Studies in Algorithms

Nir Ailon

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
By the Department of
Computer Science

September 2006

Abstract

This work is comprised of three separate parts: (1) *Lower bounds for linear degeneracy testing* (based on joint work with Bernard Chazelle [5]); (2) *Aggregating inconsistent information* (based on joint work with Moses Charikar and Alantha Newman [3]); and (3) *The fast Johnson-Lindenstrauss transform and approximate nearest neighbor searching* (based on joint work with Bernard Chazelle [6]).

The first part discusses a fundamental computational geometric problem called rSUM: given n real numbers, do any r of them sum up to 0? It is the simplest case of a more general family of problems called degeneracy testing. This seemingly naive problem is in the core of the difficulty in designing algorithms for more interesting problems in computational geometry. The computational model assumed is the linear decision tree. This model was successfully used in many other results on the computational complexity of geometric problems. This work extends and improves a seminal result by Erickson [46], and sheds light not only on the complexity of rSUM as a computational problem but also on the combinatorial structure (known as the linear arrangement) attached to it.

The second part studies optimization algorithms designed for integrating information coming from different sources. This framework includes the well-known problem of voting from the old theory of social choice. It has been known for centuries that voting and collaborative decision making is difficult (and interesting) due to certain inherent paradoxes that arise. More recently, the computational aspects of these problems have been studied, and several hardness results were proved. The recent interest in voting and the theory of social choice theory in the context of computer science was motivated by more "modern" problems related to the age of information: If several algorithms are used for approximately solving a problem using different heuristics, how do we aggregate the corresponding outputs into one single output? In some cases there are reasons to believe that an aggregate output is better than each one of the individual outputs (voters). We design improved algorithms for two important problems known as rank aggregation and consensus clustering. In our analysis we prove new results on optimization over binary relations (in particular, order and equivalence relations).

The third part revisits the computational aspects of a well-known lemma by Johnson and Lindenstrauss from the mid 80's. The Johnson-Lindenstrauss lemma states the surprising fact that any finite subset of a Euclidean space can be almost isometrically embedded in a space of dimension at most logarithmic in the size of the subset. In fact, a suitably chosen random linear transformation does the trick. The algorithmic results were quickly reaped by researchers interested in improving algorithms suffering from running time and/or space depending heavily on the dimensionality of the problem, most notably proximity based problems such as clustering and nearest neighbor searching. Many new computationally-friendly versions of the original J-L lemma were proved. These versions simplified the distribution from which the random linear transformation was chosen, but did not yield better than a constant factor improvement in its running time. In this work we define a new distribution on linear transformations with a significant computational improvement. We call it the Fast Johnson-Lindenstrauss Transform (FJLT), and show how to apply it to nearest neighbor searching in Euclidean space. In the last chapter of this part we propose a different approach (unrelated to the FJLT) for improving nearest neighbor searching in the Hamming cube. Interestingly, we achieved this latter improvement before working on the FJLT, and it provided evidence and motivation for an FJLT-type result.

Acknowledgments

First and foremost, I am grateful to my advisor Bernard Chazelle. I'm very fortunate to have had the opportunity to work under his supervision. It was a wonderful experience, and I owe the very foundation of my career to him. My graduate work and this dissertation would not have been possible without his continuous support and encouragement from my very first days at Princeton. I am looking forward to continuing our fruitful collaboration in the future. I would also like to thank Moses Charikar and Alantha Newman for our exciting joint work.

This dissertation has benefited from the insight of numerous colleagues: Jeff Erickson (for discussions on his linear-degeneracy lower bound proof), Ravi Kumar, D. Sivakumar, Shuchi Chawla, Tony Wirth, Anke van Zuylen, Lisa Fleischer, Cynthia Dwork, Steve Chien, Avrim Blum, Seffi Naor and Kunal Talwar (for discussions on ranking and clustering), Noga Alon (for discussions on hardness of minimum feedback arc-set in tournaments and consequent collaboration), Warren Schudy (for communicating a nice observation improving Lemma 3.6 in Part II), Nina Gantert, Anupam Gupta, Elchanan Mossel and Yuval Peres (for discussions on probability and the Johnson-Lindenstrauss lemma), Sariel Har-Peled, Piotr Indyk and Yuval Rabani (for discussions on nearest-neighbor searching).

For their helpful comments, I would like to thank the reviewers of this dissertation, Moses Charikar, Nick Pippenger and Bernard Chazelle.

The Department of Computer Science at Princeton University has provided me with a stimulating academic home. In particular, Melissa Lawson was always helpful and made graduate students feel welcome.

Thanks to my friends from the graduate program: Iannis Tourlakis, Diego Nehab, Miroslav Dudik, Satyen Kale, Elad Hazan, Loukas Georgiadis, Renato Werneck, Tony Wirth, Seshadhri Comandur, Ding Liu, Frances Spalding, Amit Agarwal and others. Finally, thanks to my parents Michal and Amit and my sisters Shiri and Galit for moral support and countless pep talks.

Dedicated to my beloved family

Contents

Abstract . . . iii

List of Figures . . . x

0 Preface **1**

I Lower Bounds for Linear Degeneracy Testing **3**

1 Introduction **4**

1.1 Definition of problem . . . 4

1.2 The computational model . . . 6

1.3 Previous results . . . 8

2 A Geometric Framework for Lower Bounds **12**

2.1 Terminology and conventions . . . 12

2.2 Overview of proof . . . 14

3 The Case $s = r$ **16**

3.1 Critical hyperplanes via error-correcting codes . . . 16

3.2 The Chamber . . . 20

4 The Case $s = r + 1$ **26**

4.1 What doesn't work in the previous proof? . . . 26

4.2 The tensor product construction . . . 29

5 The Case $s > r + 1$ **32**

6 General Linear Degeneracy Tests **35**

7 Concluding Remarks **37**

II Aggregating Inconsistent Information: Ranking and Clustering **38**

1 Introduction **39**

2 Rank Aggregation **42**

2.1 Definition of problem 42

2.2 Majority rules:

Condorcet winners and the Condorcet paradox 43

2.3 The Kemeny approach 44

2.4 Minimum feedback arc-set in tournaments 45

2.5 Previous work 47

2.6 New algorithms and results 48

3 Analysis of Ranking Algorithms **53**

3.1 KwikSort for MinFAS-Tour 53

3.2 KwikSort for weighted MinFAS 56

3.3 An improved approximation ratio for RankAggregation 59

3.4 LP-KwikSort for rounding the ranking LP 63

3.5 Proof of ranking polyhedral inequalities 67

4 NP-Hardness of Feedback Arc Set on Tournaments **71**

5 Consensus Clustering **74**

5.1 Definition of problem 74

5.2 Majority rules: A Condorcet equivalent 74

5.3 A Kemeny approach equivalent 75

5.4 Correlation clustering on complete graphs 76

5.5 Previous work 79

5.6 New algorithms and results 79

6 Analysis of Clustering Algorithms **83**

6.1 KwikCluster for CorrelationClustering on complete graphs 84

6.2 KwikCluster for weighted CorrelationClustering 85

6.3 An improved approximation ratio for ConsensusClustering 88

6.4 LP-KwikCluster for rounding the clustering LP 89

6.5 Proof of clustering polyhedral inequalities 93

7 Concluding Remarks **97**

III The Fast Johnson-Lindenstrauss Transform and Approximate Nearest Neighbor Searching **100**

1 Introduction **101**

1.1 History of the Johnson-Lindenstrauss transform 102

1.2 Approximate nearest neighbor searching 105

2 The Fast Johnson-Lindenstrauss Transform **108**

3 ANN **Searching in Euclidean Space** **117**

3.1 Part I: Linear-factor approximation 118

3.2 Part II: Binary search with handles 119

3.3 Poisson discretization 121

3.4 A pruned data structure 121

4 ANN **Searching Over the Hamming Cube** **124**

4.1 Improvement using linear algebra 126

4.2 No query left behind 126

5 Concluding Remarks **128**

List of Figures

1.1 A 3SUM-hard problem 5

1.2 A linear decision tree 7

2.1 Arrangements and polyhedra 13

2.2 The face C of $\mathcal{Q}$, collapsed points and critical hyperplanes 15

3.1 From error correcting code vectors to critical hyperplane normals 19

3.2 Main construction step 22

3.3 Constructing row i of M^{φ_h} 23

3.4 How the error correcting code works 24

4.1 Constructing critical hyperplanes for $s = r + 1$ 28

5.1 Constructing critical hyperplanes for $s > r + 1$ 33

1.1 Aggregating inconsistent information 40

2.1 The Condorcet paradox 43

2.2 From RankAggregation to weighted MinFAS 46

2.3 Any median problem admits a simple 2-approximation algorithm. 47

2.4 Pseudocode and diagram for KwikSort 49

2.5 Pseudocode for LP-KwikSort 52

3.1 Charging directed triangles 54

3.2 Pseudocode for PickAPerm 59

3.3 The ranking Δ-polytope 70

4.1 Blowing up G by factor k 72

5.1 An equivalent to the Condorcet paradox 75
5.2 CorrelationClustering 77
5.3 From ConsensusClustering to weighted CorrelationClustering 78
5.4 Pseudocode and diagram for KwikCluster 80
5.5 Pseudocode for LP-KwikCluster 82

6.1 Charging paradox triangles 84
6.2 Pseudocode for PickACluster 88
6.3 The clustering Δ-polytope 95

1.1 ε-Approximate nearest neighbor searching 105

2.1 The FJLT transform 109

3.1 Pseudocode for $O(n)$-ANN in Euclidean space 118
3.2 Why the $O(n)$-ANN algorithm works 119
3.3 Pseudocode for ε-ANN in Euclidean space 123

Chapter 0

Preface

The three parts of this dissertation are based on independent research efforts, and are each self contained. Nevertheless, combining the three tells the story of the two main themes found in theoretical computer science, namely, *negative* vs. *positive* results.

Negative results demonstrate the limits of computation. A classic example of such a result is the ancient Greek problem of trisecting an angle using only a compass and a straightedge. This was proven to be impossible by Wantzel (1836). In the first part we prove that solving linear degeneracy is impossible using only a certain restricted (yet natural) set of operations in less than a certain amount of time. Negative results are usually very difficult to prove because one needs to argue against all possible algorithms. Restricting the model of computation, arguing against only certain types of algorithms or making widely believed "plausible" negative assumptions (e.g. $P \neq NP$) are some of the tools often used in such proofs.

Positive results demonstrate the possibilities of computation by designing algorithms and proving guarantees on the solutions they output and the resources (time, space, randomness) they consume. Of particular interest are algorithms computing *approximations* to problems for which there is evidence (in the form of negative results) for hardness of computing exactly. In the second part, such approximation algorithms are described for several well-known hard (assuming $P \neq NP$) problems. Another exciting direction of algorithmic research is the attempt to squeeze the last drop of efficiency from existing algorithms. Classic examples are fast matrix multiplication and the fast Fourier transform (FFT). In extreme cases, when approximation is allowed, *sublinear* algorithms are possible (algorithms that read only a small sample from the input). This field enjoyed a flurry of interest in the past decade. The explosion of information and increase in storage capabilities revolutionized our lives and called for new ideas for handling data. What's considered efficient in some settings (e.g. *polynomial time/space*) is simply not efficient enough here. The third part significantly improves a basic computational technique (called *dimension reduction*) abundantly found in algorithms on massive datasets.

Part I

Lower Bounds for Linear Degeneracy Testing

Chapter 1

Introduction

1.1 Definition of problem

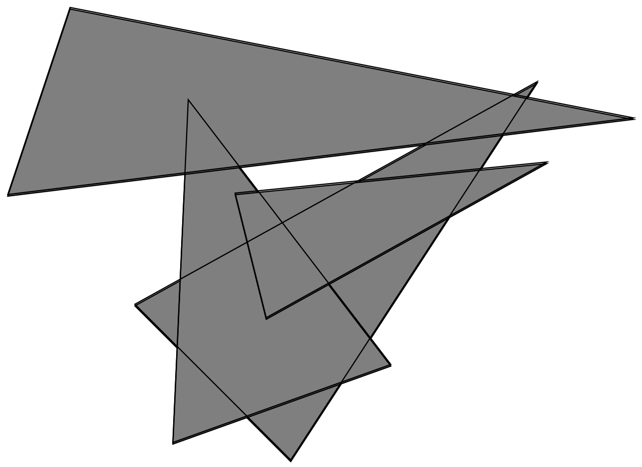
When designing algorithms in computational geometry, one often makes the simplifying assumption of the input being in *general position.* For example, if the input to the problem is a collection of points in the plane, we may require that "no three points lie on the same line". This is equivalent to the nonsingularity of the matrices

$$\begin{pmatrix} x_1 & x_2 & 1 \\ y_1 & y_2 & 1 \\ z_1 & z_2 & 1 \end{pmatrix},$$

where $x, y, z \in \mathbb{R}^2$ are any three points from the input, which is in turn equivalent to the non-vanishing of the corresponding determinant multinomials.

If the application is construction of a Voronoi diagram, then we may require that no four points lie on the same circle. This corresponds, again, to the non-vanishing of a certain multinomial evaluated at the combined coordinates of all possible choices of four input points. We call inputs that are not in general position *degenerate*, because the set of such n-dimensional[1] inputs is a (closed) subset of measure 0 in Euclidean space. Similarly, we can formulate the degeneracy of power diagrams, algebraic varieties, real semi-algebraic sets, etc. Classical "bichromatic" problems also

[1]Note that the dimension here is that of the entire input, for example, m points in $\mathbb{R}^2$ has dimension $n = 2m$.



Does the union of a given set of triangles in the plane contain a "hole"?

Figure 1.1: A 3SUM-hard problem

fall in that category: for example, checking incidence between points and hyperplanes (Hopcroft's problem), rays and triangles, lines and spheres, etc.

Though these *degenerate* cases can be handled for any algorithm with special care, it is easier to ignore them as a first approximation while implementing the algorithms, or when devising new ones. In real life, of course, we cannot assume that the input is always in general position. Degeneracies do occur, and one must take care of them. But the importance of studying degeneracies in computational geometry does not lie solely in this pessimistic approach to real life scenarios. Testing for degeneracies is an important problem in its own right. The problem 3SUM is defined as that of determining whether for an input of n real numbers $x_1, \dots, x_n$, there exist three indices $1 \leq i_1 < i_2 < i_3 \leq n$ such that $x_{i_1} + x_{i_2} + x_{i_3} = 0$. The problem 4SUM is defined similarly. There is a vast collection of geometric problems known to be 3SUM-hard and 4SUM-hard, all of which are at least as hard as rSUM (for $r = 3, 4$) via subquadratic reductions [58]. Classical examples are separating line segments by a line, testing if a union of triangles is simply connected (Figure 1.1), checking for polygon containment under translation, minimizing the Hausdorff distance between segment sets, computing the Minkowski sum of two polygons, sorting the vertices of a line arrangement, etc. [10,16,17,25,36,84]. Needless to say, the importance of elucidating the complexity of these degeneracy testing problems can hardly be overstated.

This work studies the complexity of deciding *linear* degeneracy. More precisely, given a fixed linear polynomial $f(t_1, \ldots, t_r) = \sum_{j=1}^{r} \alpha_j t_j - \alpha_0$, a point $x \in \mathbb{R}^n$ is f-degenerate if there exists a collection of distinct indices $i_1, \ldots, i_r \in [n]$ such that

$$f(x_{i_1}, \ldots, x_{i_r}) = 0 \ .$$

For simplicity we will assume that $\alpha_0 = 0$ and $\alpha_j = 1$ for $j = 0 \ldots r$. We will show in Chapter 6 that this restriction is immaterial. In this simpler case, degeneracy of x is equivalent to the existence of an increasing sequence of indices $1 \leq i_1 < \cdots < i_r \leq n$ such that $\sum_{j=1}^{r} x_{i_j} = 0$, *viz.* rSUM.

Using the standard terminology of *language* from complexity theory, we now formally define:

Definition 1.1. The language $r\mathrm{SUM} \subseteq \mathbb{R}^*$ is defined[2] as $\cup_{n\geq 1} r\mathrm{SUM}_n$, where

$$r\mathrm{SUM}_n = \bigcup_{1 \leq i_1 < \cdots < i_r \leq n} \{x : \ x_{i_1} + \cdots + x_{i_r} = 0\} \ \subseteq \mathbb{R}^n \ .$$

The hyperplane $\{x : \ x_{i_1} + \cdots + x_{i_r} = 0\} \ \subseteq \mathbb{R}^n$ is called an r-*canonical* hyperplane. We denote by $\mathcal{C}_{r,n}$ the set of all r-canonical hyperplanes in $\mathbb{R}^n$.

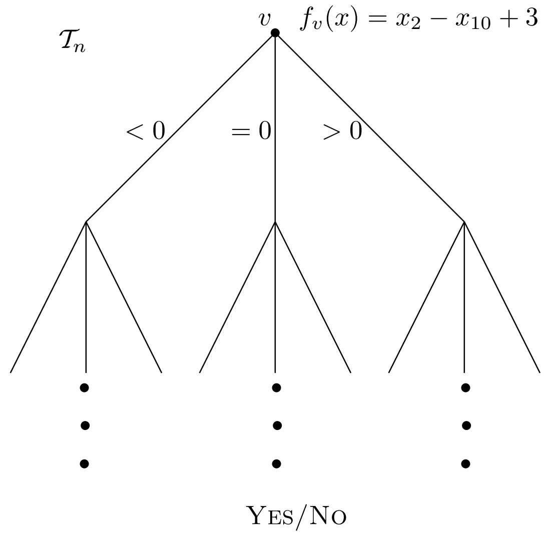
1.2 The computational model

The size of an input instance $x \in \mathbb{R}^n$ to rSUM is n. Clearly, if both r and n are part of the input, the problem is NP-complete (via a simple reduction from SubsetSum, for example). However, in this work we allow the coordinates of x to be arbitrary real numbers, and we ignore their representation size.

Instead of considering the Turing machine, we consider the *linear decision tree* model of computation. Decision trees have often shown to be realistic and effective models for proving lower bounds on the complexity of fundamental geometric problems [19,22,41,46–48,61,62,102,106,107].

Definition 1.2. A linear decision tree (LDT) algorithm $\mathcal{T}$ is a collection $\{\mathcal{T}_n\}_{n\geq 1}$ of ternary rooted trees. A linear polynomial (henceforth a *query*) $f_v \in \mathbb{R}[t_1, \ldots, t_n]$ is assigned to all internal

[2] By $\mathbb{R}^*$ we mean $\cup_{n\geq 1}\mathbb{R}^n$.



To compute $\mathcal{T}_n(x)$ for input $x \in \mathbb{R}^n$, evaluate $f_v(x)$, branch on output, and continue walking down until an output YES/NO leaf is reached.

Figure 1.2: A linear decision tree

nodes v of $\mathcal{T}_n$, and YES/NO labels are attached to the leaves.

For input $x \in \mathbb{R}^n$, the computation $\mathcal{T}(x)$ is carried out as follows. Set $v = \text{root}(\mathcal{T}_n)$. While v is not a leaf, evaluate $f_v(x)$, and if the result is < 0 (resp. $= 0$) (resp. > 0) then set v to its left (resp. middle) (resp. right) child. Finally, output the label of the leaf v.

The language decided by $\mathcal{T}$ is the locus of points in $\mathbb{R}^*$ for which $\mathcal{T}(x)$ =YES. The running time $\text{TIME}_{\mathcal{T}}(n)$ is the height of $\mathcal{T}_n$. The complexity of a language in $\mathbb{R}^*$ is the running time of the fastest linear decision tree deciding it.

Refer to Figure 1.2 for an illustration of LDT's. Note that the LDT is a *nonuniform* model of computation: the tree $\mathcal{T}_n$ as a function of n may even be undecidable.

Much research has been done on the more general algebraic decision trees and algebraic computation trees (e.g. [19]). In the former, the f_v's can be arbitrary (not necessarily linear) multinomials. In the latter, the f_v's may also contain indeterminates that are substituted by values computed in previous nodes. We will restrict our discussion to the LDT model. These extended models of computation may suggest interesting future generalizations.

We can view the query polynomials attached to the nodes v of $\mathcal{T}_n$ as hyperplanes (henceforth *query* hyperplanes) in n-space, and the evaluation process at v is geometrically viewed as testing whether the input point x is above, below, or contained in the query hyperplane $q_v \overset{\text{def}}{=} \{x : f_v = 0\}$.

The parametrized complexity of rSUM as a function of r is still poorly understood. A naive solution for rSUM is a "spaghetti" tree sequentially testing against all canonical hyperplanes. The running time of this algorithm is $O\left(\binom{n}{r}\right)$.

1.3 Previous results

The trivial $O\left(\binom{n}{r}\right)$ upper bound can be improved [46] to

$$\begin{cases} O\left(n\binom{n}{\lfloor r/2 \rfloor}\right) & r \text{ odd} \\ O\left(\binom{n}{r/2} r \log(n/r)\right) & r \text{ even} \end{cases}.$$

We call these improved algorithms *meet-in-the-middle*. The idea for r even is to compute the sums $x_{i_1} + \cdots + x_{i_{r/2}}$ for all possible $1 \le i_1 < \cdots < i_{r/2} \le n$, and store them in an ascending-order sorted list L. In each entry in the list, we also store the index set $\{i_1, \ldots, i_{r/2}\}$ generating it. Similarly, we then form a list L' of the numbers $-(x_{i'_1} + \cdots + x_{i'_{r/2}})$ for all possible $1 \le i'_1 < \cdots < i'_{r/2} \le n$. Any value shared by both lists corresponding to disjoint index sets $\{i_1, \ldots, i_{r/2}\}$ and $\{i'_1, \ldots, i'_{r/2}\}$ is a witness for degeneracy. This can be found be merging the two sorted lists, with special care taken for handling the difficulty of disjointness of two index sets. The total running time is dominated by the sorting time of $O\left(\binom{n}{r/2} r \log(n/r)\right)$. Although stated as a program in a high-level programming language, this can actually be implemented as a linear decision tree. Using the nonuniformity of the model of computation, and a result by Fredman [56], the above algorithm can be improved to a running time of $O\left(\binom{n}{r/2}\right)$.

If r is odd, then we create a sorted list L of the numbers $x_{i_1} + \cdots + x_{i_{\lfloor r/2 \rfloor}}$ corresponding to all choices of $1 \le i_1 < \cdots < i_{\lfloor r/2 \rfloor} \le n$, and similarly L' of the numbers $-(x_{i'_1} + \cdots + x_{i'_{\lfloor r/2 \rfloor}})$. Then, any $i \in [n]$ such that the lists $L + x_i$ and L' share a common value corresponding to index sets $\{i_1, \ldots, i_{\lfloor r/2 \rfloor}\}$ and $\{i'_1, \ldots, i'_{\lfloor r/2 \rfloor}\}$ that are disjoint and do not contain i is a witness for degeneracy. To find such a witness we try all $i \in [n]$, and for each i we merge the lists $L + x_i$ and L' in time linear in the size of the lists (again, taking care of the index-set disjointness technicality). The total running time is dominated by the n merges of size $O\left(\binom{n}{\lfloor r/2 \rfloor}\right)$, giving the stated running time.

More dramatic improvements can be achieved by taking more advantage of nonuniformity and the fact that there is no restriction on the hyperplanes that can be used in the linear decision tree. Indeed, it is very easy to see that the only hyperplanes used in meet-in-the-middle are hyperplanes parallel to all but at most r axes of $\mathbb{R}^n$. In other words, the normal vectors to these hyperplanes have at most r nonzeros. By a result of Meyer auf der Heide [88], a decision tree of depth $O(n^4 \log n)$ exists for rSUM, for any r. The existence of an unconstrained linear decision tree with depth poly(n, r) deciding rSUM also follows from work by Meiser [87].

Information theoretic lower bounds of $\Omega(n \log n)$ on the depth of a tree deciding rSUM in an unconstrained linear decision tree model are obtained by Dobkin and Lipton [41], and under more general nonlinear models of computation by Steele and Yao [102] and Ben-Or [19].

In this work, we will be interested in the restricted model of computation, which we more formally call an s-restricted linear decision tree:

Definition 1.3. An s-restricted linear decision tree (sLDT) is an LDT in which the linear polynomials f_v corresponding to all internal nodes v have the form

$$f_v = \sum_{j=1}^{s} \alpha_j x_{i_j} - \beta$$

for some $\alpha_1, \ldots, \alpha_s, \beta \in \mathbb{R}$ and indices $1 \leq i_1 < \cdots < i_s \leq n$. In other words, the normals q_v^* to the corresponding hyperplanes q_v have at most s nonzeros.

As stated above, meet-in-the-middle is an rLDT algorithm. It is not hard to see that an sLDT algorithm for $s < r$ cannot decide rSUM. In fact, the collection of hyperplanes $\mathcal{Q}_n$ queried by a tree $\mathcal{T}_n$ deciding rSUM_n must contain $\mathcal{C}_{r,n}$. Otherwise, take a hyperplane $C \in \mathcal{C}_{r,n} \backslash \mathcal{Q}_n$. By finiteness of $\mathcal{Q}_n$, there exists a point $x \in C$ that is not contained in any hyperplane from $\mathcal{Q}_n$. This point is degenerate (a Yes instance), but a small perturbation of it moves it to a nondegenerate point x' (a No instance) such that the straight line connecting x and x' traverses no hyperplane in $\mathcal{Q}_n$. Thus, x and x' follow the same computational path in $\mathcal{T}_n$, a contradiction.

Improving on previous work [39, 56], Erickson [46] proved that *any* rLDT deciding rSUM has depth $\Omega(n^{\lceil r/2 \rceil})$ for fixed r. His proof is quite a tour de force. It is packed with ingenious, tightly coupled arguments, and its only downside is to offer little wiggle room to try out new ideas. In particular, extending the proof to s-linear trees for $s > r$ has long been elusive. Even the case $s = r + 1$, mentioned in Yao's list of major open problems in his 2000 DIMACS lecture [108], has resisted all efforts. The contribution of this work, while far from closing the book on the problem, represents a significant advance on two fronts: (i) accommodating $s > r$ variables and (ii) allowing for larger values of r.

- We prove a lower bound of $\Omega(nr^{-3})^{\lceil r/2 \rceil}$ on the depth of any rLDT deciding rSUM. This improves on Erickson's bound[3] of $\Omega(nr^{-r})^{\lceil r/2 \rceil}$. For moderately large values of r, this improves his lower bounds from sub-constant (trivial) to exponential. Indeed, if say $r = r(n) > n^{\varepsilon}$, Erickson's bound falls below any constant while ours is of the form $2^{n^{\Omega(1)}}$. The technical underpinning of this improvement is a new adversarial strategy based on error-correcting codes.

- By using a tensor product construction based on permutation matrices, we are able to generalize the lower bound to the sLDT model for $s > r$. We show that an sLDT deciding rSUM must have depth of at least

  $$\Omega(nr^{-3})^{\frac{2r-s}{2\lceil (s-r+1)/2 \rceil}(1-\varepsilon_r)},$$

  where $\varepsilon_r > 0$ tends to 0 as $r \to \infty$. Note that for $s = r + 1$, this bound takes the simpler form

  $$\Omega(nr^{-3})^{\frac{r-1}{2}(1-\varepsilon_r)}.$$

  In words, this tells us that an "extra coefficient" in the LDT hyperplanes does not buy us too much power when compared to rLDT.

The exponential lower bound still holds for $s > r$. For any fixed $\varepsilon > 0$, the depth of an s-linear decision tree is $(nr^{-3})^{r^{\Omega(1)}}$, if $s \leq r + r^{1-\varepsilon}$. In the case $r > n^{\varepsilon}$, this gives a lower bound of $2^{n^{\Omega(1)}}$. Note that our bounds collapse if s is not $O(r)$. This is an obvious limitation of our method, but one must note that a dependency on s is inevitable. Indeed, our lower bound of $n^{\Omega(r)}$ for $s = r + O(1)$ *cannot* hold for arbitrary values of s, in view of Meyer auf der Heide and Meiser's results.

[3]Erickson's bound is stated as $\Omega(n^{\lceil r/2 \rceil})$ for r fixed, and the bound stated here is obtained by a careful analysis of his work.

Another contribution of this work is methodological. To obtain our bounds requires a whole set of new algebraic arguments, but our starting point is essentially a geometrization of Erickson's method. The main benefit is to bypass the complicated machinery of infinitesimals found in [46], obviate the need for Tarski's transfer principle, and more generally do away with analytical arguments.

To make the proof more digestible, we will begin our discussion with the geometric framework and then treat the case $s = r$. Next we will move on to the case $s = r + 1$, where we introduce the tensor product construction in its simplest form. Finally we will cover the general $s > r$ case.

Chapter 2

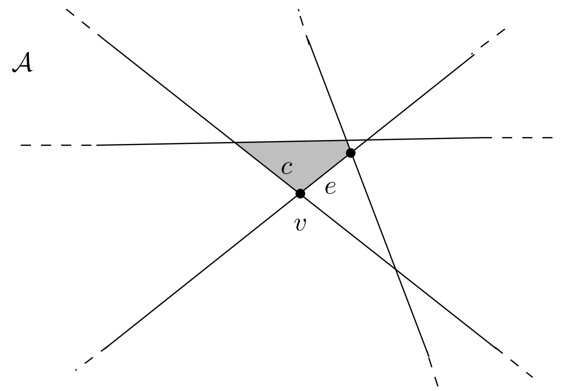
A Geometric Framework for Lower Bounds

2.1 Terminology and conventions

We will make use of standard geometric objects such as *convex polyhedra* and *arrangements*. Good introductions to the field can be found in [45, 60, 85, 111].

Given a finite collection $\mathcal{H}$ of hyperplanes in $\mathbb{R}^n$, the induced arrangement $\mathcal{A}$ is the equivalence relation on $\mathbb{R}^n$ defined as follows: $x\mathcal{A}y$ if for all hyperplanes $h \in \mathcal{H}$, either both x and y are contained in h, or both lie on same side of h. The equivalence classes are called *faces*. Each face F of $\mathcal{A}$ has a dimension, which equals the dimension of its affine closure. Faces of dimension k are called k-faces. *Vertices* are 0-faces, *edges* are 1-faces cells are n-faces, and *facets* are $(n-1)$-faces. Note that faces are relatively open in the topology induced by their affine closure.

Convex polyhedra in $\mathbb{R}^n$ are the intersection of a finite collection of closed half-spaces in $\mathbb{R}^n$. A convex polyhedron P also has faces of different dimensions, defined similarly to the arrangement faces with respect to the hyperplanes supporting the half-spaces defining the polytope. Note that the intersection of the closure of two faces of a convex polyhedron (or an arrangement) is either empty or the closure of a face. Also note that k-faces of convex polyhedra (or arrangements) can be viewed as convex polyhedra in $\mathbb{R}^k$.



Four lines (hyperplanes in two dimensions) induce an arrangement $\mathcal{A}$ with e.g. vertex v, an edge e (also a facet) and a cell c. The closure of c is a 2-dimensional bounded polyhedron (a polytope), and v, e are two of its faces.

Figure 2.1: Arrangements and polyhedra

Fix r and n, and let $\mathcal{T}_n$ be an LDT deciding rSUM_n. Let $\mathcal{Q}$ denote the collection of query hyperplanes q_v for v ranging over all internal nodes of $\mathcal{T}_n$. Let $\mathcal{A}$ denote the arrangement induced by $\mathcal{Q}$, and let $\mathcal{B}$ denote the arrangement induced by $\mathcal{C}_{r,n}$ (recall that $\mathcal{C}_{r,n}$ is the collection of canonical hyperplanes). Let $\mathcal{A}'$ denote the equivalence relation induced by the leaves of $\mathcal{T}_n$, namely $x\mathcal{A}'y$ if and only if the two computational paths corresponding to x and y are identical. Note that the equivalence classes of $\mathcal{A}'$ are convex polyhedra (we will call them faces). Clearly, $\mathcal{A}$ is a refinement of $\mathcal{A}'$ (i.e. any face of $\mathcal{A}$ is contained in some face of $\mathcal{A}'$). Also note that by definition of "$\mathcal{T}_n$ deciding rSUM_n", any face of $\mathcal{A}'$ is either entirely contained in rSUM_n (and therefore in some canonical hyperplane) or disjoint from rSUM_n.

A linear hyperplane in $\mathbb{R}^n$ is a hyperplane containing the origin. An arrangement induced by linear hyperplanes is called a *fan*. The faces of fans are polyhedral *cones*. By definition, $\mathcal{B}$ is a fan. It is easy to see that all query hyperplanes of $\mathcal{T}_n$ can be assumed to be linear. In other words, $\mathcal{A}$ is also a fan. Indeed, since $\mathcal{T}_n$ decides rSUM_n, evaluating $\mathcal{T}_n(x)$ is equivalent to evaluating $\mathcal{T}_n(\alpha x)$ for any number $\alpha > 0$. In particular, we could take α small enough so that αx lies on the same side as the origin with respect to all nonlinear query hyperplanes $q \in \mathcal{Q}$. Therefore, nonlinear queries need not be evaluated and they can be removed from $\mathcal{T}_n$.

By the last claim, we can identify all query hyperplanes as well as canonical hyperplanes with their normals. We will use h^* to denote a vector normal to the hyperplane $h \subseteq \mathbb{R}^n$, and we can then write

$$h = \{x \in \mathbb{R}^n : \langle h^*, x \rangle = 0\} .$$

Finally, we define the notion of a *distinguishing* hyperplane. A hyperplane $h \subseteq \mathbb{R}^n$ distinguishes between two points $x, y \in \mathbb{R}^n$ if either x and y lie on opposite sides of h, or exactly one of x, y is contained in h. Clearly, if $x \in \mathrm{r}SUM_n$ (a Yes instance) and $y \notin \mathrm{r}SUM_n$ (a No instance), then both computation paths of $\mathcal{T}_n(x)$ and $\mathcal{T}_n(y)$ must contain at least one hyperplane distinguishing between x and y. We will also say that a hyperplane distinguishes between a point x and a set Y if it distinguishes between x and y for all $y \in Y$.

2.2 Overview of proof

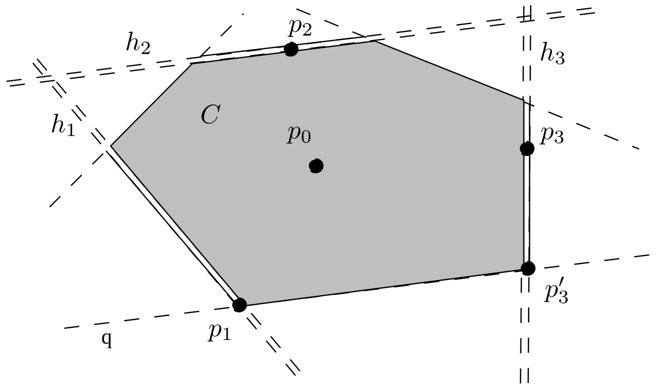
Like other geometric decision tree lower bound results, our proof will identify a notion of high complexity in the canonical (degenerate) hyperplane geometry. More precisely, the basic idea of the proof is to identify a nondegenerate face C in the arrangement $\mathcal{A}$ induced by $\mathcal{Q}$ (recall that $\mathcal{Q}$ contains the set of canonical hyperplanes). This face, which we call the *chamber*, has a high *degeneracy-complexity* measure, defined as follows. Let $S = \{p_1, \ldots, p_d\} \subseteq \partial C$ be a set of degenerate points. We say that S is *C-independent* if no two distinct points $p_i, p_j \in S$ lie on the closure of the same facet of the polytope $\bar{C}$. This is equivalent to the condition that for any $p_0 \in C$, no hyperplane from $\mathcal{Q}$ can simultaneously distinguish between p_0 and p_i and between p_0 and p_j. The degeneracy-complexity measure of C is the maximal size of a C-independent set S.

