Graph Algorithms for Planning and Partitioning

Shuchi Chawla

CMU-CS-05-184

September 30, 2005

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Avrim Blum, Chair
Anupam Gupta
R. Ravi
Moses Charikar, Princeton University

Submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy

Copyright © 2005 by Shuchi Chawla

This research was sponsored by the National Science Foundation under grant nos. IIS-0121678, CCR-0105488, CCR-0122581, CCR-9732705, CCR-0085982 and through a generous fellowship from the IBM Corporation. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: approximation algorithms, combinatorial optimization, robot navigation, vehicle routing, graph partitioning, hardness of approximation

Abstract

In this thesis, we present new approximation algorithms as well as hardness of approximation results for several planning and partitioning problems. In planning problems, one is typically given a set of locations to visit, along with timing constraints, such as deadlines for visiting them; The goal is to visit a large number of locations as efficiently as possible. We give the first approximation algorithms for problems such as ORIENTEERING, DEADLINES-TSP, and TIME-WINDOWS-TSP, as well as results for planning in stochastic graphs (Markov decision processes). The goal in partitioning problems is to partition a set of objects into clusters while satisfying "split" or "combine" constraints on pairs of objects. We consider three kinds of partitioning problems, viz. CORRELATION-CLUSTERING, SPARSEST-CUT, and MULTICUT. We give approximation algorithms for the first two, and improved hardness of approximation results for SPARSEST-CUT and MULTICUT.

To my grandmothers,
(late) Smt. Sita Devi Chawla and Smt. Satyawati Malhotra

Acknowledgements

I am grateful to my advisor, Prof. Avrim Blum, for being understanding and patient, for being a great source of information and insight, and for giving me the freedom to work at my own pace on topics of my choice. Thanks also to Dr. Cynthia Dwork and Prof. Anupam Gupta for being great mentors and role models. Prof. Moses Charikar and Prof. R. Ravi helped shape a large part of this thesis through numerous discussions; I am grateful to them for their help and advice during the final year of my graduate study. This thesis would not have been possible without the cooperation of my numerous co-authors — thanks to you all!

An enormous thanks to Aditya and Luis for maintaining my sanity (and sometimes making me lose it) during graduate school, and for being the best friends and officemates one could ask for. Finally, I would like to thank my parents and sister for their unconditional love and support through all my endeavors.

Contents

1 Introduction **1**

1.1 Path-planning problems 2

1.1.1 Overview of our results 5

1.1.2 Planning under uncertainty 6

1.2 Graph Partitioning 8

1.2.1 Overview of our results 10

2 Planning with Time Constraints **11**

2.1 Introduction 11

2.1.1 Related work 12

2.1.2 Notation and definitions 14

2.1.3 Summary of results 16

2.2 MIN-EXCESS-PATH 17

2.2.1 A segment partition 18

2.2.2 A dynamic program 20

2.2.3 An improved analysis 21

2.3 ORIENTEERING 22

2.3.1 A 3-approximation 23

2.3.2 Difficulties in obtaining a $2 + \delta$ approximation 24

2.4 Incorporating timing constraints 24

2.4.1 Using ORIENTEERING to approximate DEADLINES-TSP 25

2.4.2 Minimal vertices 26

2.4.3 A partition into sets 27

2.4.4 Finding the best collection 29

2.4.5 From deadlines to time-windows 30

2.5 A bicriteria approximation for the TIME-WINDOWS-TSP 31

2.5.1 The small margin case 32

2.5.2 The large margin case 34

2.5.3 The general case 36

2.6 Extensions 38

2.6.1 Max-Value Tree and Max-Value Cycle 38
2.6.2 Multiple-Path ORIENTEERING 38
2.6.3 ORIENTEERING on special metrics 39
2.7 Hardness of approximation 40
2.8 Concluding remarks 41

3 Planning under Uncertainty **43**
3.1 Introduction 43
3.1.1 Markov decision processes 44
3.1.2 Strategies and policies 44
3.1.3 Objectives 45
3.1.4 Stochastic optimization 47
3.1.5 Our Results 48
3.2 Maximum Discounted-Reward Path 48
3.3 Stochastic TSP 51
3.3.1 The complexity of computing the optimal strategy 52
3.3.2 The adaptivity gap 53
3.3.3 Approximation via static policies 56
3.3.4 Approximation via dynamic strategies 58
3.4 Concluding remarks 59

4 Correlation Clustering **61**
4.1 Introduction 61
4.1.1 Related and subsequent work 63
4.2 Notation and definitions 64
4.3 NP-completeness 65
4.4 A constant factor approximation for minimizing disagreements 66
4.5 A PTAS for maximizing agreements 73
4.5.1 A PTAS doubly-exponential in $1/\epsilon$ 74
4.5.2 A singly-exponential PTAS 75
4.6 Random noise 81
4.7 Extensions 82
4.8 Concluding remarks 85

5 Min-Ratio Partitioning **87**
5.1 Introduction 87
5.1.1 SPARSEST-CUT and low-distortion metric embeddings 88
5.1.2 The Unique Games Conjecture 89
5.2 Our results 90

5.3 Related and subsequent work 92
5.4 An improved approximation for SPARSEST-CUT 94
5.4.1 A high-level overview 94
5.4.2 Notation and definitions 95
5.4.3 Creating decomposition suites 97
5.4.4 Obtaining decomposition bundles: weighting and watching 101
5.4.5 Embedding via decomposition bundles 103
5.5 Hardness of approximating SPARSEST-CUT and MULTICUT 108
5.5.1 Preliminaries 108
5.5.2 Hardness of bicriteria approximation for MULTICUT 109
5.5.3 Hardness of approximating SPARSEST-CUT 114
5.5.4 Hardness of approximating MIN-2CNF≡ DELETION 116
5.5.5 Limitations of our approach 117
5.6 Concluding remarks 117

Bibliography **118**

List of Figures

1.1 Approximation factors and references for a few path-planning problems. Prior results are accompanied by references. 6

2.1 Segment partition of a path in graph G 19
2.2 Algorithm *P2P*— 3-approximation for $s - t$ ORIENTEERING 23
2.3 A bad example for Orienteering. The distance between each consecutive pair of points is 1. 24
2.4 An example illustrating the construction of the rectangles $\mathcal{R}(x, y, z)$. 27
2.5 Algorithm *DTSP*—$3 \log n$-approximation for DEADLINES-TSP 30
2.6 Division of set V_ϵ into segments S_j. Note that segments S_i, S_{i+3} and S_{i+6} are disjoint along the time axis. 32
2.7 Algorithm $\mathcal{A}$—Bicriteria approximation for the small margin case 33
2.8 Algorithm $\mathcal{B}$—Approximation for the large margin case 35
2.9 Algorithm $\mathcal{C}$—Bicriteria approximation for the general case 37

3.1 A $(6.75 + \delta)$-approximation for DISCOUNTED-REWARD-TSP 50
3.2 Exact iterative PSPACE algorithm ITER for STOCHASTIC-TSP 53

4.1 Construction of a triangle packing for Lemma 4.5 68

5.1 Projection and Bucketing 98
5.2 The distribution of projected edge lengths in the proof of Lemma 5.9. If y falls into a light-shaded interval, the pair (x, y) is split. 99

Chapter 1

Introduction

We study two classes of optimization problems on graphs, namely, planning and partitioning, and present improved approximations and hardness of approximation results for them. Broadly, the goal in these problems is to find structure in a set of objects, satisfying some constraints, such that the value of the structure is maximized (or its cost is minimized). In planning problems, the objects are tasks to be performed, and the structure is an ordering of these tasks satisfying scheduling constraints; The goal is to perform a large subset of the tasks in the most efficient manner. In partitioning problems, the structure is a grouping of the objects into classes based on their similarity or closeness.

Planning and partitioning are fundamental combinatorial problems and capture a wide-variety of natural optimization problems; Examples arise in transportation problems, supply chain management, document classification in machine learning, and reconstructing phylogenies in computational genomics. Unfortunately most of these problems are NP-hard. Therefore, a natural approach to solving them is to look for approximate solutions that can be computed in polynomial time.

In this thesis, we study the approximability of several partitioning and planning problems. We provide the first approximation algorithms for some of the problems, and improve upon previously known approximations for others. For some of these problems, we provide hardness of approximation results showing that these problems are NP-hard to approximate better than a certain factor.

We begin with some basic definitions.

