multiset of items that must be packed as a unit into patterns. This is similar to the "item gluing" of Rothvoss. The vector x represents the number (potentially fractional) of each pattern our solution uses, where feasible patterns are made up of containers of total size at most one.

We consider the *packing graph* that connects containers of y to all large-enough container slots of x, and we define the *deficiency* of x and y to be the total size of containers of y that cannot be packed into the slots given by x in an optimal (potentially fractional) assignment. We show that our algorithm produces integral x and y with $\mathbf{1}^T x \leq OPT$ and deficiency

$O(\log n)$. The deficiency can be achieved by an integral assignment, and packing the leftover containers greedily uses only $O(\log n)$ additional bins.

(1) *Improved preprocessing phase*: Let A be the constraint matrix only containing columns from $\text{supp}(x)$, and assume the rows of A are sorted by container size. For $\sigma \in 2^{-\mathbb{N}}$, define *size class* σ to be the set of all containers of size between $\frac{1}{2}\sigma$ and σ.
For all size classes σ, starting with the smallest:

(a) **Grouping:** Group the containers and round all containers down to the smallest container in their group. We do this in such a way that the total number of nonzero rows in the size class is at most $m\sigma^{-1/2}$, where $m = |\text{supp}(x)|$. This operation only changes x and the deficiency only increases by $\sigma^{1/2}$.

(b) **Container Reassignment:** For containers that appear more than $2\sigma^{-1/4}$ times within the same pattern, we combine them into larger containers. This operation updates both x and y and increases the deficiency by $O(\sigma^{3/4})$.

(2) *Improved rounding phase*: For an interval I of rows of A, we define the adjusted number of incidences as

$$\tilde{n}(I) = \sum_{C \in I, A_C \neq 0} (\|A_C\|_1 + \sigma^{-1/2}).$$

Let $v_I = \sum_{C \in I} A_C$. Note that a bound on $\tilde{n}(I)$ implies a bound on $\|v_I\|_1$ as well as the number of nonzero rows of I. Together with the bound of $\|A_C\|_\infty \leq 2\sigma^{-1/4}$, this implies $\|v_I\|_2 \leq \sqrt{\|v_I\|_1 \cdot \|v_I\|_\infty} \leq \tilde{n}(I)\sigma^{1/8}$. The preprocessing phase guarantees that the adjusted number of incidences of the entire size class σ is $O(m\sigma^{-1})$ – this allows us to choose intervals I so that v_I satisfy the conditions of Lovett-Meka, but of small enough Euclidean norm that we can guarantee the increase in deficiency in size class σ is at most $\sigma^{1/16}$.

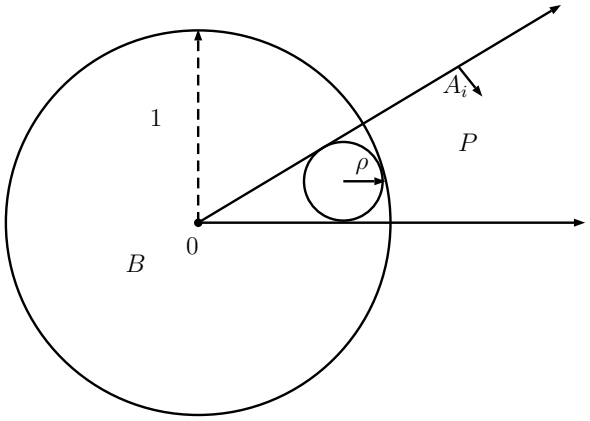


Figure 1.1: Illustration of the polyhedral cone P with radius $\rho > 0$

1.2 A New Algorithm for Linear Programming

One of the central algorithmic problems in theoretical computer science as well as in more practical areas like operations research is finding the solution to a *linear program* (LP)

$$\max\{c^T x \mid Ax \geq b\} \tag{1.2}$$

where $A \in \mathbb{R}^{m \times n}$, $c \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$. Instead of solving (1.2) directly, one can instead find a feasible point in the *open polyhedral cone*

$$P = \{x \in \mathbb{R}^n \mid Ax > \mathbf{0}\} \tag{1.3}$$

where $A \in \mathbb{R}^{m \times n}$ – using standard reductions one can interchange the representations (1.2) and (1.3) with at most a linear overhead. In addition, we may assume that P is full-dimensional. As illustrated in Figure 1.1, let $\rho > 0$ be the radius of the largest ball contained in the intersection of the feasible region P with the unit ball $B := B(\mathbf{0}, 1)$. It turns out that $\frac{1}{\rho}$ may be exponential in the input size, but $\log \frac{1}{\rho}$ is guaranteed to be polynomial.

One way to obtain the $\log \frac{1}{\rho}$ guarantee is by including a rescaling step that increases $\text{vol}(P \cap B)$ by a constant factor. Since $P \cap B$ initially contains a ball of radius ρ, the volume cannot change by more than a factor of ρ^n, and hence there will be at most $O(n \log \frac{1}{\rho})$ rescalings. A general framework is given in Algorithm 2.

Algorithm 2

FOR $\tilde{O}(n \log \frac{1}{\rho})$ phases DO:

(1) **Initial phase:** Either find $x \in P$ or give $\lambda \in \mathbb{R}^m_{\geq 0}$, $\|\lambda\|_1 = 1$ with $\|\lambda A\|_2 \leq \Delta$.

(2) **Rescaling phase:** Find an invertible linear transformation F so that $\text{vol}(F(P) \cap B)$ is a constant fraction larger than $\text{vol}(P \cap B)$. Replace P by $F(P)$.

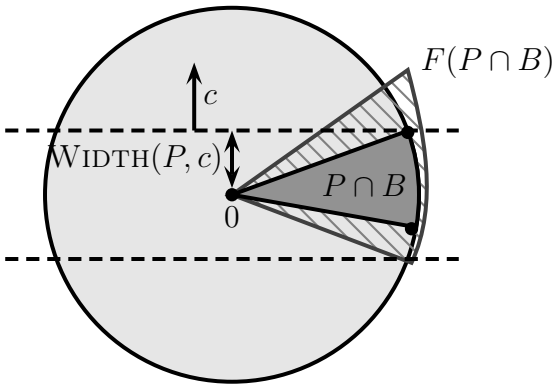


Figure 1.2: If the width of P in direction c is at most $O(\frac{1}{\sqrt{n}})$, then the linear transformation F that stretches in direction c will increase $\text{vol}(P \cap B)$ by a constant factor.

For example, Peña and Soheili showed that a smooth version of the perceptron algorithm implements the initial phase in $\tilde{O}(1/\Delta)$ iterations [PS12]. In addition, they showed that applying the linear transformation $F(x) = x + cc^T x$ to the cone P will increase the volume of $P \cap B$ by a constant factor, provided the *width* of P in direction c is at most $O(\frac{1}{\sqrt{n}})$ (See Figure 1.2.) Finally, they showed that given λ with $\|\lambda\|_1 = 1$ and $\|\lambda A\|_2 \leq \Delta$, one can find a direction in which P has width at most $O(m\Delta)$ – in particular, this shows that $\Delta = O(\frac{1}{m\sqrt{n}})$ suffices to rescale.

A classical algorithm for linear programming is based on the *multiplicative weights update* (MWU) method [AHK12]. Viewed with the right lens, this algorithm consists of performing

gradient descent on the *potential function* $\Phi(x) = \sum_{i=1}^{m} e^{-\langle A_i, x\rangle}$ – notice that as soon as $\Phi(x) < 1$, then $x \in P$. It turns out that MWU terminates in $\tilde{O}(\frac{1}{\rho^2})$ iterations, and hence it need not be polynomial in the input size.

1.2.1 Our Contribution

We demonstrate a new algorithm for linear programming. Our algorithm follows the structure of Algorithm 2, but we give new implementations for each of the phases.

(1) *Initial phase based on MWU*: We show that in $\tilde{O}(1/\Delta^2)$ iterations the MWU method can be made to implement the initial phase of Algorithm 2. The idea is that gradient descent will decrease the potential function Φ by a significant amount provided $\frac{\nabla\Phi}{\Phi}$ is large enough. On the other hand, $\frac{\nabla\Phi}{\Phi}$ takes the form λA, and so if this is small enough then we can rescale. By modifying the gradient descent approach to walk in a direction that decreases both $\Phi(x)$ and $\|\nabla\Phi(x)\|_2$, we speed up this implementation to use only $\tilde{O}(1/\Delta)$ iterations. This essentially matches the number of iterations needed for the initial phase of [PS16].

(2) *Improved rescaling phase*: We design rescaling methods that apply for a parameter of $\Delta = \Theta(\frac{1}{n})$, which improves over the threshold $\Delta = \Theta(\frac{1}{m\sqrt{n}})$ required by [PS16]. This results in a smaller number of iterations that are needed per phase until one can rescale the system. Our first rescaling method uses randomness to find a direction c of width at most $O(\sqrt{n}\Delta)$ – the linear transformation $F(x) = x + cc^T x$ then increases the volume of $P \cap B$ by a constant factor. Our second method is a deterministic and *multirank* rescaling which also guarantees a constant factor increase in the volume – in fact, under certain conditions it will actually increase the volume by up to an exponential factor.

1.3 Number Balancing and Minkowski's Theorem

The *number balancing problem* (NBP) is the following: given real numbers $a_1, \ldots, a_n \in [0, 1]$, find two distinct subsets $I_1, I_2 \subseteq [n]$ so that the difference $|\sum_{i\in I_1} a_i - \sum_{i\in I_2} a_i|$ of their sums

is minimized. Equivalently, given a vector $a \in [0,1]^n$, we want to find a vector of *signs* $x \in \{-1,0,1\}^n \setminus \{\mathbf{0}\}$ to minimize the inner product $|\langle a, x\rangle|$. Notice that NBP is a relaxation of the classical *number partitioning problem*, which requires the two subsets to partition $[n]$ [GJ79].

An application of the pigeonhole principle shows there always exists $x \in \{-1,0,1\}^n \setminus \{\mathbf{0}\}$ with $|\langle a, x\rangle| \leq \frac{n}{2^n-1}$, and with a slightly refined argument we can improve the bound to $O(\frac{\sqrt{n}}{2^n})$. In spite of these strong nonconstructive guarantees, it is less clear how well one can do constructively:

Question 1. Given a vector $a \in [0,1]^n$, for what $\delta > 0$ can one be guaranteed to find an $x \in \{-1,0,1\}^n \setminus \{\mathbf{0}\}$ with $|\langle a, x\rangle| \leq \delta$ in polynomial time?

The best known bound is given by Karmarkar and Karp's *differencing algorithm*, which guarantees $|\langle a, x\rangle| \leq n^{-\Theta(\log n)}$ [KK82a]. On the other hand, it is NP-hard to decide whether there are two disjoint subsets that sum up to the exact same value [WY92].

Another nonconstructive result is *Minkowski's Theorem*, which says that any symmetric convex body $K \subseteq \mathbb{R}^n$ of volume at least 2^n must intersect $\mathbb{Z}^n \setminus \{\mathbf{0}\}$ [Mat02]. This theorem is proved by placing translates of $\frac{1}{2}K$ at any lattice point and then inferring an overlap due to the pigeonhole principle. Again one can consider the algorithmic question:

Question 2. Given a symmetric convex body K with volume at least 2^n, for what factor ρ can one be guaranteed to find an $x \in (\rho K) \cap \mathbb{Z}^n \setminus \{\mathbf{0}\}$ in polynomial time?

This factor ρ is known to be within a polynomial factor of the approximability of the *shortest vector problem* (SVP). In particular, LLL basis reduction yields a polynomial-time algorithm for $\rho \leq 2^{O(n)}$, but a small enough polynomial-factor approximation could be enough to break lattice based cryptosystems.

1.3.1 Our Contribution

In this work, we show that the above questions are related:

(1) *Using a Minkowski oracle*: We note that the body $K := \{x \in (-2, 2) : |\langle a, x\rangle| \leq 2^{-\Theta(n)}\}$ has volume at least 2^n, and therefore an exact oracle for Minkowski's Theorem yields an algorithm to find a number balancing solution with $|\langle a, x\rangle| \leq 2^{-\Theta(n)}$. If we only have an approximate oracle, we can still obtain x with $|\langle a, x\rangle| \leq 2^{-\Theta(n)}$, but in this case $\|x\|_\infty$ may be as large as $\Theta(\rho)$. We introduce a nontrivial *self reduction* that reduces $\|x\|_\infty$ by a factor of 2, and we show that this yields an NBP algorithm that guarantees $|\langle a, x\rangle| \leq 2^{-n^{\Theta(1/\rho)}}$.

(2) *Using a number balancing oracle*: We show that an oracle for number balancing that guarantees $|\langle a, x\rangle| \leq \frac{2\sqrt{n}}{2^n}$ yields an approximate oracle for Minkowski's theorem with approximation factor $\rho = O(n^5)$. To do this, we first use lattice basis reduction to reduce to the case where K is a well-rounded ellipsoid. We demonstrate that an NBP oracle can be extended to multiple vectors without weakening the bound too much – we can then get a stronger bound by allowing $\|x\|_\infty$ to increase. Combining these, we find $x \in \mathbb{Z}^n \setminus \{\mathbf{0}\}$ whose inner product with each of the axis directions is bounded by $O(n^4)$ of the corresponding axis length, yielding an $O(n^5)$ approximation for the original set K.

Chapter 2

A LOGARITHMIC INTEGRALITY GAP FOR BIN PACKING

2.1 Introduction

One of the classical combinatorial optimization problems that is studied in computer science is *Bin Packing*. It appeared as one of the prototypical **NP**-hard problems already in the book of Garey and Johnson [GJ79] but it was studied long before in operations research in the 1950's, for example by [Eis57]. We refer to the survey of Johnson [CGJ84] for a complete historic account.

Bin packing is a good example to study the development of techniques in approximation algorithms as well. The 1970's brought simple greedy heuristics such as *First Fit*, analyzed by Johnson [Joh73] which requires at most $1.7 \cdot OPT + 1$ bins and *First Fit Decreasing* [JDU$^+$74], which yields a solution with at most $\frac{11}{9}OPT + 4$ bins (see [Dós07] for a tight bound of $\frac{11}{9}OPT + \frac{6}{9}$). Later, an *asymptotic PTAS* was developed by Fernandez de la Vega and Lueker [FdlVL81]. One of their main technical contributions was an *item grouping technique* to reduce the number of different item types. The algorithm of Fernandez de la Vega and Lueker finds solutions using at most $(1+\varepsilon)OPT + O(\frac{1}{\varepsilon^2})$ bins, while the running time is either of the form $O(n^{f(\varepsilon)})$ if one uses dynamic programming or of the form $O(n \cdot f(\varepsilon))$ if one applies linear programming techniques.

A big leap forward in approximating bin packing was done by Karmarkar and Karp in 1982 [KK82b]. First of all, they argue how a certain exponential size LP can be approximately solved in polynomial time; secondly they provide a sophisticated rounding scheme which produces a solution with at most $OPT + O(\log^2 OPT)$ bins, corresponding to an *asymptotic FPTAS*.

It will be convenient throughout this chapter to allow a more compact form of input,

where $s \in [0,1]^n$ denotes the vector of different item sizes and $b \in \mathbb{N}^n$ denotes the multiplicity vector, meaning that we have b_i copies of item type i. In this notation we say that $\sum_{i=1}^n b_i$ is the *total number of items*. The linear program that we mentioned earlier is called the *Gilmore-Gomory LP relaxation* [Eis57, GG61] and it is of the form

$$\min \left\{ \mathbf{1}^T x \mid Ax \geq b, x \geq \mathbf{0} \right\}. \tag{2.1}$$

The constraint matrix A consists of all column vectors $p \in \mathbb{Z}_{\geq 0}^n$ that satisfy $\sum_{i=1}^n p_i s_i \leq 1$. The linear program has variables x_p that give the number of bins that should be packed according to the *pattern* p.

We denote the value of the optimal fractional solution to (2.1) by OPT_f, and the value of the best integral solution by OPT. As we mentioned before, the linear program (2.1) does have an exponential number of variables, but only n constraints. A fractional solution x of cost $\mathbf{1}^T x \leq OPT_f + \delta$ can be computed in time polynomial in $\sum_{i=1}^n b_i$ and $1/\delta$ [KK82b] using the Grötschel-Lovasz-Schrijver variant of the Ellipsoid method [GLS81]. An alternative and simpler way to solve the LP approximately is via the Plotkin-Shmoys-Tardos framework [PST95] or the multiplicative weight update method. See the survey of [AHK12] for an overview.

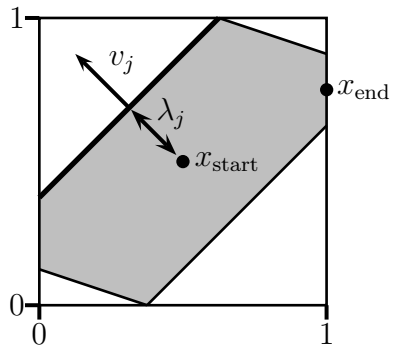
The best known lower bound on the integrality gap of the Gilmore-Gomory LP is an instance where $OPT = \lceil OPT_f \rceil + 1$; Scheithauer and Terno [ST97] conjecture that these instances represent the worst case additive gap. While this conjecture is still open, it is reasonable to expect that the best approximation algorithms will be based on rounding a solution to this amazingly strong Gilmore Gomory LP relaxation. For example, the Karmarkar-Karp algorithm operates in $\log n$ iterations in which one first groups the items such that only $\frac{1}{2}\sum_{i \in [n]} s_i$ many different item sizes remain; then one computes a basic solution x and buys $\lfloor x_p \rfloor$ times pattern p and continues with the residual instance. The analysis provides a $O(\log^2 OPT)$ upper bound on the *additive* integrality gap of (2.1).

The rounding mechanism in the recent paper of Rothvoss [Rot13] uses an algorithm by Lovett and Meka that was originally designed for discrepancy minimization. The Lovett-

Meka algorithm [LM12] can be conveniently summarized as follows:

Theorem 3 (Lovett-Meka '12)**.** *Let $v_1, \ldots, v_m \in \mathbb{R}^n$ be vectors with $x_{start} \in [0,1]^n$ and parameters $\lambda_1, \ldots, \lambda_m \geq 0$ so that $\sum_{j=1}^m e^{-\lambda_j^2/16} \leq \frac{n}{16}$. Then in randomized polynomial time one can find a vector $x_{end} \in [0,1]^n$ so that $|\langle x_{end} - x_{start}, v_j \rangle| \leq \lambda_j \cdot \|v_j\|_2$ for all $j \in \{1, \ldots, m\}$ and at least half of the entries of x_{end} are in $\{0,1\}$.*

Intuitively, the points x_{end} satisfying the linear constraints $|\langle x_{\text{end}} - x_{\text{start}}, v_j \rangle| \leq \lambda_j \cdot \|v_j\|_2$ form a polytope and the distance from the start point to the jth hyperplane is λ_j. The condition $\sum_{j=1}^m e^{-\lambda_j^2/16} \leq \frac{n}{16}$ essentially says that the polytope is going to be "large enough". The algorithm of [LM12] itself consists of a random walk through the polytope. For more details, we refer to the very readable paper of [LM12].



The bin packing approximation algorithm of Rothvoss [Rot13] consists of logarithmically many runs of Lovett-Meka. To be able to use the Lovett-Meka algorithm effectively, Rothvoss needs to rebuild the instance in each iteration and "glue" clusters of small items together to larger items. His procedure is only able to do that for items that have size at most $\frac{1}{\text{polylog}(n)}$ and each of the iterations incurs a loss in the objective function of $O(\log \log n)$. In contrast we present a procedure that can even cluster items together that have size up to $\Omega(1)$. Moreover, Rothvoss' algorithm only uses two types of parameters for the error parameters, namely $\lambda_j \in \{0, O(\sqrt{\log \log n})\}$. In contrast, we use the full spectrum of parameters to achieve only a *constant* loss in each of the logarithmically many iterations.

2.1.1 Our contribution

Our main contribution is the following theorem:

Theorem 4. *For any Bin Packing instance (s, b) with $s_1, \ldots, s_n \in [0, 1]$, one can compute a solution with at most $OPT_f + O(\log OPT_f)$ bins, where OPT_f denotes the optimal value of the Gilmore-Gomory LP relaxation. The algorithm is randomized and the expected running time is polynomial in $\sum_{i=1}^{n} b_i$.*

The recent book of Williamson and Shmoys [WS11] presents a list of 10 open problems in approximation algorithms. Problem #3 in the list is whether the Gilmore-Gomory LP has a constant integrality gap; hence we make progress towards that question.

We want to remark that the original algorithm of Karmarkar and Karp has an additive approximation ratio of $O\big(\log OPT_f \cdot \log\big(\max_{i,j}\big\{\frac{s_i}{s_j}\big\}\big)\big)$. For *3-partition* instances where all item sizes are strictly between $\frac{1}{4}$ and $\frac{1}{2}$, this results in an $O(\log n)$ guarantee, which coincides with the guarantees of Rothvoss [Rot13] and this paper if applied to those instances. A paper of Eisenbrand et al. [EPR13] gives a reduction of those instances to minimizing the discrepancy of 3 permutations. Interestingly, shortly afterwards Newman, Neiman and Nikolov [NNN12] showed that there are instances of 3 permutations that do require a discrepancy of $\Omega(\log n)$. It seems unclear how to realize those permutations with concrete sizes in a bin packing instance — however any further improvement for bin packing even in that special case with item sizes in $\big(\frac{1}{4}, \frac{1}{2}\big)$ would need to rule out such a realization as well. Rothvoss conjectures that the integrality gap for the Gilmore Gomory LP is indeed $\Theta(\log n)$.

2.2 The Packing Graph

It is well-known that for the kind of approximation guarantee that we aim to achieve, one can assume that the items are not too tiny. In fact it suffices to prove an additive gap of $O(\log \max\{n, \frac{1}{s_{\min}}\})$ where n is the number of different item sizes and $s_{\min}$ is a lower bound on all item sizes. Note that in the following, "polynomial time" means always polynomial in the total number of items $\sum_{i=1}^{n} b_i$.

Lemma 5. *Assume for a monotone function g, there is a polynomial time algorithm for bin packing instances (s, b) with $s \in [0,1]^n$ and $s_1, \ldots, s_n \geq s_{\min} > 0$ that finds a solution with at most $OPT_f + g\big(\max\{n, \frac{1}{s_{\min}}\}\big)$ bins. Then there is a polynomial time algorithm that for* all *instances finds a solution with at most $OPT_f + g(OPT_f) + O(\log OPT_f)$ bins.*

Proof. Let (s, b) be any bin packing instance with $s \in [0,1]^n$ and $b \in \mathbb{N}^n$. Let $U := \sum_{i=1}^n b_i s_i$ be the total size. Note that $U \leq OPT_f \leq 2U + 1$, so U is a good estimate on the value of the LP optimum. We split items into *large* ones $L := \{i \in [n] \mid s_i \geq \frac{1}{U}\}$ and *small* ones $S := \{i \in [n] \mid s_i < \frac{1}{U}\}$.