Lemma 2.1. *For any nondegenerate face C of $\mathcal{A}$, the degeneracy-complexity measure of C is a lower bound for the height of $\mathcal{T}_n$.*

Proof. Let $S = \{p_1, \ldots, p_d\}$ be any C-independent set[1], and let $p_0 \in C$. Consider the path of nodes $v_1, \ldots, v_l$ visited in the computation $\mathcal{T}_n(p_0)$. Since p_0 is a No instance of $\mathrm{r}SUM_n$ (a nondegenerate point), and p_i is a Yes instance (a degenerate point) for all $i = 1, \ldots, d$, it follows that at least one of the query hyperplanes $q_{v_1}, \ldots, q_{v_l}$ must distinguish between p_0 and p_i. But by the definition of C-independence, we know that any q_{v_j} can distinguish between p_0 and at most one p_i. This means that $l \geq d$, lower-bounding the height of $\mathcal{T}_n$, as required. $\square$

[1]We implicitly assume from the definition of C-independence that $S \subseteq \partial C \cap \mathrm{r}SUM_n$.

Using adversarial-type proof terminology, we can say that if the height of $\mathcal{T}_n$ is less than $|S|$ for some C-independent set S then an adversary can "collapse" the nondegenerate point $p_0 \in C$ to one of the degenerate $p_i \in S$ in a way that is indistinguishable for the algorithm $\mathcal{T}_n$. Following the terminology from [46], we will call the points in S *collapsed* points.



The hyperplanes h_1, h_2, h_3 marked by double-lines are the critical hyperplanes. The points p_1, p_2, p_3' are not C-independent, because q can simultaneously distinguish between p_0 and p_1, p_3'. The points p_1, p_2, p_3 are C-independent.

Figure 2.2: The face C of $\mathcal{Q}$, collapsed points and critical hyperplanes

This game between the algorithm and the adversary is very easy to visualize (Figure 2.2). In the actual proof, instead of directly identifying the chamber C, we will work our way backwards. We first identify a nondegenerate point p_0 and a potential set of collapsed (degenerate) points. Then we will argue (using the restrictions on the set of queries $\mathcal{Q}$ used in $\mathcal{T}_n$) that no hyperplane from $\mathcal{Q}$ can simultaneously distinguish between p_0 and two points in S. In fact, we will show the stronger condition that there is exactly one hyperplane in $\mathcal{Q}$ that distinguishes between any collapsed point $p \in S$ and p_0, and this unique hyperplane is canonical. Therefore, there is a bijection between the collapsed points and the distinguishing canonical hyperplanes, called *critical* hyperplanes and denoted by $\mathcal{H}$. The collapsed point corresponding to $h \in \mathcal{H}$ will be denoted by p_h, and will be contained in h. It is not hard to see that the set $S = \{p_h\}_{h \in \mathcal{H}}$ is a C-independent set of points, where C is the unique face of $\mathcal{A}$ containing p_0.

Consistently working our way backwards, we will start by defining the set $\mathcal{H}$ of critical hyperplanes in the next chapter. By the above discussion, the lower bound for $\mathcal{T}_n$ will be $|\mathcal{H}|$.

Chapter 3

The Case $s = r$

In this chapter we will analyze the case considered in [46], namely, lower bounds for rSUM under rLDT. We improve the dependence of the lower bound on r using the theory of error correcting codes, replacing a construction based on a Vandermonde matrix used there.

3.1 Critical hyperplanes via error-correcting codes

By padding the input if necessary, we can always assume that $n = rm$, for some integer m. This allows us to view a normal vector $h^* \in \mathbb{R}^n$ as an $r \times m$ real matrix M^{h^*}, whose rows are filled with the coordinates of h^*; i.e., $M_{ij}^{h^*} = h^*_{(i-1)m+j}$.

A canonical hyperplanes $h \in \mathcal{C}_{n,r}$ is of the form

$$\{x \in \mathbb{R}^n : \; x_{i_1} + \cdots + x_{i_r} = 0\} \; .$$

In other words, M^{h^*} has exactly r 1's and $n - r$ 0's. The canonical hyperplanes we will choose for the set of critical hyperplanes $\mathcal{H}$ will have exactly one 1 in each row.

Where to put the 1's is dictated by an error-correcting recipe meant to ensure high "independence". Throughout this chapter we use the shorthand

$$r_0 = \lceil r/2 \rceil. \tag{3.1}$$

Let t be the smallest prime greater than r, and let $\mathcal{M}$ be a Reed-Solomon code [83] of length $t-1$ and distance $r-r_0+1$ over the finite field $\mathbb{F}_t$. This means that $\mathcal{M}$ is a linear subspace of $\mathbb{F}_t^{t-1}$ with the following combinatorial property: any nonzero vector in $\mathcal{M}$ has at least $r-r_0+1$ nonzero coordinates. A constructive way to do this is to regard $\mathbb{F}_t^{t-1}$ as the ring of polynomials $\mathbb{F}_t[X]$ modulo the polynomial $X^{t-1}-1$. We then pick some primitive[1] $\beta \in \mathbb{F}_t$ and let $\mathcal{M}$ be the ideal in this ring generated by the polynomial $(X-\beta)(X-\beta^2)\cdots(X-\beta^{r-r_0})$. This ideal has dimension $k=t-1-r+r_0$ with the desired distance property (see [83] for details).

Now, define $\mathcal{M}_r$ to be the linear subspace of $\mathcal{M}$ defined by:

$$\mathcal{M}_r = \{x \in \mathcal{M} : \ x_i = 0 \text{ for } r < i \leq t-1\} \ .$$

In this way, we can think of $\mathcal{M}_r$ as a linear code of length r, distance greater than $r-r_0$ and dimension as large as $k-(t-1-r)=r_0$.

Let $u_1,\ldots,u_{r_0}$ be an independent set of vectors in $\mathcal{M}_r$. Of course, by permuting coordinates and performing column operations, we can always assume that the set is in column echelon form, i.e., the $r \times r_0$ matrix $(u_1,\ldots,u_{r_0})$ consists of the $r_0 \times r_0$ identity matrix on top of some $(r-r_0)\times r_0$ matrix.

$$(u_1,\ldots,u_{r_0}) = \overbrace{\begin{pmatrix} 1 & & \\ & \ddots & \\ & & 1 \\ * & \cdots & * \\ \vdots & & \vdots \\ * & \cdots & * \end{pmatrix}}^{r_0} \begin{array}{l} \left.\vphantom{\begin{matrix}1\\ \ddots\\ 1\end{matrix}}\right\} r_0 \\ \left.\vphantom{\begin{matrix}*\\ \vdots\\ *\end{matrix}}\right\} r-r_0 \end{array}$$

Since $\mathbb{F}_t$ is a prime field, we can naturally view the u_i's as vectors in $\mathbb{R}^r$ with coordinates in $\{0,\ldots,t-1\}$. From now on, we view them vectors as *real* vectors (and not over $\mathbb{F}_t$).

[1]Meaning that the sequence $1,\beta,\beta^2,\ldots,\beta^{q-2}$ has no repetitions.

We define $\mathcal{L}$ as:

$$\mathcal{L} = \{n_1 u_1 + \cdots + n_{r_0} u_{r_0} : \ 1 \le n_i < m/(r_0 t) \ \forall i = 1, \ldots, r_0\} \ .$$

The upper bound of $m/(r_0 t)$ is chosen so that all coordinates of vectors in $\mathcal{L}$ lie in $\{0, \ldots, m-1\}$. Note that $\mathcal{L}$ is similar to a lattice in $\mathbb{R}^n$ with basis $u_0, \ldots, u_{r_0}$, except that it has bounded coordinates.

Lemma 3.1. *Our construction of $\mathcal{L}$ satisfies the following three properties:*

(i) The first r_0 coordinates of any vector in $\mathcal{L}$ specify it uniquely.

(ii) The set $\mathcal{L}$ consists of at least $(n/r^3)^{r_0}$ vectors in $\mathbb{R}^r$ with coordinates in $\{0, \ldots, m-1\}$.

(iii) Any nonzero vector in[2] $\operatorname{span} \mathcal{L}$ *has at least $r - r_0 + 1$ nonzero coordinates.*

Proof. Part (i) follows immediately from the echelon form of the matrix formed by the basis $\{u_1, \ldots, u_{r_0}\}$.

To see part (ii), we use a well-known number-theoretic theorem by Nagura [91], stating that the interval $[x, 6x/5]$ contains a prime for any $x \ge 25$. This shows that $tr_0 \le r^2$. Therefore, for $r = \Omega(1)$ and $m = \Omega(r^2)$,

$$|\mathcal{L}| \ge \left(\frac{m}{tr_0} - 1\right)^{r_0} \ge (nr^{-3})^{r_0} \ .$$

To prove (iii), consider a nonzero element $u = \sum_{i=1}^{r_0} \alpha_i u_i$ of $\operatorname{span} \mathcal{L}$. Assume by contradiction that u has at least r_0 zero coordinates $i_1, \ldots, i_{r_0}$. We claim that if such a vector exists, then there exists another vector $u' = \sum_{i=1}^{r_0} \alpha'_i u_i$ with rational α'_i's and zeros in the exact same coordinates $i_1, \ldots, i_{r_0}$. Indeed, constraining r_0 fixed coordinates to 0 in $\operatorname{span} \mathcal{L}$ is a homogenous linear system of constraints on the numbers $\alpha_1, \ldots, \alpha_{r_0}$ with integer coefficients. This system of constraints is over the rationals, and therefore a nontrivial real solution exists if and only if a nontrivial rational one exists. Now multiply u' by the unique positive rational number a such that the coordinates of au' have no common divisor. In particular, at least one coordinate of au' is not divisible by t. Therefore, the vector $au' (\operatorname{mod} t) \in \mathbb{F}_t^r$ is a nonzero vector in $\mathcal{M}_r$ with at least r_0 nonzeros, a contradiction to its error correcting code property.

□

[2] Throughout this work, the span operator is over the *reals*.

$$\ell = \begin{pmatrix}0\\3\\1\end{pmatrix} \in \mathcal{L} \quad \Rightarrow \quad h^* = \begin{pmatrix}1&0&0&0\\0&0&0&1\\0&1&0&0\end{pmatrix}$$

$$r = 3,\ m = 4\ \ (n = mr = 12)$$

Figure 3.1: From error correcting code vectors to critical hyperplane normals

The set $\mathcal{H}$ of critical hyperplanes will be defined in bijection with $\mathcal{L}$. The hyperplane h corresponding to $\ell = (\ell_1, \ldots, \ell_r) \in \mathcal{L}$ is defined by its normal vector (in matrix notation) M^{h^*}: the coordinate ℓ_i indicates where to place the 1 in the i-th row of the matrix, i.e.,

$$M_{ij}^{h^*} = \begin{cases} 1 & j = \ell_i + 1 \\ 0 & \text{otherwise} \end{cases}.$$

By construction,

$$M^{h^*}(0, \ldots, m-1)^T \in span\,(\mathcal{L}). \tag{3.2}$$

Also, since M^{h^*} has exactly one 1 in each row and zeros elsewhere, we note that

$$M^{h^*}(1, \ldots, 1)^T = (1, \ldots, 1)^T\ . \tag{3.3}$$

The intersection $\cap\mathcal{H} = \cap_{h \in H} h$ of all the hyperplanes $h \in \mathcal{H}$ is a linear subspace of positive dimension. Indeed, it contains any multiple of the vector

$$(\underbrace{1, \ldots, 1}_{n-m}, \underbrace{1-r, \ldots, 1-r}_{m}).$$

Let $\mathcal{K}$ denote the set of query hyperplanes in $\mathcal{Q}$ that contain $\cap\mathcal{H}$. Note that $\mathcal{Q} \supseteq \mathcal{K} \supseteq \mathcal{H}$.

Lemma 3.2. *Given any* $q^* \in \mathcal{K}$,

(i) $M^{q^*}(1, \ldots, 1)^T = b\,(1, \ldots, 1)^T$ *for some real* b, *and*

(ii) $M^{q^*}(0, \ldots, m-1)^T \in \text{span}\,(\mathcal{L})$.

Proof. By elementary linear algebra, the linear hyperplane q contains $\cap\mathcal{H}$ if and only if

$$q^* \in (\cap\mathcal{H})^{\perp} = \text{span}\{h^*\}_{h\in\mathcal{H}} \ .$$

The proof of the lemma follows from assertions 3.2 and 3.3 for all $h \in \mathcal{H}$ and by linearity.

$\square$

3.2 The Chamber

Consider a query hyperplane $q \in \mathcal{Q}\backslash\mathcal{K}$. Since $q \not\supset \cap\mathcal{H}$, either q is parallel to $\cap\mathcal{H}$ (i.e. $q \bigcap (\cap\mathcal{H}) = \emptyset$) or q traverses $\cap\mathcal{H}$ (i.e. $q \bigcap (\cap\mathcal{H})$ is a subspace of dimension $(\dim \cap\mathcal{H}) - 1$). In words, the query hyperplanes outside of $\mathcal{K}$ intersect $\cap\mathcal{H}$ in strictly lower-dimensional subspaces. Therefore, by finiteness of $\mathcal{K}$, there exists $c_0 \in \cap\mathcal{H}$ and $\rho > 0$ such that the ball $B(c_0, \rho)$ centered at c_0 of radius ρ intersects none of the hyperplanes of $\mathcal{Q} \setminus \mathcal{K}$ (see Figure 3.2).

By lying on every critical hyperplane the point c_0 is highly degenerate. Moving it by some vector ψ to be specified next changes all of that (Fig. 3.2). We define the point

$$p_0 = c_0 + \psi \tag{3.4}$$

to be safely outside of the critical hyperplanes. To do that, we need a positive convex real function g, meaning one with positive second derivative; eg, $x \mapsto x^2 + 1$. For some fixed, small enough $\gamma > 0$, we define the vector $\psi \in \mathbb{R}^n$ by its matrix M^{ψ}:

$$M_{ij}^{\psi} = \begin{cases} \gamma g(j) & i \leq r_0 \\ \gamma^2 g(j) & \text{otherwise.} \end{cases} \tag{3.5}$$

Note: γ is a scaling factor that is absolutely needed. The reason we use γ^2, however, is in anticipation of the case $s > r$. We could use γ in this chapter instead.

Lemma 3.3. *The point p_0 lies outside of any canonical hyperplane and any hyperplane of $\mathcal{Q} \setminus \mathcal{K}$.*

This implies that the decision tree must output NO on input p_0. Note, however, that p_0 might still lie on a query hyperplane.

Proof. By choosing γ small enough, we can ensure that

$$\|\psi\|_2 \leq \rho/2 \ .$$

Therefore, the point p_0 lies inside $B(c_0, \rho)$, safely away from any hyperplane of $\mathcal{Q} \setminus \mathcal{K}$ (Fig. 3.2).

We have already observed that $\mathcal{Q}$ must contain all of the canonical hyperplanes. Therefore, the only danger is that p_0 lies on some canonical hyperplane $q \in \mathcal{C}_{n,r} \cap \mathcal{K}$. The normal q^* of q has the form

$$M^{q^*}_{ij} = \begin{cases} 1 & j = i_j \\ 0 & \text{otherwise} \end{cases} ,$$

for some index set $i_1, \ldots, i_r$. But this is impossible. Indeed, $c_0 \in \cap\mathcal{H} \subseteq q$, and so

$$\begin{aligned} \langle q^*, p_0 \rangle &= \langle q^*, c_0 + \psi \rangle \\ &= 0 + \langle q^*, \psi \rangle \\ &= \sum_{1 \leq i \leq r_0} \gamma g(i_j) + \sum_{r_0 < i \leq r} \gamma^2 g(i_j) > 0. \end{aligned} \tag{3.6}$$

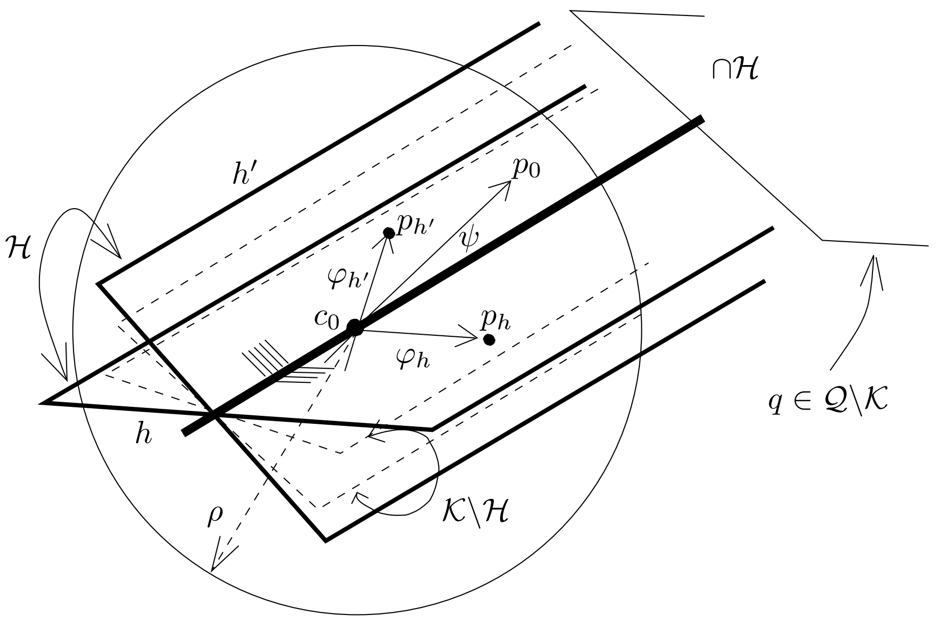
□

The chamber C we're interested in is the unique face of $\mathcal{A}$ that contains p_0. To define the map $h \in \mathcal{H} \mapsto p_h \in \partial C$ between critical hyperplanes and collapsed points, we need to introduce the vector space W spanned by the $2r$ vectors $u_k, w_k \in \mathbf{R}^n$ $(k = 1, \ldots, r)$, defined (using the matrix notation) as follows:

$$M^{u_k}_{ij} = \begin{cases} 1 & i = k \\ 0 & \text{otherwise} \end{cases} , \quad M^{v_k}_{ij} = \begin{cases} j & i = k \\ 0 & \text{otherwise} \end{cases} . \tag{3.7}$$

In other words, W consists of vectors w such that $M^w_{ij} = \alpha^w_i + \beta^w_i j$ for all i, j, for some real $\alpha^w_1, \ldots, \alpha^w_r, \beta^w_1, \ldots, \beta^w_r$. All the points p_h will lie on $\partial C \cap (p_0 + W)$. The reason for this will be made clear in case (A) in the proof of Lemma 3.4. Given $h \in \mathcal{H}$, we define a vector

$$\varphi_h \in \psi + W \tag{3.8}$$



Start from degenerate point $c_0 \in \cap\mathcal{H}$. Take a γ-small step in direction ψ to nondegenerate p_0, and collapse it onto p_h for all $h \in \mathcal{H}$. choose γ small enough so that all the action is in $B(c_0, \rho)$, safely avoiding $\mathcal{Q}\backslash\mathcal{K}$.

Figure 3.2: Main construction step

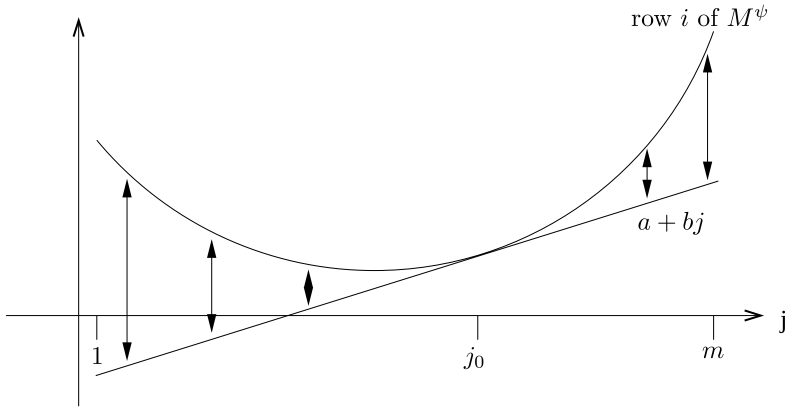
such that $M^{\varphi_h}_{ij} > 0$ (resp. $M^{\varphi_h} = 0$) if $M^{h^*}_{ij} = 0$ (resp. $M^{h^*}_{ij} \neq 0$). Note that $\psi + W$ is not necessarily a vector space. One should think of M^{φ_h} as a mask: Its rows mark with zeroes the positions where M^{h^*} is (w.l.o.g.) 1 and fill the rest with positive entries. To see that such a vector φ_h actually exists, consider the i-th row of the matrix M^{φ_h}. Let

$$\gamma_i = \begin{cases} \gamma & i \leq r_0 \\ \gamma^2 & i > r_0 \end{cases}.$$

It suffices to show that the row can satisfy constraints in a, b of the form $\gamma_i g(j) + a + bj = 0$ if j is equal to the one value j_0 where $M^{h^*}_{ij_0} = 1$, and $\gamma_i g(j) + a + bj > 0$ for any $j \neq j_0$. Feasibility is ensured by the fact that g is a convex function (see Figure 3.3).

It is immediate to check that as $\gamma \to 0^+$, one can choose φ_h so that

$$\lim_{\gamma \to 0^+} M^{\varphi_h}_{ij} = 0, \tag{3.9}$$



Row i of M^{φ_h} is the difference between row i of M^{ψ} and a straight line tangent to it at the unique coordinate j_0 for which $M^{h^*}_{ij_0} = 1$.

Figure 3.3: Constructing row i of M^{φ_h}

for all i, j (for example, one can half γ together with each entry of M^{ϕ_h}). This implies that, by scaling down γ if necessary, we can ensure that

$$\|\varphi_h\|_2 < \rho/2 \ . \tag{3.10}$$

We now define

$$p_h = c_0 + \varphi_h. \tag{3.11}$$

Lemma 3.4. *For any $h \in \mathcal{H}$, the only hyperplane $q \in \mathcal{Q}$ distinguishing between p_h and p_0 is $q = h$.*

Proof. Since p_h has 0 exactly in coordinates for which h^* has nonzeros, it follows that $p_h \in h$. By Lemma 3.3, $p_0 \notin h$. Therefore, h distinguishes between p_0 and p_h. We now show that no other query hyperplane $q \in \mathcal{Q}$ distinguishes between p_0 and p_h.

Assume $q \in \mathcal{Q} \backslash \mathcal{K}$. By (3.10), $p_h \in B(c_0, \rho)$, and $B(c_0, \rho)$ is not intersected by q. Hence, q does not distinguish between c_0 and p_h.

It remains to show that if $q \in \mathcal{K}$ and $q \neq h$ then q does not distinguish between c_0 and p_h. We distinguish between two cases.

(A) The normal q^* has a null row. But by Lemma 3.2 (i), the sum of any two rows of M^{q^*} are

$$M^{q^*} = \begin{pmatrix} 0 & 0 & * & 0 & 0 & * \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & * & 0 & * & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ * & 0 & 0 & * & 0 & 0 \end{pmatrix}$$

For $s = r = 6$, the extreme scenario for case (A) of Lemma 3.4 is when the six nonzeros (marked by $*$'s) are paired in three rows (leaving three null rows). Since the distance of the error correcting code is 4, it follows by Lemma 3.2 (ii) and Lemma 3.1 (iii) that $M^{q^*}(0, \ldots, m-1)^T = 0$.

Figure 3.4: How the error correcting code works

equal. This means that the sum of any row of M^{q^*} is zero:

$$M^{q^*}(1, \ldots, 1)^T = (0, \ldots, 0)^T \ . \tag{3.12}$$

But since M^{q^*} has at most r nonzero coordinates, it has at most $\lfloor r/2 \rfloor$ non-null rows (in order for a row to have sum 0, it must either be null or contain at least two nonzeros).

Next, by Lemma 3.2 (ii),

$$M^{q^*}(0, \ldots, m-1)^T \in \operatorname{span} \mathcal{L} \ .$$

But since at most $\lfloor r/2 \rfloor$ rows of M^{q^*} are non-null, it follows that $M^{q^*}(0, \ldots, m-1)^T$ has at most $\lfloor r/2 \rfloor < r - r_0 + 1$ nonzeros. By the error correction code properties stated in Lemma 3.1 (iii) (Figure 3.4), this implies that

$$M^{q^*}(0, \ldots, m-1)^T = (0, \ldots, 0) \ . \tag{3.13}$$

By (3.12) and (3.13), it follows that q^* is orthogonal to the space W.

Finally,

$$\begin{aligned} \langle q^*, p_h \rangle &= \langle q^*, c_0 + \varphi_h \rangle \\ &\in \langle q^*, c_0 + \psi + W \rangle \\ &= \langle q^*, c_0 + \psi \rangle + \langle q^*, W \rangle \\ &= \langle q^*, p_0 \rangle + \{0\} \end{aligned}$$

Therefore q does not distinguish between p_h and p_0, as required.

(B) The normal M^{q^*} has no null row. But since it has at most r nonzero coordinates, it has exactly one nonzero in each row. But by Lemma 3.2 (i), we can assume w.l.o.g. that this coordinate equals 1. In particular, q is a canonical hyperplane, and q^* differs from h^* in at least one nonzero position. Recall from (3.6) that

$$\langle q^*, p_0 \rangle > 0 \ .$$

We claim that also $\langle q^*, p_h \rangle > 0$. Using the fact that $c_0 \in q$,

$$\begin{aligned} \langle q^*, p_h \rangle &= \langle q^*, c_0 + \varphi_h \rangle \\ &= \langle q^*, c_0 \rangle + \langle q^*, \varphi_h \rangle \\ &= \langle q^*, \varphi_h \rangle \end{aligned} \tag{3.14}$$

Now recall that φ_h is a mask of h^* in the sense that the zero (resp. positive) coordinates of φ_h correspond to ones (resp. zeros) in h^*. Since q^* has exactly one 1 per row, and in at least one row q^* has a 1 in a position corresponding to a 0 of h^*, it follows that

$$\langle q^*, p_h \rangle = \langle q^*, \varphi_h \rangle > 0 \ .$$

Again, q does not distinguish between p_h and p_0, as required.

□

Theorem 3.5. *The depth of any r-linear decision tree for r-*SUM *is*

$$\Omega(nr^{-3})^{\lceil r/2 \rceil}.$$

Chapter 4

The Case $s = r + 1$

4.1 What doesn't work in the previous proof?

The only place where the number s actually plays a role is in the proof of Lemma 3.4. Case (A) survives almost verbatim. The only problem is that the number of null rows of M^{q^*} can be at least $r - \lfloor s/2 \rfloor$, which can be less than r_0. We fix this by strengthening the error correcting code over $\mathbb{F}_r$ to be of distance $\lfloor s/2 \rfloor + 1$. This means that r_0 (the dimension of the code over $\mathbb{F}_r$) is set to be

$$r_0 = r - \lfloor s/2 \rfloor. \tag{4.1}$$

In the present case, the setting $r_0 = \lfloor r/2 \rfloor$ will do. This affects the error-correcting code: $\mathcal{M}_r \subseteq \mathbb{F}_t^r$ has distance $r - r_0 + 1$ and dimension as large as r_0, and the size of the set of critical hyperplanes $\mathcal{H}$ is now at least $(n/r^3)^{r_0}$, for the new value of r_0.

Case (B) is far more difficult to fix. The new query $q \in Q$ that must be taken care of is one in which the rows of M^{q^*} have exactly one nonzero element, except for one of them, i_0, which has two nonzeroes (the case of one nonzero in every row having already been handled). Again we can assume that all the nonzero elements are 1, except in one fixed row i_0, where the elements are α

and $1-\alpha$, for some real $\alpha \notin \{0,1\}$. Let

$$\gamma' = \begin{cases} \gamma & i_0 \le r_0 \\ \gamma^2 & i_0 > r_0 \end{cases} .$$

Taking row i_0 into account, we can rewrite (3.6) as

$$\begin{aligned} \langle q^*, p_0 \rangle &= \langle q^*, \psi \rangle \\ &= \gamma' \alpha g(j_{i_0}) + \gamma'(1-\alpha) g(j'_{i_0}) + \sum_{\substack{i=1 \\ i \ne i_0}}^{r_0} \gamma g(j_i) + \sum_{\substack{i=r_0+1 \\ i \ne i_0}}^{r} \gamma^2 g(j_i). \end{aligned}$$

If $i_0 > r_0$ then all is well. Indeed, by making γ small enough

$$\langle q^*, p_0 \rangle = \sum_{i=1}^{r_0} \gamma g(j_i) + O(\gamma^2) > 0. \tag{4.2}$$

(This is the reason we needed to use both γ and γ^2 in (3.5)). Note that the constant hiding in the O-notation in (4.2) depends on q, but by the finiteness of Q we can choose γ so as to satisfy the inequality simultaneously for all $q \in Q$. We now check whether q can distinguish between p_0 and p_h for some $h \in \mathcal{H}$, namely, we are interested in the sign of $\langle q^*, p_h \rangle$. Consider the first r_0 rows of M^{q^*}. By Lemma 3.1 (i), there is a unique $\tilde{h} \in \mathcal{H}$ such that M^{q^*} agrees with $M^{\tilde{h}^*}$ in the first r_0 rows. For this unique $\tilde{h} \in H$, we allow q to distinguish between p_0 and $p_{\tilde{h}}$. We need to show that for all $h \in H$ distinct from $\tilde{h}$, q does not distinguish between p_0 and p_h, namely, $\langle q^*, h^* \rangle > 0$. As before, we need only consider $\langle M^{q^*}, M^{\varphi_h} \rangle$. This dot-product can be written as

$$\langle M^{q^*}_{\text{high}}, M^{\varphi_h}_{\text{high}} \rangle + \langle M^{q^*}_{\text{low}}, M^{\varphi_h}_{\text{low}} \rangle ,$$

where M_{high} (resp. M_{low}) corresponds to the first r_0 (resp. last $r - r_0$) rows in the matrix notation. Clearly, as in the arguments for Lemma 3.4 case (B), $\langle M^{q^*}_{\text{high}}, M^{\varphi_h}_{\text{high}} \rangle > 0$. By setting γ to be small enough, and recalling that $\mathcal{Q}$ and hence $\mathcal{K}$ are finite, we can make sure that $\langle M^{q^*}_{\text{high}}, M^{\varphi_h}_{\text{high}} \rangle$ dominates $\langle M^{q^*}_{\text{low}}, M^{\varphi_h}_{\text{low}} \rangle$, and hence $\langle q^*, p_h \rangle > 0$, as required.

The case $i_0 \le r_0$ is a tougher nut to crack. In fact we have not found a way of tackling it

$$M^{h^*} = \left(\begin{array}{c} \text{place any matrix from } \mathcal{P} \text{ here} \\ \hline \text{error-correct here} \end{array}\right) \begin{array}{l} \left.\vphantom{\big(}\right\} r_0 \\ \left.\vphantom{\big(}\right\} r - r_0 \end{array}$$

Figure 4.1: Constructing critical hyperplanes for $s = r + 1$

directly. Consequently, our strategy is simply to modify $\mathcal{H}$ so that this case cannot happen. Recall that, for the purpose of Lemma 3.4, we can assume that $q^* \in \mathcal{K}$. As we observed in the proof of Lemma 3.2, this implies that $q^* \in \operatorname{span}\{h^*\}_{h\in\mathcal{H}}$. Thus, our goal is to redefine a large set $\mathcal{H}$ of critical hyperplanes so that, in addition to all the properties we expect of $\mathcal{H}$, the following should hold: If q^* is a vector of $\mathbb{R}^n$ such that (i) with the exception of one row $i_0 \leq r_0$ each of the first r_0 rows of M^{q^*} consists of a single 1 with 0's everywhere else, and (ii) the exceptional row, i_0, is null everywhere except for two entries summing up to 1, then q^* *cannot* be in the span of ${h^*}_{h\in H}$.

Recall from the construction of $\mathcal{H}$ that the first r_0 rows of any M^h ($h^* \in \mathcal{H}$) completely determine the remaining rows. Furthermore, each one of the first r_0 rows can be chosen by placing a 1 arbitrarily between positions 1 and $m_0 = \lfloor m/qr_0 \rfloor$ and filling the rest of the row with 0's. So it suffices to concentrate on the first r_0 rows. Once we have the top r_0 rows, we use our Reed-Solomon code to fill in the bottom $r - r_0$ rows just as we did in the previous section.

An $r_0 \times a$ matrix is called *defective* if, with the exception of one row (called *anomalous*), each one consists of a single 1 with 0's everywhere else; furthermore the exceptional row is null everywhere except at two places. We postpone the proof of the next result.

Lemma 4.1. *There exists a set $\mathcal{P}$ $r_0 \times m$ 0/1 matrices with exactly one 1 per row between positions 1 and m_0 such that no defective $r_0 \times m$ matrix belongs to* $\operatorname{span}\mathcal{P}$ *and for any fixed $\varepsilon > 0$ and n large enough,*

$$|\mathcal{P}| \geq (nr^{-3})^{\lfloor r/2 \rfloor (1 - 1/\ln\lfloor r/2 \rfloor)(1-\varepsilon)}.$$

In view of our previous discussion, this automatically implies a lower bound on the depth of $(r+1)$-LDT's (Figure 4.1). The theorem below does not indicate what happens for small values of r. A careful examination shows that we obtain nontrivial lower bounds for any $r \geq 6$.

Theorem 4.2. *The depth of any $(r+1)$-LDT for r-SUM is at least $(nr^{-3})^{\lfloor r/2 \rfloor - o(r)}$.*

4.2 The tensor product construction

The problem fits into a general class of questions related to codes and combinatorial designs: How to build a large vector space that does not contain a family of forbidden vectors? In the case at hand, we start by building a "core" square matrix that satisfies the desired property and then show how to scale it up into an arbitrarily large rectangular matrix by using a suitable tensor product.

Let A be an $r_0 \times a$ real matrix, and let B be an $r_0 \times b$ real matrix. Following standard tensor notation, we write the element $A_{i,j}$ as A^i_j instead. We define a tensor product matrix operator

$$\otimes : \mathbf{R}^{r_0 \times a} \times \mathbf{R}^{r_0 \times b} \mapsto \mathbf{R}^{r_0 \times ab}$$

as follows: If $P = A \otimes B$ then $P^i_{j,k} = A^i_j B^i_k$. It is a mixed third-order tensor with two covariant indices and a single contravariant one.

This product extends to sets naturally. If $\mathcal{A}$ (resp. $\mathcal{B}$) is a set of $r_0 \times a$ (resp. $r_0 \times b$) real matrices, then

$$\mathcal{A} \otimes \mathcal{B} = \{\, A \otimes B \,|\, A \in \mathcal{A},\ B \in \mathcal{B} \,\}.$$

Tensor exponentiation for sets is defined by

$$\mathcal{A}^{\otimes k} = \underbrace{\mathcal{A} \otimes \cdots \otimes \mathcal{A}}_{k \text{ times}}.$$

The elements of $\mathcal{A}^{\otimes k}$ belong to the vector space $\mathcal{V}_k$ of mixed $(k+1)$st-order tensors with k covariant indices and 1 contravariant one. By fixing an ordering (say, lexicographic) of the covariant indices, we can interpret the tensors of $\mathcal{V}_k$ as $r_0 \times a^k$ matrices, and vice versa.

For our "core," we choose permutation matrices. Let Π denote the set of $r_0 \times r_0$ 0/1 matrices with exactly one 1 per row and column. The lemma below gives our tensor product its raison d'être.

Lemma 4.3. *No defective $r_0 \times r_0^k$ matrix can belong to the span of $\Pi^{\otimes k}$, for any $k \geq 1$.*

Proof. In any matrix of span(Π) each row and each column sum up to the same number, which can be assumed to be 1. Therefore, the anomalous row of a defective $r_0 \times r_0$ matrix consists of two entries, $\alpha, \alpha' \notin \{0, 1\}$ summing up to 1. Suppose that the column with the α also includes a set of ℓ ones (note that these can only be ones, and not some other nonzeros). Since the column sum is 1, we have $\alpha + \ell = 1$. Since $\alpha \neq 0, 1$, we conclude that α must be a negative integer. Therefore, $\alpha' = 1 - \alpha \geq 2$. But then the column with α' sums up to more than 1, which gives a contradiction. This proves the lemma for $k = 1$.