Approximation algorithms and hardness of approximation

As mentioned earlier, most of the problems we consider in this thesis are NP-hard. We therefore aim to approximate the objectives in polynomial time. In particular, let $\mathcal{S}$ be the set of all feasible solutions to a problem, and let $f(S)$ denote the objective that is to be maximized. Suppose that the optimal solution is given by $S^* = \text{argmax}_{S\in\mathcal{S}} f(S)$. Then, an α-approximation to the problem is a solution $S \in \mathcal{S}$ such that

$$f(S) \quad \geq \quad \frac{1}{\alpha} f(S^*) \tag{1.1}$$

Likewise, if the problem is a minimization problem, then an α-approximation to the problem is a solution $S \in \mathcal{S}$ such that

$$f(S) \quad \leq \quad \alpha f(S^*) \tag{1.2}$$

Note that in both the cases $\alpha \geq 1$. The smaller the value of α, the better the solution is. An α-approximation algorithm is a polynomial-time algorithm that produces a solution that is an α-approximation to the optimal solution for the problem.

A problem is said to be hard to approximate to within a factor of α, if constructing an α-approximation algorithm for it implies that P=NP.

1.1 Path-planning problems

Consider a repair service that receives requests for repair; Each request may be accompanied with a (monetary) value, a resource(time)-requirement for servicing it, and a time-window within which it may be serviced. The goal of the repair service is to maximize its profit, that is, the total value it accrues by serving a subset of the requests according to their requirements. In order to do so, the service must decide which requests to accept as well as in what order to schedule them in a conflict-free manner. A related problem is that of a delivery-service, that needs to deliver various packages to different locations within their time-windows. Again the goal of the delivery-man is to deliver the packages in the most efficient manner—minimizing his fuel cost, maximizing the number delivered, minimizing the amount of time taken, etc.—while satisfying constraints such as the capacity of the carrying vehicle, or picking the packages from a warehouse before delivering them. These are examples of typical path-planning problems.

Path-planning problems arise in such diverse fields as robotics, assembly analysis, virtual prototyping, pharmaceutical drug design, manufacturing, and computer animation, and have been studied extensively in Operations Research; We review some of this work towards the end of this section. Typically in these problems we are given a number of tasks to perform; Performing a task or switching from one task to another takes time; We are allowed to perform a subset of the tasks and reject the remaining; Our goal is to perform a large number of tasks as efficiently as possible. We model these tasks as locations on a map (graph) that are to be visited. The problems then involve computing a subgraph on the locations (such as a path or sequence of locations) while respecting certain constraints (for example, fixed start and terminal locations). Informally, the objective is to construct a *small subgraph* that contains *many locations* (or a set of locations of high total value).

A simple example is the TRAVELING SALESMAN PROBLEM or the TSP — here one is given a set of locations to be visited with distances between them; the goal is to visit all the locations as fast as possible. This is a fundamental combinatorial problem, and work on it dates as far back as the 1930s [2]. Several approximations are known for the problem in general, as well as for the special cases when the distances between locations form a Euclidean or planar metric [9, 11]. There has also been a large amount of work on developing heuristics for solving the problem nearly optimally (see [2] for a bibliography).

An algorithm for the TSP is indeed a useful tool in many planning applications. Consider, however, a situation where the salesman has a limited amount of time to visit locations and may not be able to visit all of them. A natural goal in that case would be to maximize the number of locations visited (or their total "value") before a certain deadline. This is known as the ORIENTEERING problem. The name "Orienteering" is derived from an outdoor sport where each player is given a map of the terrain and a set of sites to be visited, each amounting to a score; the goal is to accumulate the largest score in a limited amount of time. The ORIENTEERING problem has been studied in approximation algorithms literature for a long time, but unlike the TSP, no approximations were known for this problem with general distance metrics prior to our work. In this dissertation, we present the first approximation, a 3-approximation, to this problem.

Each of the problems we consider explores a trade-off between two objective functions: the *cost* of the constructed subgraph (a function of its length), and the *total value* spanned by it. From the point of view of exact algorithms, we need simply to specify the cost we are willing to tolerate and the value we wish to span. Most of these problems, however, are NP-hard, so we focus on approximation algorithms. We must then specify our willingness to approximate the two distinct objectives. A problem is called a *budget* problem when our goal is to *approximately* maximize the value collected subject to a budget on the cost (such as a fixed upper bound on the total length of the path). Thus ORIENTEERING is a budget problem. Conversely, a *quota* problem is one where our goal is to *approximately* minimize the cost of our solution subject to a fixed quota (lower bound) on the value collected (thus value is a feasibility constraint, while our approximated objective is cost). The TSP is an example of a quota problem, where the quota on value is the total value of all the locations. Another example is the k-TSP. Here our goal is to visit at least k (unspecified) locations and minimize the time taken to do so.

There are several algorithms known for approximating quota problems, the first of these being a 1.5 approximation to the TSP due to Christofides [44]. For the k-TSP, in particular, a series of constant-factor approximations have been developed starting with a 17-approximation due to Blum et al. [32] (where n is the total number of locations), and culminating in the recent 2-approximation due to Garg [69]. In contrast, for budget problems such as ORIENTEERING, where we are given fixed deadlines and other timing constraints on performing the tasks, no approximations were known prior to our work. Our work presents new techniques for handling such fixed timing constraints, leading to the first approximations for budget planning problems, and adding a new tool to the repertoire of the algorithm designer. We demonstrate the use of these techniques in planning problems with multiple salesmen (or delivery-men), as well as for variants where the value collected is a decreasing function of the time taken.

Path-planning problems have also been studied extensively in Operations Research (see, for example, [1]) and are known as *Vehicle Routing* problems in that literature. Several approaches have been studied for solving these problems, including heuristics such as local search, Simulated Annealing and Genetic algorithms, as well as techniques for linear programming that output optimal solutions, such as cutting plane and branch and bound methods [135, 104, 94]. These approaches, however, do not simultaneously give a guarantee on the quality of the solution produced as well as the time taken to find the solution.

A classification of path-planning problems

Depending on the constraints on the solution, or the objective to be optimized, planning problems can be classified according to various features as follows:

Cost-Value trade-off: As described earlier, depending on whether we want to approximate the cost of the solution or the value collected by it, the problem is a quota or budget problem respectively. We also consider a third class of problems, *bicriteria* approximations, where we approximate both the cost as well as the value of the solution, or a combination of the two.

Unrooted, rooted or point-to-point: In a rooted problem, the path (subgraph) must start at a specified vertex, the start. In the point-to-point variant, an end point (terminal) is also specified. No such constraints are placed on the subgraph in unrooted problems. Unrooted problems can be trivially reduced to their rooted counterparts (by trying all possible roots or starts), which can in turn be reduced to the point-to-point version of the problem. The latter is therefore the hardest of the three.

Timing constraints: The cost of a solution is typically defined in terms of the time that it takes to visit various locations. For example, in *min-length* problems, we ask for a solution of minimum length, or place a bound on the maximum length allowed. These can be generalized to *deadline* problems, where different locations have different deadlines for visiting them. In a further generalization, different locations have different *time-windows*, specified by a release time[1] and a deadline, and in order to collect value, the locations must be visited within their respective time-windows. We also consider a new objective for the point-to-point problems, known as *min-excess*. The excess of a path is its length minus the shortest distance between the start and terminal locations. The importance of this objective will be clarified in Chapter 2.

Path or tour: In the tour version of the problem, the objective is to construct a tour that starts and ends at the same vertex.

[1]This terminology is borrowed from scheduling literature, where the release time of a task refers to the time that the task is released and available to work upon.

Single or multiple vehicles: In the multi-vehicle version of these problems, the solution may contain a pre-specified number of paths (tours), with the distance to each vertex measured along the path (tour) on which that vertex lies. In *min-vehicle* problems, we ask for the minimum number of paths (tours) that satisfy timing and value constraints.

Undirected or directed: In undirected problems, the distance function between locations is symmetric—the time taken to go from a location u to another location v is the same as the time taken to go from v to u. In directed problems, the distance function is asymmetric[2].

We use a uniform naming convention for the problems. Each problem is either a min-cost or max-value or a bicriteria problem. We also specify the cost function used—excess, length, deadlines, or time-windows, and the subgraph output.