Now, we perform the *geometric grouping* from [KK82b] to the large items as follows: sort items from largest to smallest and form groups of total size between 2 and 3. Then for each group, round all items to the largest item type in its group. This procedure allows to reduce the number of different item types to U while the optimal fractional value increases to at most $OPT'_f \leq OPT_f + O(\log U)$. Now we run the assumed algorithm to assign items in L to at most $OPT'_f + g(U) \leq OPT_f + g(OPT_f) + O(\log U)$ bins. Here we are using that $OPT_f \geq U$ is an upper bound on the number of items in the modified instance and $s_{\min} := \frac{1}{U}$ is a lower bound on the item sizes in L.

Then we "sprinkle" the small items greedily over those bins. If no new bin needs to be opened, we are done. Otherwise, we know that the solution consists of k bins such that $k-1$ bins are at least $1-\frac{1}{U}$ full. This implies $U \geq (k-1)\cdot(1-\frac{1}{U})$, and hence $k \leq U+3 \leq OPT_f+3$ assuming $U \geq 2$. □

As a side remark, the reduction in Lemma 5 will choose $s_{\min} = \Theta\big(\frac{1}{OPT_f}\big)$. For our result, we use $g(k) = \Theta(\log k)$. From now on we assume that we have n different item sizes with all sizes satisfying $s_i \geq s_{\min}$ for some given parameter $s_{\min}$. Starting from a fractional solution x to (2.1) our goal is to find an integral solution of cost $\mathbf{1}^T x + O\big(\log \max\{n, \frac{1}{s_{\min}}\}\big)$. Another useful standard argument is as follows:

Lemma 6. *Any bin packing instance (s, b) can be packed in polynomial time into at most $2\sum_{i=1}^n s_i b_i + 1$ bins.*

Proof. Simply assign the items greedily and open new bins only if necessary. If we end up with k bins, then at least $k-1$ of them are at least half full, which means that $\sum_{i=1}^{n} s_i b_i \geq \frac{1}{2}(k-1)$. Rearranging gives the claim. □

Instead of packing items directly into bins, we are able to use the Lovett-Meka algorithm more effectively by first arranging items into *containers* and then packing containers into bins. For us a container is any vector $C \in \mathbb{Z}^n_{\geq 0}$, i.e. it is a multiset of items with C_i copies of item i. Let $\mathcal{C}$ be the set of all containers of size at most 1, where the *size* $s(C)$ of a container is just the sum of the sizes of its items $s(C) := \sum_{i=1}^{n} C_i s_i$. We let y_C denote the number of copies of container C, and we define the set of possible arrangements of items into containers as

$$\mathrm{Arr}(s, b) := \Big\{ y \in \mathbb{Z}^{\mathcal{C}}_{\geq 0} : \sum_{C \in \mathcal{C}} y_C C_i = b_i \text{ for all items } i \Big\}.$$

As we will pack containers into bins, we want to define a *pattern* as a vector of the form $p \in \mathbb{Z}^{\mathcal{C}}_{\geq 0}$ where p_C denotes the number of times that the pattern contains container C. Of course the sum of the sizes of the containers should be at most 1, thus

$$\mathcal{P} := \Big\{ p \in \mathbb{Z}^{\mathcal{C}}_{\geq 0} \mid \sum_{C \in \mathcal{C}} p_C \cdot s(C) \leq 1 \Big\}$$

is the set of all *(valid) patterns*.

Given a fractional solution $x \in \mathbb{R}^{\mathcal{P}}_{\geq 0}$ and an arrangement of items $y \in \mathrm{Arr}(s, b)$, we define a bipartite graph $G(x, y)$. On the left this graph has nodes $C \in \mathcal{C}$ with multiplicity y_C, corresponding to containers we need to pack. On the right it has nodes $(C, p) \in \mathcal{C} \times \mathcal{P}$ with multiplicity $x_p \cdot p_C$, corresponding to slots for containers in the patterns of our fractional solution. We connect containers to slots in which they fit. Explicitly, the edge set is $E = \{(C, (C', p)) : s(C) \leq s(C')\}$. Our goal will be to find a (fractional) assignment of containers to patterns that packs as many containers as possible.

Consider the example that is visualized in Figure 2.1. The example has $n = 3$ items of size $s = (0.3, 0.2, 0.1)$ and multiplicity vector $b = (2, 3, 4)$. Those items are arranged into containers C_1, C_2, C_3 which have multiplicities $y_{C_1} = y_{C_2} = 1$ and $y_{C_3} = 2$. Moreover, in our

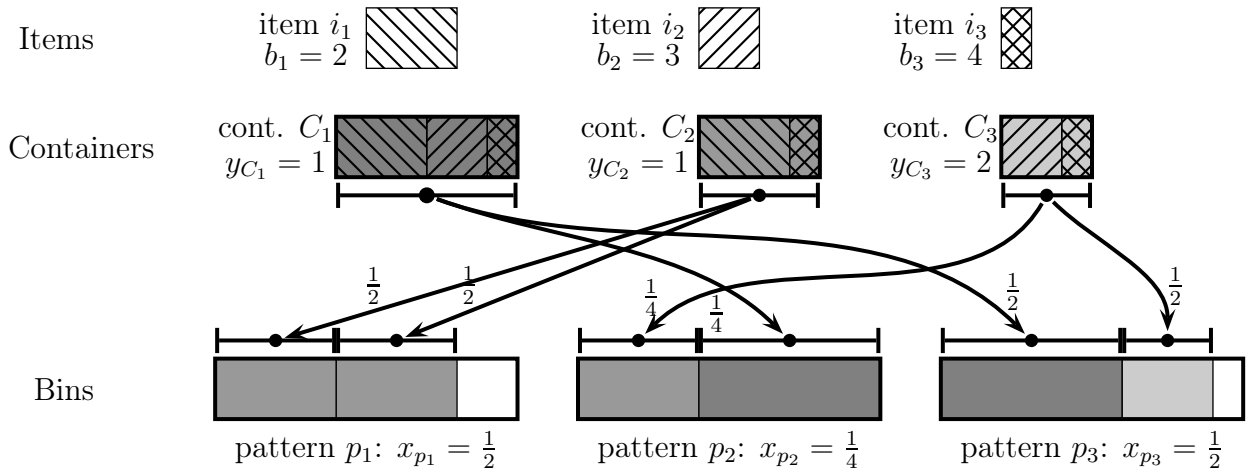


Figure 2.1: Example for assigning containers to patterns

example we have 3 patterns p_1, p_2, p_3 with fractional values $x_{p_1} = x_{p_3} = \frac{1}{2}$ and $x_{p_2} = \frac{1}{4}$. The arrows represent the assignment of containers. For example, container C_1 is assigned with a fractional value of $\frac{1}{4}$ to pattern p_2 and $\frac{1}{2}$ to p_3. Notice that we do allow container C_3 to be assigned to slots of a larger container in p_2. However, we are still unable to pack $\frac{5}{4}$ of container C_3 and $\frac{1}{4}$ of container C_1. Later we will say that the *deficiency* of the graph is $\frac{5}{4} \cdot s(C_3) + \frac{1}{4} \cdot s(C_1)$.

Slightly more generally, we say that a bipartite graph $G = (V_\ell \cup V_r, E)$ is a *packing graph* if each $v \in V_\ell \cup V_r$ has an associated size $s(v) \in [0, 1]$ and multiplicity $\text{mult}(v) \in \mathbb{R}_{\geq 0}$, and the edge set is given by $E = \{(u, v) \in V_\ell \times V_r \mid s(u) \leq s(v)\}$. Therefore we call $G(x, y)$ the *packing graph associated with x and y.*

An *assignment* in a packing graph is a function $a : E \to \mathbb{R}_{\geq 0}$ so that for any $v \in V$, we have $\sum_{e \in \delta(v)} a(e) \leq \text{mult}(v)$, where $\delta(v)$ denotes the set of edges incident to v. The *deficiency* of a packing graph is the total size of left nodes that fail to be packed in an optimal assignment. That is,

$$\text{def}(G) := \min_{a \text{ assignment of } G} \Big\{ \sum_{v \in V_\ell} s(v) \cdot (\text{mult}(v) - a(\delta(v))) \Big\}.$$

We now want to make a couple of observations about general packing graphs. First,

we can define a partial ordering on packing graphs as follows. If $G = (V_\ell \cup V_r, E)$ and $G' = (V'_\ell \cup V'_r, E')$ are packing graphs, we say $G \preceq G'$ if for any $s \geq 0$ we have

$$\sum_{v \in V_\ell, s(v) \geq s} \text{mult}(v) \leq \sum_{v \in V'_\ell, s'(v) \geq s} \text{mult}'(v)$$

and

$$\sum_{v \in V_r, s(v) \geq s} \text{mult}(v) \geq \sum_{v \in V'_r, s'(v) \geq s} \text{mult}'(v).$$

Essentially this says that at any size, G has at most as many nodes to pack and at least as much space to pack them in. In particular, we have the following.

Observation 1. *If G and G' are packing graphs with $G \preceq G'$, then def$(G) \leq$ def(G').*

Proof. Note that the condition on the left nodes allows us to find an assignment a_ℓ from V_ℓ to V'_ℓ with deficiency zero. Similarly, we can find an assignment a_r from V'_r to V_r with deficiency zero. Given any assignment a' from V'_ℓ to V'_r we define an assignment a from V_ℓ to V_r as the composition of a_ℓ, a' and a_r. The total size of left nodes we fail to pack in a is at most that of a', and so we can conclude that $\text{def}(G) \leq \text{def}(G')$. □

The edge set of these graphs is extremely simple, so that one can directly obtain the deficiency as follows.

Observation 2. *For any packing graph, an optimal assignment $a : E \to \mathbb{R}_{\geq 0}$ which attains def(G) can be obtained as follows: go through the nodes $v \in V_r$ in any order. Take the node $u \in V_\ell$ of maximum size that has some capacity left and satisfies $s(u) \leq s(v)$. Increase $a(u, v)$ as much as possible.*

Proof. Choose an arbitrary node $v \in V_r$. Let $\hat{a}$ denote the assignment of left nodes to v where v is packed with maximal-size left nodes, and let a denote an arbitrary assignment of left nodes to v. Let $G_{a,v}$ denote the packing graph with v removed from V_r and the multiplicities of left nodes u decreased by $a(u, v)$. Then notice that $G_{\hat{a},v} \preceq G_{a,v}$. In particular, $\text{def}(G_{\hat{a},v}) \leq \text{def}(G_{a,v})$ by Observation 1, and so we may as well pack v greedily. □

In future sections we further restrict ourselves to *left-integral* packing graphs — that is, for any $v \in V_\ell$, $\text{mult}(v) \in \mathbb{Z}_{\geq 0}$.

We then define the deficiency of the pair (x, y) as the deficiency of the corresponding packing graph $G(x, y)$, that is, $\text{def}(x, y) := \text{def}(G(x, y))$.

We should discuss why the arrangement into the containers and the packing graph is useful. First of all, it is easy to find *some* initial configuration.

Lemma 7. *For any bin packing instance (s, b), one can compute a "starting solution" $x \in \mathbb{R}^{\mathcal{P}}_{\geq 0}$ and $y \in Arr(s, b)$ in polynomial time so that $\mathbf{1}^T x \leq OPT_f + 1$ and $def(x, y) = 0$ with $|supp(x)| \leq n$.*

Proof. As we already argued, one can compute a fractional solution x for (2.1) in polynomial time that has cost $\mathbf{1}^T x \leq OPT_f + 1$. We simply use singleton containers $\{i\}$ for all items $i \in \{1, \ldots, n\}$ and set $y_{\{i\}} := b_i$. □

Next, we argue that our notion of deficiency was actually meaningful in recovering an assignment of items to bins.

Lemma 8. *Suppose that $x \in \mathbb{Z}^{\mathcal{P}}_{\geq 0}, y \in Arr(s, b)$ are both integral. Then there is a packing of all items into at most $\mathbf{1}^T x + 2def(x, y) + 1$ bins.*

Proof. Since x and y are both integral, all multiplicities in $G(x, y)$ will be integral and we can find an integral assignment a attaining $\text{def}(x, y)$. Buy all the patterns suggested by x. Use a to pack the containers in y. There might be containers in y that have not been assigned; their total size is $\text{def}(G(x, y))$. We pack containers greedily into at most $2\text{def}(x, y) + 1$ many extra bins using Lemma 6. □

In each iteration of our algorithm, it will be useful for us to be able fix the integral part of x and focus solely on the fractional part.

Lemma 9. *Suppose $x \in \mathbb{R}^{\mathcal{P}}_{\geq 0}, y \in \mathbb{Z}^{\mathcal{C}}_{\geq 0}$. If $\hat{x}_p = \lfloor x_p \rfloor$ for all patterns p, then there exists $\hat{y} \in \mathbb{Z}^{\mathcal{C}}_{\geq 0}$, so that $def(\hat{x}, \hat{y}) = 0$ and $def(x - \hat{x}, y - \hat{y}) = def(x, y)$.*

Proof. Let us imagine that we replace each node (C, p) in $G(x, y)$ with two copies, a "red" node and a "blue" node. The red copy receives an integral multiplicity of $\text{mult}_{\text{red}}(C, p) = \hat{x}_p p_C$ while the blue copy receives a fractional multiplicity of $\text{mult}_{\text{blue}}(C, p) = (x_p - \hat{x}_p) \cdot p_C$. Now we apply Observation 2 to find the best assignment a. Crucially, we set up the order of the right hand side nodes so that we first process the red integral nodes and then the blue fractional ones. Note that the assignment that this greedy procedure computes is optimal and moreover, the assignments for red nodes will be integral. For each container C on the left, we define $\hat{y}_C$ to be the total red multiplicity of its targets under this optimal assignment. Then $\text{def}(G(\hat{x}, \hat{y})) = 0$ and $\text{def}(G(x - \hat{x}, y - \hat{y})) = \text{def}(G(x, y))$. □

Let $\text{frac}(x) = \{p \in \mathcal{P} : x_p \notin \mathbb{Z}\}$ and $\text{supp}(x) = \{p \in \mathcal{P} : x_p > 0\}$. Now we have enough notation to state our main technical theorem:

Theorem 10. *Let (s, b) be an instance with $s_1, \ldots, s_n \geq s_{\min} > 0$. Let $y \in Arr(s, b)$ and $x \in \mathbb{R}^{\mathcal{P}}_{\geq 0}$ with $|frac(x)| \geq L \log(\frac{1}{s_{\min}})$, where L is a large enough constant. Then there is a randomized polynomial time algorithm that finds $\tilde{y} \in Arr(s, b)$ and $\tilde{x} \in \mathbb{R}^{\mathcal{P}}_{\geq 0}$ with $\mathbf{1}^T \tilde{x} = \mathbf{1}^T x$ and $def(\tilde{x}, \tilde{y}) \leq def(x, y) + O(1)$ while $|frac(\tilde{x})| \leq \frac{1}{2}|frac(x)|$.*

Notice that Lemma 9 lets us assume without loss of generality that we are working with $x \in [0, 1)^{\mathcal{P}}$, so that $\text{frac}(x) = \text{supp}(x)$. While it will take the remainder of this paper to prove the theorem, the algorithm behind the statement can be split into the following two steps:

(I) *Rebuilding the container assignment*: We will change the assignments for the pair (x, y) so that for every container of size σ the patterns in $\text{supp}(x)$ use, they use nearly $(\frac{1}{\sigma})^{1/2}$ copies, while no individual pattern in $\text{supp}(x)$ contains more than $(\frac{1}{\sigma})^{1/4}$ copies of the same container.

(II) *Application of Lovett-Meka*: We will apply the Lovett-Meka algorithm to sparsify the fractional solution x. Here, the vectors v_j that comprise the input for the LM-algorithm will correspond to sums over intervals of rows of the constraint matrix A. Recall that

the error bound provided by Lovett-Meka crucially depends on the lengths $\|v_j\|_2$. The procedure in (I) will ensure that the Euclidean length of those vectors is small.

Once we have proven Theorem 10, Theorem 4 easily follows:

Proof. We compute a fractional solution x to (2.1) of cost $\mathbf{1}^T x \leq OPT_f + 1$. In fact, we can assume that x is a basic solution to the LP and hence $|\text{frac}(x)| \leq |\text{supp}(x)| \leq n$. We construct a container assignment $y \in \text{Arr}(s, b)$ consisting only of singletons, see Lemma 7. Then for $\log(n)$ iterations, we run Theorem 10 and replace x and y with the updated $\tilde{x}$ and $\tilde{y}$.

As soon as $|\text{frac}(x)| \leq O(\log \frac{1}{s_{\min}})$, we can just buy every pattern in $\text{frac}(x)$. In each iteration the deficiency increases by at most $O(1)$. At the end, we use Lemma 8 to actually pack the items into bins. We arrive at a solution of cost $OPT_f + O(\log \max\{n, \frac{1}{s_{\min}}\})$ which is enough, using Lemma 5. □

We will describe the implementation of (I) in Section 2.3 and then (II) in Section 2.4.

2.3 Rebuilding the container assignment

In this section we assume that we are given $x \in [0,1)^{\mathcal{P}}$ with $|\text{supp}(x)| = m$. To ease notation, we will only write the nonzero parts of x, so that if $\text{supp}(x) = \{p_1, p_2, \ldots, p_m\}$, then $x = (x_{p_i})_{i=1}^m$. We update x by altering the patterns that make up its support. Even though some patterns could become identical, we continue to treat them as separate patterns.

Originally, we had defined A as the incidence matrix of the Gilmore-Gomory LP in (2.1) where the rows correspond to items. Since we are now considering patterns to be multi-sets of containers, let us for the rest of the paper redefine the meaning of A. Now, the rows of A correspond to the containers in $\mathcal{C}$ ordered from largest to smallest, and columns represent the patterns in $\text{supp}(x)$. As we perform the grouping and container-forming operations, we update the columns of the matrix. The resulting columns yield a new fractional solution $\tilde{x}$ by taking x_{p_i} copies of the pattern now in column i.

We will now describe our grouping and container reassignment operations, keeping track of what happens to the fractional solution as well as to the corresponding matrix. While it won't be immediately obvious, the point of this process is to ensure that we will be able to find sums of rows that satisfy Lovett-Meka and have small L_2 norm.

First, we need a lemma that tells us how rebuilding the fractional solution affects the deficiency. To have some useful notation, define $\text{mult}(C, x) := \sum_{p \in \mathcal{P}} \text{mult}(C, p) = \sum_{p \in \mathcal{P}} x_p p_C$ to be the number of times that the patterns cover container $C \in \mathcal{C}$.

Lemma 11. *Suppose that $t_\sigma \geq 0$ is such that*

$$\sum_{s(C) \geq s} \textit{mult}(C, \tilde{x}) \geq \begin{cases} \sum_{s(C) \geq s} \textit{mult}(C, x) & \textit{if} \quad s > \sigma \\ \sum_{s(C) \geq s} \textit{mult}(C, x) - t_\sigma & \textit{if} \quad s \leq \sigma \end{cases}$$

Then $\textit{def}(\tilde{x}, y) \leq \textit{def}(x, y) + \sigma \cdot t_\sigma$.

Proof. Let C_0 be the largest container of size at most σ, and let x' be the vector representing t_σ copies of the pattern containing a single copy of C_0. Then $G(\tilde{x}+x', y) \preceq G(x, y)$, and so by Observation 1 we have $\text{def}(G(\tilde{x} + x', y)) \leq \text{def}(G(x, y))$. But if y' is the vector representing t_σ copies of C_0, then $\text{def}(G(\tilde{x}, y)) = \text{def}(G(\tilde{x} + x', y + y'))$, since we can find an optimal assignment taking the containers in y' to those of x'. Since the total size of y' is at most σt_σ, we have $\text{def}(G(\tilde{x}, y)) \leq \text{def}(G(\tilde{x} + x', y)) + \sigma t_\sigma \leq \text{def}(G(x, y)) + \sigma t_\sigma$, and therefore $\text{def}(\tilde{x}, y) \leq \text{def}(x, y) + \sigma t_\sigma$. □

If σ is a power of 2, say $\sigma = 2^{-\ell}$ for $\ell \in \mathbb{Z}_{\geq 0}$, then we say the *size class* σ is the set of items with sizes between $\frac{1}{2}\sigma$ and σ. In this next lemma, we round containers in patterns down so that each container type in size class σ is either not used at all or is used at least $\frac{\delta}{\sigma}$ times.

Lemma 12 (Grouping)**.** *Let (s, b) be a bin packing instance with $y \in \mathbb{Z}^{\mathcal{C}}_{\geq 0}$ and $x \in [0, 1)^{\mathcal{P}}$. For any size class σ and $\delta > 0$, we can find $\tilde{x} \in [0, 1)^{\mathcal{P}}$ so that*

1. $\mathbf{1}^T \tilde{x} = \mathbf{1}^T x$

2. $|supp(\tilde{x})| \leq |supp(x)|$

3. *For each container type C in size class σ, either $mult(C, \tilde{x}) = 0$ or $s(C) \cdot mult(C, \tilde{x}) \geq \delta$. In all other size classes, the multiplicities of containers in patterns do not change.*

4. $def(\tilde{x}, y) \leq def(x, y) + O(\delta)$.

Proof. Assume containers are sorted by size, from largest to smallest. Define S_δ to be the set of containers in size class σ not satisfying condition (3) above, i.e.

$$S_\delta := \{C \text{ in size class } \sigma \mid 0 < s(C) \cdot \text{mult}(C, x) < \delta\}.$$

For a subset $H \subset S_\delta$, define the weight of H to be $w(H) := \sum_{C \in H} s(C) \cdot \text{mult}(C, x)$. Note that the weight of a single container is at most δ. Hence we can partition $S_\delta = H_1 \dot{\cup} H_2 \dot{\cup} \ldots \dot{\cup} H_r$ so that:

1. $w(H_k) \in [2\delta, 3\delta], \forall k = 1, \ldots, r-1$.

2. $w(H_r) \leq 3\delta$.

3. $C \in H_k, C' \in H_{k+1}$ implies $s(C) \geq s(C')$.