For $k > 1$, we define the tensor homomorphism $\Psi_l : \mathcal{V}_k \mapsto \mathcal{V}_{k-1}$, where

$$\Psi_l(P)^i_{j_1,\ldots,j_{l-1},j_{l+1},\ldots,j_k} = \sum_{j=1}^{r_0} P^i_{j_1,\ldots,j_{l-1},j,j_{l+1},\ldots,j_k}.$$

Let M be a defective $r_0 \times r_0^k$ matrix. By definition, its anomalous row i_0 contains two nonzero elements: the two corresponding covariant k-tuple indices, being distinct, differ in at least one index l. Since $k > 1$, there exists at least one covariant index $l' \neq l$. We easily verify that $\Psi_{l'}(M)$ is a defective $r_0 \times r_0^{k-1}$ matrix and

$$\Psi_{l'}(\Pi^{\otimes k}) = \Pi^{\otimes (k-1)}.$$

Therefore, by linearity,

$$\Psi_{l'}(M) \in \operatorname{span}(\Pi^{\otimes (k-1)}) \ .$$

By induction, this is a contradiction. $\square$

To maximize its size, we choose the set $\mathcal{P} = \Pi^{\otimes k}$ for the largest k such that $r_0^k \leq m_0 = \lfloor m/tr_0 \rfloor$. It is easy to verify that $|\mathcal{P}| = ((r_0)!)^k$, because the mapping $f : \Pi \times \cdots \times \Pi \to \mathbb{R}^{r_0 \times r_0^k}$ defined as

$$f(A_1, \ldots, A_k) = A_1 \otimes \cdots \otimes A_k$$

is an injection. Using Stirling's approximation, we find that

$$|\mathcal{P}| = |\Pi|^k = (r_0!)^k \geq (nr^{-3})^{\lfloor r/2 \rfloor (1 - 1/\ln \lfloor r/2 \rfloor)(1-\varepsilon)},$$

for any fixed $\varepsilon > 0$. Filling up each row with 0's to get the proper length m concludes the proof of Lemma 4.1. □

Chapter 5

The Case $s > r + 1$

We need a new idea to generalize the tensor product construction to higher values of s. We exploit the fact that the (hard part of the) lower bound involves only query hyperplanes whose normal vectors q^* are spanned by the normals h^* of the critical hyperplanes $h \in \mathcal{H}$. We use this to add combinatorial structure to the matrices M^{q^*} by redesigning the set $\cap\mathcal{H}$.

The first step is to redefine r_0: this is the number of top rows in the normals M^{h^*} to the critical hyperplanes in which we have "freedom" to choose where to position the 1's, whereas the remaining bottom rows serve as the error correcting code. In order to make case (A) of Lemma 3.4 work, we must have that the distance of our error correcting code $\lfloor s/2 \rfloor + 1$. This means that the dimension r_0 of the code (over $\mathbb{F}_t$) can be at most as in (4.1): $r_0 \le r - \lfloor s/2 \rfloor$.

Next, we will make an additional restriction on the definition of r_0. The idea behind this restriction will be soon made clear. We will choose $r_0 = \lambda\rho_0$, where

$$\lambda = \left\lfloor \frac{s-r}{2} \right\rfloor + 1 \quad \text{and} \quad \rho_0 = \left\lfloor \frac{r - \lfloor s/2 \rfloor}{\lambda} \right\rfloor .$$

Note that this subsumes our choice of r_0 for the cases $s = r$ and $s = r+1$ and satisfies the required constraint $r_0 \le r - \lfloor s/2 \rfloor$. Also note that this requires that s be not too large, say $s < \lfloor 3r/2 \rfloor$.

We now show how to construct normals of critical hyperplanes (Figure 5.1). Divide up the first r_0 rows of M^{h^*} into λ blocks of consecutive rows of ρ_0 rows each.

- **Step 1** Use the tensor construction of the previous chapter (the case $s = r + 1$) to produce

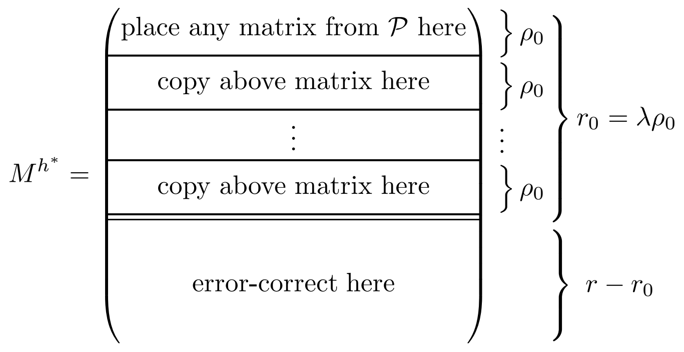


Figure 5.1: Constructing critical hyperplanes for $s > r + 1$

the top ρ_0 rows of M^{h^*}. In carrying out the construction, of course, use permutation matrices of size $\rho_0 \times \rho_0$ instead of $r_0 \times r_0$. This gives us a set $\mathcal{P}$ of matrices with the same properties as those of Lemma 4.1, except for the size of $\mathcal{P}$ and the size of the matrices, now $\rho_0 \times m$.

- **Step 2** For each matrix of $\mathcal{P}$, make λ copies of it and stack them on top of one another to produce an $r_0 \times m$ matrix.
- **Step 3** Complete the bottom $r - r_0$ rows via the error correcting code as before.

Lemma 5.1. *For any $q \in \mathcal{K}$, the top r_0 rows of M^{q^*} form an $r_0 \times m$ matrix made up of λ copies of the same $\rho_0 \times m$ matrix.*

Proof. This is a simple consequence of the fact that $q^* \in \text{span}\{h^*\}_{h \in \mathcal{H}}$. □

There is no need to revisit Lemma 3.4 in detail. Again, only case (B) is worth discussing at this point: Each row of M^{q^*} has at least one nonzero element. By analogy with the case $s = r + 1$, if all the rows with more than one nonzero have indices greater than r_0 then inequality (4.2) holds by choosing γ small enough, and we are done.

Suppose now that at least one row $i_0 \leq r_0$ contains two or more nonzeroes. By Lemma 5.1, the λ blocks that make up the top r_0 rows of M^{q^*} are identical. This shows that no block can have more than $\rho_0 + 1$ nonzeroes. Indeed, any one of them did, then so would all of the others, and their combined contribution of nonzeroes would be at least $(\rho_0 + 2)\lambda$. Added to the (at least) $r - r_0$ nonzeroes of the bottom rows, this would give us a total of at least $(\rho_0 + 2)\lambda + r - r_0 > s$ nonzero coordinates in q^*, which is ruled out. So, the only possibility left is for each block to have exactly

ρ_0 or $\rho_0 + 1$ nonzeroes: the first case was handled in the proof of Lemma 3.4, while the second one was shown to be impossible in the last section because of the tensor product construction.

The new set $\mathcal{P}$ is of size at least $(\rho_0!)^k$, where k is the largest integer such that $\rho_0^k \leq m/tr_0$. Elementary calculations show that

$$|\mathcal{P}| \geq (nr^{-3})^{\frac{2r-s}{2\lfloor (s-r)/2 \rfloor + 1}(1-\varepsilon_r)},$$

where $\varepsilon_r > 0$ tends to 0 as $r \to \infty$.

Theorem 5.2. *The depth of any s-linear decision tree for r-*SUM *is at least*

$$(nr^{-3})^{\frac{2r-s}{2\lceil (s-r+1)/2 \rceil} - o(r)}.$$

Note that for any $s \leq r + r^{1-\varepsilon}$, where $\varepsilon > 0$ is arbitrarily small constant, the depth is $(nr^{-3})^{r^{\Omega(1)}}$.

Chapter 6

General Linear Degeneracy Tests

In this chapter we show that rSUM testing is not harder than fDegeneracy testing. We define the following linear degeneracy problems.

rSUM': Given a point $x \in \mathbf{R}^{r \times m}$, do there exist indices $i_1, \ldots, i_r \in [m]$ s.t.

$$\sum_{k=1}^{r} x_{ki_k} = 0 \ ?$$

Let f be a fixed r-variate linear polynomial of the form $f = \alpha_1 t_1 + \cdots + \alpha_r t_r - \alpha_0$, where $\alpha_k \neq 0$ for $k = 1, \ldots, r$.

fDegeneracy: Given a point $x \in \mathbb{R}^n$, do there exist r distinct indices $i_1, \ldots, i_r$ s.t.

$$f(x_{i_1}, \ldots, x_{i_r}) = 0 \ ?$$

fDegeneracy': Given a point $x \in \mathbb{R}^{r \times m}$, do there exist indices $i_1, \ldots, i_r \in [m]$ s.t.

$$f(x_{1i_1}, \ldots, x_{ri_r}) = 0 \ ?$$

Observation 6.1. *Theorems 3.5, 4.2 and 5.2 apply to* rSUM'*, because the critical hyperplanes have a single one in each row.*

Claim 6.2. *The point $x = (x_{ij}) \in \mathbb{R}^{r \times m}$ is in* rSUM' *if and only if the point $y = (y_{ij}) \in R^{r \times m}$*

with

$$y_{ij} = x_{ij}/\alpha_i + \alpha_0/(\alpha_i r)$$

is in fDEGENERACY*'.*

We conclude that an LDT $\mathcal{T}$ deciding fDEGENERACY' can be transformed into an LDT deciding rSUM' while preserving the structure of the tree (its height in particular) and the number of nonzero coefficients used in the normals to the query hyperplanes attached to each internal node (and vice versa). This reduction shows that fDEGENERACY' and rSUM' have the exact same complexity, and hence, Theorems 3.5, 4.2 and 5.2 apply to fDEGENERACY' as well.

Finally, we would like to show that fDEGENERACY (for dimension $n = rm$) is at least as hard as fDEGENERACY' (for dimension $r \times m$). This is a bit more tricky. The equivalence between fDEGENERACY' and rSUM' tells us, in fact, that any sLDT deciding fDEGENERACY' for dimension $r \times m$ contains a high degeneracy-complexity chamber C, where the complexity (depending on s) is as stated in Theorems 3.5, 4.2 and 5.2. However, this does not mean that C is also a high degeneracy-complexity chamber for fDEGENERACY. The problem is that p_0 (the representative of the chamber C of the arrangement $\mathcal{A}$) must be nondegenerate for the proof to work. This was implicitly taken care of in the rSUM case; Indeed by walking in a positive direction $\psi \in \mathbb{R}^{r\times m}$ from c_0 we escaped all canonical hyperplanes, including *non conforming* canonical hyperplanes, namely, those that have multiple 1's in a single row and no 1's in others. For general fDEGENERACY this may not work, but a small perturbation of ψ will do the trick. Indeed, this nonconstructive step will almost surely escape all non-conforming canonical hyperplanes, namely, hyperplanes not of the form $\{\sum_{i=1}^{r} \alpha_i x_{ij_i} - \alpha_0 = 0\}$. (This step should be compared to Section 3.3 of [46]).

Chapter 7

Concluding Remarks

Open Problems

The techniques used in this paper break down when $r = n^{1/3}$ or when $s = \Omega(r)$. Unfortunately, even the cases $r < 6, s > r$ give trivial lower bounds, which are subsumed by the general $\Omega(n \log n)$. These cases (especially $r = 3, 4$) are important, and we hope that the new techniques introduced here will return the focus to them. Also, it would be interesting to know if the lower bounds we get for $s > r$ (and especially for $s > r + 1$) are tight for big r. This question is related to the more general problem of combinatorial design: given a linear vector space, find the largest possible collection of vectors of a given type such that no vector in their linear span violates some combinatorial property. Error correcting codes are a special case of this problem, where the combinatorial property is a bound on the number of zero coordinates in nontrivial vectors. The tensor product of permutation matrices is another case, where the combinatorial property is related to the number of nonzeros in each row.

Part II

Aggregating Inconsistent Information: Ranking and Clustering

Chapter 1

Introduction

In this part we deal with the problem of aggregating information originating from different sources. This problem lies in the intersection of economics and game theory, social choice theory, combinatorics and optimization. We start with a few examples.

1. A conference program committee selects 70 participating papers out of 250 submissions, based on a score in the range of $1-5$ given by the program committee members for each submission.

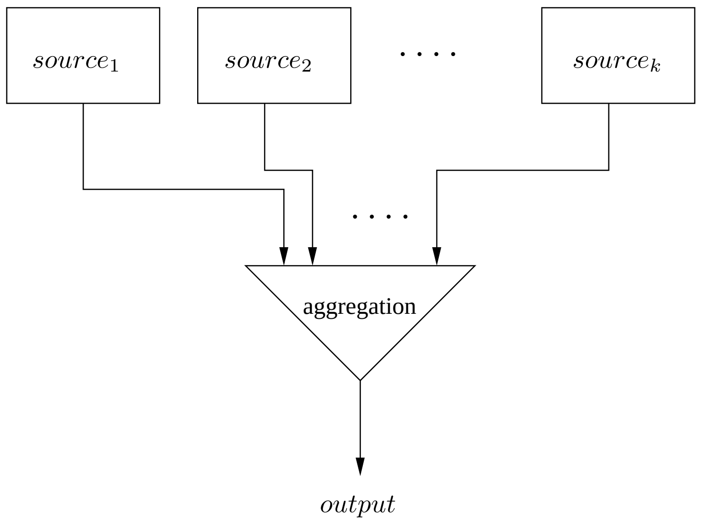
2. A stock market portfolio is built based on information gathered from different financial newspapers columnists.

3. A meta-search engine takes a search query as input and returns a cocktail of information consisting of Google search output (30%), Yahoo search output (60%) and Altavista search output (10%).

4. Several heuristics are used for retrieving relevance-ranked records from a database, and one output ranking is to be formed by combining their corresponding outputs.

5. Several heuristics are used for clustering microarray data, and one output clustering is to be formed by combining their corresponding outputs.

We categorize the above examples based on two criteria. The first criterion is *structure.* By structure we roughly mean that the different sources of information are objects in a simple logical



Though each source of information is assumed to be consistent with itself (adheres to structural constraints), inconsistencies may be present among different sources.

Figure 1.1: Aggregating inconsistent information

space (e.g space of permutations, space of k-dimensional unit vectors, space of graphs). Examples 1,3,4 and 5 deal with combining highly structured information, whereas example 2 deals with combining unstructured information.

The second criterion is *truthfulness*. In a nontruthful setting, the sources of information are selfish agents, and they could use knowledge of the aggregating algorithm as well as speculations on other agents' input to maximize a utility function by possibly lying about their preferences. Examples 1,2 and 3 may be nontruthful if the aggregation algorithm is not designed carefully. In examples 4 and 5 we assume that the different sources of information are outputs of different heuristics designed and implemented by the same agent, and therefore truthfulness is not an issue.

In this work we consider structured information, and we do not consider truthfulness issues. Specifically, we address the combinatorial optimization problems of RankAggregation and ConsensusClustering. In RankAggregation we combine *complete* rankings of a ground set of elements. This is motivated by examples 3,4 (and possibly 1) above. Note that in applications we may be interested in aggregating top-k lists, namely, ranked subsets of the ground set. Top-k rankings are a special case of the more general ranking with *ties*. We comment on these generalizations in Chapter 7. In ConsensusClustering we combine clusterings of a ground set of elements. This is motivated by example 5.

Chapter 2

Rank Aggregation

2.1 Definition of problem

The problem of ranking a set of contestants or a set of alternatives based on possibly conflicting preferences is a central problem in the areas of voting and social choice theory. Specifically, combining k different complete ranked lists (voters) on the same set of n elements (candidates) into a single ranking, which best describes the preferences expressed in the given k lists, is known as the problem of RankAggregation. This problem dates back to as early as the late 18th century when Condorcet and Borda each proposed voting systems for elections with more than two candidates [23, 34]. Both systems required each voter to supply a complete ranked list of the candidates and used aggregate information from these rankings to determine a winner. There are numerous applications in sports, databases, and statistics [43, 51] in which it is necessary to effectively combine rankings from different sources. Dwork, Kumar, Naor and Sivakumar proposed applying rank aggregation to the problem of combining rankings from different web search engines as a method of combating spam [43]. Fagin, Kumar and Sivakumar proposed applying rank aggregation to the nearest neighbor problem in high dimensions [51].

In the last half century, rank aggregation has been studied and defined from a mathematical perspective. In particular, Kemeny proposed a precise criterion for determining the "best" aggregate ranking[1] [78, 79]. The input is a set V of n candidates and k permutations of the candidates,

[1]Historically known as *Kemeny aggregation*.

$\{\pi_1, \pi_2, \ldots, \pi_k\} \subseteq S_V$. By convention, an element $\pi \in S_V$ is a bijection from V to $\{1, \ldots, |V|\}$, where $\pi(v)$ is the *rank* of element $v \in V$. The lower $\pi(v)$ is, the higher its "preference". For $u, v \in V$, we write $u <_\pi v$ to say that u has higher preference than v: $\pi(u) < \pi(v)$. The problem is, how do we output one ranking combining as much information as possible from $\pi_1, \ldots, \pi_k$?

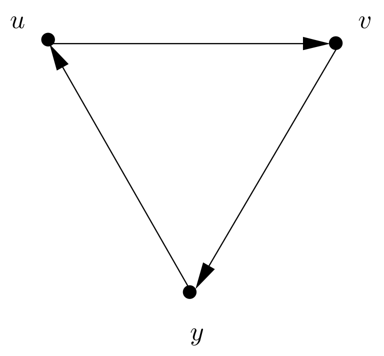
2.2 Majority rules: Condorcet winners and the Condorcet paradox

Consider the tournament[2]

$$G_{maj} = (V, A_{maj}), \tag{2.1}$$

defined as follows: $(u, v) \in A_{maj}$ if the number of integers $i \in [k]$ for which $u <_{\pi_i} v$ is greater than the number of those for which $v <_{\pi_i} u$. That is, u is "more popular" than v in the head-to-head contest induced by $\{\pi_1, \ldots, \pi_k\}$. Ties are broken arbitrarily.

A *Condorcet winner* is an element $v \in V$ that is a source in G_{maj}. A condorcet winner may not exist for a given input $\{\pi_1, \ldots, \pi_k\}$. This implies that the tournament G_{maj} may contain cycles (as demonstrated in the Condorcet paradox in Figure 2.1). Therefore, we cannot simply topologically sort G_{maj} to solve RankAggregation.



An election system among three voters over three candidates u, v, y is enough to illustrate the Condorcet paradox. If the votes are $u <_{\pi_1} v <_{\pi_1} y$, $v <_{\pi_2} y <_{\pi_2} u$ and $y <_{\pi_3} u <_{\pi_3} v$ then G_{maj} is as in the drawing.

Figure 2.1: The Condorcet paradox

[2]A tournament is a directed graph (V, A) such that for all $u, v \in V$, either $(u, v) \in A$ or $(v, u) \in A$.

2.3 The Kemeny approach

A *Kemeny optimal* ranking of the candidates is the ranking π that minimizes

$$\mathrm{cost}_{kem}(\pi) = \sum_{i}^{k} d(\pi, \pi_i) \ ,$$

where $d(\pi, \sigma)$ denotes[3] the number of pairs of candidates that are ranked in different orders by π and σ. For example, if $V = \{A, B, C, D\}$, $\pi = [A, B, C, D]$[4]and $\sigma = [B, C, A, D]$, then $d(\pi, \sigma) = 2$ since elements A and B appear in different orders in the two rankings as do elements A and C. In other words, a Kemeny optimal ranking minimizes the number of pairwise *disagreements* with the given k rankings. Note that $d(\cdot, \cdot)$ is a metric (in fact, an ℓ_1 metric). The general problem of minimizing $\sum_{y \in Y \subset X} d(x, y)$ over all possible $x \in X$ for a metric space d over X is called the median problem. Throughout this paper we will slightly abuse terminology and refer to the problem of finding a Kemeny-optimal ranking as RankAggregation.

There are many justifications to considering the Kemeny approach for RankAggregation [43]. It can be interpreted as a method for maximizing the likelihood of the votes assuming a very natural probabilistic voting model. It is *anonymous* and *neutral* (see [67] for definition of these social-choice theoretic terms). An important justification related to the previous discussion is as follows: If a Condorcet winner $v \in V$ does exist, then there exists a Kemeny-optimal ranking π with $\pi(v) = 1$. A voting system with this property is said to satisfy the *Condorcet criterion.* In fact, the Kemeny-optimal ranking satisfies the *generalized* Condorcet criterion: If there exists a subset $U \subseteq V$ such that for all $u \in U$ and $v \in V \backslash U$, $(u, v) \in A_{maj}$ then there exists a Kemeny-optimal ranking π with

$$\pi(U) = \{1, \ldots, |U|\} \ . \tag{2.2}$$

To see why this is true, notice that if there is a Kemeny-optimal solution π for which $\pi(u) = \pi(v)+1$ for some $u \in U$ and $v \in V \backslash U$, then by swapping the values of π at u and v we do not increase the cost, therefore, we remain with a Kemeny-optimal solution. Continuing until no such swap is possible, we obtain a Kemeny-optimal solution satisfying (2.2).

[3]The distance function $d(\cdot, \cdot)$ is known as the *Kendall-τ* distance.
[4]This is shorthand for $\pi(A) = 1, \pi(B) = 2, \pi(C) = 3, \pi(D) = 4$.

2.4 Minimum feedback arc-set in tournaments

As we will see in subsequent chapters, RankAggregation is closely related to the graph-theoretical problem of minimum feedback arc-set in tournaments (MinFAS-Tour). Given a directed graph $G = (V, A)$, MinFAS is the problem of finding the smallest set $A' \subseteq A$ such that $(V, A - A')$ is acyclic. The size of this set is exactly the minimal number of backward edges induced by a linear ordering of V. In MinFAS-Tour, G is a tournament. This problem turns out to be useful in studying RankAggregation, but is also interesting in its own right. For example, imagine a sports tournament where each player plays against every other player once: How should we rank the players based on these possibly nontransitive (inconsistent) outcomes?

The complementary problem to finding a minimum feedback arc set is the *maximum acyclic subgraph* problem, also known as the *linear ordering* problem.

It turns out that RankAggregation can be cast as a special case of *weighted* MinFAS. In weighted MinFAS, the input is $G = (V, w)$, where V is our ground set of vertices (candidates) and w is a nonnegative real function on ordered distinct pairs $(u, v) \in V \times V$ (we use w_{uv} as shorthand for $w(u, v)$). The goal is to find $\pi \in S_V$ minimizing

$$\text{cost}_G(\pi) = \sum_{u,v \in V:\ u <_\pi v} w_{vu} \ .$$

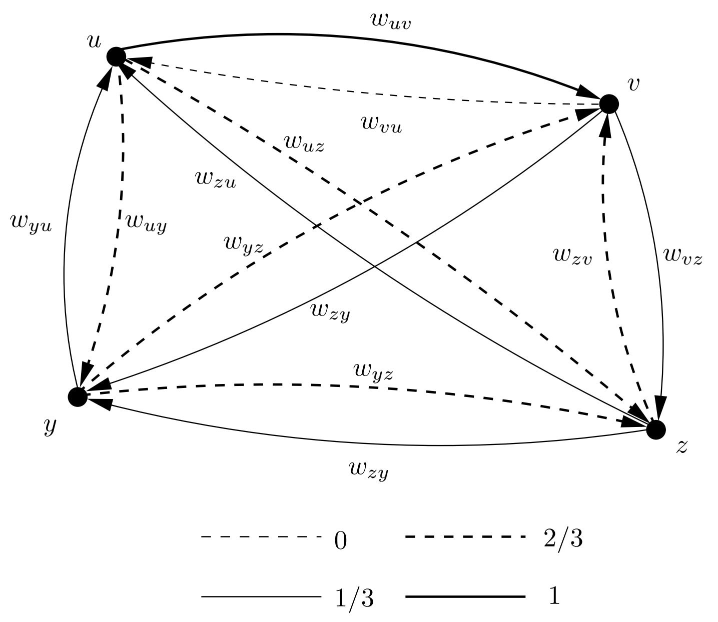
It is trivial to verify that if $w_{uv} \in \{0, 1\}$ for all distinct $u, v \in V$, then this is MinFAS (with possible anti-parallel edges). If, in addition, the weights satisfy $w_{uv} + w_{vu} = 1$ for all distinct $u, v \in V$, then this is exactly MinFAS-Tour. We generalize the notion of a tournament to arbitrary real-weighted graphs:

Definition 2.1. If $w_{uv} + w_{vu} = 1$ for all distinct $u, v \in V$ then w (or G) is said to satisfy the *tournament constraint.*

Given an input $\{\pi_1, \dots, \pi_k\} \subseteq S_V$ to RankAggregation, consider the following system w of weights (see example in Figure 2.2):

$$w_{uv} = \frac{1}{k} \left|\{i \in [k] : u <_{\pi_i} v\}\right| \ . \tag{2.3}$$

Clearly, $\text{cost}_G(\pi)$ is (up to a normalization factor of k) equivalent to $\text{cost}_{kem}(\pi)$. Henceforth, when



Weighted instance G induced by $V = \{u, v, y, z\}$, $k = 3$ voters and
$\pi_1 = [u, v, y, z]$, $\pi_2 = [u, y, z, v]$, $\pi_3 = [z, y, u, v]$.

Figure 2.2: From RankAggregation to weighted MinFAS

G is clear from the context, we will use cost(π) to denote $\text{cost}_G(\pi)$, our RankAggregation objective function.

Definition 2.2. If $w_{uy} \leq w_{uv} + w_{vy}$ for all distinct $u, v, y \in V$ then w (or G) is said to satisfy the *triangle inequality.*

Claim 2.3. *The weight system induced by* RankAggregation *in (2.3) satisfies both the tournament constraint and the triangle inequality.*

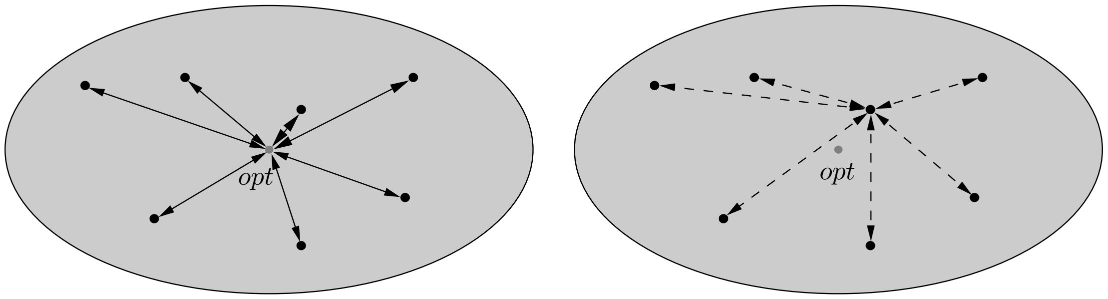
Proof. The tournament constraint is satisfied because all voters π_i either rank u before v (contributing to w_{uv}) or v before u (contributing to w_{vu}).

The triangle inequality follows from the simple observation that every voter that ranks u before y must either rank u before v or v before y. □

Note that not all weight systems w over V satisfying the tournament constraint and the triangle inequality are induced by RankAggregation. In fact, we will see that the extra structure in RankAggregation enables us to obtain better approximation guarantees that we could not argue for w not induced by RankAggregation (though satisfying both constraints).

2.5 Previous work

Recently, RankAggregation has been studied from a computational perspective. Finding a Kemeny optimal ranking is NP-hard [18] and remains NP-hard even when there are only four input permutations to aggregate [43]. This motivates the problem of finding a ranking that *approximately* minimizes the number of disagreements with the given input rankings. Several 2-approximation algorithms [37, 43] are known. In fact, one of the input rankings has cost no worse than twice that of the optimal solution. This is true for *any* median problem (Figure 2.3). A randomized version of this simple algorithm with the same (expected) 2-approximation factor which will be useful in our analysis is PickAPerm (Figure 3.2): uniformly at random output one of the input permutations.



A median problem with a set Y of 7 input points in some metric space X. The point *opt* $\in X$ is the optimal solution, minimizing the sum of distances to the points in Y. The point in Y closest to *opt* is a 2-approximation. Also, choosing a point in Y uniformly at random is an expected 2-approximation

Figure 2.3: Any median problem admits a simple 2-approximation algorithm.

Another 2-approximation algorithm is obtained as follows: instead of considering the Kemeny approach, consider the *Spearman footrule* distance d' over permutations on V, defined as follows:

$$d'(\pi, \sigma) = \sum_{v \in V} |\pi(v) - \sigma(v)| \ .$$

The Spearman footrule distance takes the actual position of the ranked items into account when comparing between π and σ. Solving median over d' can be done in polynomial time via a minimum cost matching reduction [43, 44], and it can be shown [37] that

$$d(\pi, \sigma) \leq d'(\pi, \sigma) \leq 2d(\pi, \sigma) \ .$$

This means that a Spearman footrule optimal ranking has a Kemeny cost no more than twice optimal.

MinFAS can be approximated to within $O(\log n \log \log n)$ [49, 100] and has (at least) the same approximation hardness as vertex cover [77], which is 1.36 [40]. More than a decade ago, Bang-Jensen and Thomassen conjectured that MinFAS-Tour is NP-hard [14]. However, for the past decade, no progress has been made on settling this conjecture. In contrast, the minimum feedback *vertex* set problem on tournaments is NP-hard [101] and is approximable to within 2.5 [26]

We are not aware of any approximation for MinFAS-Tour that improves on the bound for the more general MinFAS. The complementary maximization problem on tournaments seems to be easier from an approximation standpoint. Arora, Frieze and Kaplan [11] and Frieze and Kannan [57] gave PTASs for the maximum acyclic subgraph problem in dense graphs, which implies a PTAS for the problem on tournaments. Specifically, they show that if the maximum edge weight of a given digraph $G = (V, A)$ is bounded by 1 and the value of an optimal solution is OPT, then with probability at least $1 - \delta$, their algorithm finds a solution of value at least $OPT - \varepsilon n^2$ in time exponential in $1/\varepsilon^2$ and polynomial in $1/\delta$ and $n = |V|$. This implies a PTAS for MinFAS-Tour, because $OPT = \Omega(n^2)$ there.

2.6 New algorithms and results

We give improved approximation algorithms for RankAggregation and for special cases of weighted MinFAS.

We use variants of essentially the same simple algorithm. The most basic variant is KwikSort for instances of MinFAS-Tour (Figure 2.4). This is exactly the well-known textbook QuickSort invented by Hoare [66], except that we ignore the working assumption that the tournament G is acyclic.

First, we pick a random vertex u to be the “pivot” vertex. Second, we place all vertices

```
KwikSort($G = (V, A)$)
  A recursive algorithm for MinFAS-Tour

  if $V = \emptyset$ then return empty-list
  set $V_L \leftarrow \emptyset, V_R \leftarrow \emptyset$
  pick random pivot $u \in V$

  for all vertices $v \in V \setminus \{u\}$:
      if $(v, u) \in A$ then
          add $v$ to $V_L$  (place $v$ on left side)
      else (if $(u, v) \in A$)
          add $v$ to $V_R$  (place $v$ on right side)

  set $G_L \leftarrow$ sub-tournament induced by $V_L$
  set $G_R \leftarrow$ sub-tournament induced by $V_R$

  return order KwikSort($G_L$),$v$,KwikSort($G_R$)
    (concatenation of left recursion, $v$, and right recursion)
```

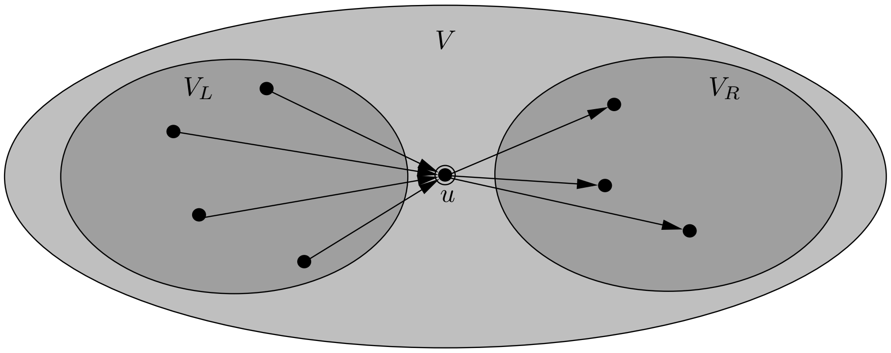


Figure 2.4: Pseudocode and diagram for KwikSort

connected to u with an in-edge on the left side of u and all vertices connected to u with an out-edge on the right side of u. We then recurse on the two tournaments induced by the vertices on each side. Our analysis of KwikSort yields a 3-approximation algorithm for MinFAS-Tour, improving on the best-known previous factor of $O(\log n \log \log n)$. We rely on a new technique for arguing a lower bound for the optimal solution by demonstrating a fractional packing of directed triangles using probabilities of certain events in the execution probability space.

To apply this algorithm to weighted MinFAS, we define a notion of a *majority* tournament $G_w = (V, A_w)$ corresponding to instance $G = (V, w)$.

Definition 2.4. Given an instance (V, w) of weighted MinFAS, the corresponding *majority* tournament $G_w = (V, A_w)$ is defined by setting $(u, v) \in A_w$ if $w_{uv} > w_{vu}$ for all distinct $u, v \in V$. If $w_{uv} = w_{vu}$, then we decide either $(u, v) \in A_w$ or $(v, u) \in A_w$ arbitrarily.

Note that although the majority tournament G_w depends on the weights of the weighted instance, it is an *unweighted* graph. Also note that if $G = (V, w)$ is induced by RankAggregation as in (2.3), then G_w is exactly G_{maj} defined in (2.1).

By outputting $\pi =$ KwikSort(G_w) we obtain an algorithm for weighted MinFAS, and in particular for RankAggregation. This algorithm is a 3-approximation whenever w satisfies the tournament constraint, and a 2-approximation if it satisfies the triangle inequality. Hence, this is yet another 2-approximation algorithm for RankAggregation. However, we show that by taking the better of KwikSort and PickAPerm we obtain an (11/7)-approximation for RankAggregation. This improved approximation ratio is due to the fact that each algorithm does relatively well on instances in which the other algorithm does relatively poorly.

Note that a simple lower bound on the value of an optimal solution for weighted MinFAS is to take the sum over all vertices $i < j$ of $\min\{w_{ij}, w_{ji}\}$. In contrast, our analysis uses a stronger lower bound based on the weight of directed triangles (Condorcet-paradox triangles) in the majority tournament. Interestingly, the analysis of our simple combinatorial algorithm bounds the integrality gap of a natural LP relaxation for MinFAS (with w satisfying the corresponding constraints).

	KwikSort	LP-KwikSort	Previous
MinFAS-Tour	3	5/2	$O(\log n \log\log n)$ [49, 100]
Weighted MinFAS w/ Tournament Constraint	5	5/2	$O(\log n \log\log n)$
Weighted MinFAS w/ Triangle Inequality	2	2 (w/ Tour. Const.)	$O(\log n \log\log n)$
RankAggregation (Better of PickAPerm & ...)	11/7	4/3	2

Table 2.1: Summary of results for ranking

Additionally, we consider the following LP relaxation [96] for weighted MinFAS:

$$\begin{aligned} \text{minimize} \sum_{u,v\in V:\ u\neq v} & x_{uv} w_{vu} \\ x_{uy} \leq x_{uv} + x_{vy} & \quad \forall u,v,y \\ x_{uv} + x_{vu} = 1 & \quad \forall u,v \\ x_{uv} \geq 0 & \quad \forall u,v \end{aligned} \tag{2.4}$$

The LP has a variable x_{uv} for all distinct $u, v \in V$. Clearly, if we imposed the constraint that $x_{uv} \in \{0, 1\}$ for all u, v, the corresponding IP would be an exact mathematical program for weighted MinFAS.

Assuming we have an optimal solution $\{x_{uv}\}$ to the LP, we run LP-KwikSort (V, x) (Figure 2.5) to round it. This is similar to KwikSort, except that after choosing a pivot vertex, instead of deterministically placing vertices on the right or left side, we randomly decide based on the value of the corresponding LP values. This results in vastly improved approximation factors. In particular, taking the best of LP-KwikSort and PickAPerm results in a (4/3)-approximation algorithm for RankAggregation. See Table 2.1 for a complete summary of the new approximation factors.