1.1.1 Overview of our results

The main contributions of our work are as follows:

- The first approximation algorithm, a 3-approximation, for the rooted max-value D-length path or the ORIENTEERING problem in general graphs, solving an open problem of [8, 19].
- The first approximation, a $\min\{3\log^2 n, O(\log D_{\max})\}$ approximation, for the max-value time-windows problem or TIME-WINDOWS-TSP, where the goal is to maximize the total values of locations visited within their time-windows, and $D_{\max}$ is the maximum deadline in the graph.
- A bicriteria approximation for TIME-WINDOWS-TSP—for any constant $c > 0$, if we allow our path to exceed the deadlines of locations by a factor of $(1 + 2^{-c})$, then the path obtains an $O(c)$-approximation to the optimal value.
- A constant-factor approximation for a new problem that we call the DISCOUNTED-REWARD-TSP, where the value at each location is discounted exponentially with time. This is a natural model for searching for a trapped individual and is the most common objective used in MDP problems (see Section 1.1.2).
- Introducing the concept of the *excess* of a path, and developing a constant-factor approximation for the new min-excess k-value path or MIN-EXCESS-PATH problem. This approximation forms a key component in all the algorithms described above.

We summarize our results on deterministic undirected path-planning problems in Figure 1.1 below. Several of our algorithms reflect a series of reductions from one approximation problem to

[2]Asymmetric distance functions arise, for example, when the travel time depends on the traffic in the two directions.

another. We therefore also report reductions between the approximation factors of various problems. Improvements in the approximations for various problems will propagate through. Note that for the MIN-EXCESS-PATH problem, the general reduction only implies an approximation of 2.5. See Section 2.2.3 for the analysis behind the improved approximation factor.

All the approximations reported in Figure 1.1 are for point-to-point, single-vehicle, path problems. The reported results hold also for the rooted, unrooted and tour variants. For the max-value problems, the same algorithms can be modified to obtain approximations for the multi-vehicle variants, increasing the factor of approximation by 1 (that is, an α-approximation becomes an $(\alpha+1)$-approximation).

Problem	Best approx.	Reduction	Hardness
min-length k-value path (k-TSP)	2 [69]	$\alpha_{k\text{TSP}}$	220/219 [118]
min-excess k-value path (MIN-EXCESS-PATH)	$2+\epsilon$	$\alpha_{\text{ex}} = \frac{3}{2}\alpha_{k\text{TSP}} - \frac{1}{2}$	220/219
max-value D-length path (ORIENTEERING)	3	$\alpha_{\text{Orient}} = \lceil \alpha_{\text{ex}} \rceil$	1481/1480
max-value deadlines path (DEADLINES-TSP)	$3\log n$	$\alpha_{\text{DTSP}} = \alpha_{\text{Orient}} \log n$	1481/1480
max-value time-windows path (TIME-WINDOWS-TSP)	$3\log^2 n$	$\alpha_{\text{TW}} = \alpha_{\text{Orient}} \log^2 n$	1481/1480
bicriteria time-windows path	$(1+2^{-c})$ for deadlines $O(c)$ for value		
max-discounted-reward path (DISCOUNTED-REWARD-TSP)	$6.75+\epsilon$	$\alpha_{\text{Disc}} = (1+\alpha_{\text{ex}})(1+\frac{1}{\alpha_{\text{ex}}})^{\alpha_{\text{ex}}}$	

Figure 1.1: Approximation factors and references for a few path-planning problems. Prior results are accompanied by references.

1.1.2 Planning under uncertainty

Often in planning problems, and optimization problems in general, the input to the problem may be revealed over time (instead of being entirely known at the start), or the outcomes of the algorithm's decisions, instead of being fixed, may be governed by random (stochastic) processes. In such a situation, the performance or cost of a solution cannot be predicted precisely beforehand, but is rather a variable depending on the actual turn of events. For example, in the case of a delivery-man, the time taken to go from one location to another may be governed by the traffic on the roads, and therefore cannot be estimated precisely before the delivery-man actually travels between the two locations, and moreover differs every time the delivery-man takes this route. In this case, it would be desirable to obtain a solution that works well most of the time. More precisely, assuming that we can model the process that governs the random aspects of the problem, one goal may be to design an algorithm that optimizes the performance of the solution *in expectation* over the randomness in the input.

We consider some planning problems in this *stochastic optimization* framework in Chapter 3. In particular, we generalize the concept of a graph or map to a *Markov decision process* (MDP). An MDP consists of a set of states (or locations) and actions to move from one location to another. Each action has a probability distribution on locations associated with it; on taking an action, the agent ends up at a location picked from the distribution associated with that action. The actions, therefore, model the uncertainty and randomness in the input to the problem. The goal of a stochastic planning problem is to navigate through the MDP with the same constraints and objectives as in the deterministic case. For example, the goal may be to visit all the locations, or at least k of them, while minimizing the expected time taken to do so.

Planning problems over MDPs arise in the field of robot navigation in AI, and therefore have been studied extensively in that literature. Certain objectives, such as reaching a goal state as quickly as possible, or frequently visiting "good" states while avoiding "bad" states, can be optimized efficiently over MDPs using techniques such as linear programming and dynamic programming. These objectives have the property that the best strategy for the agent (or robot) for optimizing these objectives depends solely on its current location and is independent of its history. In contrast, for a problem such as the TSP on a Markov decision process, the next action taken by the agent may depend on the states that have already been visited in the past, as there is no benefit gained from visiting the same state more than once. In AI terminology, such MDPs are called *time-dependent*, as the "reward" obtained by visiting a state or action changes over time (in the case of the TSP, it disappears after the first visit). This makes the solution space much richer and complex, making the problem harder to optimize. Prior to our work, no approximations were known for optimizing planning problems over these time-dependent MDPs.

In Chapter 3, we study the approximability of the STOCHASTIC-TSP, or the TSP over MDPs. We give an n-approximation to this problem, as well as a weaker $O(n^3 \log n)$-approximation using a "static" solution—one that picks actions independently of the time and history of execution. To our knowledge, these are the first approximations to this problem. We also show that the STOCHASTIC-TSP can be "iteratively-computed" in PSPACE (that is, at any step, the next best action can be computed in PSPACE).

We also consider the well-studied *infinite horizon discounted reward* [91, 120, 131] objective, where the reward at every location decreases exponentially with time, and the goal of the agent is to obtain as much reward as possible. Note that in this problem, there is no time-constraint on the length of the path taken, however, shorter paths face lesser discounting than longer paths, and so this produces what in practice turns out to be good behavior on part of the agent. As for the STOCHASTIC-TSP, we consider the version of the problem in which the reward at a node disappears after the first time the node is visited. In the context of deterministic undirected graphs, we provide the first approximation—a $6.75+\epsilon$ approximation—for this DISCOUNTED-REWARD-TSP problem.

1.2 Graph Partitioning

Breaking a graph into two or more large pieces while minimizing the interface between them, or grouping a set of objects into clusters based on the similarities between them, are classic examples of partitioning problems. Such problems arise in numerous applications, such as data mining, document classification, routing in communication networks, divide-and-conquer algorithms, VLSI layout problems, and image segmentation.

We consider partitioning problems with pairwise constraints. In particular, we are given a set of objects with "join" or "split" constraints on pairs of the objects. The goal is to divide the objects into clusters, such that pairs with a join constraint between them are clustered together, whereas those with a split constraint between them are in different clusters. There may, of course, be instances where it is not possible to produce a partition satisfying all the constraints. In such a case, the partition is allowed to violate some of the constraints while paying a cost.

A variety of partitioning problems can be cast into this framework of partitioning with pairwise constraints, each exploring a different trade-off between violating join or split constraints and displaying a different behavior in their approximability. Consider, for example, the *min-bisection* problem, where the goal is to partition a given weighted graph into two equal-sized components, minimizing the total weight of edges between the two components. In this case, we can place a split constraint between every pair of nodes, and a join constraint between pairs that are connected by an edge. Our goal would be to satisfy at least $n^2/4$ split constraints, while minimizing the number of join constraints violated.

In this thesis, we consider three kinds of partitioning problems with pairwise constraints. In the CORRELATION-CLUSTERING problem, each join and split constraint has a weight associated with it. The cost of any partition is the total weight of the constraints that it violates. This problem is NP-hard even for the case when there is a split or join constraint between each pair of objects, and each weight is 1. From the point of view of approximation, we can ask for a solution that approximately minimizes the total weight of all violated constraints, or one that approximately maximizes the total weight of all satisfied constraints. We present the first constant factor approximations for both these objectives.

A special case of the CORRELATION-CLUSTERING problem is the MULTICUT problem, where each split constraint has an infinite weight. In other words, a feasible partition must satisfy all the split constraints, and the goal is to minimize the total weight of the violated join constraints. It is immediate that the CORRELATION-CLUSTERING problem is at least as hard as MULTICUT, but in fact, it is known [37, 54] that the two problems are equally hard.

A related problem is the SPARSEST-CUT problem. Here the solution is allowed to violate split constraints as well—the goal is to minimize the average cost of violating join constraints for every split constraint satisfied, or simply put, to get the most bang for the buck. More formally, the objective function is given by the total weight of the violated join constraints divided by the total weight of satisfied split constraints.