For $k = 1, \ldots, r-1$ and containers $C \in H_k$, replace containers of type C in all patterns $p \in \text{frac}(x)$ with the smallest container type appearing in H_k. For all $C \in H_r$, remove containers of type C from all patterns $p \in \text{frac}(x)$. Call the updated vector $\tilde{x}$. We see immediately that $\mathbf{1}^T \tilde{x} = \mathbf{1}^T x$ and $|\text{supp}(\tilde{x})| \leq |\text{supp}(x)|$.

Moreover, since every container type C appearing in $\tilde{x}$ now has an entire group using it, and the weight of each container didn't change by more than a factor of 2, we have $s(C) \cdot \text{mult}(C, \tilde{x}) \geq \delta$, and so condition (3) is satisfied. To complete the proof, it remains to show that $\text{def}(G(\tilde{x}, y)) \leq \text{def}(G(x, y)) + O(\delta)$.

Now, for any i, there is at most one group H_k whose containers (partly) changed from being larger than $s(C_i)$ to smaller. The weight of this group is at most 3δ, and so $\sum_{j \leq i} \text{mult}(C_j, x) -$

$\sum_{j \le i} \text{mult}(C_j, \tilde{x}) \le \frac{6\delta}{\sigma}$. Since this holds for all i, we can apply Lemma 11 to conclude that $\text{def}(G(\tilde{x}, y)) \le \text{def}(G(x, y)) + O(\delta)$. □

We now remark what happens to the associated matrix A under this grouping operation. Write $A, \tilde{A}$ as our original and updated matrices, and $A_C, \tilde{A}_C$ as the rows for container C. For container types C in size class σ, either $\tilde{A}_C x = 0$ or $s(C) \cdot \tilde{A}_C x \ge \delta$. In particular, notice that we have either $\|\tilde{A}_C\|_1 = 0$ or $s(C) \cdot \|\tilde{A}_C\|_1 \ge \delta$. For all other size classes, $\tilde{A}_C = A_C$.

Before we introduce the next main lemma — how to reassign containers — we state a useful result about decomposing packing graphs in a nice way. For a visualization, see Figure 2.2.

Lemma 13. *Suppose $G = (V_\ell \cup V_r, E)$ is a left-integral packing graph as in Section 2.2, and that for every $v \in V_r$, we are given red and blue multiplicities so that $mult(v) = mult_{red}(v) + mult_{blue}(v)$. Suppose further that all nodes $v \in V_r$ of size greater than σ have $mult_{red}(v) = 0$. Then we can find left-integral packing graphs G_{red} and G_{blue} with the same edges, nodes, and sizes of G but with multiplicities satisfying $mult_{red} + mult_{blue} = mult$. Moreover, we have $def(G_{red}) = 0$ and $def(G_{blue}) \le def(G) + \sigma$.*

Proof. By allowing fractional red and blue multiplicities, we can find initial values for the red and blue multiplicities of left nodes so that $\text{def}(G_{\text{red}}) = 0$ and $\text{def}(G_{\text{blue}}) = \text{def}(G)$. To enforce integrality, we will update these multiplicities by swapping (fractional parts of) larger red nodes for smaller blue nodes.

Suppose nodes on the left with positive red multiplicity are ordered by size, so that $\sigma \ge s(v_1) \ge s(v_2) \ge \ldots \ge s(v_\ell)$. While the multiplicities are not all integral, let i be the index of the largest node v_i with $\text{mult}_{\text{red}}(v_i)$ not integral. If $i < \ell$, decrease $\text{mult}_{\text{red}}(v_i)$ to $\lfloor \text{mult}_{\text{red}}(v_i) \rfloor$, and increase $\text{mult}_{\text{red}}(v_{i+1})$ by the same amount. If $i = \ell$, simply decrease $\text{mult}_{\text{red}}(v_\ell)$ to $\lfloor \text{mult}_{\text{red}}(v_\ell) \rfloor$.

For clarity, let $\text{mult}'_{\text{red}}$ denote the red multiplicity at the end of this process and let $\text{mult}'_{\text{blue}} = \text{mult} - \text{mult}'_{\text{red}}$ be the remaining multiplicity. Notice that for all $v \in V_\ell$, we increase $\text{mult}'_{\text{red}}(v)$ by something less than one and then take the floor, so $\text{mult}'_{\text{red}}(v) \le \lceil \text{mult}_{\text{red}}(v) \rceil \le$

$\text{mult}(v)$. In particular, the left red multiplicity is integral and satisfies $0 \leq \text{mult}'_{\text{red}} \leq \text{mult}$, and so the same holds for the blue multiplicity. It remains to analyze the change in deficiency.

Notice that the deficiency of the red graph has not increased, since we either replace nodes with smaller nodes or decrease the multiplicity of the last node. Moreover, for any size s, the total red multiplicity of nodes at least size s has decreased by at most 1. Therefore in the complementary blue graph,

$$\sum_{v \in V_\ell : s(v) \geq s} (\text{mult}'_{\text{blue}}(v) - \text{mult}_{\text{blue}}(v)) \leq 1.$$

The additional blue nodes we fail to pack will therefore all have size at most σ and their total multiplicity will be at most 1, so the deficiency of the blue graph increases by at most σ. □

A key technical ingredient for our algorithm is to be able to replace sets of identical copies of a container in patterns of x by a bigger container that contains the union of the smaller containers. See Figure 2.3 for a visualization of the following construction.

Lemma 14. *Given a pair (x, y) with $x \in \mathbb{R}^{\mathcal{P}}_{\geq 0}$ and $y \in Arr(s, b)$, let $k \in \mathbb{N}$ and $0 < \sigma \leq 1$ be two parameters. Let $\tilde{x} \in \mathbb{R}^{\mathcal{P}}_{\geq 0}$ be the vector that emerges if for all containers C with $\frac{1}{2}\sigma \leq s(C) \leq \sigma$ and all patterns p we replace $k \cdot \lfloor \frac{p_C}{k} \rfloor$ copies of C by $\lfloor \frac{p_C}{k} \rfloor$ copies of the container that is $k \cdot C$. Then there is a $\tilde{y} \in Arr(s, b)$ so that $def(\tilde{x}, \tilde{y}) \leq def(x, y) + O(k\sigma)$.*

Proof. Consider the graph $G(x, y)$ as in Section 2.2. For every right node (C, p), we assign $\text{mult}_{\text{red}}(C, p) = k \cdot \lfloor \frac{p_C}{k} \rfloor \cdot x_p$ for C in size class σ, and $\text{mult}_{\text{red}}(C, p) = 0$ for all other C. We set $\text{mult}_{\text{blue}}(C, p) = \text{mult}(C, p) - \text{mult}_{\text{red}}(C, p)$. By Lemma 13, we can find integral red and blue multiplicities of left nodes so that $\text{def}(G_{\text{red}}) = 0$ and $\text{def}(G_{\text{blue}}) \leq \text{def}(x, y) + \sigma$. The red and blue graphs can now be treated separately, and so we restrict our attention to the red graph since it represents precisely the containers that we want to reassign.

For all nodes (C, p) on the right of the red graph, we combine the copies of C in pattern p into containers of type $k \cdot C$. For clarity we refer to these larger containers as super-containers. Similarly, we look at the containers of the left nodes, ordered from largest to smallest and

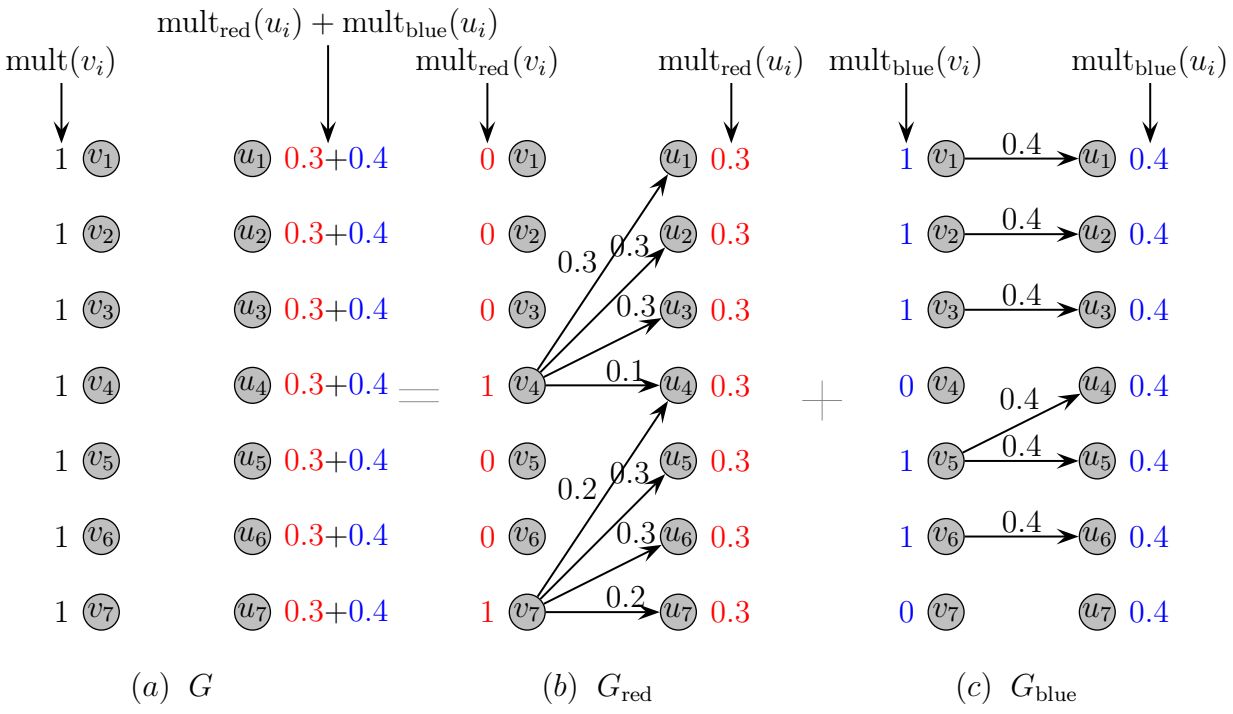


Figure 2.2: Visualization of the packing graph decomposition from Lemma 13. Here, $V_\ell = \{v_1, \ldots, v_7\}$ and $V_r = \{u_1, \ldots, u_7\}$. Assume that $s(u_i) = s(v_i)$ and $s(v_1) > \ldots > s(v_7)$. Nodes are labelled with their multiplicities. In (b) and (c) we also depict the assignments corresponding to the deficiencies.

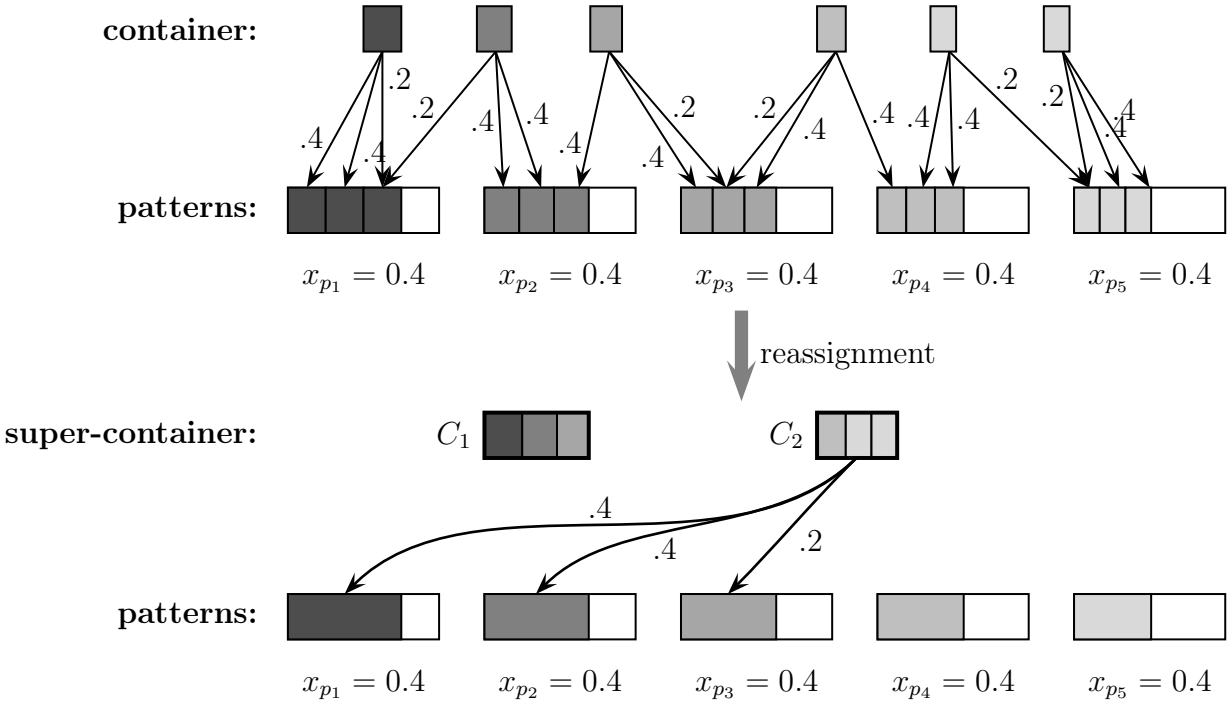


Figure 2.3: Visualization of the reassignment in Lemma 14 for $k = 3$. The upper packing graph is the red part of $G(x, y)$ with the optimal assignment a, assuming that each container has multiplicity 1. The lower graph gives the red part of $G(\tilde{x}, \tilde{y})$ with the constructed assignment a that we give in the analysis. Darker colors indicate larger containers.

taken with multiplicity. In consecutive sets of cardinality k, we combine the containers into super-containers, except perhaps fewer than k of the smallest ones. Write C_i to represent the ith largest super-container on the left.

We claim that all super-containers except C_1 can be packed into the right nodes. To see how to pack them, let a be an optimal assignment in the original red graph. For all i, a assigned the containers making up C_i to some combination of large-enough containers of total multiplicity k. All such containers became part of super-containers in the new graph, and the total multiplicity of their contribution to these super-containers is exactly 1. These super-containers are not necessarily all large enough to fit C_i, but they are all large enough to fit C_{i+1}, and this is exactly where we send C_{i+1}. With this assignment, at most one super-container and k containers were left unpacked, and so the deficiency of the updated red graph is at most $2k\sigma$.

For all containers C, we let $\tilde{y}_C = \text{mult}_{\text{red}}(C) + \text{mult}_{\text{blue}}(C)$. We note that we only changed y by rearranging the containers, and in particular we did not change the item multiplicities, so that $\tilde{y}$ is still an arrangement of the items. With this definition of $\tilde{y}$, we note that $G_{\text{red}} + G_{\text{blue}}$ is precisely $G(\tilde{x}, \tilde{y})$. We therefore have $\text{def}(G(\tilde{x}, \tilde{y})) \leq \text{def}(G(x, y)) + O(k\sigma)$, and so the total increase in deficiency is at most $O(k\sigma)$. □

Note that Lemma 14 is similar to the gluing of [Rot13], but we have no constraints on σ and we allow the number k of containers being combined to vary. We are now ready to give our second main lemma of this section, where we will see that we allow k to depend on the size class σ.

Lemma 15 (Reassigning containers)**.** *Suppose $x \in \mathbb{R}^{\mathcal{P}}_{\geq 0}, y \in Arr(s, b)$. Then there is a polynomial time algorithm to construct a pair $(\tilde{x}, \tilde{y}) \in \mathbb{R}^{\mathcal{P}}_{\geq 0} \times Arr(s, b)$ so that the following is satisfied: Let A be the constraint matrix only containing the columns from $supp(\tilde{x})$ and the rows for all containers C with $\tilde{y}_C > 0$. Then*

1. $\mathbf{1}^T \tilde{x} = \mathbf{1}^T x$.

2. $|supp(\tilde{x})| \leq |supp(x)|$.

3. $def(\tilde{x}, \tilde{y}) \leq def(x, y) + O(1)$.

4. *For any container C in size class σ one has* $\|A_C\|_\infty \leq 2\sigma^{-1/4}$.

5. *For any size class σ one has*

$$\sum_{C \text{ in class } \sigma} (\|A_C\|_1 + \sigma^{-1/2}) \leq 3m\sigma^{-1}.$$

Proof. For each size class σ, starting with the smallest, we do the following. Group the containers with $\delta = \sigma^{1/2}$. By Lemma 12, this increases the deficiency by $O(\sigma^{1/2})$. Moreover, each row will have total size at least $\sigma^{1/2}$ and the total size of all rows from size class σ cannot exceed m, the number of columns. Therefore there are at most $m\sigma^{-1/2}$ rows for size class σ. Then apply Lemma 14 with parameter $k = 2 \cdot \lfloor \sigma^{-1/4} \rfloor$. This increases the deficiency by only $O(k\sigma) = O(\sigma^{3/4})$. Since $\sigma \leq 1$, we have $k \geq 2$, and so any new containers must be in a larger size class. Therefore merging the containers of size class σ can only decrease the rows corresponding to that class. In particular, we have $\sum_{C \text{ in class } \sigma} \|A_C\|_1 \leq 2m\sigma^{-1}$ and $\sum_{C \text{ in class } \sigma} \sigma^{-1/2} \leq (m\sigma^{-1/2})\sigma^{-1/2} = m\sigma^{-1}$, so condition (5) is satisfied. After going over all size classes, we only increase the deficiency by $\sum_{\sigma \in 2^{-\mathbb{N}}} (O(\sigma^{3/4}) + O(\sigma^{1/2})) = O(1)$. □

2.4 Applying the Lovett-Meka algorithm

For the remainder of this paper, let A denote the matrix output by Lemma 15, and let C_i denote the container on row i. Assume that the rows are ordered by size, so that $s(C_1) \geq \ldots \geq s(C_t)$. For $i = 1, \ldots, t$, define $\tilde{n}(i) := \|A_{C_i}\|_1 + \sigma^{-1/2}$ to be the adjusted number of incidences of container C_i, where σ is the size class of C_i. For an interval I of such rows, write $\tilde{n}(I) = \sum_{i \in I} \tilde{n}(i)$.

Notice that in Lemma 15, item (5) gives a bound on the adjusted number of incidences in a size class. In the following, we will construct a collection of intervals, similar to what

is done in [SST] and [Mat99]. In our construction, we will bound the adjusted number of incidences in each interval, which will give us control both over the number of incidences as well as over the number of rows appearing in that interval.

Lemma 16. *Suppose $K > 0$ is a constant. For each size class σ and level $\ell \in \mathbb{Z}_{\geq 0}$ we can construct a collection of intervals $\mathcal{I}_{\sigma,\ell}$ satisfying the following:*

1. $|\mathcal{I}_{\sigma,\ell}| \leq 7^\ell \cdot (\frac{12}{K} m\sigma^{1/16} + 1)$.
2. $\forall \ell > 0$ *and* $I \in \mathcal{I}_{\sigma,\ell}$, *we have* $\tilde{n}(I) \leq (\frac{1}{2})^\ell K\sigma^{-17/16}$.
3. *If C_i is in size class σ, we can partition $\{1, \ldots, i\}$ using intervals from our collections so that for any $\ell > 0$ we use at most 7 intervals on level ℓ, all of which are in $\mathcal{I}_{\sigma,\ell}$.*

Proof. For size class σ, we first partition the rows into level 0 intervals as follows. For any row i satisfying $\tilde{n}(i) > (\frac{1}{2})K\sigma^{-17/16}$, we let $\{i\}$ be its own interval. We then subdivide the remaining rows into intervals so that $\tilde{n}(I) \leq K\sigma^{-17/16}$ for each interval I. We need a total of at most $\frac{4}{K}\sigma^{17/16}(3m\sigma^{-1}) + 1 = \frac{12}{K} m\sigma^{1/16} + 1$ intervals on level 0.

Now, given an interval I on level $\ell - 1$ with $|I| > 1$, we will subdivide I into at most 7 intervals on level ℓ. First, for any row $i \in I$ with $\tilde{n}(i) > (\frac{1}{2})^{\ell+1} K\sigma^{-17/16}$, let $\{i\}$ be its own interval. Note that there can be at most one such i. We then subdivide the remaining rows into intervals J so that $\tilde{n}(J) \leq (\frac{1}{2})^\ell K\sigma^{-17/16}$. Since none of the rows $i \in I$ became its own interval on level $\ell - 1$, we also know that $\tilde{n}(i) \leq (\frac{1}{2})^\ell K\sigma^{-17/16}$, and so this bound holds for every interval on level ℓ. The number of intervals on level ℓ is at most $7^\ell \cdot (\frac{12}{K} m\sigma^{1/16} + 1)$. Condition (3) holds by construction of the intervals. □

Let us abbreviate $\mathcal{I} := \bigcup_{\sigma,\ell} \mathcal{I}_{\sigma,\ell}$ as our entire collection of intervals constructed in Lemma 16. For an interval $I \in \mathcal{I}_{\sigma,\ell}$, we define v_I to be the sum of the rows in interval I, and we define $\lambda_I := \ell$. In other words, the parameter just denotes the level of the interval.

The input for the Lovett-Meka algorithm will consist of the pairs $\{(v_I, \lambda_I)\}_{I \in \mathcal{I}}$ where we use $\lambda_I \geq 0$ as the parameter for a constraint with normal vector v_I. Additionally, we add a

single vector $v_{\text{obj}} := \mathbf{1}$ with parameter $\lambda_{\text{obj}} := 0$ to control the objective function. There are two things to show. First we argue that the parameters are chosen so that the condition of the Lovett-Meka algorithm is actually satisfied.

Lemma 17. *Suppose that $m = |supp(x)| \geq L\log(\frac{1}{s_{\min}})$. For K, L large enough constants, one has*

$$\sum_{I \in \mathcal{I}} e^{-\lambda_I^2/16} + 1 \leq \frac{m}{16}.$$

Proof. From Lemma 16, we can calculate that

$$\begin{aligned}
\sum_{I \in \mathcal{I}} e^{-\lambda_I^2/16} &= \sum_{\sigma \in 2^{-\mathbb{N}}} \sum_{\ell \geq 0} e^{-\ell^2/16} \cdot |\mathcal{I}_{\sigma,\ell}| \\
&\leq \sum_{\sigma \in 2^{-\mathbb{N}}} \sum_{\ell \geq 0} e^{-\ell^2/16} \cdot 7^\ell \cdot (\frac{12}{K} m\sigma^{-1/16} + 1) \\
&= \frac{12m}{K} \cdot \sum_{\sigma \in 2^{-\mathbb{N}}} \sigma^{-1/16} \sum_{\ell \geq 0} e^{-\ell^2/16} \cdot 7^\ell \\
&\quad + \log(\frac{1}{s_{\min}}) \cdot \sum_{\ell \geq 0} e^{-\ell^2/16} \cdot 7^\ell \\
&\leq \frac{m}{16} - 1
\end{aligned}$$

for K, L large enough. ∎

Now, suppose we run the Lovett-Meka algorithm and obtain a solution $\tilde{x}$ with $|\text{frac}(\tilde{x})| \leq \frac{1}{2}|\text{frac}(x)|$ so that

$$|\langle v_I, x - \tilde{x}\rangle| \leq \lambda_I \cdot \|v_I\|_2 \quad \forall I \in \mathcal{I} \quad \text{and} \quad \mathbf{1}^T x = \mathbf{1}^T \tilde{x}.$$

The next step will be to combine the construction of intervals in Lemma 16 and the guarantees about the matrix from Lemma 15 to argue that L_2 norms are small and hence that the error in terms of deficiency will be small. Recall that we still assume containers are sorted so that $1 \geq s(C_1) \geq s(C_2) \geq \ldots \geq s(C_s) > 0$.