Finally, we show that MinFAS-Tour has no polynomial time algorithm assuming NP$\not\subseteq$BPP.

```
LP-KwikSort($V, x$)
  *A recursive algorithm for rounding the LP for
  weighted* MinFAS. *Given an LP solution
  $\{x_{uv}\}_{u,v \in V}$, returns an ordering on the
  vertices $V$.*

  if $V = \emptyset$ then return empty-list
  pick random pivot $u \in V$
  set $V_R = \emptyset, V_L = \emptyset$

  for all $v \in V \backslash \{u\}$:
    with probability $x_{vu}$
        add $v$ to $V_L$
    else (with probability $x_{uv} = 1 - x_{vu}$)
        add $v$ to $V_R$

  return ordering
    LP-KwikSort($V_L, x$), $u$, LP-KwikSort($V_R, x$)
```

Figure 2.5: Pseudocode for LP-KwikSort

The question of NP-hardness of MinFAS-Tour has been a long-standing conjecture of Bang-Jensen and Thomassen [14]. We show a randomized reduction from the problem of finding a minimum feedback arc set in general digraphs (which is known to be NP-hard) to the special case of tournaments. This reduction has been recently derandomized by Noga Alon [8], and the conjecture is therefore proven completely. We present the weaker randomized version here.

Chapter 3

Analysis of Ranking Algorithms

Before we prove the results for the ranking problems, we make a few notational remarks. In what follows we will use the term *directed triangle*. A directed triangle t in a directed graph $G = (V, A)$ is a set of three edges forming a directed cycle. For ease of notation, we will write $t = (u, v, y)$ to refer to the directed triangle over distinct vertices $u, v, y \in V$, namely,

$$(u, v, y) \stackrel{\text{def}}{=} \{(u, v), (v, y), (y, u)\} \subseteq A \ .$$

We may abuse notation and formally write $u \in t$ to say that u is a vertex incident to the directed triangle t, though we keep in mind that t is a set of three edges and not three vertices.

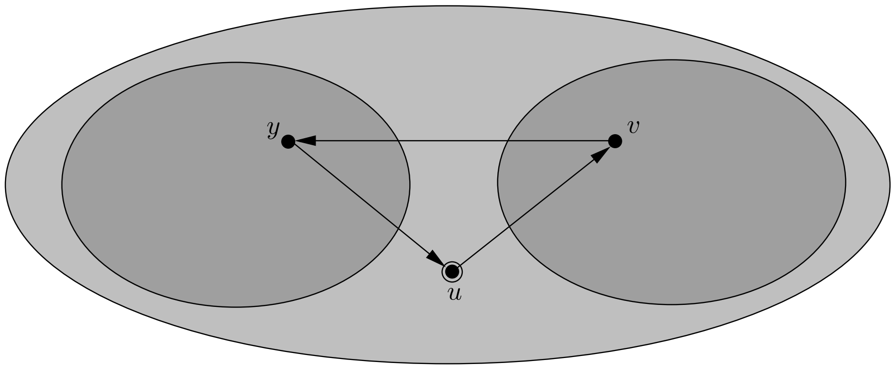
3.1 KwikSort for MinFAS-Tour

Let $G = (V, A)$ be a MinFAS-Tour instance. Consider algorithm KwikSort (Figure 2.4) for approximating it.

Theorem 3.1. *The output of* KwikSort(G) *is a random expected* 3*-approximation solution for* MinFAS-Tour*.*

Proof. Let C^{OPT} denote the cost of a fixed optimal solution. Let C^{KS} denote the (random) cost of KwikSort(G) on $G = (V, A)$. We want to show that $\mathbf{E}\left[C^{KS}\right] \leq 3C^{OPT}$.

An edge $(u, v) \in A$ becomes a backward edge if and only if there exists a third vertex y such



When a pivot is chosen among the vertices of a directed triangle $t = (u, v, y)$, the edge not incident to the pivot becomes backwards and charges the triangle a unit cost.

Figure 3.1: Charging directed triangles

that (u, v, y) form a directed triangle in G and y was chosen as the pivot when all three were input to the same recursive call (Figure 3.1). Pivoting on y would then place u on the right and v on the left, rendering edge (u, v) backward. In this case, we will charge a unit cost of the backward edge (u, v) to the directed triangle (u, v, y). We denote by T the set of directed triangles in G.

The execution of KwikSort is a recursion with a branching index of at most 2. Each node of the recursion corresponds to an input subset $I \subset V$ and a choice $u \in I$ of a pivot. Clearly, any vertex $u \in V$ is a pivot at exactly one recursion node. Let $I(u)$ denote the input subset at the recursion node at which u was chosen as pivot. In other words, u was chosen as pivot uniformly among the vertices of $I(u)$.

For any $t = (u, v, y) \in T$, denote by A_t the event

$$\{u, v, y\} \subseteq I(u) \ \vee \ \{u, v, y\} \subseteq I(v) \ \vee \ \{u, v, y\} \subseteq I(y) \ . \tag{3.1}$$

Clearly, only one of the three inclusions can occur. In words, A_t means that when one of the vertices of t is chosen as pivot, all its three vertices are input to the same recursive call. Let p_t denote the probability of event A_t. Now we observe that A_t occurs exactly when t is charged. Therefore, the expected cost of KwikSort(G) is exactly:

$$\mathbf{E}\left[C^{KS}\right] = \sum_{t \in T} p_t \ .$$

If we had a family of *disjoint* directed triangles (recall that by this we mean: *edge* disjoint), then a lower bound for C^{OPT} would be its cardinality, because any directed triangle incurs at least one backward edge in any solution. This is also true *fractionally*, as we shall see in Lemma 3.3.

Definition 3.2. A system of nonnegative weights on directed triangles $\{\beta_t\}_{t\in T}$ is a *packing* if for all $e \in A$,

$$\sum_{t:e\in t} \beta_t \leq 1 \ .$$

Lemma 3.3. *Any packing $\{\beta_t\}$ of the directed triangles is a lower bound for C^{OPT}.*

To prove the lemma, consider the following LP relaxation for MinFAS-Tour:

$$\begin{aligned} \text{minimize} \sum_{e\in A} x_e \text{ s.t.} \\ x_{e_1} + x_{e_2} + x_{e_3} \geq 1 \ \ \forall t = \{e_1, e_2, e_3\} \in T \\ x_e \geq 0 \ \ \forall e \in A \end{aligned} \tag{3.2}$$

The solution to this LP clearly lower bounds C^{OPT}. Indeed, if we interpret the variable x_e as an indicator variable to the event that e is a backward edge in some solution, then the inequality $x_{e_1} + x_{e_2} + x_{e_3} \geq 1$ encodes the fact that every directed triangle has a backward edge in any solution. It is easy to see that any packing is a feasible solution to the dual LP, and hence a lower bound for C^{OPT}.

We will demonstrate a packing using the probabilities p_t. Let $t = (u, v, y) \in T$ be a fixed directed triangle. Conditioned on the event A_t, each one of the three vertices of t was the pivot vertex with probability 1/3, because all vertices input to a recursive call are chosen as pivot with equal probability (in other words, each inclusion in (3.1) is equally likely given A_t). Therefore, any edge e of t becomes a backward edge with probability 1/3 given A_t. More formally, if we let B_e denote the event that e becomes a backward edge and $e \in t$, then

$$\mathbf{Pr}[B_e \wedge A_t] = \mathbf{Pr}[B_e|A_t]\mathbf{Pr}[A_t] = \frac{1}{3}p_t.$$

The event $B_e \wedge A_t$ means that the backwardness of edge e was charged to triangle t to which it is incident.

Observation 3.4. *For two distinct directed triangles $t, t' \in T$ such that $e \in t \cap t'$ for some edge e, the events $B_e \wedge A_t$ and $B_e \wedge A_{t'}$ are disjoint.*

Observation 3.4 is another way of saying that an edge e can be charged to only one triangle t incident to e. Therefore, for all $e \in E$,

$$\sum_{t:e\in t} \frac{1}{3} p_t \leq 1 \ . \tag{3.3}$$

So $\{p_t/3\}_{t\in T}$ is a fractional packing of T. By Lemma 3.3, $C^{OPT} \geq \sum_{t\in T} p_t/3 = \mathbf{E}[C^{KS}]/3$, as required. □

Note: As a side result, it follows that the integrality gap of LP (3.2) is at most 3.

3.2 KwikSort for weighted MinFAS

Let (V, w) be a weighted MinFAS instance, where $w \in (\mathbb{R}^+)^{n(n-1)}$. We suggest the following approximation algorithm: construct the unweighted majority tournament $G_w = (V, A_w)$ and return the ordering outputed by KwikSort(G_w). We analyze this algorithm.

For an edge $e = (u, v) \in A_w$, we let $w(e) = w_{uv}$, and $\bar{w}(e) = w_{vu}$ (by construction, $\bar{w}(e) \leq w(e)$). Fix an optimal solution π^*, and let $c^*(e)$ denote the cost incurred to it by $e = (u, v) \in A_w$, that is,

$$c^*(e) = \begin{cases} w(e) & \text{if } v <_{\pi^*} u \\ \bar{w}(e) & \text{otherwise} \end{cases} ,$$

so $C^{OPT} = \sum_{e\in A_w} c^*(e)$. Let T denote the set of directed triangles in G_w. For any $t = \{e_1, e_2, e_3\} \in T$, we define

$$c^*(t) = c^*(e_1) + c^*(e_2) + c^*(e_3)$$
$$w(t) = w(e_1) + w(e_2) + w(e_3) \ .$$

Finally, let C^{KS} denote the cost of the solution returned by KwikSort(V, G_w).

Theorem 3.5. *For an instance (V, w) of weighted* MinFAS*, if there exists a constant $\alpha > 0$ such* *that $w(t) \leq \alpha c^*(t)$ for all $t \in T$, then $\mathbf{E}[C^{KS}] \leq \alpha C^{OPT}$, i.e.* KwikSort$(G_w)$ *is an expected α-approximation solution.*

Proof. We generalize techniques presented in Section 3.1. When KwikSort is run on G_w, an edge $e \in A_w$ is *heavily charged* if it becomes a backward edge, and thus incurs the heavy cost $w(e)$. It is *lightly* charged if it incurs the light cost $\bar{w}(e)$.

Clearly, $e = (u, v) \in A_w$ is *heavily charged* if and only if a third vertex y is chosen as pivot when all three u, v, y are in the same recursive call, and (u, v, y) form a directed triangle in G_w. We charge this cost to triangle $t = (u, v, y)$.

Again we consider the set T of directed triangles in G_w, and their corresponding events A_t with probability p_t (see Section 3.1). Fix a triangle $t \in T$ with edges e_1, e_2, e_3. Conditioned on A_t, each of e_1, e_2 and e_3 are equally likely to be heavily charged, so the expected charge of t is $\frac{1}{3}p_t w(t)$.

The probability that an edge $e \in A_w$ does not incur a heavy cost (not charged to any triangle $t \in T$) is exactly

$$1 - \sum_{t:e\in t} \mathbf{Pr}[B_e \wedge A_t] = 1 - \sum_{t:e\in t} \frac{1}{3}p_t \ .$$

Therefore, $\mathbf{E}[C^{KS}] = B^{KS} + F^{KS}$, where

$$B^{KS} = \sum_{t\in T} \frac{1}{3} p_t w(t)$$
$$F^{KS} = \sum_{e\in A_w} \left(1 - \sum_{t:\ e\in t} \frac{1}{3}p_t\right) \bar{w}(e).$$

Note that the expression $(1 - \sum_{t:\ e\in t})$ for fixed $e \in A_w$ is nonnegative (see discussion in previous section). We rearrange the sum $C^{OPT} = \sum_{e\in T} c^*(e)$ as $C^{OPT} = B^{OPT} + F^{OPT}$, where

$$B^{OPT} = \sum_{t\in T} \frac{1}{3} p_t c^*(t)$$
$$F^{OPT} = \sum_{e\in A_w} \left(1 - \sum_{t:e\in t} \frac{1}{3}p_t\right) c^*(e) \ .$$

Obviously, $F^{KS} \leq F^{OPT}$, because $\bar{w}(e) \leq c^*(e)$ for any $e \in A_w$. Therefore, if for some $\alpha > 0$,

$w(t) \le \alpha c^*(t)$ for all t, then $\mathbf{E}[C^{KS}] \le \alpha C^{OPT}$ as required. ☐

Lemma 3.6. *If the weights satisfy the tournament constraint ($w_{uv} + w_{vu} = 1$), then $w(t) \le 5c^*(t)$ for all $t \in T$. If the weights satisfy the triangle inequality constraint ($w_{uv} \le w_{uy} + w_{yv}$), then $w(t) \le 2c^*(t)$.*

Proof. First assume probability constraints on the weights. In this case, we claim that $w(t) \le 5c^*(t)$. Indeed, in this case $w(e) \ge 1/2$ for all $e \in A_w$, and $\bar{w}(e) = 1 - w(e)$. Fix a triangle t containing edges e_1, e_2, e_3, and assume w.l.o.g.

$$1/2 \le w(e_1) \le w(e_2) \le w(e_3) \le 1\ . \tag{3.4}$$

Clearly, $w(t) = w(e_1) + w(e_2) + w(e_3) \le 2 + w(e_1)$. Any solution has to direct at least one of the edges in t backwards, therefore $c^*(t) \ge w(e_1)$. Since $w(e_1) \in [1/2, 1]$, we therefore have $w(t) \le 5c^*(t)$.

Consequently, KwikSort has an expected approximation ratio of at most 5 on weighted tournament instances with probability constraints on the weights.

Now we assume that the edge weights satisfy the triangle inequality. Fix $t \in T$ with edge weights $w(e_1), w(e_2), w(e_3)$. By the triangle inequality,

$$\begin{aligned} w(e_3) &\le \bar{w}(e_1) + \bar{w}(e_2) \\ w(e_1) &\le \bar{w}(e_2) + \bar{w}(e_3) \\ w(e_2) &\le \bar{w}(e_3) + \bar{w}(e_1) \end{aligned} \tag{3.5}$$

Summing up, we get $w(t) \le 2(\bar{w}(e_1) + \bar{w}(e_2) + \bar{w}(e_3))$. But $c^*(t) \ge \bar{w}(e_1) + \bar{w}(e_2) + \bar{w}(e_3)$, because the optimal solution must at least pay the lower cost at each edge. This concludes the proof.

☐

Note: In the conference version [3], a weaker bound of 3 was proven for the triangle inequality constraint only case and 2 for the combined constraints. This improvement in Lemma 3.6 is due to Warren Schudy.

Combining Theorem 3.5 and Lemma 3.6, we get:

Theorem 3.7. *Algorithm $\pi = $ KwikSort(G_w) is a 5-approximation (resp. 2-approximation) for weighted* MinFAS *with the tournament constraint (resp. with the triangle inequality).*

3.3 An improved approximation ratio for RankAggregation

Let $\{\pi_1, \ldots, \pi_k\}$ be a RankAggregation instance over some set of candidates V of size n. Consider the corresponding equivalent weighted MinFAS instance (V, w). Recall that for all $u \neq v$, w_{uv} is the fraction of input permutations ranking u before v.

As in Section 3.2, we consider the corresponding majority tournament $G_w(V, A_w)$. Theorem 3.7 shows that using KwikSort(G_w) we get a 2-approximation for this case, because both the tournament constraint and the triangle inequality are satisfied. We show in this section that the additional structure in these instances allows us to improve upon this factor. As usual, we let T denote the set of directed triangles in G_w, and we continue using the notation $w(e), \bar{w}(e), w(t), c^*(e), c^*(t)$ for $e \in A_w$ and $t \in T$ defined in Section 3.2.

As stated in Section 2.5, PickAPerm (Figure 3.2) is a 2-approximation algorithm for RankAggregation.

```
PickAPerm($\pi_1, \pi_2, \ldots \pi_k$)
  pick $i \in [k]$ uniformly at random
  return $\pi_i$
```

Figure 3.2: Pseudocode for PickAPerm

Let C^{PAP} denote the cost of PickAPerm on the RankAggregation instance. Let $G_w = (V, A_w)$ be the corresponding unweighted majority tournament. Define

$$z(e) \overset{\text{def}}{=} 2w(e)\bar{w}(e) .$$

We claim that

$$\mathbf{E}\left[C^{PAP}\right] = \sum_{e \in A_w} z(e) \ . \tag{3.6}$$

Indeed, an edge $e \in A_w$ becomes a backward (resp. forward) edge with probability $\bar{w}(e)$ (resp. $w(e)$), in which case it incurs the cost of $w(e)$ (resp. $\bar{w}(e)$). As usual, T is the set of directed triangles in G_w. For all $t = (e_1, e_2, e_3) \in T$, we let $z(t) = z(e_1) + z(e_2) + z(e_3)$. The following theorem shows how to analyze a "convex combination" of KwikSort and PickAPerm:

Theorem 3.8. *If there exist constants $\beta \in [0,1]$ and $\gamma > 0$ such that*

$$\beta w(t) + (1-\beta) z(t) \leq \gamma c^*(t) \ \forall t \in T, and$$
$$\beta \bar{w}(e) + (1-\beta) z(e) \leq \gamma c^*(e) \ \forall e \in A_w,$$

then the best of KwikSort(G_w) *and* PickAPerm$(\pi_1, \ldots, \pi_k)$ *is an expected γ-approximation for* RankAggregation.

Proof. We use the notation $C^{OPT}, F^{OPT}, B^{OPT}, F^{KS}, B^{KS}$ defined in Section 3.2.
We can now rearrange (3.6) as $\mathbf{E}\left[C^{PAP}\right] = B^{PAP} + F^{PAP}$, where

$$B^{PAP} = \sum_{t \in T} \frac{1}{3} p_t z(t) \qquad F^{PAP} = \sum_{e \in A_w} \left(1 - \sum_{t: e \in t} \frac{1}{3} p_t\right) z(e) \ . \tag{3.7}$$

Recall that the expression $(1 - \sum_{t:\ e \in t} \frac{1}{3} p_t)$ for fixed $e \in A_w$ is exactly the probability that an edge doesn't become backward in KwikSort, and hence a nonnegative number. If we now have β, γ as in the statement of the theorem, then

$$
\begin{aligned}
\beta\mathbf{E}\left[C^{KS}\right] + (1-\beta)\mathbf{E}\left[C^{PAP}\right] &= \beta B^{KS} + (1-\beta)B^{PAP} + \beta F^{KS} + (1-\beta)F^{PAP} \\
&= \sum_{t\in T}\frac{1}{3}p_t(\beta w(t) + (1-\beta)z(t)) \\
&\qquad + \sum_{e\in A_w}\left(1 - \sum_{t:e\in t}\frac{1}{3}p_t\right)(\beta\bar{w}(e) + (1-\beta)z(e)) \\
&\le \sum_{t\in T}\frac{1}{3}p_t\gamma c^*(t) + \sum_{e\in A_w}\left(1 - \sum_{t:e\in t}\frac{1}{3}p_t\right)\gamma c^*(e) \\
&= \gamma C^{OPT}\ ,
\end{aligned}
$$

where the first inequality follows from the assumptions of the theorem. We conclude that the $(\beta, 1-\beta)$-biased average of KwikSort(G_w) and PickAPerm($\pi_1, \ldots, \pi_k$) is at most an expected γ-approximation for RankAggregation. In particular, the best of the two is an expected γ-approximation. $\square$

Lemma 3.9. *For all $t \in T$,*

$$\frac{3}{7}w(t) + \frac{4}{7}z(t) \le \frac{11}{7}c^*(t)\ , \tag{3.8}$$

and for all $e \in A_w$,

$$\frac{3}{7}\bar{w}(e) + \frac{4}{7}z(e) \le \frac{11}{7}c^*(e)\ .$$

Proof. The second inequality in the lemma is obtained by verifying the simple fact that $\bar{w}(e) \le c^*(e)$ and $z(e) \le 2c^*(e)$ for all $e \in A_w$. To prove the first inequality, we can assume that

$$1/2 \le w(e_1) \le \min\{w(e_2), w(e_3)\} \le 1 \tag{3.9}$$

and that the optimal solution π^* flips only the lightest edge e_1, that is,

$$c^*(t) = w(e_1) + 1 - w(e_2) + 1 - w(e_3)\ .$$

Inequality (3.8) now becomes an inequality on $(w(e_1), w(e_2), w(e_3)) \in \mathbb{R}^3$ restricted to the polytope defined by (3.9) and the triangle inequality (3.5).

The proof can be completed by showing that the global maximum of

$$f(t) = \frac{3}{7}w(t) + \frac{4}{7}z(t) - \frac{11}{7}c^*(t)$$

is 0 on the defined polytope using standard techniques of multivariate calculus. $\square$

Note that for $(w(e_1), w(e_2), w(e_3)) = (1/2, 3/4, 3/4)$ we obtain $w(t) = 2$, $z(t) = 5/4$ and $c^*(t) = 1$, so (3.8) is tight. Theorem 3.10 follows from Theorem 3.8 and Lemma 3.9, using $\beta = 3/7$ and $\gamma = 11/7$:

Theorem 3.10. *The best of* KwikSort(G_w) *and* PickAPerm$(\pi_1, \ldots, \pi_k)$ *is an expected* (11/7)-*approximation for* RankAggregation.

In using Theorem 3.8 to derive bounds we can also take advantage of a priori knowledge of the system of weights w. We illustrate this using the special case of only $k = 3$ voters, a case of independent interest in the problem of reconstructing phylogenies from gene-order data [31, 89]:

Lemma 3.11. *If $k = 3$, then for all $t \in T$,*

$$\frac{2}{5}w(t) + \frac{3}{5}z(t) \leq \frac{6}{5}c^*(t)$$

and for all $e \in A_w$,

$$\frac{2}{5}\bar{w}(e) + \frac{3}{5}z(e) \leq \frac{6}{5}c^*(e) \ .$$

Proof. In this special case, we have that $w(e) \in \{2/3, 1\}$ for all $e \in A_w$, and $w(e_1) = w(e_2) = w(e_3) = 2/3$ for all $t = (e_1, e_2, e_3) \in T$, therefore $w(t) = 2, z(t) = 4/3$ and $c^*(t) \geq 4/3$. The inequalities can now be easily verified. $\square$

Theorem 3.12 follows from Theorem 3.8 and Lemma 3.11, using $\beta = 2/5$ and $\gamma = 6/5$:

Theorem 3.12. *The best of* KwikSort *on G_w and* PickAPerm *is an expected* (6/5)-*approximation for* RankAggregation *for $k = 3$ voters.*

3.4 LP-KwikSort for rounding the ranking LP

The techniques developed in the previous sections can be used for the analysis of LP-KwikSort (Figure 2.5) given a solution $x = \{x_{uv}\}_{u \neq v}$ to the ranking LP (2.4).

Theorem 3.13. LP-KwikSort(V, x) *returns a ranking with an expected cost of at most* 2.5 *(resp.* 2*) times the value of LP (2.4) on optimal solution x, when the weights satisfy the tournament constraint (resp. the tournament constraint and the triangle inequality). The best of* LP-KwikSort(V, x) *and* PickAPerm$(\pi_1, \ldots, \pi_k)$ *returns a ranking with an expected cost of at most* $4/3$ *times the value of the optimal solution x to LP (2.4), when the weights are induced by the* RankAggregation *instance* $\pi_1, \ldots, \pi_k$.

Note that these bounds are on the integrality gaps of the LP relaxation for the different cases. In particular, they imply the sought approximation guarantee (Table 2.1) with respect to the optimal solution. We remark that the integrality gap of LP (2.4) for MinFAS-Tour ($\{0, 1\}$-weights with the tournament constraint) can be lower bounded by $3/2$. This follows from the fact that for any tournament $G = (V, A)$ on n vertices, there is a feasible solution to the LP that has value at most $n/3$ (set $x_{uv} = 2/3$ for all $(u, v) \in A$) and there exist tournaments with no minimum feedback arc set of size smaller than $n(1/2 - \varepsilon)$, where ε is arbitrarily small (random tournaments or Payley tournaments).

Notational remark: Henceforth, we will use (u, v) to denote *unordered* pairs of vertices, that is, sets of two vertices. Additionally, we will use the term *triplet* to denote sets of three pairs incident to three vertices. We shall formally write $t = (u, v, y)$ as a shortcut for $t = \{(u, v), (v, y), (y, u)\}$. Abusing notation, we may also write $u \in t$ to say that u is incident to the triplet t, although t is a set of three pairs, and *not* three vertices.

Proof. We reduce the problem to proving global bounds on certain multinomials in high dimensional polytopes. These bounds will be stated in Lemmas 3.14 and 3.15 and proven in Section 3.5.

Let C_{LP}^{KS} denote the cost of the ordering returned by the rounding algorithm LP-KwikSort. We define the notion of pairs (u, v) that are charged *dangerously* and *safely*. The safe pairs are charged when one of their incident vertices is chosen as pivot, and the other vertex is in the same recursive call. The expected contribution of pairs that are charged safely in LP-KwikSort to the cost C_{LP}^{KS} is

$$c^*_{uv} \stackrel{\text{def}}{=} x_{uv}w_{vu} + x_{vu}w_{uv} \ . \tag{3.10}$$

This is closely related to the corresponding contribution to the LP (2.4) value (hence the term *safe*):

$$C_{LP} = \sum_{(u,v)} c^*_{uv} \ .$$

A pair (u, v) is charged dangerously when a third vertex y is chosen as pivot, all three u, v, y are in the same recursive call, and u, v are placed on opposite sides of y. The charge is w_{uv} (resp. w_{vu}) if v (resp. u) is placed on the left side of y and u (resp. v) on its right. In either case, we charge this cost to the triplet (u, v, y).

We let T denote the set of all triplets over V, and for any $t = (u, v, y) \in T$ we denote by A_t the event that all of u, v, y are in the same recursive call when one of them is chosen as pivot (similarly to our definition from Section 3.1). Let p_t denote the probability of A_t. Let B^t_{uv} denote the event that (u, v) is dangerously charged to triplet t, in that order (u is placed on the left, v on the right). Conditioned on A_t, it is clear that B^t_{uv} occurs with probability $\frac{1}{3}x_{uy}x_{yv}$. Indeed, y is chosen as pivot with probability 1/3, and conditioned on that, u is placed on the left with probability x_{uy} and v on the right with probability x_{yv}. More formally, for any $t = (u, v, y)$,

$$\mathbf{Pr}[A_t \wedge B^t_{uv}] = \mathbf{Pr}[A_t]\mathbf{Pr}[B^t_{uv}|A_t] = \frac{1}{3}p_t x_{uy}x_{yv} \ .$$

Denote

$$p^t_{uv} \stackrel{\text{def}}{=} \frac{1}{3}x_{uy}x_{yv} \ .$$

So the total expected charge on a triplet $t \in T$ is $p_t y_t$, where

$$y_t \stackrel{\text{def}}{=} \sum_{(u,v)\in t} p^t_{uv}w_{vu} + p^t_{vu}w_{uv} \ .$$

Now we notice that for any $t = (u, v, y)$ and $t' = (u, v, y')$ (two triplets sharing a pair (u, v)), the events $A_t \wedge (B^t_{uv} \vee B^t_{vu})$ and $A_{t'} \wedge (B^{t'}_{uv} \vee B^{t'}_{vu})$ are disjoint, because a pair (u, v) can be dangerously charged to exactly one triplet. Thus,

$$\sum_{t:(u,v)\in t} p_t(p^t_{uv} + p^t_{vu}) \leq 1 \ .$$

The LHS of the last expression is exactly the probability that the pair (u, v) is dangerously charged. Therefore, the total expected cost of LP-KwikSort is $\mathbf{E}\left[C^{KS}_{LP}\right] = B^{KS}_{LP} + F^{KS}_{LP}$, where

$$\begin{aligned} B^{KS}_{LP} &= \sum_t p_t y_t \\ F^{KS}_{LP} &= \sum_{(u,v)} \left(1 - \sum_{t:(u,v)\in t} p_t(p^t_{uv} + p^t_{vu})\right) c^*_{uv} \ . \end{aligned}$$

The following expression is a rearrangement of the sum $C_{LP} = \sum_{(u,v)} c^*_{uv}$: $C_{LP} = B_{LP} + F_{LP}$, where

$$\begin{aligned} B_{LP} &= \sum_t p_t \sum_{(u,v)\in t} (p^t_{uv} + p^t_{vu}) c^*_{uv} \\ F_{LP} &= \sum_{(u,v)} \left(1 - \sum_{t:(u,v)\in t} p_t(p^t_{uv} + p^t_{vu})\right) c^*_{uv} \ . \end{aligned}$$

So

$$F_{LP} = F^{KS}_{LP} \geq 0 \ . \tag{3.11}$$

We have the following lemma. We defer the proof to Section 3.5.

Lemma 3.14. *If the weight system satisfies the tournament constraints (resp. tournament constraint and triangle inequality), then for any $t \in T$,*

$$y_t \leq \tau \sum_{(u,v)\in t} (p^t_{uv} + p^t_{vu}) c^*_{uv},$$

where $\tau = 5/2$ (resp. $\tau = 2$).

As a consequence, in this case, $B^{KS}_{LP} \leq \tau B_{LP}$ and we conclude that (for the two cases in Lemma 3.14) $\mathbf{E}\left[C^{KS}_{LP}\right] \leq \tau C_{LP}$.

Although this just gives yet another 2-approximation algorithm for the rank aggregation problem, we can do better there. We couple LP-KwikSort with PickAPerm. The expected value of PickAPerm$(\pi_1, \ldots, \pi_k)$ is

$$\mathbf{E}\left[C^{PAP}\right] = \sum_{(u,v)} z_{uv},$$

where $z_{uv} = 2w_{uv}w_{vu} = 2w_{uv}(1 - w_{uv})$. We rearrange this sum as follows:

$$\mathbf{E}\left[C^{PAP}\right] = B_{LP}^{PAP} + F_{LP}^{PAP},$$

where

$$\begin{aligned} B_{LP}^{PAP} &= \sum_t p_t \sum_{(u,v)\subseteq t} (p_{uv}^t + p_{vu}^t) z_{uv} \\ F_{LP}^{PAP} &= \sum_{(u,v)} \left(1 - \sum_{t:(u,v)\in t} p_t(p_{uv}^t + p_{vu}^t)\right) z_{uv} \ . \end{aligned}$$

It is easy to see that

$$0 \leq F_{LP}^{PAP} \leq 2F_{LP} \ . \tag{3.12}$$

(This is because $z_{uv} \leq 2c_{uv}^*$, and $\sum_{t:u,v\in t} p_t(p_{uv}^t + p_{vu}^t)$ is a probability, hence ≤ 1.) We have the following lemma, (proof in Section 3.5):

Lemma 3.15. *For all $t \in T$,*

$$\frac{2}{3}y_t + \frac{1}{3}\sum_{(u,v)\in t}(p_{uv}^t + p_{vu}^t)z_{uv} \leq \frac{4}{3}\sum_{(u,v)\subseteq t}(p_{uv}^t + p_{vu}^t)c_{uv}^*.$$

As a consequence, $\frac{2}{3}B_{LP}^{KS} + \frac{1}{3}B_{LP}^{PAP} \leq \frac{4}{3}B_{LP}$.

By (3.11) and (3.12) we have $\frac{2}{3}F_{LP}^{KS} + \frac{1}{3}F_{LP}^{PAP} \leq \frac{4}{3}F_{LP}$, and combining, we conclude that

$$\frac{2}{3}\mathbf{E}\left[C_{LP}^{KS}\right] + \frac{1}{3}\mathbf{E}\left[C_{LP}^{PAP}\right] \leq \frac{4}{3}C_{LP} \ .$$

This means, in particular, that the best of LP-KwikSort(V, x) and PickAPerm$(\pi_1, \ldots, \pi_k)$ has an expected approximation ratio of at most $\frac{4}{3}$ with respect to the LP cost. This completes the proof of the theorem. □

3.5 Proof of ranking polyhedral inequalities

In this section we prove Lemmas 3.14 and 3.15. These Lemmas are equivalent to proving certain inequalities on polynomials in $\mathbb{R}^6$. We restate these inequalities for the sake of clarity, and slightly change notation to reduce indexing.

Lemmas 3.14 and 3.15 state inequalities on a single triplet t, which we fix as $(1, 2, 3)$. The interesting parameters corresponding to t are the weights w_{12}, w_{23}, w_{31} and the assignment x_{12}, x_{23}, x_{31} of the LP variables at our fixed optimal solution. Note that the tournament constraint is assumed on both x and w, and hence we can always substitute $w_{21} = 1 - w_{12}$, $w_{23} = 1 - w_{32}$ and $w_{13} = 1 - w_{31}$ to eliminate degrees of freedom (similarly for x).

For ease of notation, we let

$$\begin{array}{ccc} x_1 \overset{\text{def}}{=} x_{23} & x_2 \overset{\text{def}}{=} x_{31} & x_3 \overset{\text{def}}{=} x_{12} \\ w_1 \overset{\text{def}}{=} w_{23} & w_2 \overset{\text{def}}{=} w_{31} & w_3 \overset{\text{def}}{=} w_{12} \end{array}$$

We use $\mathbf{x} \in \mathbb{R}^3$ as shorthand for (x_1, x_2, x_3) and $\mathbf{w} \in \mathbb{R}^3$ as shorthand for (w_1, w_2, w_3). Let $\Pi \subseteq \mathbb{R}^3$ denote the *tournament constraint* polytope, that is,

$$\Pi \overset{\text{def}}{=} \{(a_1, a_2, a_3) \;:\; 0 \le a_i \le 1,\; i = 1, 2, 3\} \,. \tag{3.13}$$

Let $\Delta \subseteq \Pi$ denote the *triangle inequality and tournament constraint* polytope, that is

$$\Delta \overset{\text{def}}{=} \{(a_1, a_2, a_3) \in \Pi \;:\; 1 \le a_1 + a_2 + a_3 \le 2\} \,.$$

(It is easy to verify that this definition equivalent to the triangle inequality when the tournament constraint is also assumed).

We define three functions, *ks* (corresponding to LP-KwikSort), *pap* (corresponding to PickAPerm), and *lp* (corresponding the value of LP (2.4) on the optimal solution). The three functions are real-valued on $\mathbb{R}^6$, and defined as follows:

$$\begin{aligned}
ks(\mathbf{x},\mathbf{w}) &= x_1x_2w_3 + (1-x_1)(1-x_2)(1-w_3)\\
&\quad + x_2x_3w_1 + (1-x_2)(1-x_3)(1-w_1)\\
&\quad + x_3x_1w_2 + (1-x_3)(1-x_1)(1-w_2)
\end{aligned}$$

$$\begin{aligned}
pap(\mathbf{x},\mathbf{w}) &= (x_1x_2 + (1-x_1)(1-x_2))2w_3(1-w_3)\\
&\quad + (x_2x_3 + (1-x_2)(1-x_3))2w_1(1-w_1)\\
&\quad + (x_3x_1 + (1-x_3)(1-x_1))2w_2(1-w_2)
\end{aligned} \tag{3.14}$$

$$\begin{aligned}
lp(\mathbf{x},\mathbf{w}) &= (x_1x_2 + (1-x_1)(1-x_2))(x_3(1-w_3) + (1-x_3)w_3)\\
&\quad + (x_2x_3 + (1-x_2)(1-x_3))(x_1(1-w_1) + (1-x_1)w_1)\\
&\quad + (x_3x_1 + (1-x_3)(1-x_1))(x_2(1-w_2) + (1-x_2)w_2)
\end{aligned}$$

Lemma 3.14 is equivalent to showing that $f \overset{\text{def}}{=} ks - \frac{5}{2}lp \leq 0$ for all $(\mathbf{x},\mathbf{w}) \in \Delta \times \Pi$ and $g \overset{\text{def}}{=} ks - 2lp \leq 0$ for all $(\mathbf{x},\mathbf{w}) \in \Delta \times \Delta$. We make two simplification steps.