MULTICUT and SPARSEST-CUT are related to several fundamental problems in graphs including multicommodity flow, expansion, and metric embeddings. Algorithms for the SPARSEST-CUT problem in particular are used as subroutines in approximating several other problems such as min-bisection and min-linear-arrangement, as well as in divide-and-conquer algorithms. Both these problems have been known to be approximable to within $O(\log n)$ factors through linear programming relaxations for over a decade [109, 70, 18, 111] (n is the number of objects in the input). For the MULTICUT problem, it was recently shown that it is not possible to beat the $O(\log n)$ bound via linear programming techniques. Furthermore, the problem is known to be APX-hard, or NP-hard to approximate better than a certain small constant factor. No such hardness result was known for the SPARSEST-CUT problem. We present the first hardness results for the SPARSEST-CUT problem, and improved hardness results for MULTICUT.

On the positive side, we improve upon the decade-old $O(\log n)$ approximation to SPARSEST-CUT obtaining an $O(\log^{3/4} n)$ approximation. Our result also implies an improved low-distortion embedding from negative-type metrics into Manhattan metrics. Low-distortion metric embeddings are an important algorithmic tool in several optimization problems such as graph partitioning and nearest-neighbor problems (see [87] for a survey). The problem of embedding negative-type metrics into Manhattan metrics, in particular, is closely related to the SPARSEST-CUT problem — the best approximation achievable for the latter is exactly equal to the best distortion achievable for the former. Algorithms for embedding general metrics into the Manhattan metric with distortion $O(\log n)$ have been known for almost two decades. It is also known that this is the best possible result for general metrics. Our work is the first to obtain a better distortion for negative-type metrics than that implied by the general result.

The CORRELATION-CLUSTERING problem arises in machine learning applications such as document classification, coreference analysis in natural language processing, and image segmentation, as well as in computational biology applications such as the reconstruction of phylogenies. Our work was the first to study the approximability of this problem. We presented constant factor approximations for this problem, as well as results for a random-noise version of the problem. Our work has been followed up extensively in approximation algorithms literature. We review this follow-up work in Chapter 4.

An interesting feature of the partitioning problems we consider here is that unlike most clustering formulations, we do not need to specify the number of clusters k as a separate parameter. For example, in k-median [36, 88] or min-sum clustering [127] or min-max clustering [85], one can always get a perfect score by putting each object into its own cluster — the question is how well one can do with only k clusters. In our partitioning formulations, the optimal partition may have few or many clusters: it all depends on the join and split constraints, and their weights.

1.2.1 Overview of our results

We consider a "complete-unweighted" graph version of the CORRELATION-CLUSTERING problem—here every pair of objects has either a split or a join constraint between them, and each constraint has a weight of 1. Even in this simpler case, finding the partition with the minimum cost is NP-hard. From the point of view of approximation, two natural objectives arise: minimize the weight of violated constraints (disagreements), or, maximize the weight of satisfied constraints (agreements). We give the first approximation algorithms for both these objectives: a combinatorial constant factor approximation for the min-disagreements objective, and a polynomial time approximation scheme (PTAS) for the max-agreements objective, which, for any constant $\epsilon > 0$, outputs a $(1+\epsilon)$ approximation in $O(e^{O(\frac{1}{\epsilon})}n^2)$ time.

For weighted graphs, we show, via a reduction from the min-multiway cut problem, that the min-disagreements problem is APX-hard. That is, there is some constant $\alpha > 1$, such that it is not possible to approximate the problem to within a factor less than α. On the other hand, for the max-agreements on weighted graphs, we give evidence that the problem is APX-hard, as this would lead to an improved algorithm for the long-open problem of graph coloring. We also give results for the random case, where there is a true optimal clustering, but the function f is a noisy version of this clustering, with some small noise added independently to each edge.

For the SPARSEST-CUT problem, we present an $O(\log^{3/4} n)$ approximation, where n is the number of vertices in the graph. Our result improves upon the previously known $O(\log n)$ approximation for the problem, and is essentially based on an improved low-distortion embedding from negative-type metrics into ℓ_1 metrics. Our result also implies that n-point subsets of the Euclidean metrics (ℓ_2) embed into ℓ_1 with distortion at most $O(\log^{3/4} n)$.

Finally, we present the first hardness of approximation result for the SPARSEST-CUT problem. In particular, we show that assuming the Unique Games conjecture of Khot [101], the SPARSEST-CUT problem is NP-hard to approximate within any constant factor. Assuming a stronger, yet plausible, version of the conjecture, our reduction implies an $\Omega(\sqrt{\log\log n})$ hardness of approximation for SPARSEST-CUT. The result also extends to the MULTICUT problem and the min-disagreements CORRELATION-CLUSTERING problem on weighted graphs.

Subsequent to our work, several improved approximations have been developed, both for CORRELATION-CLUSTERING as well as SPARSEST-CUT. We discuss this work in detail in Chapters 4 and 5.

Chapter 2

Planning with Time Constraints

2.1 Introduction

Consider a FedEx delivery-man with a map of the city that needs to deliver a number of packages to various locations in the city as efficiently as possible. One classic model of such a scenario is the TRAVELING SALESMAN PROBLEM, in which we ask for the tour that visits all the sites and whose length is as short as possible. However, what if the delivery-man cannot deliver everything? For example, he may not be able to make deliveries after 7:00 p.m. In that case, a natural question to ask is for the tour that visits the maximum total reward of sites (where reward might correspond to the value of a package being delivered), subject to a constraint that the total length is at most some given bound D. This is called the (rooted) ORIENTEERING [19, 75] problem ("rooted", because we are fixing the starting location of the robot). Interestingly, while there have been a number of algorithms that given a desired reward can approximately minimize the distance traveled (which yield approximations to the unrooted ORIENTEERING problem), approximating the reward for the case of a *fixed* starting location and *fixed* hard length limit has been an open problem.

More generally, suppose that each location v has a time-window $[R(v), D(v)]$ within which it must be visited. The delivery-man obtains a reward for a location if he visits the location within its time-window. The goal of the delivery-man, as before, is to collect as much reward as possible, while respecting the time-windows. This so-called TIME-WINDOWS-TSP, classically known as the *Vehicle Routing Problem with Time Windows*, has been studied extensively in the Operations Research literature (see [1, 56] for a survey), but no approximations for general graphs were known prior to our work.

We also consider a special version of the TIME-WINDOWS-TSP, in which the release-time $R(v)$ of every vertex is zero. That is, there is no lower limit on when a location must be visited in order to obtain reward, but each location still has a deadline that must be respected. We call this special case the DEADLINES-TSP. The DEADLINES-TSP objective is natural for scheduling-type problems with sites representing jobs to be completed and distances representing job-dependent setup times. This connection is further described in Section 2.1.1.

In this chapter, we provide the first constant-factor approximations to the ORIENTEERING and TIME-WINDOWS-TSP problems, and well as a number of other *path-planning* that we discuss below. We also prove that these problems are APX-hard, or NP-hard to approximate within an arbitrarily small constant factor.

A key contribution of our work is the introduction of the *min-excess* objective. This excess of a path is the length of the path minus the shortest distance between its end-points. Informally, any path must spend a minimum amount of time equal to the distance between the its end-points, just to get to the destination; The excess of the path is the extra time spent by it to gather value along the way. Approximating the excess of a path turns out to be a crucial component in our algorithms for ORIENTEERING and TIME-WINDOWS-TSP, as well as for the DISCOUNTED-REWARD-TSP considered in Chapter 3. We provide a $2 + \epsilon$ approximation for this problem.

2.1.1 Related work

Prize-collecting traveling salesman problems

ORIENTEERING belongs to the family of the prize-collecting traveling salesman problems (PC-TSP) [19, 20, 72, 75]. Given a set of cities with non-negative values associated with them and a table of pairwise distances, a salesman needs to pick a subset of the cities to visit so as to minimize the total distance traveled while maximizing the total amount of value collected. Note that there is a trade-off between the cost of a tour and how much value it spans. The original version of the PC-TSP introduced by Balas [20] deals with these two conflicting objectives by combining them: one seeks a tour that minimizes the *sum* of the total distance traveled and the penalties (values) on cities skipped, while collecting at least a given quota amount of value. Goemans and Williamson subsequently focused on a special case of this problem in which the quota requirement is dropped, and provided a primal-dual 2-approximation algorithm for it [74].