Lemma 18. *Let C_i be a container in size class σ. Then*

$$\Big|\sum_{j \leq i} A_{C_j}(x - \tilde{x})\Big| \leq O(\sigma^{-15/16}).$$

Proof. If C_i is a container in size class σ, Lemma 16 tells us we can write the interval $\{1,\dots,i\} = \dot{\bigcup}_{I\in\mathcal{I}(i)} I$ as the disjoint union of intervals $\mathcal{I}(i) \subseteq \mathcal{I}$ so that the only intervals $I \in \mathcal{I}(i)$ with $\lambda_I > 0$ are from class σ and we only take at most 7 intervals from each level. For all such intervals on level $\ell > 0$, we have

$$\begin{aligned}
\|v_I\|_2 &\leq \sqrt{\|v_I\|_1 \cdot \|v_I\|_\infty} \\
&\leq \sqrt{\|v_I\|_1 \cdot \sum_{i\in I} \|A_{C_i}\|_\infty} \\
&\leq \sqrt{\tilde{n}(I)\cdot\sigma^{1/2}\tilde{n}(I)\cdot 2\sigma^{-1/4}} \\
&\leq \sqrt{2}\tilde{n}(I)\cdot\sigma^{1/8}.
\end{aligned}$$

Consequently, we can bound

$$\begin{aligned}
\Big|\sum_{j\leq i} A_{C_j}(\tilde{x}-x)\Big| &\leq \sum_{I\in\mathcal{I}(i)} \lambda_I \cdot \|v_I\|_2 \\
&\leq \sqrt{2}\sum_{I\in\mathcal{I}(i)} \lambda_I\cdot\tilde{n}(I)\sigma^{1/8} \\
&\leq \sqrt{2}\sum_{\ell>0} 7\ell\cdot 2^{-\ell}K\sigma^{-15/16} \\
&= O(\sigma^{-15/16}).
\end{aligned}$$

□

It remains to argue why $\mathrm{def}(\tilde{x},y) \leq \mathrm{def}(x,y) + O(1)$ for one application of Lovett-Meka. First notice that $A_{C_j}\tilde{x} = \mathrm{mult}(C_j,\tilde{x})$ and $A_{C_j}x = \mathrm{mult}(C_j,x)$. Therefore by Lemmas 11 and 18 the rounding of each size class σ increases the deficiency by at most $O(1)\cdot\sigma^{-15/16}\cdot\sigma = O(1)\cdot\sigma^{1/16}$. Summing over all size classes gives a total increase in deficiency

$$O(1)\cdot\sum_{\sigma\in 2^{-\mathbb{N}}}\sigma^{1/16} \leq O(1).$$

That finishes the proof of Theorem 10, and hence of Theorem 4.

Chapter 3

IMPROVED RESCALING METHODS FOR LINEAR PROGRAMMING ALGORITHMS

3.1 Introduction

One of the central algorithmic problems in theoretical computer science as well as in more practical areas like operations research is finding the solution to a *linear program*

$$\max\{c^T x \mid Ax \geq b\} \tag{3.1}$$

where $A \in \mathbb{R}^{m \times n}$, $c \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$. On the theoretical side, linear programming relaxations are the backbone for many approximation algorithms [WS11, Vaz01]. On the practical side, many real-world problems can either be modeled as linear programs or as integer linear programs; the latter ones are then solved using Branch & Bound or Branch & Cut methods, both of which rely on repeatedly computing solutions to linear programs [CCZ14].

The first algorithm for linear programming was the *simplex method* due to Dantzig [Dan51]. While the method performs well in practice — and is still the method of choice today — for almost any popular pivoting rule one can construct instances where the algorithm takes exponential time [KM72]. In 1979, Khachiyan [Hač79, Sch86] developed the first polynomial-time algorithm. However, despite the desirable theoretical properties, Khachiyan's *ellipsoid method* turned out to be too slow for practical applications.

In the 1980s, *interior point methods* were developed which were efficient in theory and in practice. Karmarkar's algorithm has a running time of $O(n^{3.5}L)$, where L is the number of bits in the input [Kar84]. As recently as 2015, it was shown that there is an interior-point method using only $\tilde{O}(\sqrt{\text{rank}(A)} \cdot L)$[1] many iterations; this upper bound essentially matches

[1]The $\tilde{O}$-notation suppresses any $\text{polylog}(m, n)$ terms.

known lower bound barriers [LS15].

A common way to find a polynomial-time linear programming algorithm is with a greedy type procedure along with periodic rescaling. One famous example of this is the *perceptron algorithm* [Agm54], which we will focus on in this paper. Instead of solving (3.1) directly, this method finds a feasible point in the *open polyhedral cone*

$$P = \{x \in \mathbb{R}^n \mid Ax > \mathbf{0}\} \tag{3.2}$$

where $A \in \mathbb{R}^{m \times n}$ – using standard reductions one can interchange the representations (3.1) and (3.2) with at most a linear overhead. The classical perceptron algorithm starts at the origin and iteratively walks in the direction of any violated constraint. In the worst case this method is not polynomial time, but it is still useful due to its simplicity and robustness [Agm54]. In 2004, Dunagan and Vempala [DV06] showed that using a randomized rescaling procedure, the algorithm can be modified to find a point in (3.2) in polynomial time. Explicitly, their algorithm runs in time $\tilde{O}(mn^4 \log \frac{1}{\rho})$, where $\rho > 0$ is the radius of the largest ball in the intersection of P with the unit ball $B := B(0, 1)$. A deterministic rescaling procedure was provided by Peña and Soheili in [PS16] using techniques developed by Betke and Chubanov [Chu15, Chu12, Bet04]. Their algorithm uses an improved convergence of the perceptron algorithm based on *Nesterov's smoothing technique* [Nes05, PS12]. Overall, their algorithm takes time $\tilde{O}(m^2 n^{2.5} \log \frac{1}{\rho})$.

Another classical LP algorithm that we will discuss in this paper is based on a very general algorithmic framework called the *multiplicative weights update* (MWU) method. In its general form one imagines having m *experts* who each incur some *cost* in a sequence of iterations. In each iteration we have to select a convex combination of experts so that the expected cost is minimized, where we only have information on the past costs. The MWU method initially gives all experts the same weight and in each iteration the weight of expert i is multiplied by $\exp(-\varepsilon \cdot \text{cost incurred by expert } i)$ where ε is some parameter. Then on average, the convex combination given by the weights will be nearly as good as the cost incurred by the best expert. MWU is an *online algorithm* that does not need to know

the costs in advance, and it has numerous applications in machine learning, economics and theoretical computer science. In fact, MWU has been reinvented many times under different names in the literature. Recent applications in theoretical computer science include finding fast approximations to maximum flows [CKM$^+$11], multicommodity flows [GK07, Mad10], solving LPs [PST95], and solving semidefinite programs [AHK05]. We refer to the survey of Arora, Hazan and Kale [AHK12] for a detailed overview.

When we apply the MWU framework to linear programming, the experts correspond to the linear constraints. Suppose we use this method to find a valid point in $P = \{x : Ax > \mathbf{0}\}$ where $\|A_i\|_2 = 1$ for every row A_i. At iteration t, the cost associated with expert i will be $\langle A_i, p^{(t)} \rangle$ for some vector $p^{(t)}$. Therefore the weight of expert i at time T will be $e^{-\langle A_i, x \rangle}$ where $x = \sum_{t=1}^{T} \varepsilon^{(t)} p^{(t)}$. The analysis of MWU consists of bounding the sum of the weights, which in this case is given by the *potential function* $\Phi(x) = \sum_{i=1}^{m} e^{-\langle A_i, x \rangle}$. If we choose the update vector $p^{(t)}$ to be a weighted sum of constraints at every iteration, notice that the resulting walk in $\mathbb{R}^n$ corresponds to gradient descent on Φ – in this case MWU terminates in $\tilde{O}(\frac{1}{\rho^2})$ iterations. However, ρ need not be polynomial in the input size, and in fact this method is not polynomial time in the worst case.

3.1.1 Our contribution

For reference, the general form for the rescaled LP algorithms we will present in this paper is given in Algorithm 3. Throughout this paper we will assume that P is nonempty and full-dimensional.

Algorithm 3

FOR $\tilde{O}(n \log \frac{1}{\rho})$ phases DO:

(1) **Initial phase:** Either find $x \in P$ or provide a $\lambda \in \mathbb{R}^m_{\geq 0}$, $\|\lambda\|_1 = 1$ with $\|\lambda A\|_2 \leq \Delta$.

(2) **Rescaling phase:** Find an invertible linear transformation F so that $\text{vol}(F(P) \cap B)$ is a constant fraction larger than $\text{vol}(P \cap B)$. Replace P by $F(P)$.

Our technical and conceptual contributions are as follows:

(1) *Improved rescaling:* We design a rescaling method that applies for a parameter of $\Delta = \Theta(\frac{1}{n})$, which improves over the threshold $\Delta = \Theta(\frac{1}{m\sqrt{n}})$ required by [PS16]. This results in a smaller number of iterations that are needed per phase until one can rescale the system.

(2) *Rescaling the MWU method:* We show that in $\tilde{O}(1/\Delta^2)$ iterations the MWU method can be made to implement the initial phase of Algorithm 3. The idea is that if gradient descent is making insufficient progress then the gradient must have small norm, and from this we can extract an appropriate λ. In particular, combining this with our rescaling method, we obtain a polynomial time LP algorithm based on MWU.

(3) *Faster gradient descent:* The standard gradient descent approach terminates in at most $\tilde{O}(1/\Delta^2)$ iterations, which matches the first approach in [PS16]. The more recent work of Peña and Soheili [PS12] uses *Nesterov smoothing* to bring the number of iterations down to $\tilde{O}(1/\Delta)$. We prove that essentially the same speedup can be obtained without modifying the objective function by projecting the gradient on a significant eigenspace of the Hessian.

(4) *Computing an approximate John ellipsoid:* For a general convex body K, computing a John ellipsoid is equivalent to finding a linear transformation so that $F(K)$ is well rounded. For our unbounded region P, our rescaling algorithm gives a linear transformation F so that $F(P) \cap B$ is well-rounded.

3.2 Rescaling of the Perceptron Algorithm

In this section we fix an initial phase for Algorithm 3 – in particular, the paper of Peña and Soheili gives a smooth variant of the perceptron algorithm that implements the initial phase of Algorithm 3 in time $\tilde{O}(\frac{mn}{\Delta})$ [PS16].

We then focus on the rescaling phase of the algorithm. Our main result is that we are able to rescale with $\Delta = O(\frac{1}{n})$.

Lemma 19. *Suppose $\lambda \in \mathbb{R}^m_{\geq 0}$ with $\|\lambda\|_1 = 1$ and $\|\lambda A\|_2 \leq O(\frac{1}{n})$. Then in time $O(mn^2)$ we can rescale P so that* $\mathrm{vol}(P \cap B)$ *increases by a constant factor.*

We introduce two new rescaling methods that achieve the guarantee of Lemma 19. First we show that we can extract a thin direction by sampling rows of A using a *random hyperplane*. The linear transformation that scales P in that direction, corresponding to a *rank-1 update*, will increase $\mathrm{vol}(P \cap B)$ by a constant factor.

Next we give an alternate rescaling which is no longer a rank-1 update but which has the potential to increase $\mathrm{vol}(P \cap B)$ by up to an exponential factor under certain conditions. In addition, if we take an alternate view where the cone P is left invariant and instead update the underlying *norm*, we see that this rescaling consists of adding a scalar multiple of a particular Hessian matrix to the matrix defining the norm. We also believe that this view is the right one to make potential use of the *sparsity* of the underlying matrix A, which would be a necessity for any practically relevant LP optimization method.

Combining the guarantees for the initial phase and rescaling phase gives us the following theorem:

Theorem 20. *There is an algorithm based on the perceptron algorithm that finds a point in P in time $\tilde{O}(mn^3 \log(\frac{1}{\rho}))$.*

3.2.1 Rescaling Using a Thin Direction

In this section we will show how we can rescale by finding a direction in which the cone is *thin*. First we give the formal definition of width.

Definition 1. *Define the width of the cone P in the direction $c \in \mathbb{R}^n \setminus \{\mathbf{0}\}$ as*

$$\mathrm{width}(P, c) = \frac{1}{\|c\|_2} \max_{x \in P \cap B} |\langle c, x \rangle|.$$

In [PS16], Peña and Soheili show that stretching P in a thin enough direction increases the volume of $P \cap B$ by a constant factor.

Lemma 21 ([PS16])**.** *Suppose that there is a direction $c \in \mathbb{R}^n \setminus \{\mathbf{0}\}$ with* $\text{width}(P, c) \leq \frac{1}{3\sqrt{n}}$. *Define $F : \mathbb{R}^n \to \mathbb{R}^n$ as the linear map with $F(c) = 2c$ and $F(x) = x$ for all $x \perp c$. Then*

$$vol(F(P) \cap B) \geq \frac{3}{2} \cdot vol(P \cap B).$$

Explicitly, assuming $\|c\|_2 = 1$, Lemma 21 updates our constraint matrix to $A(I - \frac{1}{2}cc^T)$. In particular, we apply a *rank-1 update* to the constraint matrix. Given a solution x to these new constraints, a solution to the original problem can be easily recovered as $(I - \frac{1}{2}cc^T)x$.

It remains to argue how one can extract a thin direction for P, given a convex combination λ so that $\|\lambda A\|_2$ is small. Here we will significantly improve over the bounds of [PS16] which require $\|\lambda A\|_2 \leq O(\frac{1}{m\sqrt{n}})$. We begin by a new generic argument to obtain a thin direction:

Lemma 22. *For any non-empty subset $J \subseteq [m]$ of constraints one has*

$$\text{width}\left(P, \sum_{i \in J} \lambda_i A_i\right) \leq \frac{\|\sum_{i=1}^m \lambda_i A_i\|_2}{\|\sum_{i \in J} \lambda_i A_i\|_2}.$$

Proof. First, note that by the full-dimensionality of P, we always have $\|\sum_{i \in J} \lambda_i A_i\|_2 > 0$. By definition of width, we can write

$$\text{width}\left(P, \sum_{i \in J} \lambda_i A_i\right) = \frac{1}{\|\sum_{i \in J} \lambda_i A_i\|_2} \max_{x \in P \cap B} \langle \sum_{i \in J} \lambda_i A_i, x \rangle.$$

Now, we know that $\langle A_i, x \rangle \geq 0$ for all $x \in P$ and so

$$\max_{x \in P \cap B} \langle \sum_{i \in J} \lambda_i A_i, x \rangle \leq \max_{x \in B} \langle \sum_{i=1}^m \lambda_i A_i, x \rangle = \|\lambda A\|_2$$

and the claim is proven. □

So in order to find a direction of small width, it suffices to find a subset $J \subseteq [m]$ with $\|\sum_{i \in J} \lambda_i A_i\|_2$ large. Implicitly, the choice that Peña and Soheili [PS16] make is to select

$J = \{i_0\}$ for $i_0 \in [m]$ maximizing λ_{i_0}. This approach gives a bound of $\| \sum_{i \in J} \lambda_i A_i \|_2 \geq \frac{1}{m}$. We will now prove the asymptotically optimal bound[2] using a random hyperplane:

Lemma 23. *Let $\lambda \in \mathbb{R}^m_{\geq 0}$ be any convex combination and $A \in \mathbb{R}^{m \times n}$ with $\|A_i\|_2 = 1$ for all i. Take a random Gaussian g and set $J := \{i \in [m] \mid \langle A_i, g \rangle \geq 0\}$. Then with constant probability $\| \sum_{i \in J} \lambda_i A_i \|_2 \geq \frac{1}{4\sqrt{\pi n}}$.*

Proof. We set $v := \frac{g}{\|g\|_2}$. Since v is unit vector we can lower bound the length of $\| \sum_{i \in J} \lambda_i A_i \|_2$ by measuring the projection on v and obtain $\| \sum_{i \in J} \lambda_i A_i \|_2 \geq \sum_{j \in J} \lambda_i \langle A_i, v \rangle$. By symmetry of the Gaussian it then suffices to argue that $\sum_{i=1}^m \lambda_i |\langle A_i, v \rangle| \geq \frac{1}{2\sqrt{\pi n}}$. First we will show that for an appropriate constant $\alpha \in (0, 1)$,

(1) $\Pr(\|g\|_2 \geq \sqrt{2n}) \leq \frac{2}{n}$

(2) $\Pr(\sum_{i=1}^n \lambda_i |\langle A_i, g \rangle| < \sqrt{\frac{1}{2\pi}}) \leq \alpha$.

Then, with probability at least $\gamma = \frac{1-\alpha}{2}$, we have $\sum_{i=1}^n \lambda_i |\langle A_i, v \rangle| \geq \frac{1}{2\sqrt{\pi n}}$.

For (1), notice that $\|g\|_2^2$ is just the chi-squared distribution with n degrees of freedom, and so it has variance $2n$ and mean n. Therefore Chebyshev's inequality tells us that $\Pr\left[\|g\|_2^2 \geq 2n\right] \leq \frac{2}{n}$. Now, for all i, $\langle A_i, g \rangle$ is a normal random variable with mean 0 and variance 1, and so the expectation of its absolute value is $\sqrt{\frac{2}{\pi}}$. Summing these up gives $\mathbb{E}\left[\sum_{i=1}^m |\langle \lambda_i A_i, g \rangle|\right] = \sqrt{\frac{2}{\pi}}$. Moreover, $\sum_{i=1}^m |\langle \lambda_i A_i, g \rangle|$ is Lipschitz in g with Lipschitz constant 1, and so[3]

$$\Pr\left(\sum_{i=1}^m |\langle \lambda_i A_i, g \rangle| < \sqrt{\frac{2}{\pi}} - t\right) \leq e^{-t^2/\pi^2}.$$

---

[2]It suffices here to consider the trivial example with $\lambda_1 = \ldots = \lambda_n = \frac{1}{n}$ and $A_i = e_i$ being the standard basis. Then $\| \sum_{i \in J} \lambda_i A_i \|_2 \leq \frac{1}{\sqrt{n}}$ for any subset J. The optimality of our rescaling can also be seen since the cone in the last iteration is $\tilde{O}(n)$-well rounded, which is optimal up to $\tilde{O}$-terms.

[3]Recall that a function $F : \mathbb{R}^n \to \mathbb{R}$ is *Lipschitz* with Lipschitz constant 1 if $|F(x) - F(y)| \leq \|x - y\|_2$ for all $x, y \in \mathbb{R}^n$. A famous concentration inequality by Sudakov, Tsirelson, Borell states that $\Pr[|F(g) - \mu| \geq t] \leq e^{-t^2/\pi^2}$, where g is a random Gaussian and μ is the mean of F under g.

Letting $t = \sqrt{\frac{1}{2\pi}}$ gives (2). By a union bound, the probability either of these events happens is at most $\alpha + \frac{2}{n}$, and so with probability at least $\frac{1-\alpha}{2}$ neither occurs, which gives us the claim. □

While the proof is probabilistic, one can use *conditional expectation* to derandomize the sampling [AS08]. More concretely, consider the function $F(g) := \sum_{i=1}^{m} \lambda_i |\langle A_i, g\rangle| - \frac{1}{10\sqrt{n}}\|g\|_2$. The proof of Lemma 23 implies that the expectation of this function is at least $\Omega(1)$. Then we can find a desired vector $g = (g_1, \ldots, g_n)$ by choosing the coordinates one after the other so that the conditional expectation does not decrease. We are now ready to prove Lemma 19, which we restate here with explicit constants.

Lemma 24. *Suppose $\lambda \in \mathbb{R}^m_{\geq 0}$ with $\|\lambda\|_1 = 1$ and $\|\lambda A\|_2 \leq \frac{1}{12n\sqrt{\pi}}$. Then in time $O(mn^2)$ we can rescale P so that* $\mathrm{vol}(P \cap B)$ *increases by a constant factor.*

Proof. Computing a random Gaussian and checking if it satisfies the conditions of Lemma 23 takes time $O(mn)$. Since the conditions will be satisfied with constant probability, the expected number of times we must do this is constant. Once the conditions are satisfied, finding a thin direction and rescaling can be done in time $O(n^3)$. Lemmas 21 and 22 guarantee we get a constant increase in the volume. □

3.2.2 Deterministic Multi-rank Rescaling

We now introduce an alternate linear transformation we can use to rescale. This is no longer a rank-1 update, but it is inherently deterministic along with other nice properties. For one thing, although we only guarantee constant improvement in the volume, under certain circumstances the rescaling can improve the volume by an exponential factor. This transformation will also take a nice form when we change the view to consider rescaling the unit ball rather than the feasible region.

Lemma 25. *Suppose $\lambda \in \mathbb{R}^m_{\geq 0}$, $\|\lambda\|_1 = 1$ and $\|\lambda A\|_2 \leq \frac{1}{10n}$. Let M denote the matrix $\sum_{i=1}^{m} \lambda_i A_i A_i^T$ and suppose $0 \leq \alpha \leq \frac{1}{\delta_{max}}$, where $\delta_{max} = \|M\|_{op}$ denotes the maximal eigenvalue of M. Define $F(x) = (I + \alpha M)^{1/2} x$. Then* $\mathrm{vol}(F(P) \cap B) \geq e^{\alpha/5} \mathrm{vol}(P \cap B)$.