1. *Linearity in* $\mathbf{w}$: The functions f and g are linear in $\mathbf{w}$ (for $\mathbf{x}$ fixed). Therefore f obtains its maximum on $(\mathbf{x},\mathbf{w})$ for $\mathbf{w}$ which is some vertex of Π, and similarly g obtains its maximum value on $(\mathbf{x},\mathbf{w})$ for $\mathbf{w}$ which is some vertex of Δ. For f it suffices to check $\mathbf{w} = (0,0,0)$ and $\mathbf{w} = (0,0,1)$ (due to symmetry), and for g it suffices to check $\mathbf{w} = (0,0,1)$. Let $\tilde{f}(\mathbf{x}) \overset{\text{def}}{=} f(\mathbf{x},0,0,0)$, $\hat{f}(\mathbf{x}) \overset{\text{def}}{=} f(\mathbf{x},0,0,1)$ and $\tilde{g}(\mathbf{x}) \overset{\text{def}}{=} g(\mathbf{x},0,0,1)$. It remains to show that $\tilde{f}, \hat{f}, \tilde{g} : \mathbb{R}^3 \to \mathbb{R}$ are bounded above by 0 on Δ.

2. *Trilinearity in* $\mathbf{x}$*:*For $i = 1,2,3$ the functions $\tilde{f}, \hat{f}$ and $\tilde{g}$ are linear in x_i when x_j's are fixed for $j \in \{1,2,3\} \setminus \{i\}$. This means that any point $\mathbf{x} \in \Delta$ such that $\mathbf{x} + t\mathbf{e}_i \in \Delta$ for all $t \in [-\varepsilon, \varepsilon]$ for some $\varepsilon > 0$ and some $i \in \{1,2,3\}$ (where $\mathbf{e}_i$ is a standard basis element of $\mathbb{R}^3$) is *not* a strict local maximum of $\tilde{f}, \hat{f}$ and $\tilde{g}$ in Δ, so these points $\mathbf{x}$ can be ignored. The points that are left are $\mathbf{x} \in \Delta$ s.t. that $x_1 + x_2 + x_3 = 1$ or $x_1 + x_2 + x_3 = 2$.

Let $H_k \subseteq \mathbb{R}^3$ denote the hyperplane $x_1 + x_2 + x_3 = k$ for $k = 1,2$, and let $\Delta_k = \Delta \cap H_k$. The closed polytopes Δ_k are two dimensional and the polynomials $\tilde{f}, \hat{f}$ and $\tilde{g}$ are of total degree 3 and maximal degree 2 in each variable. Finding their maxima on Δ_1, Δ_2 is technical and slightly tedious and we omit the details. The interested reader may find them in [4].

Lemma 3.15 is equivalent to proving that $h \overset{\text{def}}{=} 2ks/3 + pap/3 - 4lp/3 \leq 0$ for all $(\mathbf{x}, \mathbf{w}) \in \Delta \times \Delta$. The trilinearity in $\mathbf{x}$ still holds true for h, so as before we can assume that either $\mathbf{x} \in \Delta_1$ or $\mathbf{x} \in \Delta_2$. We can assume w.l.o.g. (by symmetry) that $x \in \Delta_2$, that is, $x_1 + x_2 + x_3 = 2$. When $\mathbf{x}$ is fixed, then h is a (possibly degenerate) concave paraboloid in $\mathbf{w}$. In case of nondegeneracy, its unique global maximum is obtained when $\nabla_w h = 0$, which can be easily verified to be solved by $\mathbf{w} = \mathbf{w}^* \overset{\text{def}}{=} (w_1^*, w_2^*, w_3^*)$ defined by

$$\begin{aligned} w_1^* &\overset{\text{def}}{=} \frac{x_2 x_3}{x_2 x_3 + (1 - x_2)(1 - x_3)} + 2x_1 - 1 \\ w_2^* &\overset{\text{def}}{=} \frac{x_3 x_1}{x_3 x_1 + (1 - x_3)(1 - x_1)} + 2x_2 - 1 \\ w_3^* &\overset{\text{def}}{=} \frac{x_1 x_2}{x_1 x_2 + (1 - x_1)(1 - x_2)} + 2x_3 - 1 \end{aligned} \tag{3.15}$$

(the paraboloid in $\mathbf{w}$ is degenerate if and only if any of the denominators in (3.15) are 0, equivalently $x_i = 0$ and x_j=1 for some i, j. But this implies that after possibly permuting coordinates, $\mathbf{x} = (1, 1, 0)$. But $h(1, 1, 0, \mathbf{w}) = -2w_3^2/3 \leq 0$, proving the desired assertion trivially). Since we are assuming $x_1 + x_2 + x_3 = 2$, we have that for any $1 \leq i < j \leq 3$ $x_i + x_j \geq 1$, equivalently $x_i x_j \geq (1 - x_i)(1 - x_j)$. Therefore (3.15) implies $w_i^* \geq \frac{1}{2} + 2x_i - 1$ for $i = 1, 2, 3$. Summing up, we obtain $w_1^* + w_2^* + w_3^* \geq -\frac{3}{2} + 2(x_1 + x_2 + x_3) = \frac{5}{2} > 2$. In particular, (3.15) implies that $\mathbf{w}^* \notin \Delta$, and moreover, w^* and Δ are strictly on opposite sides of H_2. Now let $\mathbf{w}' \overset{\text{def}}{=} (w_1', w_2', w_3')$ be any point in Δ. Consider the straight line ℓ passing through $\mathbf{w}'$ and $\mathbf{w}^*$, and let $\mathbf{w}''$ the intersection of this line with H_2 (Figure 3.3). Restricted to ℓ (and for our fixed $\mathbf{x} \in \Delta_2$) h is a concave parabola attaining its maximum on $\mathbf{w}^*$. Therefore $h(\mathbf{x}, \mathbf{w}'') \geq h(\mathbf{x}, w')$, and we can assume in what follows that $\mathbf{w} = \mathbf{w}'' \in H_2$ (we cannot assume that $\mathbf{w}''$ is in Δ, though we won't be needing this assumption henceforth). We change variables and let $\tilde{h} : \mathbb{R}^4 \to \mathbb{R}$ be defined by $\tilde{h}(x_1, x_2, w_1, w_2) \overset{\text{def}}{=} h(x_1, x_2, 2 - x_1 - x_2, w_1, w_2, 2 - w_1 - w_2)$. We reduced the problem to proving that $\tilde{h} \leq 0$ on

$$\{x_1, x_2 : \ x_1 \leq 1, \ x_2 \leq 1, \ x_1 + x_2 \geq 1\} \times \mathbb{R}^2 \ .$$

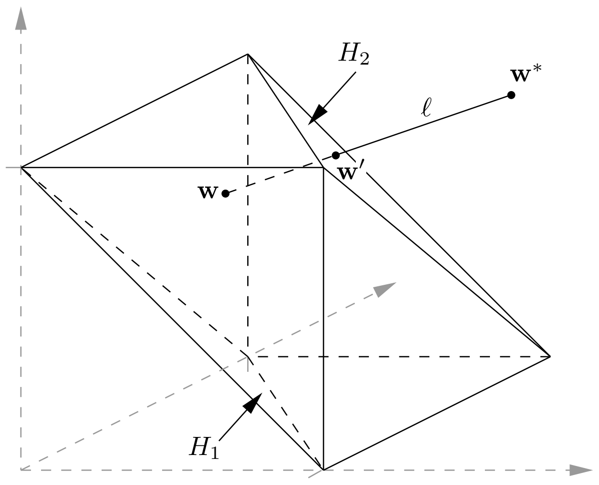


Figure 3.3: The ranking Δ-polytope

It is elementary to verify, using vanishing derivatives, that for (x_1, x_2) fixed, $\tilde{h}$ is a concave paraboloid in w_1, w_2 attaining its maximum at $(w_1, w_2) = (x_1, x_2)$. Substituting, we get

$$\tilde{h}(x_1, x_2, x_1, x_2) = -2(-1 + x_1)(-1 + x_2)(-1 + x_2 + x_3)$$

which is nonpositive because $x_1 + x_2 \geq 1$ and $x_1, x_2 \leq 1$.

$\square$

Chapter 4

NP-Hardness of Feedback Arc Set on Tournaments

All the problems referred to in Table 2.1 in Section 2.6 were previously known to be NP-hard except for MinFAS-Tour. In this section we show:

Theorem 4.1. *Unless NP⊆BPP,* MinFAS-Tour *has no polynomial time algorithm.*

Proof. We reduce to MinFAS-Tour from MinFAS, the problem of finding a minimum feedback arc set in a general directed graph (not necessarily a tournament). MinFAS is NP-hard [77] (in fact, it is MAX-SNP-hard, see [64, 92, 93]).

Let $G = (V, A)$ (with $|V| = n$) be an instance of MinFAS. Suppose we could add a set of edges A_R to G such that $(V, A \cup A_R)$ is a tournament, and such that exactly half of A_R are backward in any ordering π of V. Then by solving FasTour we would be able to recover the feedback arc set of G. This is generally impossible. However, if we add the edges A_R randomly (i.e. for every u, v such the neither (u, v) nor (v, u) are in A add (u, v) or (v, u) to A_R with equal probability) then for any π the expected number of backward edges is half $|R|$. The variance makes this approach fail. By blowing up G and using a concentration property of the random variable counting the number of backward edges in A_R, we can use this construction (see similar random digraph constructions in [92, 93]).

We pick an integer $k = \text{poly}(n)$ (chosen later). The blow-up digraph $G^k = (V^k, A^k)$ is defined

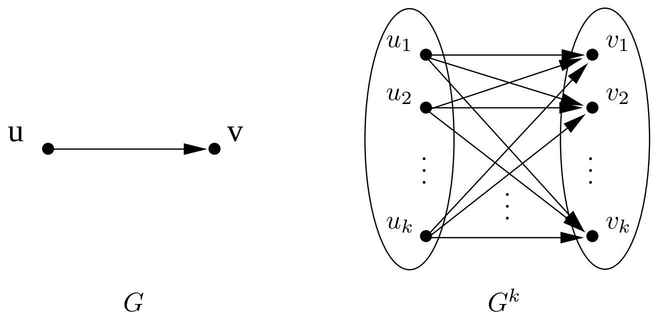


Figure 4.1: Blowing up G by factor k

as follows:

$$V^k = \bigcup_{v \in V} \{v_1, \ldots, v_k\}$$

$$A^k = \{(u_i, v_j) : \ (u, v) \in A, \ i, j \in [k]\} \ .$$

We observe that the minimum feedback arc set of G^k is exactly k^2 times the minimum feedback arc set of G. Indeed, it suffices to consider only rankings π on V^k that rank the vertices $v_1, \ldots, v_k$ as one block for all $v \in V$ (as explained in [8], if $v_i <_\pi v_j$ are not adjacent in the ranking, then either moving v_i immediately to the left of v_j or moving v_j immediately to the right of v_i will result in a ranking inducing no more feedback edges than π).

Now we turn G^k into a tournament $T^k = (V^k, A^k \cup A_R^k)$ using the construction defined above: For each non-edge $\{u, v\}$ of G^k, we randomly throw (u, v) or (v, u) into A_R^k with equal probability, completing a tournament.

For a ranking π of V^k, let $f_R(\pi)$ denote the number of feedback edges in A_R^k with respect to π. Denote by μ the expected value of $f_R(\pi)$, which is the same for all π, and can be efficiently computed. We claim that for $k = \text{poly}(n)$, with probability at least $2/3$, all rankings π satisfy $|f_R(\pi) - \mu| = O((nk)^{3/2}\sqrt{\log(nk)})$. This would imply, using the above observation, that for big enough $k = \text{poly}(n)$ the size of the minimum feedback arc set of T^k can be used to efficiently recover the size of the minimum feedback arc set of G, because $(nk)^{3/2}\sqrt{\log(nk)} = o(k^2)$.

To prove the claim, for any fixed ranking π, set a random indicator variable X_{wz}^π for every non-edge $\{w, z\}$ of G^k which equals 1 iff the edge between w and z in A_R^k is backward w.r.t. π. So $f_R(\pi) = \sum X_{wz}^\pi$. A simple application of Chernoff bounds [9] and union bound (over all possible

$(nk)!$ rankings) completes the proof of the claim.

It follows that unless FAS-DIGRAPH $\in BPP$, we cannot solve FASTOUR in polynomial time.

$\square$

We wish to thank Noga Alon for ideas significantly simplifying the proof [8]. Our initial hardness result was via max-SNP hardness of FAS-DIGRAPH, and Noga Alon pointed out that the same idea also works with the weaker NP-hardness.

Chapter 5

Consensus Clustering

5.1 Definition of problem

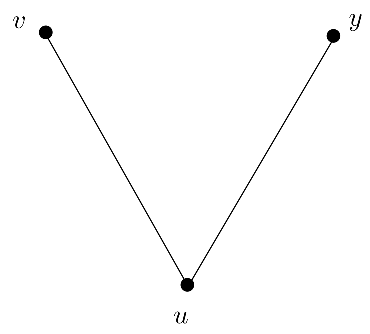
Another important case of aggregating information is the problem of integrating possibly contradictory clusterings from existing data sets into a single representative cluster. This problem is known as *consensus clustering* or *ensemble clustering* and can be applied to remove noise and incongruencies from data sets [54] or combine information from multiple classifiers [103].

The input is a set V of n elements and k clusterings $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$. A clustering $\mathcal{C}$ is an equivalence relation over V, that is, a binary relation that is *reflexive*, *symmetric* and *transitive*. We use the notation $u \equiv_{\mathcal{C}} v$ to denote that u and v are in the same class (or: u and v are *co-clustered*). We use the notation $u \not\equiv_{\mathcal{C}} v$ for the converse (in words: u and v are *separated*). The problem is, how do we output one clustering combining as much information as possible from $\mathcal{C}_1, \ldots, \mathcal{C}_k$?

5.2 Majority rules: A Condorcet equivalent

Similarly to the definition of the majority tournament in Chapter 2, we define the *pairwise majority graph*

$$G_{maj} = (V, E_{maj}) \tag{5.1}$$



Three vertices and voters are enough to illustrate the clustering-equivalent of the Condorcet paradox. If the votes are $u \equiv_{\mathcal{C}_1} v \equiv_{\mathcal{C}_1} y$, $[u \equiv_{\mathcal{C}_2} v] \not\equiv_{\mathcal{C}_2} y$, $[u \equiv_{\mathcal{C}_3} y] \not\equiv_{\mathcal{C}_3} v$, then G_{maj} is as in the drawing.

Figure 5.1: An equivalent to the Condorcet paradox

induced by $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ as follows[1]: $(u, v) \in E_{maj}$ if the number of integers i for which $u \equiv_{\mathcal{C}_i} v$ is greater than the number of integers i for which $u \not\equiv_{\mathcal{C}_i} v$. In words, the graph has an undirected edge for all pairs u, v for which the majority of voters co-cluster u and v.

We say that G_{maj} contains a paradox if there are three distinct vertices $u, v, y \in V$ such that (see Figure 5.1)

$$(u, v) \in E_{maj} \quad (u, y) \in E_{maj} \quad (v, y) \notin E_{maj} \ .$$

Indeed, if G_{maj} were paradox-free, then E_{maj} would be an equivalence relation on V, and would arguably be the best solution to the aggregation problem.

5.3 A Kemeny approach equivalent

As we did for the ranking problem, we consider the following median optimization problem: Let d denote a metric on clusterings measuring pairwise disagreements. More precisely,

$$d(\mathcal{C}, \mathcal{D}) \stackrel{\text{def}}{=} |\{(u, v) : (u \equiv_{\mathcal{C}} v \text{ and } u \not\equiv_{\mathcal{D}} v) \text{ or } (u \equiv_{\mathcal{D}} v \text{ and } u \not\equiv_{C} v)\}| \tag{5.2}$$

In words, $d(\mathcal{C}, \mathcal{D})$ measures the number of pairs (u, v) on which $\mathcal{C}$ and $\mathcal{D}$ disagree: one co-clusters them and the other separates between them. This measure of distance is also known as the

[1]In this chapter and the next, unless otherwise stated, (u, v) denotes an *unordered* pair.

Rand [97] distance. It is argued both theoretically and experimentally in [97] that it is a useful measure for evaluating quality of clustering algorithms.

For example, if $V = \{A, B, C, D\}$, $\mathcal{C} = \{\{A, B\}, \{C, D\}\}$ and[2] $\mathcal{D} = \{\{A, B, C\}, \{D\}\}$ then $d(\mathcal{C}, \mathcal{D}) = 3$ because $\mathcal{C}$ and $\mathcal{D}$ disagree exactly on the pairs $(C, D), (A, C)$ and (B, C). We let ConsensusClustering denote the median problem with respect to d, that is, given clusterings $\mathcal{C}_1, \ldots, \mathcal{C}_k$, find a clustering $\mathcal{C}$ minimizing

$$\mathrm{cost}_1(\mathcal{C}) = \sum_{i=1}^{k} d(\mathcal{C}, \mathcal{C}_k) \ . \tag{5.3}$$

Clearly, minimizing cost_1 on instances in which G_{maj} is paradox-free can be done by taking $u \equiv_{\mathcal{C}} v \iff (u, v) \in E_{maj}$.

5.4 Correlation clustering on complete graphs

Not surprisingly, ConsensusClustering is related to a well-known graph-theoretical problem called CorrelationClustering [15]. In CorrelationClustering, we are given an undirected graph $G = (V, E)$ with a label of either $(+)$ or $(-)$ on every edge. We denote by E^+ the set of $(+)$-edges and by E^- the set of $(-)$-edges, so $E = E^+ \cup E^-$ and $E^+ \cap E^- = \emptyset$. The goal is to find a clustering $\mathcal{C}$ of V minimizing (see Figure 5.2)

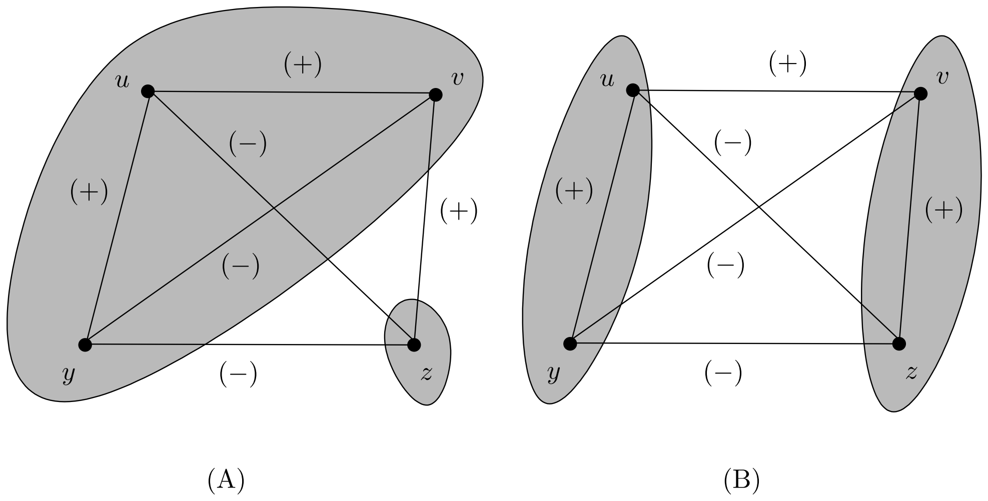
$$\begin{aligned} \mathrm{cost}_G(\mathcal{C}) = & |\{(u, v) : \ u \equiv_{\mathcal{C}} v \text{ and } (u, v) \in E^-\}| + \\ & |\{(u, v) : \ u \not\equiv_{\mathcal{C}} v \text{ and } (u, v) \in E^+\}| \ . \end{aligned}$$

In words, we are looking for the clustering minimizing the pairwise disagreement with G. In what follows, we will only deal with CorrelationClustering on *complete* graphs, namely,

$$E^+ \cup E^- = \binom{V}{2}$$

and we will use E as a shortcut for $\binom{V}{2}$.

[2]This is shorthand for $A \equiv_{\mathcal{C}} B$, $C \equiv_{\mathcal{C}} D$, $A \not\equiv_{\mathcal{C}} C$, $A \not\equiv_{\mathcal{C}} D$, $B \not\equiv_{\mathcal{C}} C$, $B \not\equiv_{\mathcal{C}} D$.



Two solutions to a CorrelationClustering instance: Solution (A) pays for $(y, v) \in E^-$ because y and v are clustered together and for $(v, z) \in E^+$ because v and z are separated. Solution (B) (optimal) pays only for $(u, v) \in E^+$ because u and v are separated.

Figure 5.2: CorrelationClustering

In *weighted* CorrelationClustering, we are given $G = (V, w^+, w^-)$, where w^+ and w^- are nonnegative weight functions on unordered pairs $(u, v) \in E$. We use w^+_{uv} and w^-_{uv} as shorthand for $w^+(u, v)$ and $w^-(u, v)$. The goal is to find a clustering $\mathcal{C}$ of V minimizing the following objective function:

$$\text{cost}_G(\mathcal{C}) = \sum_{u \equiv_{\mathcal{C}} v} w^-_{uv} + \sum_{u \not\equiv_{\mathcal{C}} v} w^+_{uv} \ .$$

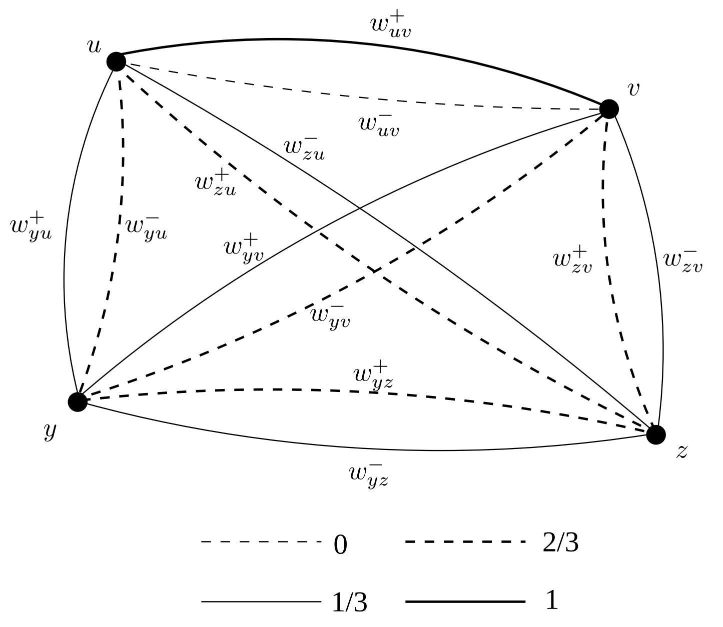
Clearly, if the weights are in $\{0, 1\}$ for all pairs (u, v) and also $w^+_{uv} + w^-_{uv} = 1$, then this is exactly CorrelationClustering on complete graphs.

Definition 5.1. If $w^+_{uv} + w^-_{uv} = 1$ for all pairs $(u, v) \in E$, then w is said to satisfy the *tournament constraint.*

(Note that we abuse the meaning of a tournament, usually referring to directed graphs.)

Given an input $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ to ConsensusClustering, consider the following system w of weights (see example in Figure 5.3):

$$\begin{aligned} w^+_{uv} &= \frac{1}{k} \left|\{i \in [k] : u \equiv_{\mathcal{C}_i} v\}\right| \\ w^-_{uv} &= \frac{1}{k} \left|\{i \in [k] : u \not\equiv_{\mathcal{C}_i} v\}\right| \end{aligned} \tag{5.4}$$



Weighted instance G induced by $V = \{u, v, y, z\}$, $k = 3$ voters and
$\mathcal{C}_1 = \{\{u, v, y, z\}\}$, $\mathcal{C}_2 = \{\{u, v\}, \{y, z\}\}$, $\mathcal{C}_3 = \{\{u, v, z\}, \{y\}\}$.

Figure 5.3: From ConsensusClustering to weighted CorrelationClustering

Clearly, for the corresponding weighted graph $G = (V, w^+, w^-)$, $\text{cost}_G(\pi)$ is (up to a normalization factor of k) equivalent to cost_1 from (5.3). Henceforth, when G is clear from the context, we will use $\text{cost}(\mathcal{C})$ to denote $\text{cost}_G(\mathcal{C})$, our ConsensusClustering objective function.

Definition 5.2. If $w^-_{uy} \le w^-_{uv} + w^-_{vy}$ for all distinct $u, v, y \in V$, then w is said to satisfy the *triangle inequality.*

Claim 5.3. *The weight system induced by* ConsensusClustering *in (5.4) satisfies both the tournament constraint and the triangle inequality.*

Proof. The tournament constraint is satisfied because all voters $\mathcal{C}_i$ either satisfy $u \equiv_{\mathcal{C}_i} v$ (contributing to w^+_{uv}) or $u \not\equiv_{C_i} v$ (contributing to w^-_{uv}). The triangle inequality follows from the simple observation that every voter that separates u from y (satisfies $u \not\equiv_{C_i} y$) either separates u from v or v from y. $\square$

5.5 Previous work

CorrelationClustering has been studied both on general and complete graphs. Both minimization and maximizing versions have been investigated. Bansal, Blum and Chawla gave the first constant factor approximation for the problem of minimizing disagreements on the complete graph [15]. Charikar, Guruswami and Wirth improved this approximation factor to 4 by rounding a linear program [30]. When the edge weights satisfy the tournament constraint, the best previous approximation factor was 7 [15,59]. When the edge weights satisfy the tournament constraint and the triangle inequality, the best previous approximation factor was 3 [59].

CorrelationClustering on complete graphs is MAX-SNP-hard [30] and ConsensusClustering is NP-hard [105], however, it is not known to be NP-hard if the number of input clusters is constant [54]. Analogously to RankAggregation (and just like any median problem), one of the voters' clusterings $\mathcal{C}_i$ is a 2-approximation. We will use a randomized version PickACluster (Figure 6.2) of this simple algorithm in our analysis, with the same (expected) approximation guarantee.

5.6 New algorithms and results

We give improved algorithms for ConsensusClustering and for certain weighted cases of CorrelationClustering. Just like in the RankAggregation case, we start with a very simple, almost greedy algorithm for approximating CorrelationClustering on complete graphs: KwikCluster (Figure 5.4).

The algorithm is simple: We choose a random pivot $u \in V$ uniformly at random, and form a cluster C consisting of u and all v connecting to u via a (+)-edge. Then we recurse on $V \backslash C$. We obtain a 3-approximation for CorrelationClustering on complete graphs, improving the best known 4 [30]. Similarly to our analysis in Chapter 3, will argue this bound by lower-bounding the optimal solution by packing paradox-triangles in G.

To apply KwikSort to weighted CorrelationClustering, we define the majority instance $G_w = (V, E_w^+, E_w^-)$ corresponding to $G = (V, w^+, w^-)$:

Definition 5.4. Given an instance (V, w^+, w^-) of weighted CorrelationClustering, the cor-

```
KwikCluster($G = (V, E^+, E^-)$)

  A recursive algorithm for CorrelationClustering on complete graphs
  if $V = \emptyset$ then return $\emptyset$
  pick random pivot $u \in V$
  set $C \leftarrow \{u\}, V' \leftarrow \emptyset$

  for all $v \in V \setminus \{u\}$:
      if $(u, v) \in E^+$ then
          add $v$ to $C$
      else (if $(u, v) \in E^-$)
          add $v$ to $V'$

  set $G' \leftarrow$ subgraph induced by $V'$

  return clusters $C$, KwikCluster($G'$)
```

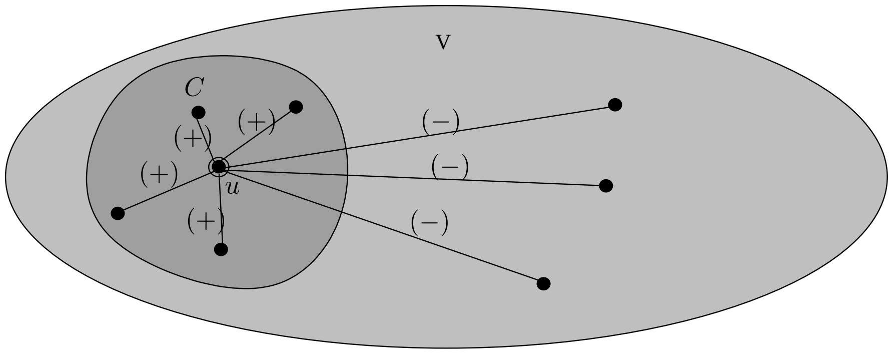


Figure 5.4: Pseudocode and diagram for KwikCluster

responding *majority* instance $G_w = (V, E_w^+, E_w^-)$ is defined by setting $(u,v) \in E_w^+$ if $w_{uv}^+ > w_{uv}^-$, otherwise $(u,v) \in E_w^-$, for all pairs u, v.

Note that if $G = (V, w^+, w^-)$ is induced by an instance of ConsensusClustering as in (5.4), then G_w is equivalent[3] to G_{maj} defined in (5.1). By outputting $\mathcal{C} = \text{KwikCluster}(G_w)$ we obtain an algorithm for weighted CorrelationClustering and, in particular for ConsensusClustering. This algorithm is a 3-approximation when the weights satisfy the tournament constraint, and a 2-approximation when they satisfy both the tournament constraint and the triangle inequality. By taking the best of KwikCluster and PickACluster we obtain an (11/7)-approximation for ConsensusClustering.

	KwikCluster	LP-KwikCluster	Previous
CorrelationClustering on complete graphs	3	5/2	4 [30]
Weighted CorrelationClustering w/ Tournament Constraint	5	5/2	7 [15, 59]
Weighted CorrelationClustering w/ Tournament Constraint & Triangle Inequality	2	2	3 [59]
ConsensusClustering (Better of PickACluster & ...)	11/7	4/3	2

Table 5.1: Summary of results for clustering

Additionally, we consider the following LP [30] for weighted CorrelationClustering:

$$\begin{aligned} &\text{minimize} \sum_{(u,v)\in E} \left(x_{uv}^+ w_{vu}^- + x_{uv}^- w_{uv}^+\right) \\ &x_{uy}^- \le x_{uv}^- + x_{yv}^- \quad \forall u,v,y \\ &x_{uv}^+ + x_{uv}^- = 1 \quad \forall u,v \\ &x_{uv}^+, x_{uv}^- \ge 0 \quad \forall u,v \end{aligned} \tag{5.5}$$

[3]In G_{maj} we used edges to denote E^+ and non-edges to denote E^-.

Clearly, if we could restrict x^+_{uv}, x^-_{uv} to be in $\{0,1\}$ for all u, v then the corresponding IP would be an exact mathematical program for weighted CorrelationClustering. Indeed, a feasible integral solution gives rise to a clustering $\mathcal{C}$ satisfying $u \equiv_{\mathcal{C}} v \iff x^+_{uv} = 1$, and the objective function is then exactly $\text{cost}(\mathcal{C})$.

```
LP-KwikCluster($G = (V, x^+, x^-)$)

  A recursive algorithm for rounding the LP for
  weighted CorrelationClustering. Given an
  LP solution $\{x^+_{uv}, x^-_{uv}\}_{u<v}$, returns a clustering
  of the vertices $V$.

  if $V = \emptyset$ then return $\emptyset$
  pick random pivot $u \in V$
  set $C \leftarrow \{u\}, V' \leftarrow \emptyset$

  for all $v \in V \backslash \{u\}$:
      with probability $x^+_{uv}$
          add $v$ to $C$
      else (with probability $x^-_{uv}$)
          add $v$ to $V'$

  set $G' \leftarrow$ subgraph induced by $V'$

  return clusters $C$, LP-KwikCluster($G'$)
```

Figure 5.5: Pseudocode for LP-KwikCluster

Assuming we have an optimal (fractional) solution $x = (x^+_{uv}, x^-_{uv})$ to the LP, we run algorithm LP-KwikCluster (V, x^+, x^-) (Figure 5.5) to round it. This is similar to KwikCluster, except that after choosing the pivot vertex, instead of deterministically placing vertices in the pivot's cluster, we randomly decide based on the value of the corresponding LP variables. This results in vastly improved approximation factors for the different scenarios we consider. In particular, by taking the best of LP-KwikSort and PickACluster, we obtain a (4/3)-approximation factor for ConsensusClustering. See Table 5.1 for a complete summary of the new approximation factors.

Chapter 6

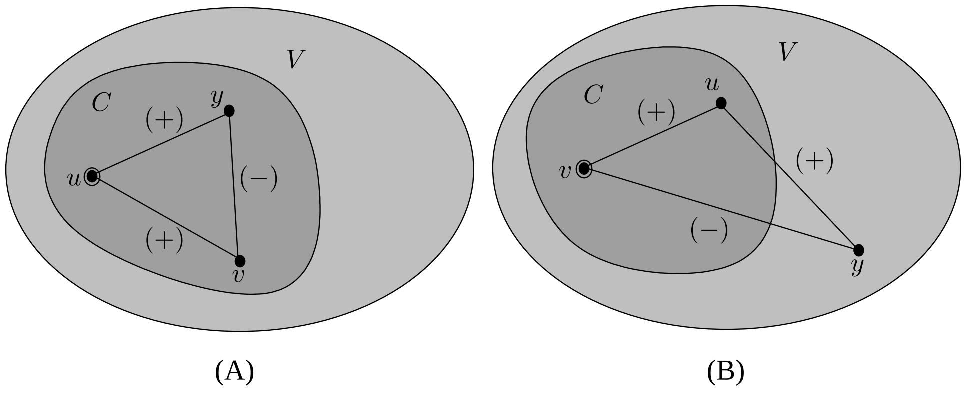
Analysis of Clustering Algorithms

Before we prove the results for the clustering problems, we make a few notational remarks. Recall that E is shorthand for $\binom{V}{2}$, and $(u, v) \in E$ refers to unordered pairs of vertices. In what follows we will use the term *triangles*. A triangle will always be a subset of three edges in E incident to three distinct vertices $u, v, y \in V$. For ease of notation, we will write (u, v, y) to refer to the triangle over distinct vertices $u, v, y \in V$, namely,

$$(u, v, y) \stackrel{\text{def}}{=} \{(u, v), (v, y), (y, u)\} \subseteq E \ .$$

We may abuse notation and write $u \in t$ to say that u is a vertex in some triangle t, though we keep in mind that t is a set of three edges and not three vertices.

If $G = (V, E^+, E^{)})$ is a complete CorrelationClustering graph then (u, v, y) is a *paradox* triangle in G if exactly two of $(u, v), (v, y), (y, u)$ belong to E^+. This is exactly the paradox presented in Section 5.2. Paradox triangles will play a role in the clustering analysis similar to that of directed triangles in Chapter 3.



When a pivot is chosen among the vertices of a paradox-triangle $t = (u, v, y)$, the edge not incident to the pivot is violated and charges the triangle a unit cost. In (A) u is the pivot, and in (B) v is the pivot.

Figure 6.1: Charging paradox triangles

6.1 KwikCluster for CorrelationClustering on complete graphs

Let $G = (V, E^+, E^-)$ be a complete CorrelationClustering instance. Consider algorithm KwikCluster (Figure 5.4) for approximating it.

Theorem 6.1. *The output of* KwikCluster(G) *is a random expected* 3*-approximation solution for* CorrelationClustering *on complete graphs.*

Proof. Let $\mathcal{C}$ be the (random) output of KwikCluster(G). The main idea is to charge edges contributing to cost($\mathcal{C}$) to paradox triangles (Figure 6.1). An edge $(u, v) \in E^+$ is charged if u and v are separated in $\mathcal{C}$. This happens if a third vertex $y \in V$ is chosen as pivot when all of y, u, v are input to the same recursive call to KwikCluster, and either

$$(y, u) \in E^+ \text{ and } (y, v) \in E^-$$

or

$$(y, v) \in E^+ \text{ and } (y, u) \in E^- \ .$$

The violation of (u, v) in this case will be charged to the triangle (u, v, y).

An edge $(u, v) \in E^-$ is charged if u and v are co-clustered in $\mathcal{C}$. This happens if a third vertex

$y \in V$ was chosen as pivot, and

$$(y,u),(y,v) \in E^+ .$$

The violation of (u,v) will be charged to the triangle (u,v,y).