An alternative approach to the bicriteria optimization is to optimize just one of the objectives while enforcing a fixed bound on the other. For example, in a quota-driven version of the PC-TSP, called k-TSP, every node has a value of one unit and the goal is to minimize the total length of the tour, while visiting at least k nodes. Similarly, ORIENTEERING can be viewed as a budget-driven version of the PC-TSP, since we are maximizing total amount of value collected, while keeping the distance traveled below a certain threshold.

There are several constant-factor approximations known for the k-TSP problem [12, 19, 32, 39]—the best being a recent 2-approximation due to Garg [69]—as well as for other quota-driven problems such as Steiner-tree [125], and Min-Latency [30, 39, 71]. Most of these results are based on a classic primal-dual algorithm for the Prize Collecting Steiner Tree problem, due to Goemans and Williamson [74].

The algorithms for k-TSP extend easily to the *unrooted* version of the ORIENTEERING problem in which we do not fix the starting location [19]. In particular, given a tour (cycle) of value Π whose length is cD for some $c > 1$, we can just break the cycle into c pieces of length at most D, and then take the best one, whose total value will be at least Π/c. Noting that an optimal cycle of length $2D$ must span at least as much value as an optimal path of length D (since one could just traverse the path forward and then back), we get a $2c$-approximation guarantee on the amount of value contained in a segment we pick. However, this does not work for the rooted problem because the "best piece" in the above reduction might be far from the start. In contrast, there is no previously known $O(1)$ approximation algorithm for the rooted ORIENTEERING Problem in general graphs. Arkin et al. [8] give a constant-factor approximation to the rooted ORIENTEERING problem for the special case of points in the plane.

Vehicle routing problems with time-windows

Vehicle routing problems have been studied extensively in Operations Research over the past several decades. For the vehicle routing problem with time-windows, various heuristics [55, 126, 134, 133], such as local search, Simulated Annealing and Genetic algorithms, as well as cutting plane and branch and bound methods [135, 104, 94] have been studied for solving this problem optimally. Optimal algorithms for stochastic inputs have also been proposed. In the approximation algorithms literature, there has been work on geometric versions of this problem. Several constant factor approximations [21, 137, 95] have been proposed for the case of points on a line. For general graphs, Chekuri and Kumar [41] give a constant-factor approximation when there are a constant number of different deadlines (or time windows). Our work is the first to give approximation guarantees for the general case with arbitrary deadlines or arbitrary time-windows.

Another generalization of the ORIENTEERING problem, the time-dependent ORIENTEERING problem, deals with a stochastic network topology. In this problem, the lengths of edges in the graph are known functions of time. The version in which the ratio of the maximum length to the minimum length over all edges and times is bounded by a constant, is approximable [65] within a factor of 2. Nothing is known for the general version of the problem.

The Scheduling Perspective: Another motivation for path-planning problems comes from a classic scheduling problem known as *scheduling with sequence dependent setup times* [5, 35, 142, 143]. In this problem we are given several jobs released at time zero, each having a production duration p_j and a delivery date D_j. In addition, for each pair of jobs, there is a setup time s_{ij}, which is incurred when job j is undertaken following job i. The goal is to schedule jobs as efficiently as possible on a single machine. If we assume the s_{ij} are symmetric (e.g., they depend on some notion of similarity between the jobs), then this can be modeled as an undirected graph problem by assigning a length of s_{ij} to the edge between i and j. Production durations can also be incorporated by adding $p_j/2$ to every edge incident on the node j, and subtracting the same from the deadline D_j.

The scheduling setting is fairly general and models many problems. For example, one objective may be to minimize the makespan, which is equivalent to the TRAVELING SALESMAN PROBLEM problem. The objective of minimizing the completion-time of jobs is equivalent to the min-latency problem. Likewise, the objective of completing as many jobs as possible by their delivery dates is equivalent to the DEADLINES-TSP problem. Interestingly, meeting target delivery dates has been declared as the most important scheduling objective for production planners by Wisner and Siferd [142]. Several heuristics [77, 90] have been proposed for this (and other related problems), however, no approximations were known prior to our work.

2.1.2 Notation and definitions

In path-planning problems, we are given a weighted undirected graph $G = (V, E)$ that represents a map of the locations to be visited. Vertices $v \in V$ denote locations, and a length function $\ell : E \to \mathbb{R}^+$ on edges denotes the time taken (or distance traveled) to go from one location to another. In addition, each location is equipped with a *value* or *reward*, $\pi : V \to \mathbb{R}^+$. Let $s \in V$ denote a special node called the *start* or *root*, and $t \in V$ denote the *terminal* vertex.

Our goal is to construct a path in the graph that starts at s, terminates at t, and efficiently visits many locations in between. In particular, each of our problems explore a trade-off between two objective functions: the *cost* of the constructed subgraph (a function of its total length), and the *total value* spanned by it. From the point of view of exact algorithms, we need simply to specify the cost we are willing to tolerate and the value we wish to span. Most of these problem, however, are NP-hard, so we focus on approximation algorithms. We must then specify our willingness to approximate the two distinct objectives.

A problem is called a *quota* problem when our goal is to *approximately* minimize the cost of our solution subject to a fixed quota (lower bound) on the value collected (thus value is a feasibility constraint, while our approximated objective is cost). Conversely, a *budget* problem is one where our goal is to *approximately* maximize the value collected subject to a budget on the cost (such as a fixed upper bound on the total length of the path). We also consider *bicriteria* approximations where we approximate both the cost as well as the value of the solution.

For example, if we define the cost of a solution to be the total length of the edges contained in it, then the corresponding quota problem is the min-length k-value path problem, or the k-TSP: here the goal is to construct a path covering k value and minimize the cost of doing so. Likewise the corresponding budget problem is the max-value D-length path problem or ORIENTEERING, where the goal is to construct a path of length at most D that collects the maximum possible reward. We consider these problems in the first part of this chapter.

In the second part of the chapter, we consider path-planning problems with different timing constraints on visiting different locations. In particular, each vertex in the graph is equipped with a release time $R : V \to \mathbb{R}^+$ and a deadline $D : V \to \mathbb{R}^+$. The goal is to visit a large amount of value within the appropriate time-windows defined by the release times and deadlines. Accordingly, the value collected by a path is the total value of vertices v visited within the time interval $[R(v), D(v)]$. The corresponding budget problem, or the TIME-WINDOWS-TSP problem, aims to maximize the reward collected while obeying all time-windows exactly. In the corresponding quota and bicriteria problems, the cost of the solution is measured by the factor by which the solution exceeds deadlines.

We now introduce some more notation, to be used throughout this chapter. For a path P visiting u before v, let $\ell_P(u, v)$ denote the length along P from u to v. Let $\mathrm{d}(u, v)$ denote the length of the *shortest* path from node u to node v. For ease of notation, we use let $\ell_P(v)$ to denote $\ell_P(s, v)$ and d_v to denote $\mathrm{d}(s, v)$. For a set of nodes $V' \subseteq V$, let $\pi(V') = \sum_{v \in V'} \pi(v)$. For a set of edges $E' \subseteq E$, let $\ell(E') = \sum_{e \in E'} \ell(e)$. We denote the optimal solution to our problems by P^*.

The "min-excess" objective

The main subroutine in our algorithms requires also introducing a variation on the cost function, closely related to the length of the path. Define the *excess* of a path P from u to v, $\epsilon_P(u, v)$, to be $\ell_P(u, v) - \ell(u, v)$, that is, the difference between that path's length and the shortest distance between u and v in the graph. We may now ask for a minimum-excess path that visits at least k value. Obviously, the minimum-excess path of total value k is also the minimum-*length* path of total value k; however, a path of a constant factor times minimum length may have more than a constant-factor times the minimum excess. We therefore consider separately the MIN-EXCESS-PATH problem. Note that an (s, t) path approximating the optimum excess ϵ by a factor α will have length $\mathrm{d}(s, t) + \alpha\epsilon \leq \alpha(\mathrm{d}(s, t) + \epsilon)$ and therefore approximates the minimum cost path by a factor α as well. Achieving a good approximation to this MIN-EXCESS-PATH problem will turn out to be a key ingredient in our approximation algorithms.

Preliminaries

We assume without loss of generality that all nodes are reachable from the start within their respective deadlines. (That is, for all vertices v, the deadline $D(v)$ is larger than the release time $R(v)$ as well as the shortest distance of v from the start, $\mathrm{d}(v)$.)