Proof. First notice that M is symmetric positive semi-definite with trace 1. Therefore the eigenvalues of $I + \alpha M$ take the form $1 + \alpha\delta_i$ where $0 \leq \delta_i \leq \alpha$ and $\sum_{i=1}^{n} \delta_i = 1$. Note that since $\alpha\delta_i \leq 1$, we can lower bound the eigenvalues by $1 + \alpha\delta_i \geq e^{\alpha\delta_i/2}$. Therefore

$$\det(I + \alpha M) \geq \prod_{i=1}^{n} e^{\alpha\delta_i/2} = \exp\left(\frac{\alpha}{2}\sum_{i=1}^{n}\delta_i\right) = e^{\alpha/2}.$$

In particular, $\det(F) \geq e^{\alpha/4}$.

So far we have shown that $\mathrm{vol}(F(P \cap B))$ is significantly larger than $\mathrm{vol}(P \cap B)$. However, the desired bound is on $\mathrm{vol}(F(P) \cap B)$, and so we need to ensure that we do not lose too much of the volume when we intersect with the unit ball. It turns out the bound on $\|\lambda A\|_2$ will allow us to do precisely this.

For any $x \in P \cap B$, we get the bound

$$\|F(x)\|_2^2 = x^T x + \alpha \sum_{i=1}^{m} \lambda_i \langle A_i, x\rangle^2 \leq 1 + \alpha \sum_{i=1}^{m} \lambda_i \langle A_i, x\rangle \leq 1 + \alpha\|\lambda A\|_2 \leq 1 + \frac{\alpha}{10n}.$$

The point is that every element of $F(P \cap B)$ has length at most $1 + \frac{\alpha}{20n}$, and so intersecting with the unit ball will not lose more volume than shrinking by a factor of $1 + \frac{\alpha}{20n}$. In particular, the volume decreases by at most $(1 + \frac{\alpha}{20n})^{-n} \geq e^{-\alpha/20}$, and so we have

$$\begin{aligned}
\mathrm{vol}(F(P) \cap B) &\geq e^{-\alpha/20}\,\mathrm{vol}(F(P \cap B)) \\
&\geq e^{-\alpha/20} \cdot e^{\alpha/4}\,\mathrm{vol}(P \cap B) \\
&\geq e^{\alpha/5} \cdot \mathrm{vol}(P \cap B).
\end{aligned}$$

□

Note that one always has $\delta_{\max} \leq 1$ and hence in any case one can choose $\alpha \geq 1$. Therefore if $\|\lambda A\|_2 \leq \frac{1}{10n}$, we get constant improvement in $\mathrm{vol}(P \cap B)$. In fact, if the eigenvalues of M happen to be small, we could get up to exponential improvement. This computation can be carried out in time $O(mn^2)$ and so Lemma 25 proves Lemma 19 and hence Theorem 20.

3.2.3 An Alternate View of Rescaling

Instead of applying a linear transformation to the cone P, there is an equivalent view where instead one applies a linear transformation to the unit ball. We will now switch the view in the sense that we fix the cone P, but we update the norm in each rescaling step so that the unit ball becomes more representative of P.

Recall that a symmetric positive definite matrix $H \in \mathbb{R}^{n\times n}$ induces a *norm* $\|x\|_H := \sqrt{x^T H x}$. Note that also H^{-1} is a symmetric positive definite matrix [4] and $\|\cdot\|_{H^{-1}}$ is the *dual norm* of $\|\cdot\|_H$. In this view we assume the rows A_i of A are normalized so that $\|A_i\|_{H^{-1}} = 1$.

Let $B_H := \{x \in \mathbb{R}^n \mid \|x\|_H \leq 1\}$ be the unit ball for the norm $\|\cdot\|_H$. Note that B_H is always an *ellipsoid*. We will measure progress in terms of the fraction of the ellipsoid B_H that lies in the cone P, namely $\mu(H) := \frac{\text{vol}(B_H \cap P)}{\text{vol}(B_H)}$. The goal of the rescaling step will then be to increase $\mu(H)$ by a constant factor. Note that we initially have $\mu(H) = \mu(I) \geq \rho^n$, and at any time $0 \leq \mu(H) \leq 1$, so we can rescale at most $O(n \log \frac{1}{\rho})$ times.

In this view, Lemma 25 takes the following form:

Lemma 26. *Let $H \in \mathbb{R}^{n\times n}$ be symmetric with $H \succ 0$. Suppose $\lambda \in \mathbb{R}^m_{\geq 0}$ with $\|\lambda\|_1 = 1$ and $\|\lambda A\|_{H^{-1}} \leq \frac{1}{10n}$ and let $M := \sum_{i=1}^m \lambda_i A_i A_i^T$. Let $0 \leq \alpha \leq \frac{1}{\delta_{\max}}$, where $\delta_{\max} := \|H^{-1}M\|_{op}$. Then for $\tilde{H} := H + \alpha M$ one has $\mu(\tilde{H}) \geq e^{\alpha/5} \cdot \mu(H)$.*

Algorithm 4 illustrates the multi-rank rescaling under the alternate view. Notice that the algorithm updates the norm matrix by adding a scalar multiple of the Hessian matrix of the MWU potential function discussed in Section 3. Moreover, throughout the algorithm our matrix H will have the form $I + \sum_{i=1}^m h_i A_i A_i^T$ for some $h_i \geq 0$. Note that this allows fairly compact representation as we only need $O(m)$ space to encode the coefficients h_i that define the norm matrix.

[4] An easy way to see this is to write $H = \sum_{j=1}^n \mu_j u_j u_j^T$ as the eigendecomposition of H. Then $H^{-1} = \sum_{j=1}^n \frac{1}{\mu_j} u_j u_j^T$ is the inverse; clearly all eigenvalues are positive and the inverse has the same spectrum as H.

Algorithm 4

FOR $\tilde{O}(n \log \frac{1}{\rho})$ phases DO:

(1) **Initial phase:** Either find $x \in P$ or give $\lambda \geq 0$, $\|\lambda\|_1 = 1$ with $\|\lambda A\|_{H^{-1}} \leq O(\frac{1}{n})$.

(2) **Rescaling phase:** Update $H := H + \alpha M$, where $M = \sum_{i=1}^{m} \lambda_i A_i A_i^T$.

3.3 Rescaling for the MWU algorithm

In this section we show that the same rescaling methods can be used to make the MWU method into a polynomial time LP algorithm. In particular, we show that a *MWU phase* can implement the initial phase of Algorithm 3 or 4. For ease of notation, we will assume $H = I$, but we can recover the general case by replacing A_i by $H^{-1/2}A_i$ and replacing the update direction p by $H^{1/2}p$.

Recall that the MWU algorithm corresponds to gradient descent on a particular potential function. First we show how the standard gradient descent approach implements the initial phase. We then introduce a modified gradient descent, which speeds up the MWU phase to $\tilde{O}(1/\Delta)$ iterations. Combining this with our rescaling results gives the following:

Theorem 27. *There is an algorithm based on the MWU algorithm that finds a point in P in time $\tilde{O}(mn^{\omega+1} \log(\frac{1}{\rho}))$, where $\omega \approx 2.373$ is the exponent of matrix multiplication.*

3.3.1 Standard Gradient Descent

Consider the potential function $\Phi(x) = \sum_{i=1}^{m} e^{-\langle A_i, x \rangle}$, where $\|A_i\|_2 = 1$ for all rows A_i. Notice that $\Phi(0) = m$ and that if $\Phi(x) < 1$ then $\langle A_i, x \rangle > 0$ for all i, and hence $x \in P$. In this section we analyze standard gradient descent on Φ, starting at the origin. Notice that the gradient takes the form

$$\nabla \Phi(x) = -\sum_{i=1}^{m} e^{-\langle A_i, x \rangle} A_i.$$

If we let $\lambda_i = \frac{1}{\Phi(x)} e^{-\langle A_i, x\rangle}$, we see that $\|\lambda\|_1 = 1$ and $\lambda A = -\frac{\nabla\Phi(x)}{\Phi(x)}$. In particular, if at any iteration this vector has small Euclidean norm, then we will be able to rescale. It remains to show, therefore, that if this vector has large Euclidean norm, then we get sufficient decrease in the potential function.

We begin by establishing some useful notation. Given $x \in \mathbb{R}^n$, define $\lambda_i = \frac{1}{\Phi(x)} e^{-\langle A_i, x\rangle}$, $y = -\frac{\nabla\Phi(x)}{\Phi(x)} = \sum_{i=1}^m \lambda_i A_i$ and $M = \frac{\nabla^2\Phi(x)}{\Phi(x)} = \sum_{i=1}^m \lambda_i A_i A_i^T$. Even though all three depend on x, we will not denote that here to keep the notation clean. We assume now that $\|A_i\|_2 = 1$ for all i.

Lemma 28. *Suppose $x \in \mathbb{R}^n$ and abbreviate $y = -\frac{\nabla\Phi(x)}{\Phi(x)}$. Then*

$$\Phi(x + \frac{1}{2}y) \leq \Phi(x) \cdot e^{-\|y\|_2^2/4}.$$

Proof. First note that since $\|\lambda\|_1 = 1$ and $\|A_i\|_2 = 1$, we know that $|\langle A_i, y\rangle| \leq 1$ for all i. In our analysis we will also use the fact that for any $z \in \mathbb{R}$ with $|z| \leq 1$ one has $e^z \leq 1 + z + z^2$. We obtain the following.

$$\begin{aligned}
\Phi(x + \frac{1}{2}y) &= \sum_{i=1}^m e^{-\langle A_i, x + \frac{1}{2}y\rangle} = \sum_{i=1}^m e^{-\langle A_i, x\rangle} e^{-\frac{1}{2}\langle A_i, y\rangle} \\
&\leq \Phi(x) \cdot \sum_{i=1}^m \lambda_i (1 - \frac{1}{2}\langle A_i, y\rangle + \frac{1}{4}\langle A_i, y\rangle^2) \\
&\leq \Phi(x) \cdot (1 - \frac{1}{4}\|y\|_2^2).
\end{aligned}$$

□

Thus as long as $\|y\|_2 \geq \Omega(\frac{1}{n})$, gradient descent will decrease the potential function by a factor of $e^{-\Theta(1/n^2)}$ in each iteration, and so in at most $O(n^2 \ln(m))$ iterations we arrive at a point x with $\Phi(x) < 1$.

3.3.2 Modified Gradient Descent

With $\Delta = \Theta(\frac{1}{n})$, the standard gradient descent approach implements the initial phase of Algorithm 3 in $\tilde{O}(n^2)$ iterations. It turns out we can get the same guarantee in $\tilde{O}(n)$ iterations

by choosing a more sophisticated update direction. While we do not know how to guarantee an update direction that decreases $\Phi(x)$ by factor of more than $e^{-\Theta(\|y\|^2)}$, we are able to decrease the *product* of $\Phi(x)$ and $\|\nabla\Phi(x)\|_2$ a lot faster. First we give general bounds for the change in Φ and $\|\nabla\Phi\|_2$ under an arbitrary update step.

Lemma 29. *For any $0 < \varepsilon \leq 1$ and $p \in \mathbb{R}^n$ with $\|p\|_2 \leq 1$, we have*

$$\Phi(x + \varepsilon p) \leq \Phi(x) \cdot (1 - \varepsilon\langle y, p\rangle + \varepsilon^2 p^T M p).$$

Proof. Notice that since $\|p\|_2 \leq 1$ and $\|A_i\|_2 = 1$ we have $|\langle A_i, \varepsilon p\rangle| \leq 1$ by the Cauchy-Schwarz inequality. Writing out the definitions we obtain

$$\begin{aligned}
\Phi(x + \varepsilon p) &= \sum_{i=1}^{m} e^{-\langle A_i, x+\varepsilon p\rangle} \\
&= \sum_{i=1}^{m} e^{-\langle A_i, x\rangle} e^{-\varepsilon\langle A_i, p\rangle} \\
&\overset{(*)}{\leq} \sum_{i=1}^{m} e^{-\langle A_i, x\rangle}(1 - \varepsilon\langle A_i, p\rangle + \varepsilon^2\langle A_i, p\rangle^2) \\
&= \Phi(x) \cdot \sum_{i=1}^{m} \lambda_i(1 - \varepsilon\langle A_i, p\rangle + \varepsilon^2 p^T A_i A_i^T p) \\
&= \Phi(x) \cdot (1 - \varepsilon\langle y, p\rangle + \varepsilon^2 p^T M p).
\end{aligned}$$

In $(*)$ we use the estimate that for any $z \in \mathbb{R}$ with $|z| \leq 1$ one has $e^z \leq 1 + z + z^2$. □

Lemma 30. *For any $0 < \varepsilon \leq 1$ and $p \in \mathbb{R}^n$ with $\|p\|_2 \leq 1$, we have*

$$\|\nabla\Phi(x + \varepsilon p)\|_2 \leq \|\nabla\Phi(x)\|_2 \cdot \left(1 - \varepsilon\frac{\langle y, Mp\rangle}{\|y\|_2^2} + \varepsilon^2\Big(\frac{\langle p, Mp\rangle}{\|y\|_2} + \frac{\|Mp\|_2^2}{\|y\|_2^2}\Big)\right).$$

Proof. For any z with $|z| \leq 1$, we have $e^z = 1 + z + \eta z^2$ for some $\eta \in \mathbb{R}$ with $|\eta| \leq 1$. In particular, since $\|A_i\|_2 = 1$ and $\|p\|_2 \leq 1$, we have $|\langle A_i, \varepsilon p\rangle| \leq 1$ and so we have such an η_i

for each i.

$$\begin{aligned}
\frac{\|\nabla\Phi(x+\varepsilon p)\|_2}{\Phi(x)} &= \frac{1}{\Phi(x)}\Big\|\sum_{i=1}^{m}(-A_i)\cdot e^{-\langle A_i,x+\varepsilon p\rangle}\Big\|_2 \\
&= \frac{1}{\Phi(x)}\Big\|\sum_{i=1}^{m}(-A_i)e^{-\langle A_i,x\rangle}e^{-\varepsilon\langle A_i,p\rangle}\Big\|_2 \\
&= \Big\|\sum_{i=1}^{m}(-A_i)\cdot\lambda_i\cdot\Big(1-\varepsilon\langle A_i,p\rangle+\varepsilon^2\cdot\eta_i\langle A_i,p\rangle^2\Big)\Big\|_2 \\
&\le \Big\|\sum_{i=1}^{m}(-A_i)\cdot\lambda_i\cdot(1-\varepsilon\langle A_i,p\rangle)\Big\|_2+\varepsilon^2\cdot\Big\|\sum_{i=1}^{m}(-A_i)\cdot\lambda_i\eta_i\langle A_i,p\rangle^2\Big\|_2 \\
&\le \Big\|\sum_{i=1}^{m}\lambda_iA_i-\varepsilon\sum_{i=1}^{m}\lambda_iA_i\langle A_i,p\rangle\Big\|_2+\varepsilon^2\cdot\sum_{i=1}^{m}\underbrace{\|A_i\|_2}_{=1}\cdot\lambda_i\cdot\underbrace{|\eta_i|}_{\le 1}\cdot\langle A_i,p\rangle^2 \\
&\le \Big\|y-\varepsilon\cdot Mp\Big\|_2+\varepsilon^2\cdot\sum_{i=1}^{m}\lambda_i\langle A_i,p\rangle^2 \\
&= \left(\|y\|_2^2-2\varepsilon y^TMp+\varepsilon^2\|Mp\|_2^2\right)^{1/2}+\varepsilon^2p^TMp \\
&\le \|y\|_2\cdot\left(1+2\frac{1}{\|y\|_2^2}\left(-\varepsilon\langle y,Mp\rangle+\varepsilon^2\|Mp\|_2^2\right)\right)^{1/2}+\varepsilon^2p^TMp \\
&\le \|y\|_2\left(1+\frac{1}{\|y\|_2^2}\left(-\varepsilon\langle y,Mp\rangle+\varepsilon^2\|Mp\|_2^2\right)\right)+\varepsilon^2p^TMp \\
&\le \|y\|_2\left(1+\frac{1}{\|y\|_2}\varepsilon^2p^TMp+\frac{1}{\|y\|_2^2}\left(-\varepsilon\langle y,Mp\rangle+\varepsilon^2\|Mp\|_2^2\right)\right)
\end{aligned}$$

Recalling that $\nabla\Phi(x)=-\Phi(x)y$ finishes the proof. □

We remark that in the case when y is an eigenvector of M with eigenvalue α, we can choose $p=\frac{y}{\|y\|}$ and $\varepsilon=\min\{\frac{1}{2},\frac{\|y\|_2}{4\alpha}\}$ and guarantee that $\Phi(x)\|\nabla\Phi(x)\|_2$ decreases by a factor of $e^{-\Omega(\|y\|_2)}$. For the general case, it turns out that it suffices to find a vector p that is close enough in angle to y and so that either Mp is also close in angle to y, or $\|Mp\|_2$ is very small.

Lemma 31. *Suppose $p\in\mathbb{R}^n$ with $\|p\|_2\le 1$ and constant $a>0$ is such that either*

1. *$\langle y,p\rangle\ge\frac{\|y\|_2}{(\log n)^a}$ and $\langle y,Mp\rangle\ge\frac{\|Mp\|_2\cdot\|y\|_2}{(\log n)^a}$ or*
2. *$\langle y,p\rangle\ge\frac{\|y\|_2}{(\log n)^a}$ and $\|Mp\|_2\le O\left(\frac{1}{poly(n)}\right)$.*

Then as long as $\|y\|_2 \geq \Omega(\frac{1}{n})$*, choosing* $\varepsilon = \min\left\{\frac{\|y\|_2}{4(\log n)^{2a}\|Mp\|_2}, \frac{1}{2(\log n)^a}\right\}$ *gives*

$$\|\nabla\Phi(x+\varepsilon p)\|_2 \cdot \Phi(x+\varepsilon p) \leq \|\nabla\Phi(x)\|_2 \cdot \Phi(x) e^{-\tilde{\Theta}(1/n)}.$$

Proof. Let $\varepsilon = \min\left\{\frac{\|y\|_2}{4(\log n)^{2a}\|Mp\|_2}, \frac{1}{2(\log n)^a}\right\}$. Then by Lemma 29, we have

$$\Phi(x+\varepsilon p) \leq \Phi(x) \cdot (1 - \varepsilon\langle y, p\rangle + \varepsilon^2\|Mp\|_2) \leq \Phi(x) \cdot (1 - \varepsilon\frac{\|y\|_2}{(\log n)^a} + \varepsilon^2\|Mp\|_2).$$

Assume first that we are in Case 1. By Lemma 30, since we know $\varepsilon \leq \frac{1}{2(\log n)^a}$, we have

$$\|\nabla\Phi(x+\varepsilon p)\|_2 \leq \|\nabla\Phi(x)\|_2 \cdot \Big(1 - \frac{\varepsilon}{2}\frac{\|Mp\|_2}{\|y\|_2(\log n)^a} + \varepsilon^2\frac{\|Mp\|_2^2}{\|y\|_2^2}\Big).$$

If $\varepsilon = \frac{1}{2(\log n)^a}$, then $\|Mp\|_2 \leq \frac{\|y\|_2}{2(\log n)^a}$. Using this, $\Phi(x)$ will decrease by $e^{-\tilde{\Theta}(\|y\|_2)}$, and $\|\nabla\Phi(x)\|_2$ will decrease.

On the other hand, if $\varepsilon = \frac{\|y\|_2}{4(\log n)^{2a}\|Mp\|_2}$, then $\Phi(x)$ will decrease, and $\|\nabla\Phi(x)\|_2$ decreases by $e^{-\tilde{\Theta}(1)}$. Together, these show that the product decreases by a factor of $e^{-\tilde{\Theta}(\|y\|_2)}$.

If we are in Case 2, the only thing that might change is that when $\varepsilon = \frac{1}{2(\log n)^a}$ we might have $\|\nabla\Phi(x)\|_2$ increase by up to $e^{O(1/\mathrm{poly}(n))}$. Since $\|y\|_2 \geq \Omega(\frac{1}{n})$ this is the dominating term, and so we will still get the appropriate decrease. □

The idea for computing a direction p to satisfy Lemma 31 is to project y onto an appropriate eigenspace of M. More formally, suppose $M = \sum_{j=1}^n \alpha_j v_j v_j^T$ is the eigendecomposition of M. Notice that $\alpha_j \in [0,1]$ for all j.

Given $K = \mathrm{polylog}(n)$, define $S_k = \mathrm{span}\{v_j \mid 2^{-k} \leq \alpha_j \leq 2^{-k+1}\}$ for $1 \leq k < K$ and define $S_K = \mathrm{span}\{v_j \mid \alpha_j \leq 2^{-K+1}\}$. If $z_k = \mathrm{proj}_{S_k} z$, where $z = \frac{y}{\|y\|_2}$, there must be some $k \leq K$ with $\langle z, z_k\rangle \geq \frac{1}{K}$. Now, if $k < K$, then we have $\langle z, Mz_k\rangle \geq \frac{1}{2K}\|Mz_k\|_2$, and hence z_k satisfies Case 1 of Lemma 31. If $k = K$, then we have $\|Mz_K\|_2 \leq \frac{1}{2^K}$, and hence Case 2 of Lemma 31 is satisfied. Unfortunately, computing these S_k would require computing the eigendecomposition of M. To speed up the computation of this vector, we instead do the following.

Lemma 32. *Suppose y and M are as above, and define $z = \frac{y}{\|y\|_2}$. For $k = 1, ..., K := polylog(n)$, define $z_k = (I - \frac{1}{2}M)^{2^k} z$. Then for some $k \leq K$, $p = z_k$ satisfies the conditions of Lemma 31.*

Such an update direction can be computed in time $\tilde{O}(mn^{\omega-1})$. By Lemmas 31 and 32 taking this step decreases $\Phi(x)\|\nabla\Phi(x)\|_2$ by $e^{-\Omega(1/n)}$ provided $\|\lambda A\|_2 \geq \Omega(\frac{1}{n})$. After $\tilde{O}(n)$ updates we obtain x with $\|\lambda A\|_2 \cdot \Phi(x)^2 \leq O(\frac{1}{n})$. This implies that either $\Phi(x) < 1$, in which case $x \in P$, or $\|\lambda A\|_2 \leq O(\frac{1}{n})$, and then we can rescale. This completes the proof of Theorem 27. In the remainder of this section, we give the proof of Lemma 32.

Proof. Let $N = \frac{1}{2}M$ and suppose the eigenvectors of N are unit vectors $v_1, ..., v_n$ with eigenvalues $\alpha_1, ..., \alpha_n \leq \frac{1}{2}$. We compute

$$z_k = \sum_{j=1}^{n} (1 - \alpha_j)^{2^k} \langle z, v_j \rangle v_j.$$

$$Nz_k = \sum_{j=1}^{n} \alpha_j (1 - \alpha_j)^{2^k} \langle z, v_j \rangle v_j.$$

For any k, we can get the following bounds.