In both cases we notice that a triangle consisting of (u,v,y) can be charged for a violation only if it is a paradox triangle. The proof now is very similar to that of Theorem 3.1 for ranking. We let T denote the set of paradox triangles in G. Remember that exactly one edge of t belongs to E^-. we argue that there is a bijection between the charged triangles and edges in violation. We denote by p_t the probability that triangle t is charged, hence $\mathbf{E}[\mathrm{cost}(\mathcal{C})] = \sum_{t\in T} p_t$. Finally, we argue that $\sum_{t\in T} p_t/3$ is a lower bound for the optimal solution, because $\{p_t/3\}_{t\in T}$ is a *packing* of triangles $t \in T$, and any packing of the triangles lower bounds the optimal solution by LP duality. This proves the expected 3-approximation guarantee. □

6.2 KwikCluster for weighted CorrelationClustering

Now let (V, w^+, w^-) be a weighted Correlation-Clustering instance, where $w^+, w^- \in (\mathbb{R}^+)^E$. Unlike weighted FasTour, we will *only* consider weight systems that satisfy the tournament constraint $w^+_{uv} + w^-_{uv} = 1$.

Consider running KwikCluster(G_w) as an approximation algorithm for instance G, where $G_w = (V, E^+_w, E^-_w)$ is the (unweighted) majority instance attached to G, as in Definition 5.4. Theorem 6.2 is analogous to Theorem 3.7:

Theorem 6.2. *Algorithm* KwikCluster(G_w) *is a* 5*-approximation (resp.* 2*-approximation) for weighted* CorrelationClustering *with the tournament constraint (resp. with the tournament constraint and triangle inequality).*

Theorem 6.2 is a consequence of the following Theorem 6.3 and Lemma 6.4. The proof is almost identical to that of Theorem 3.7, with paradox triangles (Figure 5.1) in G_w replacing the role of directed triangles in tournaments. For each edge $e = (u,v) \in E$ we let

$$w(e) = \begin{cases} w^+_{uv} & w^+_{uv} > w^-_{uv} \\ w^-_{uv} & \text{otherwise} \end{cases} \qquad \bar{w}(e) = \begin{cases} w^-_{uv} & w^+_{uv} > w^-_{uv} \\ w^+_{uv} & \text{otherwise} . \end{cases}$$

For each paradox triangle $t = \{e_1, e_2, e_3\} \subseteq E$ in G_w we assign a weight:

$$w(t) = w(e_1) + w(e_3) + w(e_3) \ .$$

We let T denote the set of paradox triangles in G_w.

Now we define a random variable $C^{KC} = \text{cost}_G(\mathcal{C})$, where $\mathcal{C}$ is the output of KwikCluster(G_w). We let $C^{OPT} = \sum_{e \in E} c^*(e)$ denote the cost of some fixed optimal solution $\mathcal{C}^*$ for G, where $c^*(e)$ is the charge for edge e (i.e. either $w(e)$ if $\mathcal{C}^*$ violates e, or $\bar{w}(e)$ otherwise). Finally, for all $t = \{e_1, e_2, e_3\} \in T$, let $c^*(t)$ denote $c^*(e_1) + c^*(e_2) + c^*(e_3)$.

Theorem 6.3. *For an instance $G = (V, w^+, w^-)$ of weighted* CorrelationClustering*, if there exists a constant $\alpha > 0$ such that $w(t) \leq \alpha c^*(t)$ for all $t \in T$, then $\mathbf{E}[C^{KC}] \leq \alpha C^{OPT}$, i.e.* KwikCluster$(G_w)$ *is an expected α-approximation solution.*

Proof. As in the proof of Theorem 3.5, we charge parts of C^{KC} to events in the probability space of KwikCluster. Whenever an edge of some paradox triangle $t = \{e_1, e_2, e_3\} \in T$ is in violation due to the choice of one of its vertices as pivot, we charge the cost of the violated edge (the one not incident to the pivot) to t. Conditioned on t being charged, each one of its three edges is equally likely to be violated, and hence its expected charge is $(w(e_1) + w(e_2) + w(e_3))/3$. If p_t is the probability that t is charged, we get

$$C^{KC} = B^{KC} + F^{KC},$$

where B^{KC} counts the cost charged to paradox triangles, and F^{KS} counts the rest:

$$\begin{aligned} B^{KC} &= \sum_{t \in T} \frac{1}{3} p_t w(t) \\ F^{KC} &= \sum_{e \in E} \left(1 - \sum_{t \in T:\ e \in t} \frac{1}{3} p_t\right) \bar{w}(e) \end{aligned}$$

Similarly, we decompose C^{OPT} as:

$$C^{OPT} = B^{OPT} + F^{OPT},$$

where

$$B^{OPT} = \sum_{t=\{e_1,e_2,e_3\}\in T} \frac{1}{3} p_t (c^*(e_1) + c^*(e_2) + c^*(e_3)) = \sum_{t\in T} \frac{1}{3} p_t c^*(t)$$

$$F^{KC} = \sum_{e\in E} \left(1 - \sum_{t\in T:\ e\in t} \frac{1}{3} p_t\right) c^*(e)\ .$$

Recall that $\bar{w}(e) \leq c^*(e)$ and that $\left(1 - \sum_{t\in T:\ e\in t} \frac{1}{3} p_t\right) \geq 0$ for all $e \in E$ (the latter inequality captures the fact that only one paradox triangle can be charged for the violation of a given pair). The statement of the theorem follows. $\square$

Lemma 6.4. *If the weights satisfy the tournament constraint ($w^+_{uv} + w^-_{uv} = 1$), then $w(t) \leq 5c^*(t)$ for all $t \in T$. If, in addition, the weights satisfy the triangle inequality ($w^-_{uv} \leq w^-_{vy} + w^-_{yu}$), then $w(t) \leq 2c^*(t)$.*

Proof. For the tournament constraint only case, we fix $t = \{e_1, e_2, e_3\} \in T$ and we can assume w.l.o.g. that $1/2 \leq w(e_1) \leq w(e_2) + w(e_3) \leq 1$. It suffices to prove that

$$w(t) = w(e_1) + w(e_2) + w(e_3) \leq 5w(e_1)$$

because clearly $w(e_1) \leq c^*(e_1) + c^*(e_2) + c^*(e_3)$ (any optimal solution has to violate at least one edge of t, and therefore pays at least the price of the lightest one). Proving this is identical to the first part of the proof of Theorem 3.6.

To show the second part, we notice that the tournament constraint together with the triangle inequality imply:

$$\begin{aligned} w(e_1) &\leq \bar{w}(e_2) + \bar{w}(e_3) \\ w(e_2) &\leq \bar{w}(e_3) + \bar{w}(e_1) \\ w(e_3) &\leq \bar{w}(e_1) + \bar{w}(e_2) \end{aligned} \tag{6.1}$$

(Note that if, say, $e_1 \in E^-_w, e_2 \in E^+_w, e_3 \in E^+_w$ then the first inequality in (6.1) is exactly the triangle inequality, but the other two inequalities require both the triangle inequality and the tournament constraint[1]). The proof continues as in the second part of the proof of Theorem 3.6. $\square$

[1]It is possible to formally define the triangle inequality constraint to immediately yield (6.1) without the need of

Theorem 6.2 implies that KwikCluster(G_w) is an expected 2-approximation for the ConsensusClustering instance $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$, where G is the induced weighted CorrelationClustering instance.

6.3 An improved approximation ratio for ConsensusClustering

As we did for RankAggregation in Section 3.3, by coupling KwikCluster and PickACluster we obtain an improved algorithm for ConsensusClustering.

Theorem 6.5. *The best of* KwikCluster(G_w) *and* PickACluster($\mathcal{C}_1, \ldots, \mathcal{C}_k$) *has an expected approximation ratio of at most* $\frac{11}{7}$ *for* ConsensusClustering.

```
PickACluster($\mathcal{C}_1, \ldots, \mathcal{C}_k$)
  pick $i \in [k]$ uniformly at random
  return $\mathcal{C}_i$
```

Figure 6.2: Pseudocode for PickACluster

Proof. We repeat the proof of Theorem 6.5 almost verbatim. We let the random variable C^{PAC} denote the cost of $\mathcal{C} =$ PickACluster($\mathcal{C}_1, \ldots, \mathcal{C}_k$). We notice

$$\mathbf{E}\left[C^{PAC}\right] = \sum_{e \in E} z(e) \ ,$$

where $z(e)$ is the expected cost PickACluster pays on edge $e \in E$. With probability $w(e)$ this cost is $\bar{w}(e)$, and with probability $\bar{w}(e)$, this cost is $w(e)$. Therefore, $z(e) = 2w(e)\bar{w}(e)$. We now decompose $\mathbf{E}\left[C^{PAC}\right]$ to F^{PAC} and B^{PAC} similarly to (3.7).

We decomposed the optimal cost C^{OPT}, the expected KwikCluster cost $\mathbf{E}\left[C^{KC}\right]$ and the expect PickACluster cost $\mathbf{E}\left[C^{PAC}\right]$ to B-costs (charging paradox-triangles in G_w) and to F-

the tournament constraint, but we prefer Definition 5.2 because it captures the idea that w_{uv}^{-} is a metric measuring the amount of "separation" between u and v.

costs (charging edges). The required approximation guarantee is argued separately for the B-costs and for the F-costs, using equivalents to Theorem 3.8 and Lemma 3.9. □

6.4 LP-KwikCluster for rounding the clustering LP

We now consider the analysis of LP-KwikCluster on CorrelationClustering and ConsensusClustering. Let $x = \{x^+_{uv}, x^-_{uv}\}_{(u,v)\in E}$ be an optimal solution to the LP (5.5) corresponding to an instance $G = (V, w^+, w^-)$ of weighted CorrelationClustering. (Recall that E is simply our shorthand for $\binom{V}{2}$). The following theorem establishes the results in the center column of Table 5.1.

Theorem 6.6. LP-KwikCluster(V, x^+, x^-) *returns a clustering with an expected cost of at most* 2.5 *(resp.* 2*) times the value of LP (5.5) on optimal solution x, when the weights satisfy the tournament constraint (resp. the tournament constraint and the triangle inequality). The best of* LP-KwikCluster(V, x^+, x^-) *and* PickACluster$(\mathcal{C}_1, \ldots, \mathcal{C}_k)$ *returns a clustering with an expected cost of at most* $4/3$ *times the value of the optimal solution x to LP (5.5), when the weights are induced by the* ConsensusClustering *instance* $\mathcal{C}_1, \ldots, \mathcal{C}_k$.

Proof. Define

$$c^*_{uv} = x^+_{uv} w^-_{uv} + x^-_{uv} w^+_{uv} \ .$$

Let C_{LP} denote the value of LP (5.5) on x, which can be written as

$$C_{LP} = \sum_{(u,v)\in E} c^*_{uv} \ .$$

Let T be the set of all triangles over V, namely,

$$T = \{(u, v, y) : \ u, v, y \in V \text{ distinct}\} \ .$$

For a triangle $t = (u, v, y) \in T$, as usual, we let A_t denote the event that one of u, v, y was chosen as pivot when the other two vertices are in the same recursive call to LP-KwikCluster. Let $p_t = \mathbf{Pr}[A_t]$.

Let $B^t_{\{v\}y} \subseteq A_t$ denote the event that u was the vertex chosen as pivot, v is co-clustered with u, and y is separated from u. The probability $p^t_{\{v\}y}$ of $B^t_{\{u\}y}$ *conditioned* on A_t is

$$\frac{1}{3}x^+_{uv}x^-_{uy} \ ,$$

because each vertex was equally likely to be chosen as pivot (hence 1/3), and x^+_{uv} (resp. x^-_{uy}) is the probability of v co-clustered with u (resp. y separated from u). The cost corresponding to (v, y) incurred by this choice is w^+_{vy}, because v and y are separated in the output of LP-KwikCluster. This cost is charged to t.

Let $B^t_{\{vy\}} \subseteq A_t$ denote the event that u was the vertex chosen as pivot, and both v, y are co-clustered with u. The probability $p^t_{\{vy\}}$ of $B^t_{\{vy\}}$ *conditioned* on A_t is clearly

$$\frac{1}{3}x^+_{uv}x^+_{uy} \ .$$

The cost corresponding to (v, y) incurred by this choice is w^-_{vy}, because v and y are co-clustered in the output of LP-KwikCluster. This cost is, again, charged to t.

Finally, let $B^y_{uv} \subseteq A_t$ denote the event that u was the vertex chosen as pivot, and both v, y are separated from u. The cost corresponding to (v, y) incurred by this choice is yet to be determined in a deeper recursive call to LP-KwikCluster, and t is *not* charged in this case.

The expected charge on t conditioned on A_t, which we denote by y_t, is exactly:

$$y_t \overset{\text{def}}{=} \sum_{e=(u,v)\in t} \left((p^t_{\{u\}v} + p^t_{\{v\}u})w^+_{uv} + p^t_{\{uv\}}w^-_{uv} \right) \ .$$

We now claim that for all $e = (u, v) \in E$,

$$\sum_{t:e\in t} p_t(p^t_{\{u\}v} + p^t_{\{v\}u} + p^t_{\{uv\}}) \leq 1 \ . \tag{6.2}$$

Indeed, the expression inside the sum in (6.2) is exactly the probability that the cost corresponding to e incurred by LP-KwikSort is charged to a triangle t incident to e. It is easy to see that this cost can be charged to at most one triangle, and hence (6.2) is a sum of probabilities of pairwise-disjoint events, in particular, a probability.

Let C^{KC}_{LP} denote the cost of the output clustering of LP-KwikCluster. We decompose the expected cost $\mathbf{E}[C^{KC}_{LP}]$ to B^{KC}_{LP}, corresponding to triangle charges, plus F^{KC}_{LP}, corresponding to edge charges (all the rest). Note that an edge $e = (u, v) \in E$ is charged to itself exactly if one of its endpoints was chosen as pivot, when both its endpoints are input to the same recursive call to LP-KwikCluster. Conditioned on this event, the expected charge is exactly c^*_{uv}, matching the corresponding LP charge. Using our notation, we can now write:

$$B^{KC}_{LP} = \sum_t p_t y_t$$

$$F^{KC}_{LP} = \sum_{e=(u,v)\in E} \left(1 - \sum_{t:\ e\in t} p_t(p^t_{\{u\}v} + p^t_{\{v\}u} + p^t_{\{uv\}})\right) c^*_{uv}.$$

Using our analysis of the probability space induced by the execution of LP-KwikCluster, we can also decompose the LP value C_{LP} as $B_{LP} + F_{LP}$, where

$$B_{LP} = \sum_t p_t \sum_{e=(u,v)\in t} (p^t_{\{u\}v} + p^t_{\{v\}u} + p^t_{\{uv\}})c^*_{uv}$$

$$F_{LP} = \sum_{e=(u,v)\in E} \left(1 - \sum_{t:e\in t} p_t(p^t_{\{u\}v} + p^t_{\{v\}u} + p^t_{\{uv\}})\right) c^*_{uv} \ .$$

The proof of the following lemma is deferred to Section 6.5.

Lemma 6.7. *If the weight system satisfies the probability constraints (resp. probability constraints and triangle inequality constraints), then for any $t \in T$,*

$$y_t \leq \tau \sum_{e=(u,v)\in t} (p^t_{\{u\}v} + p^t_{\{v\}u} + p^t_{\{uv\}})c^*_{uv} \ ,$$

where $\tau = 5/2$ (resp. $\tau = 2$).

As a result, we get (using (6.2)) that

$$\mathbf{E}[C^{KC}_{LP}] = B^{KC}_{LP} + F^{KC}_{LP} \leq \tau(B_{LP} + F_{LP}) = \tau C_{LP} \ ,$$

for $\tau = 2$ or $\tau = 5/2$ (depending on the weights). This proves the first part of the theorem.

For ConsensusClustering: Let C^{PAC} denote the value of PickACluster. Clearly,

$$\mathbf{E}\left[C^{PAC}\right] = \sum_{e=(u,v)\in E} z_{uv} \ ,$$

where $z_{uv} = 2w^+_{uv}w^-_{uv}$ is the expected contribution of pair $(u,v) \in E$ to the cost. Again, decompose $\mathbf{E}\left[C^{PAC}\right]$ as $B^{PAC}_{LP} + F^{PAC}_{LP}$, where

$$\begin{aligned} B^{PAC}_{LP} &= \sum_t p_t \sum_{e=(u,v)\in t} (p^t_{\{u\}v} + p^t_{\{v\}u} + p^t_{\{uv\}})z_{uv} \\ F^{PAC}_{LP} &= \sum_{e=(u,v)\in E} \left(1 - \sum_{t:e\in t} p_t(p^t_{\{u\}v} + p^t_{\{v\}u} + p^t_{\{uv\}})\right) z_{uv} \ . \end{aligned}$$

The proof of the following lemma is deferred to Section 6.5.

Lemma 6.8. *For all $t \in T$,*

$$\frac{2}{3}y_t + \frac{1}{3}\sum_{e=(u,v)\in t}(p^t_{\{u\}v} + p^t_{\{v\}u} + p^t_{\{uv\}})z_{uv} \le \frac{4}{3}\sum_{e=\{u,v\}\in t}(p^t_{\{u\}v} + p^t_{\{v\}u} + p^t_{\{uv\}})c^*_{uv} \ .$$

Lemma 6.8 implies (using (6.2)) that

$$\frac{2}{3}B^{KS}_{LP} + \frac{1}{3}B^{PAC}_{LP} \le \frac{4}{3}B_{LP} \ .$$

It is easy to see that $z_{ij} \le 2c^*_{ij}$, therefore, $F^{PAC}_{LP} \le 2F_{LP}$ and consequently

$$\frac{2}{3}F^{KS}_{LP} + \frac{1}{3}F^{PAC}_{LP} \le \frac{4}{3}F_{LP} \ .$$

Combining, we conclude that $\frac{2}{3}\mathbf{E}\left[C^{KS}_{LP}\right] + \frac{1}{3}\mathbf{E}\left[C^{PAC}\right] \le \frac{4}{3}C_{LP}$. This proves that by flipping a $(2/3, 1/3)$-biased coin and returning the output of LP-KwikCluster or PickACluster based on the outcome is an expected $(4/3)$-approximation algorithm for ConsensusClustering. In particular, the better of the two is a $(4/3)$-approximation algorithm. This completes the proof of Theorem 3.13.

□

6.5 Proof of clustering polyhedral inequalities

We prove Lemmas 6.7 and 6.8. Fix a triangle t consisting of three arbitrary vertices $t = (1, 2, 3) \subseteq V$. The two lemmas state inequalities on a single triangle t which we can assume to equal our fixed triangle, without loss of generality. The interesting parameters corresponding to t are the weights $w^-_{12}, w^-_{23}, w^-_{23}$ and the assignment $x^-_{12}, x^-_{23}, x^-_{31}$ of the LP variables at our fixed optimal solution. Note that the tournament constraint is assumed on both x and w, and hence we can always substitute $w^+_{12} = 1 - w^-_{12}$, $w^+_{23} = 1 - w^-_{23}$ and $w^+_{31} = 1 - w^-_{31}$ (similarly for x).

For ease of notation, we let

$$\begin{array}{ccc} x_1 \overset{\text{def}}{=} x^-_{23} & x_2 \overset{\text{def}}{=} x^-_{31} & x_3 \overset{\text{def}}{=} x^-_{12} \\ w_1 \overset{\text{def}}{=} w^-_{23} & w_2 \overset{\text{def}}{=} w^-_{31} & w_3 \overset{\text{def}}{=} w^-_{12} \end{array}$$

Let $\Pi \subseteq \mathbb{R}^3$ denote the *tournament constraint* polytope as defined in (3.13). Let $\Delta \subseteq \Pi$ denote the *triangle inequality and tournament constraints* polytope for clustering, that is,

$$\Delta = \{(a_1, a_2, a_3) \in \Pi \ : \ a_3 \leq a_1 + a_2, \ a_1 \leq a_2 + a_3, \ a_2 \leq a_3 + a_1\} \ .$$

We define three functions, *kc* (corresponding to LP-KwikCluster), *pac* (corresponding to PickACluster), and *lp* (corresponding to the value of LP(5.5) on the fixed optimal solution).

The functions are real-valued on $\mathbb{R}^6$ and defined as follows:

$$\begin{aligned}
kc(\mathbf{x},\mathbf{w}) &= (1-x_1)(1-x_2)w_3 + (x_1(1-x_2)+(1-x_1)x_2)(1-w_3)\\
&\quad + (1-x_2)(1-x_3)w_1 + (x_2(1-x_3)+(1-x_2)x_3)(1-w_1)\\
&\quad + (1-x_3)(1-x_1)w_2 + (x_3(1-x_1)+(1-x_3)x_1)(1-w_2)
\end{aligned}$$

$$\begin{aligned}
pap(\mathbf{x},\mathbf{w}) &= ((1-x_1)(1-x_2)+(1-x_1)x_2+x_1(1-x_2))2w_3(1-w_3)\\
&\quad + ((1-x_2)(1-x_3)+(1-x_2)x_3+x_2(1-x_3))2w_1(1-w_1) \qquad (6.3)\\
&\quad + ((1-x_3)(1-x_1)+(1-x_3)x_1+x_3(1-x_1))2w_2(1-w_2)
\end{aligned}$$

$$\begin{aligned}
lp(\mathbf{x},\mathbf{w}) &= ((1-x_1)(1-x_2)+(1-x_1)x_2+x_1(1-x_2))(x_3(1-w_3)+(1-x_3)w_3)\\
&\quad + ((1-x_2)(1-x_3)+(1-x_2)x_3+x_2(1-x_3))(x_1(1-w_1)+(1-x_1)w_1)\\
&\quad + ((1-x_3)(1-x_1)+(1-x_3)x_1+x_3(1-x_1))(x_2(1-w_2)+(1-x_2)w_2)
\end{aligned}$$

Lemma 6.7 is equivalent to showing that $f = kc - \frac{5}{2}lp \leq 0$ for all $(\mathbf{x},\mathbf{w}) \in \Delta \times \Pi$ and $g = kc - 2lp \leq 0$ for all $(\mathbf{x},\mathbf{w}) \in \Delta \times \Delta$. We make the two simplification steps as before.

1. *Linearity in* $\mathbf{w}$: The functions f and g are linear in $\mathbf{w}$ (for $\mathbf{x}$ fixed). Arguing as before, it suffices to analyze f on $\mathbf{w} = (0,0,0)$, $\mathbf{w} = (0,0,1)$, $\mathbf{w} = (0,1,1)$ and $\mathbf{w} = (1,1,1)$, and g on $\mathbf{w} = (0,0,0), \mathbf{w} = (0,1,1), \mathbf{w} = (1,1,1)$.

2. *Trilinearity in* $\mathbf{x}$*:* For $i = 1,2,3$ the functions f and g are linear in x_i when x_j's are fixed for $j \in \{1,2,3\} \setminus \{i\}$. This means that any point $\mathbf{x} \in \Delta$ such that $\mathbf{x} + t\mathbf{e}_i \in \Delta$ for all $t \in [-\varepsilon, \varepsilon]$ for some $\varepsilon > 0$ and some $i \in \{1,2,3\}$ (where $\mathbf{e}_i$ is a standard basis element of $\mathbb{R}^3$) is *not* a strict local maximum of f, g in Δ, so these points $\mathbf{x}$ can be ignored. The points that are left are $\mathbf{x} \in \Delta_1 \cup \Delta_2 \cup \Delta_3$ where $\Delta_i = \Delta \cap H_i$ for $i = 1,2,3$ and $H_1 = \{(a_1,a_2,a_3) \in \mathbb{R}^3 : \ a_1 = a_2 + a_3\}$, $H_2 = \{(a_1,a_2,a_3) \in \mathbb{R}^3 : \ a_2 = a_3 + a_1\}$, $H_3 = \{(a_1,a_2,a_3) \in \mathbb{R}^3 : \ a_3 = a_1 + a_2\}$ (Figure 6.3).

The functions f, g restricted to one of the finitely many "interesting" points $\mathbf{w}$ and to $\mathbf{x} \in \Delta_i$ for some $i \in \{1,2,3\}$ can be represented as polynomials of total degree 3 and maximal degree 2

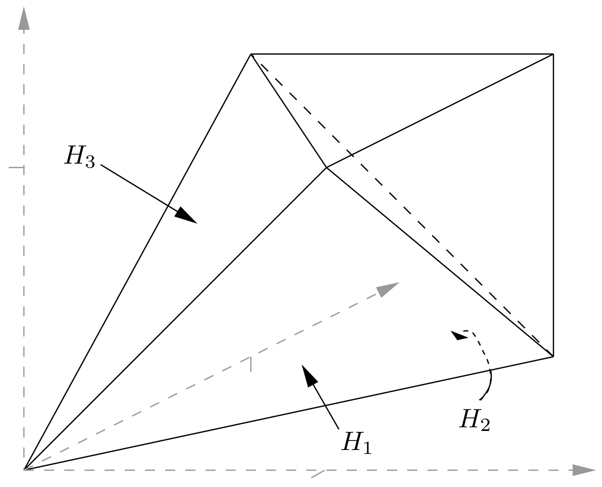


Figure 6.3: The clustering Δ-polytope

in each variable. We omit the tedious yet elementary details of showing that in these domains, $f, g \leq 0$.

Lemma 6.8 is equivalent to proving that $h = 2kc/3 + pap/3 - 4lp/3 \leq 0$ for all $(\mathbf{x}, \mathbf{w}) \in \Delta \times \Delta$. We prove this assertion as follows. *Using symmetries of h*: Let $(\mathbf{x}, \mathbf{w})$ be some local maximum of h in $\Delta \times \Delta$. Assume there is an index $i \in \{1, 2, 3\}$ such that all of $x_i, x_{i+1}, w_i, w_{i+1} \notin \{0, 1\}$ (the index arithmetic is modulo 3). Without loss of generality, assume that $x_1, x_2, w_1, w_2 \notin \{0, 1\}$. Since $(\mathbf{x}, \mathbf{w})$ is a local maximum of h on $\Delta \times \Delta$, and since $x_1, x_2, w_1, w_2 \notin \{0, 1\}$, the derivatives of h on the hyperplane $H = \{(\mathbf{x}, \mathbf{w}) + t(1, -1, 0, 0, 0, 0) + s(0, 0, 0, 1, -1, 0) : \ t, s \in \mathbb{R}\}$ must vanish at $t = s = 0$. One verifies that h is a polynomial of total degree 2 in t, s on H, and the derivatives vanish at the unique point $t = (x_2 - x_1)/2, s = (w_2 - w_1)/2$. Therefore, we may assume that $x_1 = x_2$ and $w_1 = w_2$. Now if in addition $x_3, w_3 \notin \{0, 1\}$ then we use the same argument (switching the roles of the variables), and we can assume that $x_1 = x_2 = x_3, w_1 = w_2 = w_3$. It is trivial to show that $h \leq 0$ under this constraint.

Boundary cases: We can now assume that either: (1) at least two of $x_1, x_2, x_3, w_1, w_2, w_3$ are in $\{0, 1\}$, or, (2) $x_1 = x_2, w_1 = w_2$ and at least one of x_3, w_3 are in $\{0, 1\}$. In addition, the function h is trilinear in $\mathbf{x}$, so we may assume (as above) that $\mathbf{x} \in H_1 \cup H_2 \cup H_3$. This reduces the problem to proving inequalities for polynomials of total degree at most 4 and maximal degree at most 3 (resp. 2) in each x-variable (resp. w-variable), in 3-dimensional polytopes. We omit the tedious yet elementary details of this case-by-case proof.

Chapter 7

Concluding Remarks

RankAggregation using sorting algorithms

KwikSort is in fact the well-known *quick-sort* algorithm for ordered data with transitivity violations. Can we use other standard sorting algorithms, such as *merge-sort* to obtain similar approximation algorithms?

Derandomization

Derandomization of the algorithms analyzed in this work is an interesting problem. Recently, there has been considerable progress in this direction. Anke van Zuylen [104] shows how to derandomize the LP rounding algorithm for both ranking and clustering problems with either probability constraints or triangle inequality constraints, with slightly weaker bounds. She also achieves bounds for the triangle inequality constraints *only* case for clustering, assuming a slightly stronger and more descriptive notion of the triangle inequality which we did not consider here. Extending her analysis to an algorithm that takes the best of the new (deterministic) LP-rounding algorithm and PickAPerm (or Pick-A-Cluster) is an interesting open question. In addition, Coppersmith et. al. show that ordering a weighted tournament by in-degree is a 5-approximation for weighted Fas-Tournament with probability constraints.

Tight instances

Finding tight examples for the algorithms presented in this work is an interesting problem. For weighted weighted FAS-TOURNAMENT and weighted CORRELATION-CLUSTERING with probability constraints, Warren Schudy communicated the following tight example for the KWIKSORT and KWIKCLUSTER, respectively. It suffices to consider unweighted instances (weights are $0, 1$). For the ranking problem, take an acyclic tournament and flip the edge connecting the lowest and the highest ranked vertices. The optimal solution pays 1. KWIKSORT pays $n-2$ if the lowest or highest ranked vertices are chosen as pivot in the first step, otherwise 1. Therefore, the expected ratio is $3(n-2)/n$, which tends to 3 as $n \to \infty$. For the clustering problem set all edges to $(+)$ except for one which is set to $(-)$. The optimal solution pays 1 by clustering all the vertices together. KWIKCLUSTER pays $n-2$ if one of the two vertices incident to the unique $(-)$-edge is chosen as pivot in the first step, otherwise the optimal cost of 1, giving an expected ratio of $3(n-2)/n$. Finding tight examples for the triangle inequality cases as well as for the aggregation problems remains an open problem.

Generalized ranking

One of the main motivations for RANKAGGREGATION comes from fighting search engine spam [43,44]. By aggregating outputs of different search engines, the effect of spam (artificially increasing the rank of a web page with respect to a search engine) can be reduced. In this application, however, it is more likely that each search engine provides us with a partial list of web pages (relevant to a given search query). Fagin et al [50] discuss the difficulties of dealing with partial (top-k) rankings and compare several methods of measuring the distance between two top-k lists. A top-k list can be viewed as a special case of ranking with *ties*. A ranking with ties on a set V is a binary relation $\leq$ that is reflexive ($u \leq u$ for all $u \in V$), transitive ($u \leq v$ and $v \leq y$ implies $u \leq y$) and comparable (for all u, v, either $u \leq v$ or $v \leq u$). Note that this is not a *partial order relation* because there is no antisymmetry (we allow both $u \leq v$ and $v \leq u$ for $u \neq v$). A ranking with ties induces a natural equivalence relation $\equiv$ on V, in which $u \equiv v$ if both $u \leq v$ and $v \leq u$. A top-k list is a ranking with ties in which the elements not among the top-k are an equivalence class ranked lower than the top-k elements (each one of which is an equivalence singleton). Another interesting example of ranking with ties is *questionnaires*: A questionnaire is a form in which a participant evaluates a list of items (e.g. films) by choosing a crude value (e.g. "very good", "good", "bad", "very bad") for each item (see example 1. in Chapter 1). Lacking exact numerical interpretations of the crude values, a reasonable alternative is to consider the obvious induced pairwise relation: If X is "good" and Y is "bad" then X is preferred over Y by the participant. This gives rise to a natural ranking with ties, in which all items with the same crude value are an equivalence class. Aggregating information from questionnaires is a highly standard task in fields such as experimental psychology. There is ongoing work on extending the techniques presented here to aggregation of ranking with ties.

Generalized clustering

A depth-M hierarchical clustering of dataset is a collection $\mathcal{C}_1, \ldots, \mathcal{C}_M$ of regular clusterings in which each $\mathcal{C}_i$ is a refinement of $\mathcal{C}_{i+1}$ for $i = 1, \ldots, M-1$. This means that each cluster in $\mathcal{C}_i$ is contained in some cluster of $\mathcal{C}_{i+1}$. A hierarchical clustering can be naturally viewed as a uniform-height tree, with the root representing the entire dataset, its children representing the clusters of $\mathcal{C}_M$ and so on down to the leaves, representing the individual dataset elements. The corresponding tree metric on the dataset is an *ultrametric* ($d(u,v) \leq \max\{d(u,y) + d(y,v)\}$). The problem of fitting dissimilarity data to an ultrametric is a central problem in statistics, and also arises in phylogeny, where the objective is to learn the evolution tree by fitting a tree metric on taxa. It can also be viewed as a hierarchical generalization of CorrelationClustering, and some of the techniques developed in this work have been successfully used for this problem by Ailon and Charikar [2].

Other open problems

- Is RankAggregation NP-Hard for three permutations [43, 44]?
- Is ConsensusClustering NP-Hard for a constant number of clusters [54, 105]?
- Can we approximate weighted CorrelationClustering with triangle inequalities, but no probability constraints?

Part III

The Fast Johnson-Lindenstrauss Transform and Approximate Nearest Neighbor Searching

Chapter 1

Introduction

Metric embeddings have been proven to be very useful algorithmic tools. In the most general setting, one wishes to map a finite metric space (U, d_U) to another metric space (V, d_V) belonging some restricted family, without distorting the pairwise distances too much. If we denote the mapping by Φ, we usually care about minimizing the relative distortion, which we can define here as

$$\max_{x_1, x_2 \in U} \frac{|d_U(x_1, x_2) - d_V(\Phi(x_1), \Phi(x_2))|}{d_U(x_1, x_2)} .$$

(Much of the computer science literature uses the expression $\|\Phi\|_{Lip} \cdot \|\Phi^{-1}\|_{Lip}$ as a measure of distortion, where $\| \cdot \|_{Lip}$ is the *Lipschitz norm* of a function between metric spaces. Also, much work related to this subject makes the technical assumption that Φ is 1-Lipschitz, namely, it does not stretch pairwise distances, but we do not assume that here.) The metric space Y usually belongs to a restricted family of metric spaces which is easier to work with.

Linial, London and Rabinovich in their seminal paper [82] first considered the algorithmic applications of metric embeddings. Embedding into ℓ_1 metrics is especially important in optimization of hard cut problems in graphs. Embedding into tree metrics is useful in optimization of hard network design and clustering problems.

In many cases, we are interested in embedding high-dimensional normed metrics spaces into low dimensional normed spaces. Papadimitriou et al [95] use low-dimensional embeddings to speed up computation of low-rank approximation of matrices. Schulman [99] used them for efficiently approximating certain metric clustering problems. Indyk [71] used them for data streaming applications. Indyk et al [74] and Kushilevitz et al [81] used low-dimensional embeddings to speed up approximate nearest neighbor searching. The latter example is the main motivation for this work, though we hope to see other applications as well.

Our main result is a new low-distortion embedding of ℓ_2^d into $\ell_p^{O(\log n)}$ $(p = 1, 2)$, where n is the number of points. The possibility of obtaining such an embedding with distortion $(1 + \varepsilon)$ for any $\varepsilon > 0$ using the Johnson-Lindenstrauss (JL) transform has long been known. In this work we consider the complexity of implementing this embedding, and show a nontrivial way to obtain a significant speedup. We call our new embedding the *Fast-Johnson-Lindenstrauss-Transform* (FJLT). Naive implementations of JL incur only a polynomial cost, which is enough for some applications. However, for other applications, such as approximate nearest neighbor searching, sublinear algorithms are sought. Applying JL is the bottleneck of some of the fastest algorithms. We apply the $p = 1$ case to improve them.