We also assume that all values are integers in the range $\{1, \ldots, n^2\}$—this allows us to "guess" the value collected by a solution in some of our algorithms, by trying out all possible integer values less than n^3. We can make this assumption by scaling the values down such that the maximum value is exactly n^2 (this guarantees that $\mathcal{O}$ gets at least n^2 reward). We then round each value down to its nearest integer, losing an additive amount of at most n, which is a negligible multiplicative factor. This negligible factor does mean that an approximation algorithm for a max-value problem with guarantee c on polynomially bounded inputs has (weaker) guarantee "arbitrarily close to c" on arbitrary inputs. For the MIN-EXCESS-PATH problem, we do not make this bounded-value assumption. This implies that the running time of our algorithm is linear in the total value in the graph (the minimum value at any location being 1). Note however that in our algorithms for the max-value problems, we may use a bicriteria version of min-excess in which we approximate the value obtained to within a $(1 + O(1/n))$ factor and excess to within an α_{ex} factor. We may then use the bounded value assumption, and our running time is again bounded by a polynomial in n.

Some of our algorithms (for the DEADLINES-TSP and TIME-WINDOWS-TSP problems) use as a subroutine a dynamic program that computes the maximum reward achievable between two specified end points for all possible path lengths. So, strictly speaking, the running time of these algorithms has a polynomial dependence on n and $D_{\max}$. This would be undesirable if $D_{\max}$ is exponential in n. However, we can use the following idea to reduce the running time to polynomial in n and $\log D_{\max}$. For all possible reward values (that are integers between 1 and n^3), we use binary search to find the smallest length for which that reward can be obtained. So we only need to compute and store polynomially many different lengths for each start and end pair. For ease of exposition, we ignore this detail while describing the algorithms.

Although all our algorithms run in polynomial time, the actual running times are fairly large. The MIN-EXCESS-PATH algorithm (Section 2.2.2) requires $O(\Pi n^4)$ applications of the k-TSP subroutine, where Π is the total value in the graph. Algorithm *P2P* for the ORIENTEERING problem (Figure 2.2) requires $O(n^2)$ applications of the algorithm for the MIN-EXCESS-PATH problem. The $O(\log n)$ approximation for DEADLINES-TSP (Figure 2.5) requires $O(n^6)$ applications of Algorithm *P2P*. The bicriteria approximation (Algorithm $\mathcal{C}$, see Figure 2.9) requires $O(n^5 \log \frac{1}{\epsilon})$ applications of the Algorithm *P2P*. Note that these are worst case running times for the algorithms, and the actual running times may be much smaller.

2.1.3 Summary of results

We present the first approximation, a 3-approximation to the ORIENTEERING problem, solving an open problem of [8, 19]. Central to our result is a constant-factor approximation for the MIN-EXCESS-PATH problem defined above, which uses an algorithm for the min-cost k-value path problem as a subroutine.

Using an approximation of $\alpha_{k\text{TSP}}$ for the min-length k-value path problem (k-path problem in [39]), as a subroutine, we get an $\alpha_{\text{ex}} = \frac{3}{2}\alpha_{k\text{TSP}} - \frac{1}{2}$ approximation for the MIN-EXCESS-PATH problem. We use this algorithm for MIN-EXCESS-PATH to obtain a $\lceil \alpha_{\text{ex}} \rceil$ approximation for ORIENTEERING. Our final approximation factors for these problems are $2 + \epsilon$ and 3 respectively, based on an improved analysis of a $(2 + \epsilon)$-approximation for min-length k-value path due to Chaudhuri et al. [39] (see Section 2.2.3).

We develop a $3 \log n$ approximation for the DEADLINES-TSP, based on the 3-approximation that we provide for the ORIENTEERING problem, and further extend this to a $3 \log^2 n$ approximation for TIME-WINDOWS-TSP.

We also give an algorithm for the TIME-WINDOWS-TSP problem, that achieves a bicriteria optimization — it achieves a constant factor approximation to the reward, while exceeding the deadlines by a small factor. In particular, for any given $\epsilon > 0$, suppose that we define $\Pi_P^\epsilon = \sum_{v:R(v)\le \ell_P(v)<(1+\epsilon)D(v)} \Pi(v)$ to be the value collected by a path P if all the deadlines are inflated by a factor of $1+\epsilon$. Then our algorithm collects value $\Pi_P^\epsilon \ge \Omega(\frac{1}{\log(1/\epsilon)})\Pi_{P^*}$. That is, for a constant ϵ, our algorithm obtains a constant fraction of the reward, while exceeding deadlines by a $1+\epsilon$ factor. Note that unlike typical bicriteria results, our approximation is logarithmic in $1/\epsilon$ rather than linear. In particular, this implies an $O(\log D_{\max})$-approximation for the TIME-WINDOWS-TSP problem, where $D_{\max}$ is the maximum deadline in the graph, by taking $\epsilon = 1/D_{\max}$. This gives an asymptotically better approximation than our $O(\log^2 n)$-approximation if all the deadlines in the graph are polynomially bounded in n.

In Section 2.6 we also give constant-factor approximations to several related problems, including the max-value tree problem—the "dual" to the k-MST (min-cost tree) problem—and max-value cycle. Finally, using the APX-hardness of TRAVELING SALESMAN PROBLEM on bounded metrics [60], we prove that the ORIENTEERING, MIN-EXCESS-PATH, DEADLINES-TSP and TIME-WINDOWS-TSP are APX-hard.

Our approximation and inapproximability results are summarized in Figure 1.1 in Chapter 1.

2.2 MIN-EXCESS-PATH

Recall that in the min-excess k-value path problem, we are given a value quota k; Our goal is to construct a path from s to t with value at least k and (approximately) minimum excess. This problem is closely related to the k-TSP, or the min-length k-value problem, where the goal is to construct a path with k value of (approximately) minimum length. The two problems are equivalent in terms of optimal solutions. However, in terms of approximation, min-excess is harder than k-TSP.

In this section, we present a dynamic-programming based algorithm for the min-excess problem, that uses a k-TSP approximation algorithm as a black-box. We present an analysis of our algorithm relating the approximation factor achieved to the approximation factor for k-TSP guaranteed by the black-box algorithm. In particular, an $\alpha_{k\text{TSP}}$ approximation to k-TSP guarantees a $\frac{3}{2}\alpha_{k\text{TSP}} - \frac{1}{2}$ approximation to the min-excess problem via our algorithm. Using a 2-approximation to k-TSP due to Garg [69], this gives us a 2.5-approximation to min-excess.

Towards the end of this section, we present an improved analysis based on a $2+\delta$ approximation to k-TSP due to Chaudhuri et al. [39], that gives a $2+\delta$ approximation to min-excess.

Recall that we use P^* to denote the optimal solution, that is the shortest path from s to t with $\pi(P^*) \ge k$. Let the excess of this path be denoted $\epsilon^* = \epsilon_{P^*}(s,t) = \ell(P^*) - \mathrm{d}(s,t)$. The key idea behind our algorithm for Min-Excess is the observation that two extreme cases of the problem — one when the optimal solution has nearly zero excess, and second when P^* has a very large excess — can be approximated easily using previously known techniques. We describe these extreme cases first.

For the first case, suppose that the optimum solution path encounters all its vertices in increasing order of distance from s. We call such a path *monotonic*. We can find this optimum monotonic path via a simple dynamic program: for each possible value p and for each vertex i in increasing order of distance from s, we compute the minimum excess path that starts at vertex s, ends at i, and collects value at least p. In order to do so, we guess the last vertex visited by the optimal path before i—say it is $j < i$—then the path is given by the previously computed optimal path from s to j that collects value $p - \pi(i)$, followed by the shortest path from j to i. We try all possible values of j, and pick the one that gives a path with the minimum excess. Once this process ends, the solution to the problem is given by the path corresponding to t and collecting value k.

In the second case, suppose that the excess ϵ^* is larger than the distance $\mathrm{d}(s,t)$. Then, an $\alpha_{k\text{TSP}}$-approximation to k-TSP on this instance returns a path with total length $\alpha_{k\text{TSP}}(\mathrm{d}(s,t) + \epsilon^*) \leq \mathrm{d}(s,t) + (2\alpha_{k\text{TSP}} - 1)\epsilon^*$. Therefore, we get a $2\alpha_{k\text{TSP}} - 1$ approximation to excess.

2.2.1 A segment partition

We solve the general case by breaking the optimum path into *segments* that are either monotonic (so can be found optimally as just described) or "wiggly" (generating a large amount of excess). We show that the total length of the wiggly portions is comparable to the excess of the optimum path; our solution uses the optimum monotonic paths and approximates the length of the wiggly portions by a constant factor, yielding an overall increase proportional to the excess.

In order to simplify the following discussion, we assume that no two locations in the graph are at exactly the same distance from the start s. We can ensure this by adding small offsets to the lengths of all the edges in the graph.