(1) If $\alpha_j \geq \frac{k}{2^k}$, then $(1 - \alpha_j)^{2^k} \leq e^{-\alpha_j 2^k} \leq e^{-k} \leq 2^{-k}$.

(2) If $\alpha_j \leq \frac{1}{2^k}$, then $(1 - \alpha_j)^{2^k} \geq e^{-2\alpha_j 2^k} \geq e^{-2}$.

In particular, (1) implies that $\alpha_j(1 - \alpha_j)^{2^k} \leq \frac{k}{2^k}$ for all j, and hence $\|Nz_k\|_2 \leq \frac{k}{2^k}$. On the other hand, (2) implies that $\langle z, z_k \rangle \geq \frac{1}{Ke^2}$ for some $k \leq K$. If $k < K$, we also have $\langle z, Nz_k \rangle \geq \frac{1}{Ke^2} 2^{-k-1}$. Combining these, we are guaranteed there exists $k \leq K$ with $\langle z, z_k \rangle \geq \frac{1}{e^2 K}$ and so that either $\langle z, Mz_k \rangle \geq \frac{\|Mz_k\|_2}{2e^2K^2}$ if $k < K$, or $\|Mz_K\|_2 \leq \frac{K}{2^{K-1}}$. In particular, this z_k satisfies the conditions of Lemma 31. □

3.4 Computing an Approximate John Ellipsoid

It turns out that our algorithm implicitly computes an approximate John ellipsoid for the considered cone P. Recall that a classical theorem of John [Joh48] shows that for any closed, convex set $Q \subseteq \mathbb{R}^n$, there is an ellipsoid E and a center z so that $z + E \subseteq Q \subseteq z + nE$. The bound of n is tight in general — for example for a simplex — but it can be improved to $\sqrt{n}$ for symmetric sets. This is equivalent to saying that for each convex body, there is a linear transformation that makes it *n-well rounded*. Here, a body Q is α-well rounded if $z + r \cdot B \subseteq Q \subseteq z + \alpha \cdot r \cdot B$ for some center $z \in \mathbb{R}$ and radius $r > 0$ [Bal97].

Suppose first that we use the modified gradient descent version of our algorithm but we only terminate if $\Phi(x) < \frac{1}{e}$. Note that this change in the algorithm can only increase the worst case running times by a constant factor. The final MWU phase will terminate in at most $T = \tilde{O}(n)$ steps, and the step size is always bounded by $\frac{1}{2}$. Therefore if the final MWU phase outputs x we have $\|x\|_2 \leq \frac{T}{2}$ and $\langle A_i, x\rangle \geq 1$ for all i. In particular, we have $B(\frac{1}{T}x, \frac{1}{T}) \subseteq P \cap B \subseteq B(\frac{1}{T}x, 2)$, which shows that $P \cap B$ is $\tilde{O}(n)$-well rounded.

Alternatively one can run the algorithm with standard gradient descent and a fixed step size of $\varepsilon := \Theta(\frac{1}{n})$ and only terminate when $\Phi(x) < \frac{1}{m}$. In the final phase we then obtain that $\|x\|_2 \leq O(n \ln m)$, and $\langle A_i, x\rangle \geq \ln m$ for all i. Therefore the final set $P \cap B$ will be $O(n)$-well rounded, thus removing the logarithmic terms suppressed by the $\tilde{O}$ notation.

3.5 Another possible speed-up

An alternate way to improve the running time of the MWU phase is by noting that we can actually rescale under somewhat weaker conditions. Recall that Lemma 28 with $p = \frac{y}{\|y\|_2}$ guarantees that $\Phi(x+\varepsilon p) \leq \Phi(x) \cdot (1 - \varepsilon\|y\|_2 + \varepsilon^2 p^T M p)$. Provided $p^T M p \leq O(\sqrt{n}\|y\|_2)$, we can choose $\varepsilon = \Theta(\frac{1}{\sqrt{n}})$ to get a decrease of $e^{-\Omega(\|y\|_2/\sqrt{n})}$. In particular, in $\tilde{O}(n^{3/2})$ iterations, the MWU phase guarantees one of three conditions: $x \in P$, $\|y\|_2 \leq O(\frac{1}{n})$, or $p^T M p \geq \Omega(\sqrt{n}\|y\|_2)$. We already showed that the second condition allows us to rescale – here we will show that we can also rescale using the last condition. First note that we have the following:

Lemma 33. *Suppose $x \in P \cap B$. Then $\|Mx\|_2 \leq \|\lambda A\|_2$. In particular, if we have the eigendecomposition $M = \sum_{j=1}^n \delta_j v_j v_j^T$, where $\|v_j\|_2 = 1$, then $\text{width}(P, v_j) \leq \frac{\|\lambda A\|_2}{\delta_j}$ for all j with $\delta_j > 0$.*

Proof. Note that for any $x \in P \cap B$, we have $\|Mx\|_2 \leq \sum_{i=1}^m \lambda_i |\langle A_i, x\rangle| \leq \|\lambda A\|_2$. On the other hand, we have $Mx = \sum_{j=1}^n \delta_j v_j \langle v_j, x\rangle \implies \|Mx\|_2^2 = \sum_{j=1}^n \delta_j^2 \langle v_j, x\rangle^2$. Therefore $\sum_{j=1}^n \delta_j^2 \langle v_j, x\rangle^2 \leq \|\lambda A\|_2^2$. In particular, for any $x \in P \cap B$ and for any j, we have $|\langle v_j, x\rangle| \leq \frac{\|\lambda A\|_2}{\delta_j}$. Therefore for any j we have $\text{width}(P, v_j) \leq \frac{\|\lambda A\|_2}{\delta_j}$. □

Using the results of Section 3.2, if $\delta_{\max} \geq p^T M p \geq \sqrt{n} \cdot \|\lambda A\|_2$, we can rescale to get a constant increase in the volume of $P \cap B$. We also remark that in the case when $\|\lambda A\|_2 \leq O(\frac{1}{n})$, Lemma 33 allows us to eliminate the square root from the transformation of Lemma 25 – that is, the function $F(x) = (I + \frac{1}{\delta_{\max}} M)x$ also allows us to rescale. In fact, we can use Lemma 33 to show a more general rescaling that works under either of the rescaling hypotheses. Let $\|\cdot\|_F$ denote the Frobenius norm, that is, $\|M\|_F^2 = \sum_{j=1}^n \delta_j^2$.

Lemma 34. *Suppose $n\|\lambda A\|_2^2 \leq \frac{1}{4}\|M\|_F^2$. Then the function $F(x) = (I + \frac{1}{\delta_{\max}^2} M^2)^{1/2} x$ gives*

$$\text{vol}(F(P) \cap B) \geq \exp(\|M\|_F^2 / 8\delta_{\max}^2) \cdot \text{vol}(P \cap B).$$

Proof. The eigenvalues of F take the form $(1 + \frac{\delta_j^2}{\delta_{\max}^2})^{1/2} \geq \exp(\delta_j^2 / 4\delta_{\max}^2)$. Therefore we have $\det(F) \geq \exp(\|M\|_F^2 / 4\delta_{\max}^2)$. On the other hand, for any $x \in P \cap B$, we have

$$\|F(x)\|_2^2 = \|x\|_2^2 + \frac{1}{\delta_{\max}^2}\|Mx\|_2^2 \leq 1 + \frac{1}{\delta_{\max}^2}\|\lambda A\|_2^2 \leq \exp(\|\lambda A\|_2^2 / \delta_{\max}^2),$$

and so intersecting with the unit ball decreases the volume by at most $\exp(n\|\lambda A\|_2^2 / 2\delta_{\max}^2)$. By our hypothesis, this is bounded by $\exp(\|M\|_F^2 / 8\delta_{\max}^2)$. □

Note that the improvement will always be at least a constant since $\sum_{j=1}^n \delta_j^2 \geq \delta_{\max}^2$, and it is also possible that it could get up to exponential improvement. Rescaling under these weaker conditions allows us to get the following.

Lemma 35. *The version of the algorithm described above has an overall running time of $\tilde{O}(mn^{3.5} \log 1/\rho)$.*

Proof. In at most $\tilde{O}(n^{3/2})$ iterations, standard gradient descent with step size $\varepsilon = \Theta(\frac{1}{\sqrt{n}})$ guarantees $x \in P$, $\|\lambda A\|_2 \leq \Theta(\frac{1}{n})$, or $\sqrt{n}\|\lambda A\|_2 \leq \Theta(p^T M p)$. Each iteration takes time $O(mn)$ and so the MWU phase takes total time $\tilde{O}(mn^{2.5})$ until we are able to rescale. We perform at most $\tilde{O}(n \log \frac{1}{\rho})$ rescalings, and so the total running time is $\tilde{O}(mn^{3.5} \log \frac{1}{\rho})$. □

Independent publication. The multi-rank rescaling was also discovered in a parallel and independent work by Dadush, Vegh and Zambelli [DVZ16] (see their Algorithm 5).

Chapter 4

NUMBER BALANCING IS AS HARD AS MINKOWSKI'S THEOREM

4.1 Introduction

One of *six basic NP-complete problems* of Garey and Johnson [GJ79] is the *partition problem* that for a list of numbers $a_1, \ldots, a_n$ asks whether there is a partition of the indices so that the sums of the numbers in both partitions coincide. Partition and related problems like knapsack, subset sum and bin packing are some of the fundamental classical problems in theoretical computer science with numerous practical applications; see for example the textbooks [MT90, KPP04] and the article of Mertens [Mer06]. In this paper, we study a variant called the *number balancing problem* (NBP), where the goal is to find two disjoint subsets $I_1, I_2 \subseteq \{1, \ldots, n\}$ so that the difference $|\sum_{i \in I_1} a_i - \sum_{i \in I_2} a_i|$ is minimized. Equivalently, given a vector of numbers $a = (a_1, \ldots, a_n) \in [0,1]^n$, we want to find a vector of *signs* $x \in \{-1, 0, 1\}^n \setminus \{\mathbf{0}\}$ so that $|\langle a, x \rangle| = |\sum_{i=1}^n x_i a_i|$ is minimized. Woeginger and Yu [WY92] studied this problem under the name "equal-subset-sum" and showed that it is NP-hard to decide whether there are two disjoint subsets that sum up to the exact same value. This version has also been extensively studied in combinatorics [Lun88, Boh96, LY11].

On the positive side, it is not hard to prove that there is always a solution with exponentially small error. Suppose that $a_1, \ldots, a_n \in [0,1]$. Consider the list of 2^n many numbers $\sum_{i=1}^n a_i x_i$ for all $x \in \{0,1\}^n$. All these numbers fall into the interval $[0, n]$, hence by the *pigeonhole principle*, we can find two distinct vectors $x, x' \in \{0,1\}^n$ with $|\sum_{i=1}^n a_i x_i - \sum_{i=1}^n a_i x'_i| \leq \frac{n}{2^n - 1}$. Then $x - x'$ gives the desired solution. Note that the bound can be slightly improved to $O(\frac{\sqrt{n}}{2^n})$ by using the fact that due to *concentration of measure* effects, for a constant fraction of vectors $x \in \{0,1\}^n$, the sums $\sum_{i=1}^n a_i x_i$ fall into an interval

of length $\sqrt{n}$ (instead of n).

However, since these arguments rely on the pigeonhole principle, they are non-constructive. Restricting the non-constructive argument to polynomially many "pigeons" provides a simple polynomial time algorithm to find at least an $x \in \{-1, 0, 1\}^n \setminus \{\mathbf{0}\}$ with $|\langle a, x\rangle| \leq \frac{1}{\text{poly}(n)}$ for an arbitrarily small polynomial. Interestingly, the only known polynomial time algorithm that gives a better guarantee is Karmarkar and Karp's *differencing algorithm* [KK82a] which provides the bound $|\langle a, x\rangle| \leq n^{-c\log(n)}$ for some constant $c > 0$. Their algorithm uses a recursive scheme; find $\Theta(n)$ pairs of numbers a_i of distance at most $\Theta(\frac{1}{n})$ and create an instance consisting of their differences, then recurse.

This leads to the natural question: *Given $a_1, \ldots, a_n \in [0, 1]$, what upper bound on $|\sum_{i=1}^n a_i x_i|$ can be guaranteed if $x \in \{-1, 0, 1\}^n \setminus \{\mathbf{0}\}$ is to be chosen in polynomial time?* While answering this question directly seems out of reach, we note that NBP falls into the class PPP [Pap94], where good solutions are known to exist due to the pigeonhole principle. It is reasonable therefore to study the relationship between this problem and other problems in PPP.

Recall that given linearly independent vectors $b_1, \ldots, b_n \in \mathbb{R}^n$, a *(full rank) lattice* is the set $\Lambda := \{\sum_{i=1}^n \lambda_i b_i : \lambda_i \in \mathbb{Z}\ \forall i = 1, \ldots, n\}$. The set $\{b_1, \ldots, b_n\}$ is called a *basis* for Λ and we define $\det(\Lambda) := |\det(B)|$. For any lattice $\Lambda \subseteq \mathbb{R}^n$ with $\det(\Lambda) \geq 1$, *Minkowski's Theorem* tells us that any symmetric convex body $K \subseteq \mathbb{R}^n$ of volume at least 2^n must intersect $\Lambda \setminus \{\mathbf{0}\}$, see for example [Mat02]. This theorem is proven by placing translates of $\frac{1}{2}K$ at any lattice point and then inferring an overlap due to the pigeonhole principle. Again, one can consider the algorithmic question: *given a symmetric convex body K with volume at least 2^n, for what factor ρ can one be guaranteed to find an $x \in (\rho K) \cap \mathbb{Z}^n \setminus \{\mathbf{0}\}$ in polynomial time?*

We would like to point out that this factor ρ is within a polynomial factor of the approximability of the Shortest Vector Problem (SVP), the problem of finding the shortest[1] nonzero vector in a lattice. One direction follows from the fact K can be sandwiched between

[1]SVP can be defined for any norm, but anywhere the norm is not specified we consider the Euclidean norm.

two ellipsoids that differ by a factor of $\sqrt{n}$ [Joh48]. In the other direction, a reduction of Lenstra and Schnorr shows that given a polynomial-time oracle to find a vector of length at most $f(n)$ in a lattice with $\det(\Lambda) \leq 1$, there is a polynomial-time algorithm to find a vector within $f(n)^2$ of the shortest vector [Lov86]. This is a nontrivial reduction that uses the assumed oracle both on the original lattice and on its dual. Note that Minkowski's theorem guarantees the existence of a lattice vector of length at most $O(\sqrt{n})$.

The complexity of SVP is of great theoretical and practical interest. As a rarity in theoretical computer science, SVP admits (NP $\cap$ coNP)-certificates for a value that is at most a factor $O(\sqrt{n})$ away from the optimum [AR05], while the best known hardness under reasonable complexity assumptions lies at a subpolynomial bound of $n^{\Theta(1/\log\log n)}$ [HR07]. The famous LLL-algorithm [LLL82] can find a $2^{n/2}$-approximation in polynomial time (the generalized *block reduction method* of Schnorr [Sch87] brings the factor down to $2^{n\log\log(n)/\log(n)}$). On the other hand, a polynomial factor approximation of SVP would be enough to break lattice-based cryptosystems [Ajt96, MR09].

It is not hard to use an *exact* oracle for Minkowski's Theorem to find a good number balancing solution, since the body $K := \{x \in (-2, 2) : |\sum_{i=1}^{n} x_i a_i| \leq \Theta(\frac{n}{2^n})\}$ has volume at least 2^n. However, it is not clear how we could use an *approximate* oracle. For example, it is known that the LLL-algorithm can be used to find a nonzero integer vector $x \in \rho K$ for a factor of $\rho = \text{poly}(n) \cdot 2^{n/2}$. While the error guarantee of $|\sum_{i=1}^{n} a_i x_i| \leq \text{poly}(n) \cdot 2^{-n/2}$ outperforms the Karmarkar-Karp algorithm, we only know that $\|x\|_\infty < 2\rho$, which means that x will not be a valid solution if $\rho > 1$.[2] This leads us to the next question: *what factor ρ is needed for Minkowski's Theorem to improve over Karmarkar-Karp's bound*?

We have seen that in a certain sense NBP can be reduced to an oracle for Minkowski's Theorem, and in fact we will show that there is also a direct reduction to SVP in the ℓ^∞ norm. This brings us to the question about the reverse: *given an oracle that solves NBP within an exponentially small error, can this give a non-trivial oracle for the Shortest Vector*

[2] If $\rho \leq 2 - \varepsilon$, then one can still obtain an error of $|\sum_{i=1}^{n} a_i x_i| \leq 2^{-\Theta(\varepsilon n)}$, but this breaks down if $\rho \geq 2$.

Problem or Minkowski's Theorem?

4.1.1 Our Contribution

In this work, we provide some answers to the questions raised above, by relating the complexity of the number balancing problem to Minkowski's Theorem and the Shortest Vector Problem. First we give the precise definitions of the problems we will consider.

Definition 2. *Suppose $p \in [1, \infty]$ with $B^p(\mathbf{0}, 1)$ the closed unit ball in the ℓ^p norm. For $\delta \geq \frac{\Omega(\sqrt{n})}{2^n}$ and $\rho \geq 1$, we define the following problems.*

- *δ-**NBP**: Given $a \in [0,1]^n$, find $x \in \{-1, 0, 1\}^n \setminus \{\mathbf{0}\}$ with $|\langle a, x \rangle| \leq \delta$.*
- *ρ-**Minkowski Problem**: Given a lattice Λ and a symmetric convex body[3] $K \subseteq \mathbb{R}^n$ with $vol_n(K) \geq 2^n \det(\Lambda)$, find a vector $x \in (\rho K) \cap \Lambda \setminus \{\mathbf{0}\}$.*
- *ρ-**PromiseSVP**$_p$: Given a lattice $\Lambda \subset \mathbb{R}^n$ with $vol(B^p(\mathbf{0}, 1)) \geq 2^n \det(\Lambda)$, find a vector $x \in \Lambda \setminus \{\mathbf{0}\}$ with $\|x\|_p \leq \rho$.*

As already discussed, δ-NBP and the ρ-Minkowski Problem will always have a solution by nonconstructive arguments. Moreover, we notice that ρ-PromiseSVP$_p$ is just the ρ-Minkowski Problem on an ℓ^p ball, and so it is also guaranteed to have a solution. Notice also that ρ-PromiseSVP$_p$ would also follow immediately from a ρ-approximation to SVP in the ℓ^p norm, since a short enough vector is guaranteed to exist. We would like to stress that polynomial-time algorithms for any of the three problems are not known to be inconsistent with P $\neq$ NP. The hardness results of SVP do not apply to ρ-PromiseSVP$_p$ since we do not require it to give a shortest vector of the lattice.

We provide the following reduction:

---

[3]We assume K is given to us by a separation oracle.

Theorem 36. *Suppose there is a polynomial-time algorithm for the ρ-Minkowski problem for polytopes K with $O(n)$ facets. Then there is a polynomial-time algorithm for δ-NBP where $\delta := 2^{-n^{\Theta(1/\rho)}}$.*

In fact, to obtain an algorithm for δ-NBP, it suffices to have an approximate Minkowski oracle for the linear transformation of a cube, which is equivalent to an oracle for ρ-PromiseSVP$_\infty$.

Theorem 37. *Suppose that there is a polynomial-time algorithm for ρ-PromiseSVP$_\infty$. Then there is a polynomial-time algorithm for δ-NBP, where $\delta := 2^{-n^{\Theta(1/\rho)}}$.*

In particular an oracle for $\rho \leq c' \log(n)/\log\log(n)$ would imply an improvement over Karmarkar-Karp's algorithm, where $c' > 0$ is a small enough constant.

Finally, we can also prove that an oracle with exponentially small error for number balancing would provide an approximation for Minkowski's problem:

Theorem 38. *Suppose that there is a polynomial-time algorithm for δ-NBP with $\delta \leq 2^{\sqrt{n}}/2^n$. Then there is a polynomial-time algorithm for the ρ-Minkowski problem for $\rho = O(n^5)$. Here it suffices to have a separation oracle for the convex body $K \subseteq \mathbb{R}^n$.*

In fact, we will show that when K is an ellipsoid whose volume is that of a unit Euclidean ball, we can get within a factor of $O(n^{4.5})$. Using the reduction of Lenstra and Schnorr, this implies a $O(n^9)$ approximation for SVP.

4.2 Reducing Number Balancing to Minkowski's Theorem

In this section we will show how to solve NBP with an oracle for Minkowski's problem. The idea is to consider a hypercube intersected with the constraint $|\langle a, x\rangle| \leq \delta$, and to show that this set has large enough volume. If we have an exact Minkowski oracle, this gives us $x \in \{-1, 0, 1\}^n \setminus \{\mathbf{0}\}$ as desired. Here we state a more general version which uses only a ρ-approximate Minkowski oracle, and then show how we can use this more general version to solve NBP with a weaker bound. We present the proof in the full version of this paper.