1.1 History of the Johnson-Lindenstrauss transform

By the JL lemma [73,75,85], n points in Euclidean space can be projected down to $k = c\varepsilon^{-2}\log n$ dimensions while incurring a distortion of at most $1 + \varepsilon$, where $c > 0$ is some global constant. More precisely, for any set V of n points in d dimensions, taking $k = c\varepsilon^{-2}\log n$ (for some global constant c) suffices to ensure that with constant probability, for all $x, y \in V$

$$\sqrt{\frac{k}{d}}\|x - y\|_2(1 - \varepsilon) \le \|\Phi x - \Phi y\|_2 \le \sqrt{\frac{k}{d}}\|x - y\|_2(1 + \varepsilon) \ .$$

The original JL transform was defined in Johnson and Lindenstrauss's seminal paper [75]. It is, in fact, no more than a projection Φ from d dimensions onto a randomly chosen k-dimensional subspace. This is useful provided $k < d$, which will be implied by the assumption

$$n = 2^{O(\varepsilon^2 d)} \ . \tag{1.1}$$

Following [75], researchers (Frankl and Maehara [55] and later Dasgupta and Gupta [35]) suggested variants and simplifications of their design and/or proof together with improvements on the constant c. The two papers, though simplifying and improving the original JL result, do not depart from it in the sense that the random projection they assumed is equivalent to the one assumed in [75]. If we represent this projection by a $k \times d$ real matrix Φ, then the rows of the matrix can be computed as follows: The first row is a random unit vector uniformly chosen[1] from S^{d-1} (the $(d-1)$-dimensional unit sphere in $\mathbb{R}^d$). The second row is a random unit vector from the space orthogonal to the first row, the third is a random unit vector from the space orthogonal to the first two rows, and so on. (In order to choose a random unit vector in $\mathbb{R}^m$, one can choose m i.i.d. normally distributed random variables, and normalize the resulting vector to achieve norm 1.) The resulting matrix is a projection onto a random k-dimensional subspace of $\mathbb{R}^n$ (together with a choice of an orthogonal basis, having no affect on the ℓ_2-norm). This choice of Φ satisfies the following three properties (which, in fact, completely characterize it):

- Spherical symmetry: For any orthogonal matrix $A \in O(d)$, ΦA and Φ have the same distribution.
- Orthogonality: The rows of Φ are orthogonal to each other.
- Normality: The rows of Φ are unit-length vectors.

The first to depart from the above distribution on Φ were Indyk and Motwani [74] who dropped the orthogonality and the normality conditions. They noticed that in order to obtain the JL guarantee, one can independently choose every entry of Φ using the distribution $N(0, 1/d)$. The norm of each row of Φ becomes a random variable. Furthermore, the rows are no longer necessarily orthogonal. Normality and orthogonality are satisfied only on expectation. Indeed, the squared ℓ_2 norm of every row of Φ is 1 on expectation, and the inner product of every two rows is 0 on expectation. The independence of the entries in Φ make the proof much simpler. Note that the distribution on Φ remains spherically symmetric.

The next bold and ingenious step was taken by Achlioptas [1] who dropped the spherical symmetry condition. He noticed that the only property of Φ needed for the transformation to work is that $(\Phi_i \cdot x)^2$ is tightly concentrated around mean $1/d$ for all unit vectors x, where Φ_i is the i'th row of Φ. The distribution he proposed is very simple: Choose each entry of Φ uniformly from $\{+d^{-1/2}, -d^{-1/2}\}$ (note that the normality condition is restored). The motivation in [1] was

[1]By this we mean the Haar measure on the sphere.

to make random projections easier to use in practice. Indeed, flipping coins is much easier than choosing gaussian-distributed numbers. More surprisingly, he shows that if the entries of Φ are chosen independently according to the following distribution:

$$\begin{cases} +(d/3)^{-1/2} & \text{with probability } 1/6 \\ 0 & \qquad\qquad\qquad\quad 2/3 \\ -(d/3)^{-1/2} & \qquad\qquad\qquad\quad 1/6 \end{cases},$$

then the JL guarantee holds (note that normality is now dropped). The nice property of this distribution is that it is *relatively sparse*: On expectation, a (2/3)-fraction of Φ is 0. Assuming we want to apply Φ on many points in $\mathbb{R}^d$ in a real-time setting, by keeping a linked list of all the nonzeros of Φ during preprocessing we get a 3-fold speedup in the transformation computation, compared to a naive matrix-vector multiplication.

In all the aforementioned methods, projecting a point requires multiplication by a dense k-by-d matrix; and so mapping each point takes $O(d \log n)$ time (for fixed ε). Is that optimal?

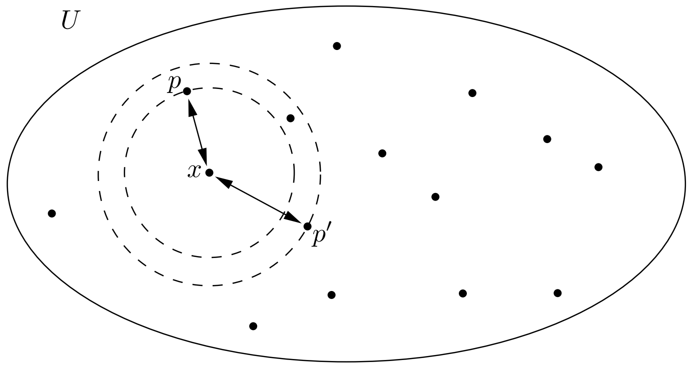
The attempt to sparsify Φ and obtain a super-constant speedup is the first point of departure for this work. A lower bound of Alon [7] dashes any hope of reducing the number of rows of Φ by more than a factor of $O(\log(1/\varepsilon))$, so the obvious question is whether the matrix can be made sparse. Bingham and Mannila [21] considered sparse projections heuristics for dimension-reduction based algorithms, and noticed that in practice they seem to give a considerable speedup with little compromise in quality. Achlioptas's method [1] can be used to reduce the density of Φ by only a constant factor, but one cannot simply go much further because a sparse matrix will typically distort a sparse vector. To prevent this, we use a central concept from harmonic analysis known as the *Heisenberg principle* (so named because it is the key idea behind the Uncertainty Principle): A signal and its spectrum cannot be both concentrated. With this in mind, we precondition the random projection with a Fourier transform (via an FFT) in order to isometrically enlarge the support of any sparse vector. To prevent the inverse effect, i.e., the sparsification of dense vectors, we randomize the Fourier transform. The resulting FJLT shares the low-distortion characteristics of a random projection but with a lower complexity. As stated above, it embeds ℓ_2 into ℓ_p, for $p = 1, 2$. The running time of the FJLT is $O(d \log d + \min\{d\varepsilon^{-2} \log n, \varepsilon^{p-4} \log^{p+1} n\})$, which outperforms the $O(d\varepsilon^{-2}\log n)$ complexity of its predecessors. Note the simpler form of $O(d\log d + \varepsilon^{-3}\log^2 n)$ for ℓ_1 and $n = 2^{O(\varepsilon d)}$ (subsumed by 1.1).

1.2 Approximate nearest neighbor searching

For a fixed metric space (U, d_U) and a finite subset (database) $P \subseteq U$, *ε-approximate nearest neighbor* (ε-ANN) searching (Figure 1.1) is the problem of preprocessing U so that given a query $x \in U$ a point $p \in P$ satisfying

$$d_U(x,p) \leq (1+\varepsilon) d_U(x,p') \ \ \forall p' \in P$$

can be efficiently returned. In other words, we are interested in a point p further from x by a factor of no more than $(1+\varepsilon)$ compared with the closest point to x in P. We will be interested in the Euclidean $(\mathbb{R}^d, \ell_2)$ and the Hamming cube $(\{0,1\}^d, \ell_1 \equiv \ell_2^2)$ cases.



Given a point x from a metric space X, return $p \in P$ minimizing $d(x,p)$, or any point p' satisfying $d(x,p') \leq (1+\varepsilon)d(x,p)$.

Figure 1.1: ε-Approximate nearest neighbor searching

This problem has received considerable attention lately. There are two good reasons for this: (i) ANN boasts more applications than virtually any other geometric problem [72]; (ii) allowing a small error ε makes it possible to break the curse of dimensionality.

There is an abundant literature on (approximate) nearest neighbor searching [12, 13, 20, 24, 29, 32, 33, 52, 68, 69, 72, 74, 80, 81, 90, 109, 110]. The early solutions typically suffered from the curse of dimensionality, but the last decade has witnessed a flurry of new algorithms that "break the curse" (see [72] for a recent survey). A few milestones deserve special mention in the context of this work. Kleinberg [80] gave two algorithms for ANN in ℓ_2^d. The first one runs in $O((d\log^2 d)(d+\log n))$ time but requires storage exponential in d. The second one runs in $O(n+d\log^3 n)$ time, improving on the trivial $O(nd)$ algorithm, and requires only $O(dn \text{ polylog } n)$ space. Both algorithms are based on the key idea of *projection onto random lines*, which was used in subsequent results as well as in this work.

The first algorithms with query times of $\text{poly}(d, \log n)$ and polynomial storage (for fixed ε) were those of Indyk and Motwani [74] and Kushilevitz, Ostrovsky, and Rabani [81]. The first reference describes a reduction from ε-ANN to *approximate point location in equal balls* (ε-PLEB) for ℓ_2^d (as well as other metric spaces). The ε-PLEB problem is that of outputting a point $p \in P$ such that $\|x-p\|_2 \le r$ for some $r > 0$ if such a point exists, and null if all points p satisfy $\|x-p\|_2 > (1+\varepsilon)r$. Based on a new space decomposition (of independent interest), Indyk and Motwani showed how to reduce an ε-ANN query to $O(\log^2 n)$ queries to an ε-PLEB oracle, a reduction later improved to $O(\log(n/\varepsilon))$ by Har-Peled [63]. The PLEB reduction can be thought of as a way of performing a binary search on the (unknown) distance to the nearest neighbor, while overcoming the possible unboundedness of the search space (we overcome this problem by using a different, simpler technique). One approach to PLEB is to use LSH (locality-sensitive hashing [74]), which requires $O(n^{1/(1+\varepsilon)})$ query time and near-quadratic (for small ε) storage. Another approach is to use dimension reduction techniques: using the methods of [63, 70, 74], this provides a query time of $O(\varepsilon^{-2} d \log n)$ with $n^{O(\varepsilon^{-2})}$ storage. We mention here that the dimension reduction overwhelms the running time of the algorithm: to remedy this was, in fact, was another point of departure for our work on FJLT. Kushilevitz et al. [81] described an ingenious (but intricate) reduction from ℓ_2^d to the Hamming cube, which results in $O(\varepsilon^{-2} d^2 \text{polylog}\, n)$ query time and polynomial storage (again, for fixed ε).

ANN searching over the Hamming cube does not suffer from the "unbounded binary search" problem. Kushilevitz et al. [81] gave a random-projection based algorithm with a query time of $O((d\log d)\varepsilon^{-2}\log n)$—an extra $\log\log d$ factor can be shaved off [28]. We improve the running time of their algorithm to $O((d+\varepsilon^{-2}\log n)\log d)$, which is the best to date (using polynomial storage[2]). Again, we show how to optimize the dimension reduction step in their algorithm, but

[2]Assuming constant ε.

this time over GF (2).

We present two new, faster ANN algorithms. Note that both of them contain additional improvements, independent of FJLT.

- One works for the d-dimensional Euclidean space. It stores n points in $\mathbb{R}^d$ and answers any ε-ANN query in $O(d \log d + \varepsilon^{-3} \log^2 n)$ time, while using $n^{O(\varepsilon^{-2})}$ storage. The solution is faster than its predecessors (at least for subexponential n) and considerably simpler.
- The other works for the d-dimensional Hamming cube. It stores n points in the d-dimensional Hamming cube and answers any ε-ANN query in $O((d + \varepsilon^{-2} \log n) \log d)$ time, while using $d^2 n^{O(\varepsilon^{-2})}$ storage. This improves on the best previous query time of $O((d \log d)\varepsilon^{-2} \log n)$ [81].

Chapter 2

The Fast Johnson-Lindenstrauss Transform

The transform (denoted FJLT) is a random distribution of linear mappings from $\mathbb{R}^d$ to $\mathbb{R}^k$, where the embedding dimension k is set to be $c\varepsilon^{-2}\log n$, for some large enough constant $c = c(p)$ Recall that $p \in \{1, 2\}$ refers to the type of embedding we seek: $\ell_2^d \mapsto \ell_p^k$.

We may assume w.l.o.g. that $d = 2^m > k$. We will also assume that $n \geq d$ and $d = \Omega(\varepsilon^{-1/2})$ (otherwise the dimension of the reduced space is linear in the original dimension).

A random embedding $\Phi \sim \text{FJLT}(n, d, \varepsilon, p)$ can be obtained as a product of three real-valued matrices (Figure 2.1): $\Phi = PHD$. The matrices P and D are random and H is deterministic:

- $\boldsymbol{P}$ = k-by-d matrix whose elements are independent mixtures of 0 with an unbiased normal distribution of variance q^{-1}, where

  $$q = \min\left\{\Theta\left(\frac{\varepsilon^{p-2}\log^p n}{d}\right), 1\right\} .$$

  More precisely, $P_{ij} \sim N(0, q^{-1})$ with probability q, and $P_{ij} = 0$ with probability $1 - q$.

- $\boldsymbol{H}$ = d-by-d normalized Walsh-Hadamard matrix:

  $$H_{ij} = d^{-1/2}(-1)^{\langle i-1, j-1\rangle} ,$$

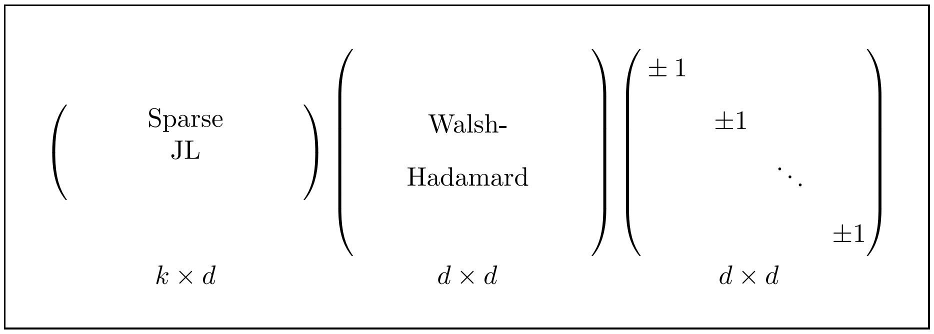


Figure 2.1: The FJLT transform

where $\langle i, j\rangle$ is the dot-product (modulo 2) of the m-bit vectors i, j expressed in binary.

- $\boldsymbol{D}$ = d-by-d diagonal matrix, where each D_{ii} is drawn independently from $\{-1, 1\}$ with probability 1/2.

The Walsh-Hadamard matrix encodes the discrete Fourier transform over the additive group $\mathrm{GF}\,(2)^d$: its FFT is particularly simple and requires $O(d \log d)$ time. It follows that, with high probability (which we make precise in Lemma 2.1), the mapping Φx of *any* vector $x \in \mathbb{R}^d$ can be computed in time $O(d \log d + qd\varepsilon^{-2} \log n)$ (all running times are expected over the random bits of the algorithm). We now make our statement on the FJLT precise.

Lemma 2.1. [The FJLT Lemma] *Fix a set X of n vectors in $\mathbb{R}^d$, $\varepsilon < 1$, and $p \in \{1, 2\}$. Let $\Phi \sim \mathrm{FJLT}$. With probability at least $2/3$, the following two events occur:*

1. *For all $x \in X$,*

$$(1 - \varepsilon)\alpha_p \|x\|_2 \le \|\Phi x\|_p \le (1 + \varepsilon)\alpha_p \|x\|_2 \ ,$$

where $\alpha_1 = k\sqrt{2\pi^{-1}}$ and $\alpha_2 = k$.

2. *The mapping $\Phi : \mathbb{R}^d \to \mathbb{R}^k$ requires*

$$O\left(d \log d + \min\{d\varepsilon^{-2} \log n, \varepsilon^{p-4} \log^{p+1} n\}\right)$$

operations.

Remark: By repeating the construction $O(\log(1/\delta))$ times we can amplify the success probability to $1 - \delta$ for any $\delta > 0$. If we know X, it is possible to test for success. In the ANN example presented in the next chapter, however, we do not know X. In that application, X is the set of differences between the query point x and the database points $p \in P$:

$$X = \{x - p : \ p \in P\} \ .$$

Although P is known, the query point x is information we obtain online after the preprocessing. However, one can amplify the probability of success of ANN by repeating the construction $O(\log(1/\delta))$ times, running the nearest neighbor algorithm w.r.t. each construction and taking the nearest among the outputs.

Proof. Without loss of generality, we can assume that $\varepsilon < \varepsilon_0$ for some suitably small ε_0. Fix some $x \in X$, and define the random variable $u = HDx = (u_1, \ldots, u_d)^T$. Assume w.l.o.g. that $\|x\|_2 = 1$. Note that u_1 is of the form $\sum_{i=1}^d a_i x_i$, where each $a_i = \pm\, d^{-1/2}$ is chosen independently and uniformly. A Chernoff-type argument shows that, with probability at least $1 - 1/20$,

$$\max_{x \in X} \|HDx\|_\infty = O(d^{-1/2}\sqrt{\log n})\,. \tag{2.1}$$

Indeed,

$$\mathbf{E}\,[e^{tdu_1}] = \prod_i \mathbf{E}\,[e^{tda_ix_i}] = \prod_i \cosh(t\sqrt{d}\,x_i) \le e^{t^2 d\|x\|_2^2/2} \ . \tag{2.2}$$

Hence, for any $s > 0$, by Markov's inequality (plugging $t = sd$ in (2.2)),

$$\begin{aligned}\mathbf{Pr}[|u_1| \ge s] &= 2\mathbf{Pr}[e^{sdu_1} \ge e^{s^2 d}] \\ &\le 2\mathbf{E}\,[e^{sdu_1}]/e^{s^2 d} \\ &\le 2e^{s^2 d\|x\|_2^2/2 - s^2 d} \\ &= 2e^{-s^2 d/2} \le 1/(20nd)\end{aligned}$$

for $s = \Theta(d^{-1/2}\sqrt{\log n})$, from which (2.1) follows by a union bound over all $nd \le n^2$ coordinates of the vectors $\{HDx : \ x \in X\}$.

Assume from now on that (2.1) holds, i.e., $\|u\|_\infty \le s$ (where $u = HDx$ for any $x \in X$, which we keep fixed). It is convenient (and harmless) to assume that $m \overset{\text{def}}{=} s^{-2}$ is integral.

Note that $\|u\|_2 = \|x\|_2$ because both H and D are isometries. Define:

$$y = (y_1, \ldots, y_k)^T = Pu = \Phi x \ .$$

By the definition of the FJLT, y_1 is obtained as follows: pick random i.i.d. indicator variables $b_1, \ldots, b_d$, where each b_j equals 1 with probability q, and random i.i.d. variables $r_1, \ldots, r_d$ distributed $N(0, q^{-1})$. Then set $y_1 = \sum_{j=1}^d r_j b_j u_j$. Let $Z = \sum_{j=1}^d b_j u_j^2$. By the 2-stability of the normal distribution, $(y_1 | Z = z) \sim N(0, q^{-1}z)$. Note that all of $y_1, \ldots, y_k$ are i.i.d (given u), and we can similarly define corresponding random i.i.d. variables $Z_1 (= Z), Z_2, \ldots, Z_k$. The expectation of these random variables is:

$$\mathbf{E}\left[Z\right] = \sum_{j=1}^{d} u_j^2 \mathbf{E}\left[b_j\right] = q \tag{2.3}$$

Let u^2 formally denote $(u_1^2, \ldots, u_d^2) \in (\mathbb{R}^+)^d$. By our assumption, u^2 lives in the following d-dimensional polytope:

$$\mathcal{P} = \{ \, (a_1, \ldots, a_d) : \, 0 \le a_j \le m^{-1} \, \forall j \ \text{ and } \sum_{j=1}^{d} a_j = 1 \, \} \ .$$

Let $u^* \in \mathbb{R}^d$ denote a vector such that u^{*2} is a vertex of $\mathcal{P}$. By symmetry, there will be no loss of generality in what follows if we fix:

$$u^* = (\underbrace{m^{-1/2}, \ldots, m^{-1/2}}_{m}, \underbrace{0, \ldots, 0}_{d-m}) \ .$$

The point u^* will be convenient for identifying extremal cases in the analysis of Z. We will use Z^* to denote the random variable Z corresponding to the case $u = u^*$. Immediately we identify that $Z^* \sim m^{-1} B(m, q)$ (in words, the binomial distribution with parameters m, q multiplied by the constant m^{-1}). Consequently,

$$\mathbf{var}(Z^*) = q(1-q)/m \ . \tag{2.4}$$

The ℓ_1 case: We choose

$$q = \min\{1/(\varepsilon m), 1\} = \min\left\{\Theta\left(\frac{\varepsilon^{-1}\log n}{d}\right), 1\right\} .$$

We now bound certain moments of Z (over the random b_i's).

Lemma 2.2. *For any $t > 1$, $\mathbf{E}[Z^t] = O(qt)^t$, and*

$$(1-\varepsilon)\sqrt{q} \le \mathbf{E}[\sqrt{Z}\,] \le \sqrt{q} .$$

Proof. For the case $q = 1$ the claim is trivial because then Z is constant 1. So we assume $q = 1/(\varepsilon m) < 1$.

It is easy to verify that $\mathbf{E}[Z^t]$ is a convex function of u^2, and hence achieves its maximum at a vertex of $\mathcal{P}$, namely, when $u = v$. So it suffices to prove the moment upper-bounds for Z^*, which conveniently behaves like a (scaled) binomial. By standard bounds on the binomial moments,

$$\mathbf{E}[Z^{*t}] = O(m^{-t}(mqt)^t) = O(qt)^t,$$

proving the first part of the lemma.

By Jensen's inequality and (2.3),

$$\mathbf{E}[\sqrt{Z}\,] \le \sqrt{\mathbf{E}[Z]} = \sqrt{q} .$$

This proves the upper-bound side of the second part of the lemma. To prove the lower-bound side, we notice that $\mathbf{E}[\sqrt{Z}\,]$ is a concave function of u^2, and hence achieves its minimum when $u = v$. So it suffices to prove the desired lower bound for $\mathbf{E}[\sqrt{Z^*}\,]$. Since $\sqrt{x} \ge 1 + \frac{1}{2}(x-1) - (x-1)^2$ for all $x \ge 0$,

$$\begin{aligned}\mathbf{E}[\sqrt{Z^*}\,] &= \sqrt{q}\,\mathbf{E}[\sqrt{Z^*/q}\,] \\ &\ge \sqrt{q}\left(1 + \frac{1}{2}\mathbf{E}[Z^*/q - 1] - \mathbf{E}[(Z^*/q-1)^2]\right) .\end{aligned} \tag{2.5}$$

By (2.3) $\mathbf{E}\,[Z^*/q-1\,]=0$ and using (2.4),

$$\begin{aligned}\mathbf{E}\,[(Z^*/q-1)^2\,] &= \mathbf{var}(Z^*/q) = (1-q)/(qm)\\ &\le 1/(qm) = \varepsilon\ .\end{aligned}$$

Plugging this into (2.5) gives $\mathbf{E}\,[\sqrt{Z^*}\,] \ge \sqrt{q}\,(1-\varepsilon)$, as required. $\square$

Since the expectation of the absolute value of $N(0,1)$ is $\sqrt{2\pi^{-1}}$, by taking conditional expectations

$$\mathbf{E}\,[|y_1|] = \sqrt{2/q\pi}\,\mathbf{E}\,[\sqrt{Z}\,]\ ,$$

and by Lemma 2.2,

$$(1-\varepsilon)\sqrt{2\pi^{-1}} \le \mathbf{E}\,|y_1| \le \sqrt{2\pi^{-1}}\,. \tag{2.6}$$

We now show that $\|y\|_1$ is sharply concentrated around its mean $\mathbf{E}\,[\|y\|_1] = k\mathbf{E}\,[|y_1|]$. To do this, first we bound the moments of $|y_1| = |\sum_j b_j r_j u_j|$. For any integer $t \ge 0$, by taking conditional expectation,

$$\mathbf{E}\,[|y_1|^t] = \mathbf{E}\,[(q^{-1}Z)^{t/2}]\,\mathbf{E}\,[|U|^t]\ ,$$

where $U \sim N(0,1)$. It is well known that $\mathbf{E}\,[|U|^t] = O(t)^{t/2}$; therefore, by Lemma 2.2,

$$\mathbf{E}\,[|y_1|^t] = O(t)^t\ .$$

It follows that the moment generating function

$$\begin{aligned}\mathbf{E}\,[e^{\lambda|y_1|}] &= 1+\lambda\mathbf{E}\,[|y_1|] + \sum_{t>1}\mathbf{E}\,[|y_1|^t]\lambda^t/t!\\ &\le 1+\lambda\mathbf{E}\,[|y_1|] + \sum_{t>1} O(t)^t\lambda^t/t!\end{aligned}$$

converges for any $0 \le \lambda < \lambda_0$, where λ_0 is an absolute constant, and

$$\mathbf{E}\,[e^{\lambda|y_1|}] = 1+\lambda\mathbf{E}\,[|y_1|] + O(\lambda^2) = e^{\lambda\mathbf{E}\,[|y_1|]+O(\lambda^2)}\ .$$

By independence,

$$\mathbf{E}\,[e^{\lambda\|y\|_1}] = (\mathbf{E}\,[e^{\lambda|y_1|}])^k = e^{\lambda\mathbf{E}\,[\|y\|_1]+O(\lambda^2 k)} \ .$$

By Markov's inequality and (2.6),

$$\begin{aligned}\mathbf{Pr}[\|y\|_1 \geq (1+\varepsilon)\mathbf{E}\,[\|y\|_1] &\leq \mathbf{E}\,[e^{\lambda\|y\|_1}]/e^{\lambda(1+\varepsilon)\mathbf{E}\,[\|y\|_1]} \\ &\leq e^{-\lambda\varepsilon\mathbf{E}\,[\|y\|_1]+O(\lambda^2 k)} \\ &\leq e^{-\Omega(\varepsilon^2 k)} \ ,\end{aligned}$$

for some $\lambda = \Theta(\varepsilon)$. The constraint $\lambda < \lambda_0$ entails that ε be smaller than same absolute constant. A similar argument leads to a similar lower tail estimate. Our choice of k ensures that, for any $x \in X$, $\|\Phi x\|_1 = \|y\|_1$ deviates from its mean by at most ε with probability at least $1-1/20$. By (2.6), this mean, $k\mathbf{E}\,[|y_1|]$, is itself concentrated (up to a relative error of ε) around $\alpha_1 = k\sqrt{2\pi^{-1}}$; rescaling ε by a constant factor and ensuring (2.1) proves the ℓ_1 claim of the first part of the FJLT lemma.

The ℓ_2 case: We now choose

$$q = \min\left\{\frac{c_1 \log^2 n}{d}, 1\right\} \ ,$$

for a large enough constant c_1.

Lemma 2.3. *With probability at least* $1 - 1/20n$,

1. $q/2 \leq Z_i \leq 2q$ *for all* $i = 1, \ldots, k$, *and*

2. $kq(1-\varepsilon) \leq \sum_{i=1}^k Z_i \leq kq(1+\varepsilon)$.

Proof. If $q = 1$ then Z is the constant q and the claim is trivial. Otherwise, $q = c_1 d^{-1}\log^2 n < 1$. For any real λ, the function

$$f_\lambda(u_1^2, \ldots, u_d^2) = \mathbf{E}\,[e^{\lambda Z}]$$

is convex. Therefore, it achieves its maximum on the vertices of the polytope $\mathcal{P}$ (as in the proof of Lemma 2.2). Hence, as argued before, $\mathbf{E}\,[e^{\lambda Z}] \leq \mathbf{E}\,[e^{\lambda Z^*}]$. We conclude the proof of the first part by using standard tail estimates on the scaled binomial Z^* derived from bounds on its moment generating function $\mathbf{E}\,[e^{\lambda Z^*}]$ (e.g. [9]), and union bounding.

For the second part, let $S = \sum_{i=1}^{k} Z_i$. Again, the moment generating function of S is bounded above by that of $S^* \sim m^{-1}B(mk, q)$ (all Z_i's are distributed as Z^*), for which it is immediate to check the required concentration bound. $\square$

Henceforth we will assume that the premise of Lemma 2.3 holds with respect to all choices of $x \in X$. Using the union bound, this happens with probability at least $1 - 1/20$. For each $i = 1, \ldots, k$ the random variable $y_i^2 q/Z_i$ is distributed χ^2 with 1 degree of freedom. It follows that, conditioned on Z_i, $\mathbf{E}\,[y_i^2] = Z_i/q$ and the moment generating function of y_i^2 is

$$\mathbf{E}\,[e^{\lambda y_i^2}] = (1 - 2\lambda Z_i/q)^{-1/2} \ .$$

Given any $\lambda > 0$ less than some fixed λ_0, and for large enough ξ, the moment generating function converges and equals:

$$\mathbf{E}\,[e^{\lambda y_i^2}] \le e^{\lambda Z_i/q + \xi\lambda^2 (Z_i/q)^2}$$

(we used the fact that $Z_i/q = O(1)$ from the first part of Lemma 2.3). Therefore, by independence,

$$\mathbf{E}\,[e^{\lambda \sum_{i=1}^{k} y_i^2}] \le e^{\lambda \sum_{i=1}^{k} (Z_i/q + \xi\lambda^2 \sum_{i=1}^{k} (Z_i/q)^2)} \ ,$$

and hence

$$\begin{aligned} &\mathbf{Pr}[\sum_{i=1}^{k} y_i^2 > (1+\varepsilon)\sum_{i=1}^{k} Z_i/q] \\ &\qquad = \mathbf{Pr}[e^{\lambda \sum_{i=1}^{k} y_i^2} > e^{(1+\varepsilon)\lambda \sum_{i=1}^{k} Z_i/q}] \\ &\qquad \le \mathbf{E}\,[e^{\lambda \sum_{i=1}^{k} y_i^2}]/e^{(1+\varepsilon)\lambda \sum_{i=1}^{k} Z_i/q} \\ &\qquad \le e^{-\varepsilon\lambda \sum_{i=1}^{k} Z_i/q + \xi\lambda^2 \sum_{i=1}^{k} (Z_i/q)^2} \ . \end{aligned} \tag{2.7}$$

If we plug

$$\lambda = \frac{\varepsilon \sum_{i=1}^{k} (Z_i/q)}{2\xi \sum_{i=1}^{k} (Z_i/q)^2}$$

into (2.7) and assume that ε is smaller than some global ε_0, we avoid convergence problems (we used the assumption $\sum_{i=1}^{k} (Z_i/q)/\sum_{i=1}^{k} (Z_i/q)^2 \le 2$ following the premise of Lemma 2.3). We now conclude that

$$\mathbf{Pr}[\sum_{i=1}^{k} y_i^2 > (1+\varepsilon)k] \le e^{-\Omega(\varepsilon^2 k)} \ .$$

A similar technique can be used to bound the left tail estimate. By choosing $k = c\varepsilon^{-2}\log n$ for some large enough c, using the union bound, and possibly rescaling ε, we conclude the ℓ_2 case of the first part of the FJLT lemma.

Running time: Computing Dx takes $O(d)$ time, because D is a diagonal matrix. Computing $H(Dx)$ takes $O(d \log d)$ time using the Walsh-Hadamard transform. Finally, computing $P(HDx)$ takes $O(|P|)$ time, where $|P|$ is the number of nonzeros in P. This number is distributed $B(nk, q)$. It is now immediate to verify that

$$\mathbf{E}\left[|P|\right] = O(\varepsilon^{p-4} \log^{p+1} n) \ .$$

Using a Markov bound, we conclude the proof for the running time guarantee of the FJLT lemma. This concludes the proof of the FJLT lemma, up to possible rescaling of ε.

□

Chapter 3

ANN Searching in Euclidean Space

We give a new ANN algorithm for ℓ_2^d that differs (and improves on) its predecessors in its use of *handles*: a bootstrapping device that allows for fast binary searching on the distance to the nearest neighbor. Plugging in the FJLT automatically provides a further improvement. Let P be a set of n points in $\mathbb{R}^d$. Given $x \in \mathbb{R}^d$, let $p_{\min}$ be its nearest neighbor in P. Recall that the answer to an ε-ANN for x is any point $p \in P$ such that

$$\|x - p\|_2 \leq (1 + \varepsilon)\|x - p_{\min}\|_2 \ .$$

The algorithm has two stages. First (part I), we compute an answer q to an $O(n)$-ANN query for x. This opens the way for a second stage, where we answer the ε-ANN query for x within the (smaller) set $P_x = P \cap \mathcal{B}_2(q, 2\|x - q\|_2)$, where $\mathcal{B}_2(q, r)$ denotes the ℓ_2-ball of radius r centered at q. The key property of P_x is that it contains $p_{\min}$ and the distance from x to its furthest neighbor in P_x is only $O(n\|x - p_{\min}\|_2)$. This sets the stage for a binary search over a bounded domain (part II). Instead of reducing the problem to an ANN query over Hamming cube (as in [81]), we embed P directly into a low-dimensional ℓ_1-space, which we then discretize. We get a two-fold benefit from our use of handles and of the FJLT. The entire scheme is outlined as pseudocode in Figure 3.3.

```
$O(n)$-ANN

  PREPROCESS($P$)
    set $v \leftarrow$ random direction in $\mathbb{R}^d$
    precompute $v^T p$ for all $p \in P$
    build binary search tree for exact 1-dimensional nearest neighbor
      with respect to $D_v(x,p) = |v^T x - v^T p|$

  QUERY($x$)
    return exact nearest neighbor of $x$ with respect to $D_v$
```

Figure 3.1: Pseudocode for $O(n)$-ANN in Euclidean space

3.1 Part I: Linear-factor approximation

To find an $O(n)$-ANN for query x, we choose a random direction $v \in \mathbb{R}^d$ and return the *exact* nearest neighbor with respect to the pseudometric D_v defined as

$$D_v(x,p) = |v^T x - v^T p| \ .$$

This can be done in $O(d+\log n)$ time by preprocessing all the $(v^T p)$'s (Figure 3.1). As we shall see in Lemma 3.1, this returns an $O(n)$-ANN of x (with respect to the full d-dimensional Euclidean space) with constant probability. By repeating the procedure $O(\log \delta^{-1})$ times (with independently drawn vectors v) and keeping the output point nearest to x, we increase the probability of success to the $O(n)$-ANN query to $1-\delta$ for any arbitrarily small $\delta > 0$.

Let $p_{\min}$ denote the exact nearest neighbor of x in the d-dimensional Euclidean space, and let $p^v_{\min}$ denote the (random) nearest neighbor of x with respect to D_v.

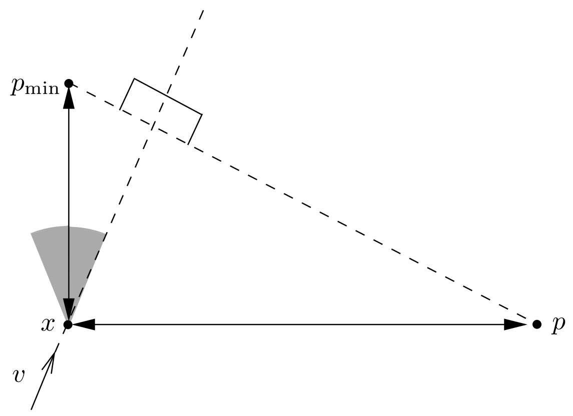
Lemma 3.1.

$$\mathbf{E}\left[\|x - p^v_{\min}\|_2\right] = O(n\|x - p_{\min}\|_2) \ .$$

Proof. Let $\chi(p)$ be the indicator variable of the event $D_v(x,p) \le D_v(x, p^v_{\min})$. Elementary trigonometry (Figure 3.2) shows that

$$\mathbf{E}\left[\chi(p)\right] = O(\|x - p_{\min}\|_2 / \|x - p\|_2) \ .$$

Clearly, $\|x - p^v_{\min}\|_2 \leq \sum_{p \in P} \chi(p)\|x - p\|_2$; therefore, by linearity of expectation,



$\mathbf{E}\,[\chi(p)]$ is maximized when $x - p$ is orthogonal to $x - p_{\min}$. In this case, if the random direction v is in the gray area, $\chi(p) = 1$. This event happens with probability $\frac{2}{\pi}\arctan(\|x - p_{\min}\|_2/\|x - p\|_2) = O(\|x - p_{\min}\|_2/\|x - p\|_2)$.