Consider the optimal path P^* from s to t. We divide it into segments in the following manner. For any real d, define $f(d)$ as the number of edges on P^* with one end-point at distance less than d from s and the other end-point at distance at least d from s. Note that $f(d) \geq 1$ for all $0 \leq d \leq \mathrm{d}(s,t)$ (it may also be nonzero for some $d \geq \mathrm{d}(s,t)$). Note also that f is piecewise constant, changing only at distances equal to vertex distances. We break the real line into intervals according to f: the *type one intervals* are the maximal intervals on which $f(d) = 1$; the type 2 intervals are the maximal intervals on which $f(d) \geq 2$. These intervals partition the real line (out to the maximum distance reached by the optimum solution) and alternate between types 1 and 2. Let the interval boundaries be labeled $0 = b_1 < b_2 \cdots b_m$, where b_m is the maximum distance of any vertex on the path, so that the i^{th} interval is (b_i, b_{i+1}). Note that each b_i is the distance label for some vertex. Let V_i be the set of vertices whose distance from s falls in the i^{th} interval. Note that the optimum path traverses each set V_i exactly once—one of any two adjacent intervals is of type 1; if the path left this interval and returned to it then $f(d)$ would exceed 1 within the interval. Thus, the vertices of P^* in set V_i form a contiguous *segment* of the optimum path that we label as $S_i = P^* \cap V_i$.

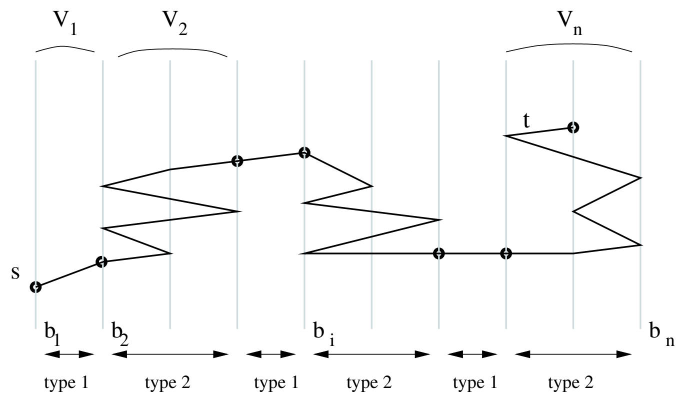


Figure 2.1: Segment partition of a path in graph G

A segment partition is shown in Figure 2.1.

Note that for each i, there may be (at most) one edge crossing from V_i to V_{i+1}. To simplify the next two lemmas, let us split that edge into two with a vertex at distance b_i from s, so that every edge is completely contained in one of the segments (this can be done since one end-point of the edge has distance exceeding b_i and the other end-point has distance less than b_i).

Lemma 2.1. *A segment S_i of type 1 has length at least $b_{i+1} - b_i$. A segment S_i of type 2 has length at least $3(b_{i+1} - b_i)$, unless it is the segment containing t in which case it has length at least $3(\mathrm{d}(t) - b_i)$.*

Proof. The length of segment S_i is lower bounded by the integral of $f(d)$ over the i^{th} interval. In a type 1 interval the result is immediate. For a type 2 interval, note that $f(d) \geq 1$ actually implies that $f(d) \geq 3$ by a parity argument—if the path crosses distance d twice only, it must end up at distance less than d. □

Corollary 2.2. *The total length of type-2 segments is at most $3\epsilon^*/2$.*

Proof. Let ℓ_i denote the length of segment i. We know that the length of P^* is $\mathrm{d}(t) + \epsilon^* = \sum \ell_i$. At the same time, we can write

$$\mathrm{d}(t) \leq b_m = \sum_{i=1}^{m-1}(b_{i+1} - b_i) \leq \sum_{i \text{ type } 1} \ell_i + \sum_{i \text{ type } 2} \ell_i/3$$

It follows that

$$\epsilon^* = \sum \ell_i - \mathrm{d}(t) \geq \sum_{i \text{ type } 2} 2\ell_i/3$$

Multiplying both sides by $3/2$ completes the proof. □

Having completed this analysis, we note that the corollary remains true even if we do not introduce extra vertices on edges crossing interval boundaries. The crossing edges are considered to belong to the type 1 segments that they cross, but this only decreases the total length of type 2 segments.

2.2.2 A dynamic program

Our algorithm computes, for each interval that might be an interval of the optimum solution, a segment corresponding to the optimum solution in that interval. It then uses a dynamic program to paste these fragments together using (and paying for) edges that cross between segments. The segments we compute are defined by 4 vertices: the closest-to-s and farthest-from-s vertices, c and f, in the interval (which define the start- and end-points of the interval: our computation is limited to vertices within that interval), and the first and last vertices, x and y, on the segment within that interval. They are also defined by the amount p of value we are required to collect within the segment. There are therefore $O(\Pi n^4)$ distinct segments to compute, where Π is the total value in the graph.

For each segment we find a nearly optimal type 1 and type 2 solution. In order to compute a type-1 solution, we compute an optimum (shortest) monotonic path from x to y that collects value p. To compute an approximately optimal type-2 solution, we use the k-TSP routine that approximates to within a constant factor the minimum length of a path that starts at x, finishes at y, stays within the boundaries of the interval defined by c and f, and collects value at least p.

Given the optimal type 1 and near-optimal type-2 segments determined for each set of 4 vertices and value, we can find the optimal way to paste some subset of them together, obtaining k value and using minimum length, using a dynamic program. Note that the segments corresponding to P^* are considered in this dynamic program, so our solution will be at least as good as the one we get by using the segments corresponding to the ones on the optimum path (i.e., using the optimum type-1 segments and using the approximately optimum type-2 segments). Therefore, we need only show that approximating the segments of P^* returns a path with small excess.

We now focus on the segments corresponding to the optimum path P^*. Consider the segments S_i of length ℓ_i on the optimum path. If S_i is of type 1, our algorithm will find a (monotonic) segment with the same end-points collecting the same amount of value of no greater length. If S_i is of type 2, our algorithm (through its use of subroutine k-TSP) will find a path with the same end-points collecting the same value over length at most $\alpha_{k\text{TSP}}\ell_i$. Let L_1 denote the total length of the optimum type 1 segments, together with the lengths of the edges used to connect between segments. Let L_2 denote the total length of the optimum type 2 segments. Recall that $L_1 + L_2 = \mathrm{d}(t) + \epsilon^*$ and that (by Corollary 2.2) $L_2 \leq \frac{3}{2}\epsilon^*$. By concatenating the optimum type-1 segments and the approximately optimum type-2 segments, the dynamic program can (and therefore will) find a path collecting the same total value as P^* of total length at most

$$\begin{aligned} L_1 + \alpha_{k\mathrm{TSP}} L_2 &= L_1 + L_2 + (\alpha_{k\mathrm{TSP}} - 1)L_2 \\ &\le \mathrm{d}(t) + \epsilon^* + (\alpha_{k\mathrm{TSP}} - 1)\left(\frac{3}{2}\epsilon^*\right) \\ &= \mathrm{d}(t) + \left(\frac{3}{2}\alpha_{k\mathrm{TSP}} - \frac{1}{2}\right)\epsilon^*. \end{aligned}$$

In other words, we approximate the minimum excess to within a factor of $\frac{3}{2}\alpha_{k\mathrm{TSP}} - \frac{1}{2}$. Using the 2-approximation to k-TSP due to Garg [69], we obtain an approximation ration of 2.5.

2.2.3 An improved analysis

Our approximation guarantee for min-excess path derived above is based on treating the k-TSP subroutine as a “black-box”. In this section we show how to slightly improve our approximation guarantee for min-excess path problem by exploiting the details of the min-cost path algorithm derived from the work of Chaudhuri et al. [39].

We begin with a brief description of the Chaudhuri et. al algorithm. In their paper, Chaudhuri et al. provide a subroutine for constructing a tree containing nodes s and t that spans at least k vertices[1] and has cost at most $(1 + \frac{1}{2}\delta)$ times the cost of the shortest s-t path with k vertices, for any fixed constant δ. To construct an s-t path from the tree obtained by the algorithm of Chaudhuri et al., we can double all the edges, except those along the tree path from s to t. This gives us a partial “Euler tour” of the tree that starts at s and ends at t. Clearly, the cost of such a path is at most $(2 + \delta)$ times the cost of the shortest s-t path spanning value k, for any fixed constant δ, and is therefore a $(2 + \delta)$-approximation to k-TSP.