Theorem 39. *Suppose we have a polynomial-time algorithm for the ρ-Minkowski problem, and let $k > 0$ be any positive integer. Then, for any $a \in [0,1]^n$, there is a polynomial-time algorithm to find $x \in \mathbb{Z}^n \setminus \{\mathbf{0}\}$ with $\|x\|_\infty \leq k$ and so that $|\langle a, x\rangle| \leq n\left(\frac{\rho}{k+1}\right)^{n-1}$.*

Proof. For ease of notation, let

$$\delta := n\left(\frac{\rho}{k+1}\right)^{n-1}.$$

Now consider the body

$$K := \left\{x \in \left(-\frac{k+1}{\rho}, \frac{k+1}{\rho}\right)^n : |\langle a, x\rangle| \leq \delta\right\}.$$

Obviously this is a symmetric convex body. For $\alpha \in [-\delta, \delta]$, consider the $(n-1)$-dimensional slice

$$K(\alpha) := \left\{x \in \left(-\frac{k+1}{\rho}, \frac{k+1}{\rho}\right)^n : \langle a, x\rangle >= \alpha\right\}$$

of it. Let us consider the $(n-1)$-dimensional volume $\mathrm{vol}_{n-1}(K(\alpha))$ of that slice. Then we can write the volume of K as

$$\mathrm{vol}_n(K) = \int_{-\delta}^{\delta} \mathrm{vol}_{n-1}(K(\alpha))\, d\alpha.$$

Moreover

$$\left(\frac{2(k+1)}{\rho}\right)^n = \mathrm{vol}_n\left(-\frac{k+1}{\rho}, \frac{k+1}{\rho}\right)^n = \int_{-\frac{n(k+1)}{\rho}}^{\frac{n(k+1)}{\rho}} \mathrm{vol}_{n-1}(K(\alpha))\, d\alpha.$$

since the slices are empty if $|\alpha| > \frac{n(k+1)}{\rho}$. By symmetry $\mathrm{vol}_{n-1}(K(\alpha)) = \mathrm{vol}_{n-1}(K(-\alpha))$. Moreover, by convexity of K, for $\alpha \geq 0$, the quantity $\mathrm{vol}_{n-1}(K(\alpha))$ is monotonically non-increasing. Thus

$$\begin{aligned}\mathrm{vol}_n(K) = \int_{-\delta}^{\delta} \mathrm{vol}_{n-1}(K(\alpha))\, d\alpha &\geq \frac{\delta}{\left(\frac{n(k+1)}{\rho}\right)} \cdot \int_{-\frac{n(k+1)}{\rho}}^{\frac{n(k+1)}{\rho}} \mathrm{vol}_{n-1}(K(\alpha))\, d\alpha \\ &= \frac{\delta}{\left(\frac{n(k+1)}{\rho}\right)} \cdot \left(\frac{2(k+1)}{\rho}\right)^n.\end{aligned}$$

Now, using the fact that $\delta = n\left(\frac{\rho}{(k+1)}\right)^{n-1}$, we get $\mathrm{vol}_n(K) \geq 2^n$. Hence, using our ρ-approximate Minkowski oracle, we can find a vector

$$x \in (\rho K \cap \mathbb{Z}^n) \setminus \{\mathbf{0}\} = ((-(k+1),(k+1))^n \cap \mathbb{Z}^n) \setminus \{\mathbf{0}\}.$$

In particular, this gives $x \in \mathbb{Z}^n$ with $|\langle a, x\rangle| \leq n\left(\frac{\rho}{k+1}\right)^{n-1}$ and $\|x\|_\infty \leq k$. □

Suppose, for instance, that $\rho = (2-\epsilon)$ for some $\epsilon \in (0,1]$. Then we can pick $k = 1$ and get $|\langle a, x\rangle| \leq 2^{-\Theta(\epsilon n)}$ with $\|x\|_\infty \leq 1$ and $x \in \mathbb{Z}^n \setminus \{\mathbf{0}\}$. However, this line of arguments breaks down for $\rho \geq 2$ as these would in general not produce feasible solutions for number balancing.

Instead of using an oracle for the ρ-Minkowski Problem one can directly use an oracle for ρ-PromiseSVP$_\infty$. We need the following theorem:

Theorem 40. *Suppose that there is a polynomial-time algorithm for ρ-PromiseSVP$_\infty$. Then for any $a \in [0,1]^n$ there is a polynomial-time algorithm to find $x \in \mathbb{Z}^n \setminus \{\mathbf{0}\}$ with $\|x\|_\infty \leq k$ and so that $|\langle a, x\rangle| \leq 2nk\rho(\frac{\rho}{k})^n$.*

Proof. When $\rho(\frac{\rho}{k})^n \geq \frac{1}{2}$, we have $2nk\rho(\frac{\rho}{k})^n \geq 1$, in which case $x = (1, 0, \cdots, 0)$ trivially does the job. We can hence assume that $\rho(\frac{\rho}{k})^n < \frac{1}{2}$.

Let I_n be the $n \times n$ identity matrix and consider the lattice Λ generated by the $(n+1) \times (n+1)$-dimensional matrix

$$B := \begin{pmatrix} \frac{\rho}{k} I_n & \mathbf{0} \\ \frac{1}{2nk}(\frac{k}{\rho})^n a^T & (\frac{k}{\rho})^n \end{pmatrix}.$$

Note that $\det(B) = 1$. Let $x \in \Lambda$ be the vector returned by the algorithm. Then $y := B^{-1}x \in \mathbb{Z}^{n+1}$. Since $\|x\|_\infty \leq \rho$ and $x = By$, we have $|y_i| \leq k$ for $i = 1, \ldots, n$.

Moreover,

$$\begin{aligned}\rho \geq |x_{n+1}| &= \left| y_{n+1}\left(\frac{k}{\rho}\right)^n + \sum_{i=1}^{n} \frac{1}{2nk}\left(\frac{k}{\rho}\right)^n y_i a_i \right| \\ &\geq |y_{n+1}|\left(\frac{k}{\rho}\right)^n - \left| \sum_{i=1}^{n} \frac{1}{2nk}\left(\frac{k}{\rho}\right)^n y_i a_i \right| \\ &\overset{|y_i|\leq k}{\geq} |y_{n+1}|\left(\frac{k}{\rho}\right)^n - \frac{1}{2}\left(\frac{k}{\rho}\right)^n,\end{aligned}$$

which implies $|y_{n+1}| \leq \frac{1}{2} + \rho(\frac{\rho}{k})^n$. Since $\rho(\frac{\rho}{k})^n < \frac{1}{2}$ and $y_{n+1} \in \mathbb{Z}$, we must have $y_{n+1} = 0$.

Therefore, $\left|\sum_{i=1}^{n} \frac{1}{2nk}\left(\frac{k}{\rho}\right)^n y_i a_i\right| = |x_{n+1}| \leq \rho$, and hence $|\sum_{i=1}^{n} y_i a_i| \leq 2nk\rho\left(\frac{\rho}{k}\right)^n$, so the vector $(y_1, \ldots, y_n)$ does the job. □

However, we still face a problem in the case $\rho \geq 2$. It turns out that we can design a *recursive self-reduction* to allow us to use larger ρ. The main technical argument is to transform an algorithm that finds $x \in \mathbb{Z}^n \setminus \{\mathbf{0}\}$ with $\|x\|_\infty \leq k$ for $k \geq 2$ into an algorithm that finds vectors $x \in \mathbb{Z}^n \setminus \{\mathbf{0}\}$ with $\|x\|_\infty \leq \frac{k}{2}$, with a bounded decay in the error $|\langle a, x\rangle|$. Applying this recursively gives the following lemma.

Lemma 41. *Suppose that there is a polynomial-time algorithm that for any $a' \in [0,1]^n$ finds a vector $x' \in \{-k, \ldots, k\}^n \setminus \{\mathbf{0}\}$ with $|\langle a', x'\rangle| \leq 2^{-n}$. If $k \leq \frac{\log n}{6 \log\log n}$, then there is also a polynomial-time algorithm that for any $a \in [0,1]^n$ finds a vector $x \in \mathbb{Z}^n$ with $|\langle a, x\rangle| \leq 2^{-n^{\frac{1}{3k}}}$ and $x \in \{-1, 0, 1\}^n \setminus \{\mathbf{0}\}$.*

Before we go through the self reduction, we show how Lemma 41 gives Theorems 36 and 37.

Proof. If $\rho \geq \frac{\log n}{48 \log\log n}$, then $2^{-n^{\Theta(1/\rho)}} = 2^{-\log^{O(1)} n}$. By choosing a proper constant on the exponent, this can be achieved with the Karmarkar-Karp algorithm. So we only need to work with $\rho < \frac{\log n}{48 \log\log n}$.

Now suppose we have a polynomial-time algorithm for the ρ-Minkowski Problem (resp. ρ-PromiseSVP$_\infty$). If we take $k = 3\rho$, Theorem 39 (resp. Theorem 40) gives a polynomial-time algorithm to find x with $\|x\|_\infty \leq k$ and $|\langle a, x\rangle| \leq 2^{-n}$.

Moreover, $k = 3\rho \leq \frac{\log n}{16 \log\log n}$, and hence the condition of Lemma 41 is satisfied. Then the bound given by Lemma 41 is $2^{-n^{\frac{1}{3k}}} \leq 2^{-n^{\Theta(1/\rho)}}$. □

We now prove Lemma 41. The way we do the self-reduction is the following. We partition our set of n numbers into subsets of size $\sqrt{n}$. First, for each subset ℓ, we find a number $b_\ell \neq 0$ for which we can (approximately) express $b_\ell, 2b_\ell, \ldots, kb_\ell$ as linear combinations of elements of that subset using only coefficients in $\{-\lfloor \frac{k}{2} \rfloor, \ldots \lfloor \frac{k}{2} \rfloor\}$. We then run our assumed algorithm on $b_1, \ldots, b_{\sqrt{n}}$ to obtain $y \in \{-k, \ldots, k\}^n$ with $\langle b, y \rangle = \sum_{\ell=1}^{\sqrt{n}} y_\ell b_\ell$ being small. Since each of the summands can be expressed more efficiently in terms of our original set of numbers, we obtain a good solution x with coefficients in $\{-\lfloor \frac{k}{2} \rfloor, \ldots \lfloor \frac{k}{2} \rfloor\}$.

The following two lemmas go through this argument more precisely. Note that the interesting parameter choice is $r := \lceil k/2 \rceil$, so that the size of the coefficients is halved.

Lemma 42. *Let $r, k \in \mathbb{N}$ be parameters with $0 < r < k$ and let $\delta \geq 0$. Let $\alpha_1, \ldots, \alpha_k \in \mathbb{R}$ so that $|\sum_{i=1}^k i \cdot \alpha_i| \leq \delta$ and abbreviate $\beta := \alpha_r + \ldots + \alpha_k$. Then for any $j \in \{0, \ldots, k\}$ one can find coefficients $\lambda_{i,j} \in \mathbb{Z}$ with $|\lambda_{i,j}| \leq \max\{r-1, k-r\}$ and $|j \cdot \beta - \sum_{i=1}^k \lambda_{i,j} \alpha_i| \leq \delta$.*

Proof. By symmetry it suffices to consider $j \geq 0$. For $j \in \{0, \ldots, r-1\}$, we can obviously write

$$j \cdot \beta = j \cdot \alpha_r + \ldots + j \cdot \alpha_k.$$

Now consider $j \in \{r, \ldots, k\}$. The trick is to use that

$$j \cdot \beta - \sum_{i=1}^k i\alpha_i = \sum_{i=r}^k j \cdot \alpha_i - \sum_{i=1}^k i\alpha_i = \sum_{i=r}^k (j-i)\alpha_i + \sum_{i=1}^{r-1} (-i)\alpha_i.$$

If we inspect the size of the used coefficients, then for $i \in \{1, \ldots, r-1\}$ we have $|-i| \leq r-1$ and for $i \in \{r, \ldots, k\}$ we have $|j-i| \leq k-r$. □

Lemma 43. *Let $k, r \in \mathbb{N}$ be parameters with $0 < r < k$. Let $f : \mathbb{N} \to \mathbb{R}$ be a non-negative function such that $f(n) \geq 4 \log n$. Suppose that there is a polynomial-time algorithm that for any $a' \in [0,1]^n$ finds a vector $x' \in \{-k, \ldots, k\}^n \setminus \{\mathbf{0}\}$ with $|\langle a', x' \rangle| \leq 2^{-f(n)}$. Then there is also a polynomial-time algorithm that for any $a \in [0,1]^n$ finds a vector $x \in \mathbb{Z}^n \setminus \{\mathbf{0}\}$ with $|\langle a, x \rangle| \leq 2^{-2f(\lfloor \sqrt{n} \rfloor)/3}$ and $\|x\|_\infty \leq \max\{r-1, k-r\}$.*

Proof. Let $a \in [0,1]^n$ be the given vector of numbers. To keep notation simpler, we will assume n is a perfect square. If it is not, we can replace $\sqrt{n}$ by $\lfloor\sqrt{n}\rfloor$. Split $[n]$ into blocks $I_1, \ldots, I_{\sqrt{n}}$ each of size $|I_\ell| = \sqrt{n}$. For each block I_ℓ we use the oracle to find a vector $x_\ell \in \{-k, \ldots, k\}^n \setminus \{\mathbf{0}\}$ with $\mathrm{supp}(x_\ell) \subseteq I_\ell$ so that $|\langle a, x_\ell\rangle| \leq 2^{-f(\sqrt{n})}$. If for any ℓ one has $\|x_\ell\|_\infty \leq r-1$, then we simply return $x := x_\ell$ and are done. Otherwise, we write the vector as $x_\ell = \sum_{i=1}^k i \cdot x_{\ell,i}$ with vectors $x_{\ell,1}, \ldots, x_{\ell,k} \in \{-1, 0, 1\}^n$. Note that these vectors will have disjoint support and $\mathrm{supp}(x_{\ell,1}), \ldots, \mathrm{supp}(x_{\ell,k}) \subseteq I_\ell$. Moreover we know that for every ℓ there is at least one index $i \in \{r, \ldots, k\}$ with $x_{\ell,i} \neq \mathbf{0}$.

Now define a vector $b \in \mathbb{R}^{\sqrt{n}}$ with $b_\ell := \sum_{i=r}^k \langle a, x_{\ell,i}\rangle$. Note that if for any ℓ we have $|b_\ell| \leq 2^{-f(\sqrt{n})}$, then we can set $x = \sum_{i=r}^k x_{\ell,i}$ and we are done. Therefore we may assume that $|b_\ell| > 2^{-f(\sqrt{n})}$ for all ℓ. Also note that since the $x_{\ell,i}$ have disjoint support, we have $\|b\|_\infty \leq \sqrt{n}$. We run the oracle again to find a vector $y \in \{-k, \ldots, k\}^{\sqrt{n}} \setminus \{\mathbf{0}\}$ so that $|\langle b, y\rangle| \leq \sqrt{n} \cdot 2^{-f(\sqrt{n})}$. For each block $\ell \in [\sqrt{n}]$ we can use Lemma 42 to find integer coefficients $\lambda_{\ell,i}$ with $|\lambda_{\ell,i}| \leq \max\{r-1, k-r\}$ so that

$$\left| y_\ell \cdot b_\ell - \sum_{i=1}^k \lambda_{\ell,i} \cdot \langle a, x_{\ell,i}\rangle \right| \leq 2^{-f(\sqrt{n})}.$$

We define

$$x := \sum_{\ell=1}^{\sqrt{n}} \sum_{i=1}^k \lambda_{\ell,i} x_{\ell,i}.$$

Then $\|x\|_\infty \leq \max\{r-1, k-r\}$ since the $x_{\ell,i}$'s have disjoint support and $\|x_{\ell,i}\|_\infty \leq 1$ for all ℓ, i. Moreover, since there is some $y_\ell \neq 0$ and $|b_\ell| > 2^{-f(\sqrt{n})}$, we have $x \neq \mathbf{0}$.

Finally we inspect that

$$|\langle a, x\rangle| \leq |\langle y, b\rangle| + \sum_{\ell=1}^{\sqrt{n}} \left| y_\ell b_\ell - \sum_{i=1}^k \lambda_{\ell,i} \langle a, x_{\ell,i}\rangle \right| \leq 2\sqrt{n} \cdot 2^{-f(\sqrt{n})} \leq 2^{-2f(\sqrt{n})/3}.$$

The last line comes from the fact that when $f(n) \geq 4 \log n$, we have $2\sqrt{n} \leq 2^{\frac{2}{3}\log n} = 2^{\frac{4}{3}\log\sqrt{n}} \leq 2^{\frac{1}{3}f(\sqrt{n})}$. ∎

Now we can apply Lemma 43 recursively to prove Lemma 41.

Proof. Suppose $k \leq \frac{\log n}{6 \log\log n}$ and set $r = \lceil k/2 \rceil$. Consider the function $f_t(n) = 2^{-t} n^{2^{-t}}$ for $t = 0, \ldots, \lceil \log k \rceil$, and notice that $2^{-t} \geq \frac{1}{2k} \geq 2\frac{\log\log n}{\log n}$. Therefore we have

$$f_t(n) = 2^{-t} n^{2^{-t}} \geq \frac{1}{2k} n^{1/2k} \geq (2\frac{\log\log n}{\log n}) \cdot \log^2 n \geq 4 \log n.$$

Finally, notice that $\frac{2}{3} f_t(\lfloor \sqrt{n} \rfloor) \geq \frac{1}{2} f_t(\sqrt{n}) = f_{t+1}(n)$. We are now able to apply Lemma 43. In particular, suppose we have an oracle to find x with $\|x\|_\infty \leq 2^{-t}k$ and $|\langle a, x \rangle| \leq 2^{-f_t(n)}$. Then Lemma 43 gives us an oracle to find x with $\|x\|_\infty \leq 2^{-(t+1)}k$ and $|\langle a, x \rangle| \leq 2^{-f_{t+1}(n)}$.[4]

Running this $\lceil \log k \rceil \leq \log 2k$ times gives a bound of $2^{-f_{\log 2k}(n)} = 2^{-n^{1/2k}/2k} \leq 2^{-n^{1/3k}}$. Here we use the fact that when $k \leq \frac{1}{6}\frac{\log n}{\log\log n}$, we have $\frac{n^{\frac{1}{2k}}}{2k} \geq n^{\frac{1}{3k}}$. □

4.3 Reducing Minkowski's Theorem to Number Balancing

In this section we show that for small enough δ, an oracle for δ-NBP can be used to design an algorithm for ρ-Minkowski's Problem, where ρ is polynomial in n. The first helpful insight is that any symmetric convex body can be approximated within a factor of $\sqrt{n}$ using an *ellipsoid* [Lov90, GLS12]. Recall that an ellipsoid is a set of the form

$$\mathcal{E} = \Big\{ x \in \mathbb{R}^n \mid \sum_{i=1}^{n} \frac{1}{\lambda_i^2} \cdot \langle x, a_i \rangle^2 \leq 1 \Big\} \tag{4.1}$$

with an *orthonormal basis* $a_1, \ldots, a_n \in \mathbb{R}^n$ defining the *axes* and positive coefficients $\lambda_1, \ldots, \lambda_n$ that describe the *lengths of the axes*[5]. Overall, our reduction will operate in two steps:

(i) By combining *John's Theorem* with *lattice basis reduction*, we can show that it suffices to find integer points in an ellipsoid that is *well-rounded*, meaning that the lengths of the axes are bounded.

(ii) We show that a number balancing oracle allows a self-reduction to a generalized form where inner products with n vectors have to be minimized and additionally the solution space is $\mathbb{Z}^n$ instead of $\{-1, 0, 1\}^n$.

[4]Here we ignore the dependence of t on n - notice that t is nondecreasing in n, so replacing $t(n)$ by $t(\sqrt{n})$ only increases $f_t(n)$.

[5]Strictly speaking, the length of axis i is $2\lambda_i$, but we will continue calling λ_i the "axis length".

We begin by proving (ii) and postpone (i) until the end of this section.

4.3.1 A self-reduction to a generalized form of number balancing

Recall that for $\delta > 0$, we defined δ-NBP as follows: given $a \in [0,1]$, find a vector $x \in \{-1,0,1\}^n \setminus \{\mathbf{0}\}$ with $|\langle a, x\rangle| \leq \delta$. Notice that we may allow for vectors $a \in [-1,1]^n$ without changing the problem, as one can flip the signs of x as needed to accommodate for the changes in sign of a. The main technical result of this section is the following reduction:

Theorem 44. *Suppose there is a polynomial-time algorithm for δ-NBP with $\delta = \frac{g(n)}{2^n}$ and $g(n) \leq 2^{n/2}$. Then there is a polynomial-time algorithm that on input $a_1, \ldots, a_n \in [-1,1]^n$ and $0 < \lambda_1 \leq \ldots \leq \lambda_n \leq 2^n$ with $\prod_{i=1}^n \lambda_i \geq 1$, finds a vector $x \in \mathbb{Z}^n \setminus \{\mathbf{0}\}$ with*

$$|\langle x, a_i\rangle| \leq O(n^4) \cdot \lambda_i \cdot g(4n^2)^{1/n} \quad \forall i = 1, \ldots, n.$$

In particular if $g(n) \leq 2^{\sqrt{n}}$, then the right hand side in Theorem 44 simplifies to just $O(n^4) \cdot \lambda_i$. We will show this by introducing two extensions of the number balancing oracle. The first extension gives a weaker bound in terms of the error parameter, but allows for multiple vectors in $[-1,1]^n$. In the second extension, we extend the range of coefficients from $\{-1,0,1\}$ to $\{-Q, \ldots, Q\}$ which leads to a much stronger error bound.

Lemma 45. *Suppose there is a polynomial-time algorithm for δ-NBP. Then there is a polynomial-time algorithm that given $a_1, \ldots, a_k \in [-1,1]^n$ and $\delta_1, \ldots, \delta_k \leq \frac{1}{2}$ with $\prod_{i=1}^k \delta_i \geq \delta$ finds a vector $x \in \{-1,0,1\}^n \setminus \{\mathbf{0}\}$ with $|\langle a_i, x\rangle| \leq 2n^2\delta_i$ for all $i = 1, \ldots, k$.*

Proof. The idea is that we will discretize all of the vectors and then run our oracle on their sum. The vector that we obtain will then have small inner product with all of the a_i. To define the discretization $\tilde{a}_i$, round elements of a_i down to the nearest multiple of $2n\delta_i$, and then multiply by $\prod_{j<i} \delta_j$. Defining $\tilde{a}_i$ this way, notice that for all i we have $|\langle \tilde{a}_i, x\rangle| \leq n \prod_{j<i} \delta_j$ for any x satisfying $\|x\|_\infty \leq 1$.