Figure 3.2: Why the $O(n)$-ANN algorithm works

$$\mathbf{E}\,[\|x - p^v_{\min}\|_2] \leq \sum_{p \in P} \|x - p\|_2\, \mathbf{E}\,[\chi(p)] = O(n\|x - p_{\min}\|_2).$$

□

3.2 Part II: Binary search with handles

Assume that the previous step succeeds in returning an $O(n)$-ANN q for x. From now on, we can confine our search within the working set P_x. Note that there are only $O(n^2)$ distinct sets P_x and, given x, P_x can be found by binary search in $O(d + \log n)$ time. If the diameter of P_x is small enough, say $\Delta(P_x) \leq \frac{1}{2}\varepsilon\|x - q\|_2$, then q is a satisfactory answer to the ε-ANN query for x. By precomputing all diameters, we can test this in constant time and be done if the outcome is positive. So, assume now that $\Delta(P_x) > \frac{1}{2}\varepsilon\|x - q\|_2$. The distance from x to any point of P_x lies in the interval $I(P_x) = [L(P_x), R(P_x)]$, where

$$\begin{aligned} L(P_x) &= \Omega(\Delta(P_x)/n) \\ R(P_x) &= O(\varepsilon^{-1}\Delta(P_x))\ . \end{aligned} \tag{3.1}$$

Each step t in the binary search is associated with two items, l_t and p_t:

1. A *search radius* $l_t \in I$: with high probability, the t-th step in the binary search finds out whether there is a point in P_x at distance at most $(1+\varepsilon)l_t$ to x (*success*) or whether all points are at distance at least l_t to x (*failure*). For initialization, we set l_1 to the high endpoint $R(P_x)$ of the search interval. For $t > 1$, we follow the standard binary search scheme by updating
$$l_t = \begin{cases} l_{t-1} - 2^{-t+1} l_1 & \textit{success} \\ l_{t-1} + 2^{-t+1} l_1 & \textit{failure} \end{cases} .$$
2. A *handle* $p_t \in P_x$ such that $\|x - p_t\|_2 \le 2l_t$. We set $p_1 = q$. In case of *success* in step $t-1$, p_t is updated as the witness point of distance at most $(1+\varepsilon)l_1$ to x. In case of *failure* there is no update: $p_t = p_{t-1}$.

The number $t_{\max}$ of binary search steps is designed to be the smallest integer a such that the search interval size of $\Theta(R(P_x)2^{-a})$ is sensitive enough to detect a relative error of ε with respect to $L(P_x)$. More precisely, we need

$$R(P_x)2^{-t_{\max}} = O(\varepsilon L(P_x)) ,$$

which solves to $t_{\max} = O(\log(n/\varepsilon))$.

Determining whether the step is a *success* (together with a witness) or a *failure* will be done in a transformed space. The transformation will be a composition of two steps. The first step will be a random normalized $(\ell_2 \to \ell_1)$ FJLT matrix Φ applied to the points in P and to x. By the FJLT lemma, with high probability, for any point $p \in P$,

$$(1-\varepsilon)\|x - p\|_2 \le \|\Phi x - \Phi p\|_1 \le (1+\varepsilon)\|x - p\|_2 .$$

As usual, $k = O(\varepsilon^{-2} \log n)$ denotes the embedding dimension.

The second step will be a random discretization of $\mathbb{R}^k$ that will allow table look-up. The discretization could be done be selecting a randomly shifted regular grid in each one of the k dimensions (with carefully chosen cell size). We choose a slightly different discretization that makes the analysis easier.

3.3 Poisson discretization

At step t, for $i = 1, \ldots, k$, consider a random two-sided infinite Poisson process Ψ_i with rate k/l_t. This implies that the number of random points in every interval $[a, a+\tau)$ obeys a Poisson distribution with expectation $\tau k/l_t$. Given $u \in \mathbb{R}^k$, define the quantization $T(u)$ of u to be the k-dimensional integer vector, whose i-th coordinate is the signed count of Poisson events between 0 and u_i, ie,

$$T(u)_i = \begin{cases} |\Psi_i \cap [0, u_i)| & \text{if } u_i \geq 0 \\ -|\Psi_i \cap [u_i, 0)| & \text{otherwise.} \end{cases}$$

We use T to define a pseudometric D_T over $\mathbb{R}^d$:

$$D_T(x, y) = \|T(\Phi x) - T(\Phi y)\|_1 \ .$$

For fixed $x, y \in \mathbb{R}^d$, $D_T(x, y)$ is a Poisson variable with rate $k\|\Phi(x-y)\|_1/l_t$. The point process might fail its purpose, so we say that $p \in P_x$ is *reliable at step t* if either

1. $\|x - p\|_2 \leq l_t$ and $D_T(x, p) \leq (1 + \frac{1}{2}\varepsilon)k$, or
2. $\|x - p\|_2 > l_t$ and $\frac{1}{(1+\frac{1}{2}\varepsilon)} k\|x-p\|_2/l_t \leq D_T(x,p) \leq (1+\frac{1}{2}\varepsilon)k\|x-p\|_2/l_t$.

If all points in P_x are reliable with respect to x and l_t, we say that step t is reliable. By our choice of k, concentration bounds [9] for the Poisson distribution show that all steps in the binary search are reliable.

3.4 A pruned data structure

Step reliability ensures that all the required information for the binary search is contained in the vector $T(\Phi x)$ (up to possible rescaling of ε); and so we can use that discrete vector to index a lookup table $S : \mathbb{Z}^k \to P_x$. The entry $S[v]$ stores a point $p \in P_x$ such that $T(\Phi(p))$ is the nearest ℓ_1-neighbor of v among all the transformed points $\{\, T(\Phi p') : p' \in P_x \,\}$. In particular, $S[T(\Phi x)]$ is the exact nearest neighbor of x in P_x with respect to D_T. For simplicity we defined the lookup table to require infinite memory. We will show how to prune the data structure shortly.

The idea is to navigate through the binary search process at the current step based on $S[T(\Phi(x))] \in P_x$. Let p' denote this point. If $D_T(x, p') \leq (1 + \frac{1}{2}\varepsilon)k$, then we conclude that $\|x - p'\|_2 \leq l_t(1+\varepsilon)$, and the step is a *success* with witness p'. If $D_T(x, p') > (1 + \frac{1}{2}\varepsilon)k$, then we conclude that all points $p \in P_x$ satisfy $\|x - p\|_2 > l_t$, and the step is a *failure*.

We now show how to prune S. We use reliability of step t and the invariant of the binary search handle p_t to assert that

$$D_T(x, p_t) \leq 2k(1 + 2\varepsilon) \; ; \tag{3.2}$$

therefore, only the points x in the D_T-metric ball specified by (3.2) need to be stored (say, in a pruned k-way tree). For fixed p_t and l_t, the number of vectors $T(\Phi x)$ that satisfy (3.2) is exactly the number of integral points $(n_1, \ldots, n_k) \in \mathbb{Z}^k$ such that

$$|n_1| + \cdots + |n_k| \leq 2k(1 + 2\varepsilon) \; .$$

This puts the storage requirement at $\binom{3k}{k-1}2^{O(k)} = 2^{O(k)} = n^{O(\varepsilon^{-2})}$. If we implement the table as a weight-balanced radix tree, the lookup time is only $O(k)$. The complete binary search takes $O(k \log(n/\varepsilon)) = O(\varepsilon^{-2} \log^2(n/\varepsilon))$ time. Obviously, we may assume $\varepsilon > n^{-O(1)}$ (otherwise the naive algorithm is faster), so the binary search time is $O(\varepsilon^{-2} \log^2 n)$.

In preprocessing, we compute discretization maps T and tables S for all possible working sets P_x, search radii l and handles $p \in P_x$. In case (3.2) does not hold, $S[T(\Phi x)]$ will be defined to return the value *exception* (and the search fails).

Theorem 3.2. *Given a set P of n points in ℓ_2^d, for any $\varepsilon > 0$, there is a randomized data structure of size $n^{O(\varepsilon^{-2})}$ that can answer any ε-ANN query in time $O(d \log d + \varepsilon^{-3} \log^2 n)$ with high probability (over the preprocessing).*

□

```
$\varepsilon$-ANN

  PREPROCESS($P$)
    preprocess for $O(n)$-ANN
    set $\Phi \leftarrow$ random $(\ell_2 \rightarrow \ell_1)$ FJLT matrix
    precompute $\Phi p$ for all $p \in P$
    for all $O(n^2)$ possible $P_x$
      for all possible handles $p \in P_x$
        for all possible $O(\log(n/\varepsilon))$ binary search radii $l$
          precompute random discretization $T : \mathbb{R}^k \rightarrow \mathbb{Z}^k$
          precompute table $S : \mathbb{Z}^k \rightarrow P_x \cup \{exception\}$

  QUERY($x$)
    set $q \leftarrow O(n)$-ANN of $x$
    set $P_x \leftarrow \{p \in P : \|p - q\| \leq 2\|x - q\|\}$  ($O(\log n)$ time using preprocessing)
    compute reduction $\Phi x$

    set $l \leftarrow R(P_x)$  (radius)
    set $p \leftarrow q$  (handle)
    for $t = 1, 2, \ldots, t_{\max}(= O(\log(n/\varepsilon)))$  (binary search loop)
      identify precomputed $S, T$ corresponding to $P_x, p, l$
      set $p' \leftarrow S[T(\Phi x)]$
      if $p' = exception$
        break for-loop (with possible warning)
      else if $D_T(x, p') > (1 + \frac{1}{2})k$  (failure)
        set $l \leftarrow l + 2^{-t}$
      else (success)
        set $l \leftarrow l - 2^{-t}$
        set $p \leftarrow p'$

    return $p$
```

At iteration t in the binary search loop, variables p, l in the program correspond to p_t, l_t in Section 3.2.

Figure 3.3: Pseudocode for ε-ANN in Euclidean space

Chapter 4

ANN Searching Over the Hamming Cube

Kushilevitz et al. [81] gave an algorithm for ANN queries over $\{0,1\}^d$. Its bottleneck is the repeated multiplication of the query point by various random matrices. Our improvement is based on the observation that, although most of these matrices are dense, by using some algebra over GF (2), one can decompose them into a sparse part together with a dense part that is of low complexity.

As usual, $P \subseteq \{0,1\}^d$ will denote the fixed database of n points in the d-dimensional Hamming cube. Given a query $x \in \{0,1\}^d$, $p_{\min} \in P$ is the exact nearest neighbor. We let $\mathcal{B}_1(z,r)$ denote the Hamming ball of radius r (inclusive) around z.

The ANN data structure of [81] supports binary search on the unknown distance $D(x, p_{\min}) = \|x - p_{\min}\|_1$, using d separate preprocessed sub-structures, $\mathcal{S}_l$ $(1 \le l \le d)$. Each one of these structures is meant to handle queries whose targeted nearest neighbors are at the distance l. To supply enough randomness so that *every* query succeeds with high probability (over preprocessing), each $\mathcal{S}_l$ itself is a collection of σ similarly built data structures $\mathcal{S}_{l,j}$. For any $j = 1, \ldots, \sigma$, $\mathcal{S}_{l,j}$ consists of:

- a random k-by-d matrix $R^{l,j}$ whose elements are chosen independently in $\{0,1\}$, with the probability of a 1 being $1/2l$;

- a table $T_{l,j} : \{0,1\}^k \to P$, initialized as follows:
  - Set all entries to ∞; then,
  - for each $p \in P$ in turn, set $T_{l,j}[\mathcal{B}_1(z, k(\mu(l) + \frac{1}{3}\varepsilon'))]$ to p, where

$$\begin{aligned} z &\overset{\text{def}}{=} R^{l,j} p \pmod 2 \\ \mu(l) &\overset{\text{def}}{=} \frac{1}{2}(1 - (1 - \frac{1}{2l})^l) \\ \varepsilon' &\overset{\text{def}}{=} \Theta(1 - e^{-\varepsilon/2}) \ . \end{aligned}$$

Lemma 4.1. **[81]** *Assume that $n \geq \log d$, and set $\sigma = cd \log d$ and $k = c\varepsilon^{-2} \log n$, for a large enough constant c. With high probability, the following holds true for any query $x \in \{0,1\}^d$ and any $1 \leq l \leq d$: Given a random $j \in \{1, \ldots, \sigma\}$, with high probability, the point $T_{l,j}[R^{l,j} x]$, if finite, is at distance at most $(1+\varepsilon)l$ from x. Furthermore, if x's nearest neighbor is at distance at most l, then the point in question, indeed, is finite.*

For a random j, we say that a query point x *passes the l-test* if the point $T_{l,j}[R^{l,j}x]$ is finite. (Note that passing is not an intrinsic property of x but a random variable.) The test is called *reliable* if both of the high-probability events in the lemma hold. Assuming reliability, failure of the test means that x's nearest neighbor lies at distance greater than l, while success yields a neighbor of x at distance at most $(1+\varepsilon)l$.

This immediately suggests [81] an ANN algorithm. Beginning with $l = \lceil d/2 \rceil$, run an l-test on x and repeat for $l/2$ if it passes and $3l/2$ if it fails; then, proceed in standard binary search fashion. Suppose for a moment that all the l-tests are reliable. Then, the binary search terminates with the discovery of an index l and a point $p \in P$ that is at most $(1+\varepsilon)l$ away from x, together with the certainty that the distance from x to its nearest neighbor exceeds l. Obviously, the point p is an acceptable answer to the ε-ANN query.

We cannot count on the reliability of every test used in the binary search. But, as in [81], we can overcome this problem by using the fault-tolerant techniques of [53] for computing with unreliable information. Note also that we may assume from now on that $n \geq \log d$: Indeed, having fewer than $\log d$ points gives us a naive (exact) algorithm with $O(d \log d)$ query time. The storage is $d^2 n^{O(\varepsilon^{-2})}$ and the query time is $O(d(\log d)\varepsilon^{-2} \log n)$. To improve this time bound, we seek to exploit the sparsity of the random matrices. That alone cuts down the query time to $O(d\varepsilon^{-2}\log n)$ in a trivial manner, the worst case being a query x that itself belongs to P.

4.1 Improvement using linear algebra

Linear algebra gives room for further improvement. For expository purposes, it is convenient to start the binary search with a d-test, so that the first test in the search is always successful. In general, consider the case where an l-step is to be performed and let l' be the last previous successful test in the search. Note that $l \geq l'/2$. The algorithm is now in possession of a *handle*, namely, a point $p \in P$ at distance at most $(1+\varepsilon)l'$ from x. The cost of the current l-test is that of computing $y = R^{l,j}x$ for a random j.

The main idea is to evaluate y as $R^{l,j}x = R^{l,j}(x+2p) = R^{l,j}(x+p) + R^{l,j}p$ over GF (2). Here is the benefit of this decomposition: The point $x+p$ has at most $(1+\varepsilon)l'$ ones, and obviously only the corresponding columns of $R^{l,j}$ are relevant in computing $R^{l,j}(x+p)$. Assuming that the 1's within each column of $R^{l,j}$ are linked together in a list, the time for computing $R^{l,j}(x+p)$ is proportional to $d+k$ plus the number N of ones within the relevant columns of $R^{l,j}$. The d comes from identifying the 1's of $x+p$, the k comes from initializing the result vector in $\mathrm{GF}(2)^k$ as zero, before scanning through the linked lists of 1's in the relevant columns. By construction, the expected value of N is at most $k(1+\varepsilon)l'(1/2l) \leq 2k$ (over the randomness of the matrix). By precomputing all the points $\{\, R^{l,j}q \mid q \in P \,\}$ in preprocessing (which adds only a factor of n to the storage), we can retrieve $R^{l,j}p$ in $O(k)$ time. In short, we can complete this binary search step in $O(d+k)$ expected time, instead of the previous $O(dk)$ bound.

4.2 No query left behind

There is only one problem: The expectation of the query time is defined over the randomness of *both* the query algorithm and the preprocessing. To remove this dependency on the preprocessing, we must ensure that, for *any* query x, the expected running time of any binary search step is $O(d+k)$ over the random choices of the index j during query answering: We call this the *NQLB policy* (for "no query left behind").

It suffices to show that, for any l and any subset $V \subseteq 2^{\{1,\ldots,d\}}$ of column indices, the total number of ones within all the columns (indexed by V) of all the matrices $R^{l,j}$ $(1 \leq j \leq \sigma)$ is $O(\sigma k|V|/l)$. This number is a random variable $Y = \sum_{1 \leq i \leq \sigma k|V|} y_i$, where each y_i is chosen independently in $\{0,1\}$ with a probability $1/2l$ of being 1. A Chernoff bound shows that $Y = O(\sigma k|V|/l)$ with probability at least $1 - 2^{-\Omega(\sigma k|V|/l)}$. Summing over all l, V, we find that the probability of violating the NQLB policy is at most

$$\sum_{l=1}^{d}\sum_{v=1}^{d}\binom{d}{v}2^{-\Omega(\sigma k v/l)},$$

which is arbitrary small.

Theorem 4.2. *Given a set P of n points in the d-dimensional Hamming cube and any $0 < \varepsilon < 1$, there exists a random data structure of size $d^2 n^{O(\varepsilon^{-2})}$ that can answer any ε-ANN query in time $O((d + \varepsilon^{-2}\log n)\log d)$ in the sense that with high probability over its construction, uniformly for all possible queries x,*

1. *with high probability over the choice of $j \in \{1,\ldots,\sigma\}$ a correct ε-ANN is returned [81], and,*
2. *the expected running time over the choice of $j \in \{1,\ldots,\sigma\}$ is $O((d + \varepsilon^{-2}\log n)\log d)$.*

Chapter 5

Concluding Remarks

Applications and improvements

The FJLT can potentially improve other proximity-related problems such as closest pair, furthest neighbor and clustering. Sarlós [98] recently discovered that the FJLT can be used to improve a result by Drineas et al [42] on fast approximate ℓ_2-regression.

A natural question is whether one can combine the FJLT with Achlioptas's [1] approach of using ± 1 matrices. More precisely, we would like to each each element of the sparse matrix Φ as 0 with probability $1-q$ and uniformly ± 1 (instead of a normal distribution) with probability $1-q$. Matousek [86] recently showed that this is indeed possible without extra cost for the $(\ell_2 \rightarrow \ell_2)$ embedding case and with a multiplicative cost of an additional ε^{-1} for the $(\ell_1 \rightarrow \ell_1)$ case (also affecting the ε-ANN application).

The ANN application presented here suffers from the $n^{O(1/\varepsilon^2)}$-space requirement, an almost insurmountable implementation bottleneck for small ε. It is natural to ask if the space and time could be traded off so that an algorithm with running time $O(\varepsilon^{-2} d \log n)$ (comparable to [63, 74] and [81]) uses significantly less space.

The Kac random walk

We propose an alternative FJLT transform which we conjecture to be at least as good as the one described in this paper, yet much more elegant. This transform is based on the following random walk on the orthogonal group on $\mathbb{R}^{d \times d}$, defined by Kac [76]. At time $t = 0$, the random walk is at the identity matrix: $U_0 = Id$. At time $t > 0$, we choose two random coordinates $1 \leq i_t < j_t \leq d$ and a random angle $\theta_t \in [0, 2\pi)$, and set $U_{t+1} = R_{i_t, j_t, \theta_t} U_t$, where $R_{i,j,\theta}$ is a rotation of the (i, j)-plane by angle θ. Clearly U_t is an orthogonal matrix for all $t \geq 0$. The walk has the Haar measure on the group of orthogonal matrices as its unique stationary distribution. For any fixed $x \in \mathbb{R}^d$, computation of $U_T x$ is extremely efficient: for $t = 1, \ldots, T$ replace x_{i_t} (resp. x_{j_t}) with $x_{i_t} \cos\theta_t + x_{j_t} \sin\theta_t$ (resp. $-x_{i_t} \sin\theta_t + x_{j_t} \cos\theta_t$). The Kac version of FJLT is defined as follows: compute $U_T x$ for all vectors $x \in X \subseteq \mathbb{R}^d$, and return the projection onto the first $O(\varepsilon^{-2} \log |X|)$ coordinates of the resulting vectors. How small can T be in order to ensure the same guarantee as the original JL dimension-reduction technique?

Kac defined this walk in the context of statistical physics in an attempt to simplify and understand Boltzmann's equation. Since then, much attention has been given to it from the viewpoints of pure and applied mathematics. For example, it can be used to efficiently estimate high-dimensional spherical integrals [65]. Its spectral properties are by now well understood [27, 38, 94]. We conjecture that $O(d \log d + poly(\log n, \varepsilon^{-1}))$ steps suffice, and propose this as an interesting problem.

Lower bounds for FJLT

It is natural to ask what is the fastest randomized linear mapping with the Johnson-Lindenstrauss guarantee. More precisely:

Question 5.1. *What is the lower bound on the expected depth of a randomized linear circuit $C_{n,d} : \mathbb{R}^d \mapsto \mathbb{R}^{O(\varepsilon^{-2} \log n)}$ such that given any set $X \subseteq \mathbb{R}^d$ of n vectors, with probability at least $2/3$, $\alpha\|x\|_2(1-\varepsilon) \leq \|C_{n,d}(x)\|_p \leq \alpha\|x\|_2(1+\varepsilon)$ for all $x \in X$, for some $\varepsilon > 0$, $p \in \{1, 2\}$ and α ?*

NQLB for ε-ANN in Euclidean space

Can we achieve a *no query left behind* guarantee for ANN in Euclidean space as we did for the Hamming cube case in Chapter 4? A union bound over the finite number of queries was the main ingredient used for making sure that with high probability, the preprocessing construction worked for all queries simultaneously. An adversary is powerless even if he knew the random bits used in the preprocessing. In the Euclidean space, however, an adversary with access to the preprocessing random bits may be able to choose difficult queries. Perhaps a bounded VC-dimension argument could be used to argue that such an attack is not possible by the adversary.

Bibliography

[1] D. Achlioptas. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *J. Comput. Syst. Sci.*, 66(4):671–687, 2003.

[2] N. Ailon and M. Charikar. Fitting tree metrics: Hierarchical clustering and phylogeny. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2005.

[3] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: ranking and clustering. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 684–693, Baltimore, MD, 2005.

[4] N. Ailon, M. Charikar, and A. Newman. Proofs of conjectures in 'aggregating inconsistent information: ranking and clustering'. *Technical Report, Princeton University*, TR-719-05, 2005.

[5] N. Ailon and B. Chazelle. Lower bounds for linear degeneracy testing. *Journal of the ACM*, 52(2):157–171, 2005.

[6] N. Ailon and B. Chazelle. Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform. In *Proceedings of the 38st Annual Symposium on the Theory of Compututing (STOC)*, pages 557–563, Seattle, WA, 2006.

[7] N. Alon. Problems and results in extremal combinatorics–I. *Discrete Mathematics*, 273(1-3):31–53, 2003.

[8] N. Alon. Ranking tournaments. *SIAM Journal on Discrete Mathematics*, to appear.

[9] N. Alon and J. Spencer. *The probabilistic method.* John Wiley, 2nd edition, 2000.

[10] E. M. Arkin, Y.-J. Chiang, M. Held, J. S. B. Mitchell, V. Sacristan, S. S. Skiena, and T.-C. Yang. On minimum-area hulls. *Algorithmica*, 21:119–136, 1998.

[11] S. Arora, A. Frieze, and H. Kaplan. A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. In *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 24–33, Burlington, VT, 1996.

[12] S. Arya and D. M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 271–280, Austin, Texas, United States, 1993.

[13] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*, 45(6):891–923, 1998.

[14] J. Bang-Jensen and C. Thomassen. A polynomial algorithm for the 2-path problem in semicomplete graphs. *SIAM Journal of Discrete Mathematics*, 5(3):366–376, 1992.

[15] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning Journal (Special Issue on Theoretical Advances in Data Clustering)*, 56(1–3):89–113, 2004. Extended abstract appeared in FOCS 2002, pages 238–247.

[16] G. Barequet and S. Har-Peled. Polygon containment and translational min-Hausdorff-distance between segment sets are 3SUM-hard. *International Journal of Computational Geometry and Applications*, 11:465–474, 2001.

[17] A. H. Barrera. Finding an $o(n^2 \log n)$ algorithm is sometimes hard. In *Proceedings of the 8th Canadian Conference on Computational Geometry*, pages 289–294, Ottawa, Ontario, Canada, 1996.

[18] J. Bartholdi, C. A. Tovey, and M. Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6(2):157–165, 1989.

[19] M. Ben-Or. Lower bounds for algebraic computation trees. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing (STOC)*, pages 80–86, 1983.

[20] M. W. Bern. Approximate closest-point queries in high dimensions. *Infomation Processing Letters*, 45(2):95–99, 1993.

[21] E. Bingham and H. Mannila. Random projection in dimensionality reduction: applications to image and text data. In *Knowledge Discovery and Data Mining*, pages 245–250, 2001.

[22] A. Björner, L. Lovász, and A. Yao. Linear decision trees: volume estimates and topological bounds. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC)*, pages 170–177, Victoria, British Columbia, Canada, 1992.

[23] J. C. Borda. Mémoire sur les élections au scrutin. *Histoire de l'Académie Royale des Sciences*, 1781.

[24] A. Borodin, R. Ostrovsky, and Y. Rabani. Lower bounds for high dimensional nearest neighbor search and related problems. In *Proceedings of the 31st Annual Symposium on the Theory of Compututing (STOC)*, pages 312–321, 1999.

[25] P. Bose, M. J. van Kreveld, and G. T. Toussaint. Filling polyhedral molds. In *Proceedings of the 3rd Workshop on Algorithms and Data Structures (WADS)*, pages 210–221, 1993.

[26] M.-C. Cai, X. Deng, and W. Zang. An approximation algorithm for feedback vertex sets in tournaments. *SIAM Journal on Computing*, 30(6):1993–2007, 2001.

[27] E. A. Carlen, M. C. Carvalho, and M. Loss. Determination of the spectral gap for Kac's master equation and related stochastic evolutions. *http://www.citebase.org/cgi-bin/citations?id=oai:arXiv.org:math-ph/0109003*, 2001.

[28] A. Chakrabarti and O. Regev. An optimal randomised cell probe lower bound for approximate nearest neighbour searching. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 473–482, 2004.

[29] T. M. Chan. Approximate nearest neighbor queries revisited. *Discrete & Computational Geometry*, 20(3):359–373, 1998.

[30] M. Charikar, V. Guruswami, and A. Wirth. Clustering with qualitative information. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 524–533, Boston, 2003.

[31] K. Chaudhuri, K. Chen, R. Mihaescu, and S. Rao. On the tandem duplication-random loss model of genome rearrangement. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 564–570, 2006.

[32] K. L. Clarkson. An algorithm for approximate closest-point queries. In *Proceedings of the 10th Annual ACM Symposium on Computational Geometry (SoCG)*, pages 160–164, 1994.

[33] K. L. Clarkson. Nearest neighbor queries in metric spaces. *Discrete & Computational Geometry*, 22(1):63–93, 1999.

[34] M.-J. Condorcet. Éssai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix. 1785.

[35] S. DasGupta and A. Gupta. An elementary proof of the Johnson-Lindenstrauss lemma. *Technical Report, UC Berkeley*, 99-006, 1999.

[36] M. de Berg, M. M. de Groot, and M. H. Overmars. Perfect binary space partitions. *Computational Geometry: Theory and Applications*, 7(1-2):81–91, 1997.

[37] P. Diaconis and R. Graham. Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society, Series B*, 39(2):262–268, 1977.

[38] P. Diaconis and L. Saloff-Coste. Bounds for Kac's master equation. *Comm. Math. Phys.*, 209(3):729–755, 2000.

[39] M. Dietzfelbinger. Lower bounds for sorting of sums. *Theoretical Computer Science*, 66(2):137–155, 1989.

[40] I. Dinur and S. Safra. On the importance of being biased. In *Proceedings of the 34th Annual Symposium on the Theory of Compututing (STOC)*, pages 33–42, 2002.

[41] D. P. Dobkin and R. J. Lipton. On the complexity of computations under varying sets of primitives. *Journal of Computer and Systems Science*, 18:86–91, 1979.

[42] P. Drineas, M. W. Mahoney, and S. Muthukrishnan. Sampling algorithms for ℓ_2 regression and applications. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Miami, Florida, United States, 2006.

[43] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *Proceedings of the Tenth International Conference on the World Wide Web (WWW10)*, pages 613–622, Hong Kong, 2001.

[44] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation revisited. *Manuscript*, 2001.

[45] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, 1987.

[46] J. Erickson. Lower bounds for linear satisfiability problems. *Chicago Journal of Theoretical Computer Science*, 18, 1999.

[47] J. Erickson. New lower bounds for convex hull problems in odd dimensions. *SIAM Journal on Computing*, 28(4):1198–1214, 1999.

[48] J. Erickson and R. Seidel. Better lower bounds on detecting affine and spherical degeneracies. *Discrete & Computational Geometry*, 18:239–240, 1997.

[49] G. Even, J. S. Naor, M. Sudan, and B. Schieber. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998.

[50] R. Fagin, R. Kumar, and D. Sivakumar. Comparing top k lists. In *Proceedings of the fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 28–36, Baltimore, 2003.

[51] R. Fagin, R. Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 301–312, San Diego, 2003.

[52] M. Farach-Colton and P. Indyk. Approximate nearest neighbor algorithms for Hausdorff metrics via embeddings. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 171–180, 1999.

[53] U. Feige, D. Peleg, P. Raghavan, and E. Upfal. Computing with unreliable information. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC)*, pages 128–137, 1990.

[54] V. Filkov and S. Skiena. Integrating microarray data by consensus clustering. In *Proceedings of International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 418–425, Sacramento, 2003.

[55] P. Frankl and H. Maehara. The Johnson-Lindenstrauss lemma and the sphericity of some graphs. *Journal of Combinatorial Theory Series A*, 44:355–362, 1987.

[56] M. L. Fredman. How good is the information theory bound in sorting? *Theoretical Computer Science*, 1:355–361, 1976.

[57] A. Frieze and R. Kannan. Quick approximations to matrices and applications. *Combinatorica*, 19(2):175–220, 1999.

[58] A. Gajentaan and M. Overmars. On a class of $o(n^2)$ problems in computational geometry. *Computational Geometry: Theory and Applications*, 5:165–185, 1995.

[59] A. Gionis, H. Mannila, and P. Tsaparas. Clustering aggregation. In *Proceedings of the 21st International Conference on Data Engineering (ICDE)*, Tokyo, 2005. To appear.

[60] J. E. Goodman. *Handbook of Discrete and Computational Geometry*. Chapman & Hall/CRC, 2004.

[61] D. Grigoriev, M. Karpinski, F. Meyer auf der Heide, and R. Smolensky. A lower bound for randomized algebraic decision trees. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing (STOC)*, pages 612–619, Philadelphia, Pennsylvania, United States, 1996.

[62] D. Grigoriev, M. Karpinski, and N. Vorobjov. Lower bound on testing membership to a polyhedron by algebraic decision and computation trees. *Discrete & Computational Geometry*, 17(2):191–215, 1997.

[63] S. Har-Peled. A replacement for Voronoi diagrams of near linear size. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 94–103, Las Vegas, Nevada, USA, 2001.

[64] J. Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48:798–859, 2001.

[65] W. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrica*, 57:95–109, 1970.

[66] C. Hoare. Quicksort. *Computer Journal*, 5(1), 1962.

[67] J. Hodge and R.E.Klima. *The Mathematics of Voting and Elections: A Hands-On Approach*, volume 22 of *Mathematical World*. AMS, 2000.

[68] P. Indyk. On approximate nearest neighbors in non-Euclidean spaces. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 148–155, 1998.

[69] P. Indyk. Dimensionality reduction techniques for proximity problems. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 371–378, 2000.

[70] P. Indyk. *High-dimensional computational geometry*. Thesis, Stanford University, 2000.

[71] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 189–197, 2000.

[72] P. Indyk. Nearest neighbors in high-dimensional spaces. In *Handbook of Discrete and Computational Geometry*. CRC Press, 2004.

[73] P. Indyk and J. Matousek. Low-distortion embeddings of finite metric spaces. In *Handbook of Discrete and Computational Geometry*. CRC Press, 2004.

[74] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 604–613, 1998.

[75] W. B. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

[76] M. Kac. *Probability and related topics in physical science.* Wiley Interscience, New York, N.Y., USA, 1959.

[77] R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–104. Plenum Press, New York, 1972.

[78] J. Kemeny and J. Snell. *Mathematical Models in the Social Sciences.* Blaisdell, New York, 1962. Reprinted by MIT Press, Cambridge, 1972.

[79] J. G. Kemeny. Mathematics without numbers. *Daedalus*, 88:571–591, 1959.

[80] J. M. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)*, pages 599–608, 1997.

[81] E. Kushilevitz, R. Ostrovsky, and Y. Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. *SIAM Journal on Computing*, 30(2):457–474, 2000.

[82] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995.

[83] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error Correcting Codes.* North Holland, 1977.

[84] J. Matousek. On geometric optimization with few violated constraints. *Discrete & Computational Geometry,* 14:365–384, 1995.

[85] J. Matousek. *Lectures on Discrete Geometry.* Springer, May 2002.

[86] J. Matousek. On variants of the Johnson-Lindenstrauss lemma. *Private communication*, 2006.

[87] S. Meiser. Point location in arrangements of hyperplanes. *Information and Computation*, 106:286–303, 1993.

[88] F. Meyer auf der Heide. A polynomial linear search algorithm for the n-dimensional knapsack problem. *Journal of the ACM*, 31(3):668–676, 1984.

[89] B. M. Moret, J. Tang, and T. Warnow. *Reconstructing phylogenies from gene-content and gene-order data*, chapter 12 in *Mathematics of evolution and phylogeny.* Clarendon press, Oxford, 2005.

[90] S. Muthukrishnan and S. C. Sahinalp. Simple and practical sequence nearest neighbors with block operations. In *Proceedings of the 13th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 262–278, 2002.

[91] J. Nagura. On the interval containing at least one prime number. In *Proc. Japan Acad. 28*, pages 177–181, 1952.

[92] A. Newman. Approximating the maximum acyclic subgraph. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, June 2000.

[93] A. Newman and S. Vempala. Fences are futile: On relaxations for the linear ordering problem. In *Proceedings of the Eighth Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 333–347, 2001.

[94] I. Pak. Using stopping times to bound mixing times. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 953–954, 1999.

[95] C. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala. Latent semantic indexing: A probabilistic analysis. In *Proceedings of the 17th Annual Symposium of Database Systems*, pages 159–168, 1998.

[96] C. N. Potts. An algorithm for the single machine sequencing problem with precedence constraints. *Mathematical Programming*, 13:78–87, 1980.

[97] W. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971.

[98] T. Sarlós. Improved approximation algorithms for large matrices via random projections. *Private communication*, 2006.

[99] L. Schulman. Clustering for edge-cost minimization. In *Proceedings of the 32nd Annual Symposium on Theory of Computing (STOC)*, pages 547–555, 2000.

[100] P. Seymour. Packing directed circuits fractionally. *Combinatorica*, 15:281–288, 1995.

[101] E. Speckenmeyer. On feedback problems in digraphs. *Graph Theoretic Concepts in Computer Science, Lecture Notes in Computer Science*, 411:218–231, 1989.

[102] J. M. Steele and A. C.-C. Yao. Lower bounds for algebraic decision trees. *Journal of Algorithms*, 3(1):1–8, 1982.

[103] A. Strehl. *Relationship-based Clustering and Cluster Ensembles for High-dimensional Data Mining.* PhD dissertation, University of Texas at Austin, May 2002.

[104] A. van Zuylen. Deterministic approximation algorithms for ranking and clustering problems. *Technical Report, Cornell University*, #1431, 2005.

[105] Y. Wakabayashi. The complexity of computing medians of relations. *Resenhas*, 3(3):323–349, 1998.

[106] A. C.-C. Yao. Algebraic decision trees and Euler characteristics. *Theor. Comput. Sci.*, 141(1&2):133–150, 1995.

[107] A. C.-C. Yao. Decision tree complexity and Betti numbers. *J. Comput. Syst. Sci.*, 55(1):36–43, 1997.

[108] A. C.-C. Yao. Why I'm an optimist. In *DIMACS Workshop on Intrinsic Complexity of Computation*, pages 10–13, apr 2000.

[109] P. N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 311–321, 1993.

[110] P. N. Yianilos. Locally lifting the curse of dimensionality for nearest neighbor search (extended abstract). In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 361–370, 2000.

[111] G. Ziegler. *Lectures on Polytopes (Graduate Texts in Mathematics).* Springer, 2001.