Now let $c = \tilde{a}_1 + \ldots + \tilde{a}_k$. By our oracle, we can find $x \in \{-1,0,1\}^n \setminus \{\mathbf{0}\}$ with $|\langle c, x\rangle| \leq \delta$. Recall that $\delta \leq \prod_{i=1}^k \delta_i \leq n \prod_{j \leq k} \delta_j$. We have

$$\begin{aligned} |\langle \tilde{a}_1, x\rangle| &\leq |\langle \tilde{a}_2, x\rangle| + \ldots + |\langle \tilde{a}_k, x\rangle| + |\langle c, x\rangle| \leq n\delta_1 + n\prod_{j\leq 2}\delta_j + \ldots + n\prod_{j\leq k}\delta_j \\ &\leq n\delta_1 \cdot \left(1 + \frac{1}{2} + \ldots + \frac{1}{2^k}\right) < 2n\delta_1. \end{aligned}$$

Therefore $|\langle \tilde{a}_1, x\rangle| = 0$. Similarly, for $1 < i \leq k$, if $|\langle \tilde{a}_1, x\rangle|, \ldots, |\langle \tilde{a}_{i-1}, x\rangle| = 0$, then we have

$$|\langle \tilde{a}_i, x\rangle| \leq |\langle \tilde{a}_{i+1}, x\rangle| + \ldots + |\langle \tilde{a}_k, x\rangle| + |\langle c, x\rangle| < 2n \prod_{j \leq i} \delta_j,$$

and hence $|\langle \tilde{a}_i, x\rangle| = 0$ for all i. Notice that by definition of $\tilde{a}_i$ we have $\| \prod_{j<i} \delta_j a_i - \tilde{a}_i \|_\infty \leq 2n \prod_{j \leq i} \delta_j$. Therefore $|\langle \prod_{j<i} \delta_j a_i, x\rangle| \leq 2n^2 \prod_{j \leq i} \delta_j$, and so we can conclude that $|\langle a_i, x\rangle| \leq 2n^2\delta_i$. □

Now we come to a second reduction that takes the oracle constructed in Lemma 45 as a starting point:

Lemma 46. *Assume there exists a polynomial-time algorithm for δ-NBP where $\delta = f(n)$. Let $a_1, \ldots, a_k \in [-1,1]^n$ be given with parameters $\delta_1, \ldots, \delta_k \leq \frac{1}{2}$ and a number Q that is a power of 2 and satisfies $\prod_{i=1}^k \delta_i \geq f(n \log Q)$. Then in polynomial time we can find a vector $x \in \{-Q, \ldots, Q\}^n \setminus \{\mathbf{0}\}$ with $|\langle a_i, x\rangle| \leq \delta_i Q \cdot 2(n \log Q)^2$ for all $i = 1, \ldots, k$.*

Proof. For each $i = 1, \ldots, k$, we define $b_i \in [-1,1]^{n \log Q}$ by $b_i(j,\ell) = a_i(j) 2^{-\ell}$ for $j = 1, \ldots, n$ and $\ell = 1, \ldots, \log Q$. Since $\prod_{i=1}^k \delta_i \geq f(n \log Q)$, we can apply Lemma 45 to find $y \in \{-1,0,1\}^{n \log Q} \setminus \{\mathbf{0}\}$ with $|\langle b_i, y\rangle| \leq \delta_i \cdot 2(n \log Q)^2$.

Now define $x \in \{-Q, \ldots, Q\}^n \setminus \{\mathbf{0}\}$ by $x_j := Q \sum_{\ell=1}^{\log Q} 2^{-\ell} y_{j\ell}$. Then for $i = 1, \ldots, k$ we have

$$\delta_i \cdot (2n \log Q)^2 \geq |\langle b_i, y\rangle| = \Big| \sum_{j=1}^{n} \underbrace{\sum_{\ell=1}^{\log Q} y_{j\ell} 2^{-\ell}}_{= x_j/Q} a_i(j) \Big| = \frac{1}{Q} \cdot |\langle a_i, x\rangle|$$

and rearranging gives the claim. □

Finally we come to the proof of Theorem 44.

Proof. Suppose that the oracle has parameter $f(n) = \rho(n)/2^n$. Suppose that $a_1, \ldots, a_n \in [-1,1]^n$ and $\lambda_1, \ldots, \lambda_n > 0$ with $\prod_{i=1}^n \lambda_i \geq 1$. We choose $Q := 2^{4n}$, which is a power of 2. Define $\delta_i = \lambda_i \cdot f(n \log Q)^{1/n}$. Note that $\delta_i \leq 2^{3n/2} \cdot f(4n^2)^{1/n} \leq \frac{1}{2}$ since $f(n) \leq 2^{-n/2}$. Then $\prod_{i=1}^n \delta_i \geq f(n \log Q)$, and so by Lemma 46 we can find $y \in \{-Q, \ldots, Q\}^n \setminus \{\mathbf{0}\}$ with

$$\begin{aligned} |\langle a_i, y \rangle| \leq Q\delta_i \cdot 2(n \log Q)^2 &\leq Q\lambda_i \cdot f(n \log Q)^{1/n} \cdot 2(n \log Q)^2 \\ &= \lambda_i \cdot \rho(4n^2)^{1/n} \cdot 2 \cdot (4n^2)^2. \end{aligned}$$

□

4.3.2 A reduction to well-rounded ellipsoids

Using John's Theorem [Joh48], the convex body K in Theorem 38 can be approximated by an ellipsoid $\mathcal{E}$ as defined in Eq. (4.1). The natural approach will then be to apply Theorem 44 to the axes of the ellipsoid. However, it will be crucial that the lengths of the axes of the ellipsoid are bounded by $2^{O(n)}$. We will now argue how to make an arbitrary ellipsoid well rounded.

Let us denote $\lambda_{\max}(\mathcal{E}) := \max\{\lambda_i : i = 1, \ldots, n\}$ as the maximum length of an axis. Recall that a matrix $U \in \mathbb{R}^{n \times n}$ is *unimodular* if $U \in \mathbb{Z}^{n \times n}$ and $|\det(U)| = 1$. In particular, the linear map $T : \mathbb{R}^n \to \mathbb{R}^n$ with $T(x) = Ux$ is a bijection on the integer lattice, meaning that $T(\mathbb{Z}^n) = \mathbb{Z}^n$. It turns out that one can use the *lattice basis reduction* method by Lenstra, Lenstra and Lovász [LLL82] to find a unimodular linear transformation that "regularizes" any given ellipsoid. Note that it suffices to work with the regularized ellipsoid $T(\mathcal{E})$ since $\mathrm{vol}_n(T(\mathcal{E})) = \mathrm{vol}_n(\mathcal{E})$ and if we find a point $x \in (\rho T(\mathcal{E})) \cap \mathbb{Z}^n$, then by linearity $T^{-1}(x) \in \rho\mathcal{E}$ and $T^{-1}(x) \in \mathbb{Z}^n$.

Given $b_1, \ldots, b_n \in \mathbb{R}^n$ we define the *Gram-Schmidt orthogonalization* iteratively as $\hat{b}_j = b_j - \sum_{i<j} \mu_{ij} \hat{b}_i$, where $\mu_{ij} = \frac{\langle b_j, \hat{b}_i \rangle}{\|\hat{b}_i\|_2^2}$. Notice that we can then write $b_j = \hat{b}_j + \sum_{i<j} \mu_{ij} \hat{b}_i$. In particular, suppose B is the matrix with columns $b_1, \ldots, b_n$ and $\hat{B}$ is the matrix with columns

$\hat{b}_1, \ldots, \hat{b}_n$. Then $B = \hat{B}V$ for an upper triangular matrix V with ones along the diagonal and $V_{ij} = \mu_{ij}$ for $i < j$.

Definition 3. *Let $B \in \mathbb{R}^{n \times n}$ be a lattice basis and let μ_{ij} be the coefficients from Gram-Schmidt orthogonalization. The basis is called* LLL reduced *if*

- *(Coefficient-reduced): $|\mu_{ij}| \leq \frac{1}{2}$ for all $1 \leq i < j \leq n$.*
- *(Lovász condition): $\|\hat{b}_i\|_2^2 \leq 2\|\hat{b}_{i+1}\|_2^2$ for $i = 1, ..., n-1$.*

LLL reduction has been widely used in diverse fields such as integer programming and cryptography [NV10]. One property of the LLL reduced basis is that the eigenvalues of the corresponding matrix B are bounded away from 0:

Lemma 47. *Let B denote the matrix with columns $b_1, \ldots, b_n$. If $b_1, \ldots, b_n$ is an LLL-reduced basis with $\|b_i\|_2 \geq 1$ for all i, then $\|Bx\|_2 \geq 2^{-3n/2} \cdot \|x\|_2$ for all $x \in \mathbb{R}^n$.*

Proof. Let $\hat{B}$ denote the Gram-Schmidt orthogonalization of B, with columns $\hat{b}_1, \ldots, \hat{b}_n$. Now, for any k, we can use the properties of LLL reduction to gain the following bound (proposition (1.7) in [LLL82]).

$$1 \leq \|b_k\|_2^2 = \|\hat{b}_k\|_2 + \sum_{i<k} \mu_{ik}^2 \|\hat{b}_i\|_2^2 \leq \left(1 + \frac{1}{4}\sum_{i<k} 2^{k-i}\right) \cdot \|\hat{b}_k\|_2^2 \leq 2^k \cdot \|\hat{b}_k\|_2^2.$$

In particular, $\|\hat{b}_k\|^2 \geq 2^{-n}$ for all $k = 1, \ldots, n$. Now let V be the matrix so that $B = \hat{B}V$, and let $x \in \mathbb{R}^n$ with $\|x\|_2 = 1$. Let k denote the largest index with $|x_k| \geq 2^{-k}$. Then

$$|(Vx)_k| = \left|x_k + \sum_{j>k} \mu_{kj} x_j\right| \geq |x_k| - \frac{1}{2}\sum_{j>k} |x_j| \geq 2^{-k} - \frac{1}{2}\sum_{j>k} 2^{-j} \geq 2^{-n}.$$

Now, by the orthogonality of $\hat{b}_1, \ldots, \hat{b}_n$, we have

$$\|Bx\|_2^2 = \|\hat{B}Vx\|_2^2 = \sum_{i=1}^{n} (Vx)_i^2 \|\hat{b}_i\|_2^2 \geq |(Vx)_k|^2 \cdot 2^{-n} \geq 2^{-3n}.$$

Taking square roots gives the claim. □

Lemma 48. *Let $\mathcal{E} = \{x \in \mathbb{R}^n : \|Ax\|_2^2 \leq 1\}$ be an ellipsoid. Then in polynomial time, we can find one of the following:*

(1) A vector $x \in \mathcal{E} \cap \mathbb{Z}^n \setminus \{\mathbf{0}\}$

(2) A linear transformation T so that $\lambda_{max}(T(\mathcal{E})) \leq 2^{3n/2}$ and $T(x) = Ux$ for a unimodular matrix U.

Proof. Use the algorithm of [LLL82] to find a unimodular matrix U such that $B = AU$ is LLL reduced. Let $b_1, \ldots, b_n$ denote the columns of B. Notice that if $\|b_i\|_2 \leq 1$, then $A^{-1}b_i \in \mathcal{E} \cap \mathbb{Z}^n$, and so we are done. So assume now that $\|b_i\|_2 \geq 1$ for all i.

Define $T(x) = U^{-1}x$, and notice that $T(\mathcal{E}) = \{x \in \mathbb{R}^n : \|Bx\|_2^2 \leq 1\}$. We then have

$$\lambda_{\max}(T(\mathcal{E})) = \max_{x \in f(\mathcal{E})} \|x\|_2 = \max_{\|Bx\|_2 \leq 1} \|x\|_2 = \max_{x \neq 0} \frac{\|x\|_2}{\|Bx\|_2} \leq 2^{3n/2},$$

where the last inequality follows from Lemma 47. □

Finally we can prove one of our main results, Theorem 38.

Proof. Let $K \subseteq \mathbb{R}^n$ be a convex body with $\mathrm{vol}_n(K) \geq 2^n$. We compute an ellipsoid[6] $\mathcal{E} = \{x \in \mathbb{R}^n \mid \sum_{i=1}^n \frac{1}{\lambda_i^2}\langle x, a_i\rangle^2 \leq 1\}$ so that $\frac{1}{5\sqrt{n}}\mathcal{E} \subseteq K \subseteq \frac{1}{5}\sqrt{n}\mathcal{E}$. Then

$$2^n \cdot 5^n \cdot n^{-n/2} \leq \mathrm{vol}_n(K) \cdot 5^n \cdot n^{-n/2} \leq \mathrm{vol}_n(\mathcal{E}) = \underbrace{\mathrm{vol}_n(B(\mathbf{0}, 1))}_{\leq 5^n n^{-n/2}} \cdot \prod_{i=1}^n \lambda_i.$$

and hence $\prod_{i=1}^n \lambda_i \geq 1$. We apply Lemma 48 to either find an integer point in $\mathcal{E}$ and we are done, or we find a unimodular transformation T so that the ellipsoid $\tilde{\mathcal{E}} := T(\mathcal{E})$ has all axes of length at most $2^{O(n)}$. Suppose the latter case happens. We write $\tilde{\mathcal{E}} = \{x \in \mathbb{R}^n \mid \sum_{i=1}^n \frac{1}{\tilde{\lambda}_i^2}\langle x, \tilde{a}_i\rangle^2 \leq 1\}$ and observe that still $\prod_{i=1}^n \tilde{\lambda}_i \geq 1$ as the volume of the ellipsoid has not changed. We make use of the δ-approximation for the number balancing

[6]Note that there *exists* an ellipsoid that approximates K within a factor of $\sqrt{n}$ and if K is a polytope with m facets, then this ellipsoid can be found in time polynomial in n and m. However, if one only has a separation oracle for K, then the best factor achievable in polynomial time is n [GLS12].

problem to apply Theorem 44 to the vectors $\tilde{a}_1, \ldots, \tilde{a}_n$ and parameters $\tilde{\lambda}_1, \ldots, \tilde{\lambda}_n$ and obtain a vector $x \in \mathbb{Z}^n \setminus \{\mathbf{0}\}$ with $|\langle \tilde{a}_i, x \rangle| \leq \tilde{\lambda}_i \cdot O(n^4)$. Then $\sum_{i=1}^n \frac{1}{\tilde{\lambda}_i^2} \langle a_i, x \rangle^2 \leq O(n^9)$ and hence $x \in O(n^{4.5}) \cdot \tilde{\mathcal{E}}$. Then $T^{-1}(x) \in (O(n^5) \cdot K) \cap (\mathbb{Z}^n \setminus \{\mathbf{0}\})$. □

BIBLIOGRAPHY

[Agm54] S. Agmon. The relaxation method for linear inequalities. *Canadian Journal of Mathematics*, 6:382–392, 1954.

[AHK05] S. Arora, E. Hazan, and S. Kale. Fast algorithms for approximate semidefinite programming using the multiplicative weights update method. In *46th IEEE FOCS*, pages 339–348, 2005.

[AHK12] S. Arora, E. Hazan, and S. Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(6):121–164, 2012.

[Ajt96] M. Ajtai. Generating hard instances of lattice problems. In *Proceedings of the 28th STOC*, pages 99–108. ACM, 1996.

[AR05] D. Aharonov and O. Regev. Lattice problems in NP cap conp. *J. ACM*, 52(5):749–765, 2005.

[AS08] N. Alon and J. H. Spencer. *The Probabilistic Method*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons Inc., Hoboken, NJ, third edition, 2008. With an appendix on the life and work of Paul Erdős.

[Bal97] Keith Ball. An elementary introduction to modern convex geometry. In *in Flavors of Geometry*, pages 1–58. Univ. Press, 1997.

[Bet04] Ulrich Betke. Relaxation, new combinatorial and polynomial algorithms for the linear feasibility problem. *Discrete & Computational Geometry*, 32(3):317–338, 2004.

[Boh96] T. Bohman. A sum packing problem of erdös and the conway-guy sequence. *Proceedings of the AMS*, 124(12):3627–3636, 1996.

[CCZ14] M. Conforti, G. Cornuejols, and G. Zambelli. *Integer Programming*. Springer Publishing Company, Incorporated, 2014.

[CGJ84] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson. Approximation algorithms for bin-packing—an updated survey. In *Algorithm design for computer system design*, volume 284 of *CISM Courses and Lectures*, pages 49–106. Springer, Vienna, 1984.

[Chu12] Sergei Chubanov. A strongly polynomial algorithm for linear systems having a binary solution. *Math. Programming*, 134(2):533–570, 2012.

[Chu15] Sergei Chubanov. A polynomial projection algorithm for linear feasibility problems. *Math. Program.*, 153(2):687–713, November 2015.

[CKM+11] P. Christiano, J.A. Kelner, A. Madry, D.A. Spielman, and S. Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proc. of the 43rd ACM Symposium on Theory of Computing*, pages 273–282, New York, NY, USA, 2011.

[Dan51] G.B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In *Activity Analysis of Production and Allocation*, Cowles Commission Monograph No. 13, pages 339–347. John Wiley & Sons, Inc., New York, NY; Chapman & Hall, Ltd., London, 1951.

[DV06] J. Dunagan and S. Vempala. A simple polynomial-time rescaling algorithm for solving linear programs. *Math. Program.*, 114(1):101–114, 2006.

[DVZ16] Daniel Dadush, László A. Végh, and Giacomo Zambelli. Rescaling algorithms for linear programming - part I: conic feasibility. *CoRR*, abs/1611.06427, 2016.

[Dós07] G. Dósa. The tight bound of first fit decreasing bin-packing algorithm is FFD(I) <= 11/9OPT(I) + 6/9. In B. Chen, M. Paterson, and G. Zhang, editors, *ESCAPE*, volume 4614 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 2007.

[Eis57] K. Eisemann. The trim problem. *Management Science*, 3(3):279–284, 1957.

[EPR13] F. Eisenbrand, D. Pálvölgyi, and T. Rothvoß. Bin packing via discrepancy of permutations. In *Transactions on Algorithms (Special Issue for SODA 2011)*, 2013.

[FdlVL81] W. Fernandez de la Vega and G. S. Lueker. Bin packing can be solved within $1 + \varepsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981.

[GG61] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9:849–859, 1961.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, New York, 1979.

[GK07] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM J. Comput.*, 37(2):630–652, 2007.

[GLS81] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[GLS12] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric algorithms and combinatorial optimization*, volume 2, pages 122–125. Springer, 2012.

[Hač79] L.G. Hačijan. A polynomial algorithm in linear programming. *Dokl. Akad. Nauk SSSR*, 244(5):1093–1096, 1979.

[HR07] I. Haviv and O. Regev. Tensor-based hardness of the shortest vector problem to within almost polynomial factors. pages 469–477, 2007.

[HR16] Rebecca Hoberg and Thomas Rothvoss. An improved deterministic rescaling for linear programming algorithms. *CoRR*, abs/1612.04782, 2016.

[HR17] Rebecca Hoberg and Thomas Rothvoss. A logarithmic additive integrality gap for bin packing. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2616–2625, 2017.

[HRRY16] Rebecca Hoberg, Harishchandra Ramadas, Thomas Rothvoss, and Xin Yang. Number balancing is as hard as minkowski's theorem and shortest vector. *CoRR*, abs/1611.08757, 2016.

[JDU+74] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4):299–325, 1974.

[Joh48] F. John. Extremum problems with inequalities as subsidiary conditions. In *Studies and Essays Presented to R. Courant on his 60th Birthday, January 8, 1948*, pages 187–204. Interscience Publishers, Inc., New York, N. Y., 1948.

[Joh73] D. S. Johnson. *Near-optimal bin packing algorithms*. PhD thesis, MIT, Cambridge, MA, 1973.

[Kar84] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[KK82a] N. Karmarkar and R. Karp. The differencing method of set partitioning. Technical report, CS Division, UC Berkeley, 1982. http://digitalassets.lib.berkeley.edu/techreports/ucb/text/CSD-83-113.pdf.

[KK82b] N. Karmarkar and R. M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *23rd annual symposium on foundations of computer science (Chicago, Ill., 1982)*, pages 312–320. IEEE, New York, 1982.

[KM72] V. Klee and G. Minty. How good is the simplex algorithm? In *Inequalities, III (Proc. Third Sympos., UCLA, 1969; dedicated to the memory of Theodore S. Motzkin)*, pages 159–175. Academic Press, New York, 1972.

[KPP04] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack problems*. Springer, 2004.

[LLL82] A. Lenstra, H. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[LM12] S. Lovett and R. Meka. Constructive discrepancy minimization by walking on the edges. In *FOCS*, pages 61–67, 2012.

[Lov86] László Lovász. *An algorithmic theory of numbers, graphs and convexity*. SIAM, 1986.

[Lov90] L. Lovász. *Geometric algorithms and algorithmic geometry*. American Mathematical Society, 1990.

[LS15] Y. Lee and A. Sidford. A new polynomial-time algorithm for linear programming. 2015. https://arxiv.org/abs/1312.6677.

[Lun88] W. Lunnon. Integer sets with distinct subset-sums. *Mathematics of Computation*, 50(181):297–320, 1988.

[LY11] V. Lev and R. Yuster. On the size of dissociated bases. *the electronic journal of combinatorics*, 18(1):P117, 2011.

[Mad10] A. Madry. Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms. In *Proc. of the 42nd ACM Symposium on Theory of Computing*, pages 121–130, New York, NY, 2010.

[Mat99] J. Matoušek. *Geometric discrepancy*, volume 18 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, 1999. An illustrated guide.

[Mat02] J. Matousek. *Lectures on Discrete Geometry*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[Mer06] S. Mertens. The easiest hard problem: Number partitioning. *Computational Complexity and Statistical Physics*, 125(2):125–139, 2006.

[MR09] Daniele Micciancio and Oded Regev. Lattice-based cryptography. In *Post-quantum cryptography*, pages 147–191. Springer, 2009.

[MT90] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., New York, NY, USA, 1990.

[Nes05] Y. Nesterov. Excessive gap technique in nonsmooth convex minimization. *SIAM Journal on Optimization*, 16(1):235–249, 2005.

[NNN12] A. Newman, O. Neiman, and A. Nikolov. Beck's three permutations conjecture: A counterexample and some consequences. In *FOCS*, pages 253–262, 2012.

[NV10] P. Nguyen and B. Vallée. The lll algorithm. *Information Security and*, 2010.

[Pap94] Christos H Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *Journal of Computer and system Sciences*, 48(3):498–532, 1994.

[PS12] J. Peña and N. Soheili. A smooth perceptron algorithm. *SIAM J. Optim.*, 22(2):728–737, 2012.

[PS16] J. Peña and N. Soheili. A deterministic rescaled perceptron algorithm. *Math. Program.*, 155(1-2):497–510, 2016.

[PST95] S. A. Plotkin, D. B. Shmoys, and É. Tardos. Fast approximation algorithms for fractional packing and covering problems. *Math. Oper. Res.*, 20(2):257–301, 1995.

[Rot13] T. Rothvoß. Approximating bin packing within O(log OPT * log log OPT) bins. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 20–29, 2013.

[Sch86] A. Schrijver. *Theory of linear and integer programming*. Wiley-Interscience Series in Discrete Mathematics. John Wiley & Sons, Ltd., Chichester, 1986. A Wiley-Interscience Publication.

[Sch87] C. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theor. Comput. Sci.*, 53:201–224, 1987.

[SST] J. H. Spencer, A. Srinivasan, and P. Tetali. The discrepancy of permutation families. Unpublished Manuscript.

[ST97] G. Scheithauer and J. Terno. Theoretical investigations on the modified integer round-up property for the one-dimensional cutting stock problem. *Operations Research Letters*, 20(2):93 – 100, 1997.

[Vaz01] V. Vazirani. *Approximation algorithms*. Springer, 2001.

[WS11] D. P. Williamson and D. B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[WY92] G. Woeginger and Z. Yu. On the equal-subset-sum problem. *Information Processing Letters*, 42(6):299–302, 1992.