In fact, if the optimal path P^* has length $\mathrm{d}(t) + \epsilon^*$, then the tree has length at most $(1 + \frac{1}{2}\delta)(\mathrm{d}(t) + \epsilon^*)$. We convert this tree into a path from s to t by doubling all edges, except for the ones on the tree path from s to t. Noting that the total cost of “non-doubled” edges is $\mathrm{d}(s,t)$, we get a path from s to t of length at most $(2 + \delta)(\mathrm{d}(t) + \epsilon^*) - \mathrm{d}(t) = (1 + \delta)\mathrm{d}(t) + (2 + \delta)\epsilon^*$. This stronger guarantee gives us an improved guarantee on the performance of the min-excess algorithm described above.

In particular, suppose that we apply the k-TSP algorithm to a segment of type-2 with end points x and y, and having an optimum min-excess path with length $\ell = \mathrm{d}(x,y) + \epsilon$. Then we get a path from x to y with the same value and length at most $(1 + \delta)\ell + \epsilon$, for any fixed small constant δ.

Now we can apply Corollary 2.2 to obtain an approximation in terms of the excess. However, note that the ϵ in the above expression is the excess of the path between the nodes u and v, and is not the same as the difference of the excess of the path P^* at v and its excess at u. In order to obtain

[1]The algorithm can be transformed easily to obtain a tree spanning a given target *value*—to each node v with a value π_v, we attach $\pi_v - 1$ leaves with zero-length edges, and run the algorithm on this new graph.

a bound in terms of the latter, let ϵ_u denote the excess of P^* from s to u, and ϵ_v the excess of P^* from s to v. Then, $\ell = (\mathrm{d}(v) + \epsilon_v) - (\mathrm{d}(u) + \epsilon_u) \leq \mathrm{d}(u, v) + \epsilon_v - \epsilon_u$. Therefore, $\epsilon \leq \epsilon_v - \epsilon_u$, and the Chaudhuri et al. algorithm returns a path of length at most $(1 + \delta)\ell + \epsilon_v - \epsilon_u$.

The dynamic program finds a path collecting the same total value as P^* and of total length at most

$$
\begin{aligned}
L_1 + (1+\delta)L_2 + \sum_{\text{type 2 segments}} (\epsilon_v - \epsilon_u) \quad &\leq \quad L_1 + (1+\delta)L_2 + \epsilon^* \\
&= \quad \mathrm{d}(t) + 2\epsilon^* + \delta L_2 \\
&\leq \quad \mathrm{d}(t) + 2\epsilon^* + \frac{3\delta}{2}\,\epsilon^*
\end{aligned}
$$

where the last statement follows from Corollary 2.2. Therefore, we get an approximation ratio of $2 + \delta'$ for the min-excess problem, for any small constant δ'.

2.3 ORIENTEERING

In the max-value D-length path problem, a.k.a. ORIENTEERING, we are given a deadline D and the goal is to find a path of length at most D that obtains the maximum possible value.

In this section we present the first constant-factor approximation for this problem in general metrics, based on the algorithm for min-excess developed in the previous section. The min-excess algorithm on any instance returns a path with the same amount of value as the optimal path, but has a slightly larger length. The main idea behind our ORIENTEERING approximation is to divide the optimal path into segments, such that we can short-cut some of these segments in order to gain time, and utilize this extra time to approximate excess on the remaining segments. Using this idea, an α_{ex} approximation to min-excess gives us a $\lceil \alpha_{\text{ex}} \rceil$ approximation to the ORIENTEERING problem. In particular, using the $(2 + \delta)$-approximation for min-excess from the previous section or the 2.5-approximation from Section 2.2.2, we achieve a 3-approximation to ORIENTEERING.

We begin with some properties of excess. Let P be any $u - v$ path that visits nodes in the order $u_0 = u, u_1, \ldots, u_l = v$. Recall that $\epsilon_P(u_i, u_j)$ denotes the excess of path P from u_i to u_j — $\epsilon_P(u_i, u_j) = \ell_P(u_i, u_j) - \mathrm{d}(u_i, u_j)$.

Fact 1 $\epsilon_P(u_0, u_i)$ is increasing in i.

Fact 2 The excess function is sub-additive. That is, for any nodes u_i, u_j, u_k with $0 \leq i < j < k \leq l$, $\epsilon_P(u_i, u_j) + \epsilon_P(u_j, u_k) \leq \epsilon_P(u_i, u_k)$.

Proof. The first fact follows from the triangle inequality. The second follows by rewriting $\epsilon_P(u_i, u_j) + \epsilon_P(u_j, u_k)$ as $\ell_P(u_i, u_j) - \mathrm{d}(u_i, u_j) + \ell_P(u_j, u_k) - \mathrm{d}(u_j, u_k) = \ell_P(u_i, u_k) - \mathrm{d}(u_i, u_j) - \mathrm{d}(u_j, u_k)$ and observing that $\mathrm{d}(u_i, u_j) + \mathrm{d}(u_j, u_k) \geq \mathrm{d}(u_i, u_k)$ by the triangle inequality. $\square$

2.3.1 A 3-approximation

We now present our 3-approximation for ORIENTEERING. The algorithm *P2P* is given in Figure 2.2. Informally, the algorithm proceeds by guessing two points on the optimal path. It then approximates the three resulting subpaths using the MIN-EXCESS-PATH algorithm while respecting the deadline D, and picks the one that obtains the most value. Note that by design, the algorithm produces a path of length at most D. We just need to show that it visits enough points.

Input: Graph $G = (V, E)$; special nodes s and t; length bound D.
Output: Path P from s to t with $\pi(P) \geq \frac{1}{3}\pi(P^*)$ and length at most D.

---

1. For every pair of nodes u and v, we will consider a path which proceeds from s to u, then visits some vertices while traveling from u to v, and then travels directly from v to t. The allowed excess of the path from u to v is $\epsilon(u, v) = D - \mathrm{d}(s, u) - \mathrm{d}(u, v) - \mathrm{d}(v, t)$. Using the $2 + \delta$-approximation to the minimum excess problem from Section 2.2, or the 2.5-approximation from Section 2.2.2, we compute a path from u to v of excess at most $\epsilon(u, v)$ that visits at least as many vertices as the best path from u to v with excess $\epsilon(u, v)/3$.
2. We now pick the pair (u, v) that maximizes the total reward collected on the computed path. We then travel from s to u via a straight line, then u to v via the chosen path, then v to t.

Figure 2.2: Algorithm *P2P*— 3-approximation for $s - t$ ORIENTEERING

Theorem 2.3. *The algorithm* P2P *is a 3-approximation to the point-to-point orienteering problem.*

Proof. Consider the optimum path P^* from s to t. Break this path into three pieces, each having at least $1/3$ of the total value value (including their end-points), and let u and v be the end-points of the portion such that $\epsilon_{P^*}(u, v)$ is smallest. Consider now the path O' that travels directly from s to u, then follows P^* to v, and then travels directly from v to t. By traveling along O' rather than P^*, we save a total of at least $\epsilon_{P^*}(s, u) + \epsilon_{P^*}(v, t)$. Now using Fact 2, and because we chose u, v such that $\epsilon_{P^*}(u, v)$ is the smallest of the three segments, we conclude that we save a total of at least $2\epsilon_{P^*}(u, v)$; that is, $\ell_{P^*}(s, t) - \ell_{O'}(s, t) \geq 2\epsilon_{P^*}(u, v)$. This is enough to pay for the added length produced by applying the min-excess approximation from u to v. Formally, we have $\ell_{O'}(s, t) = \mathrm{d}(s, u) + \mathrm{d}(u, v) + \epsilon_{P^*}(u, v) + \mathrm{d}(v, t)$, and by definition, $\epsilon(u, v) = D - \mathrm{d}(s, u) - \mathrm{d}(u, v) - \mathrm{d}(v, t) \geq \ell_{P^*}(s, t) - \ell_{O'}(s, t) + \epsilon_{P^*}(u, v)$. Plugging in the above inequality, we get that $\epsilon(u, v) \geq 3\epsilon_{P^*}(u, v)$. Since the min-excess algorithm gets at least as much value as the best possible path from u to v with excess $\epsilon(u, v)/3$, it is guaranteed to get at least as much as this portion of the optimal path, which is at least $1/3$ of the total value of the optimal path. $\square$

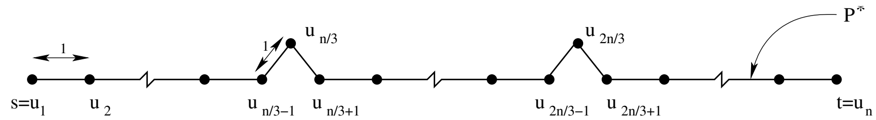


Figure 2.3: A bad example for Orienteering. The distance between each consecutive pair of points is 1**.**

2.3.2 Difficulties in obtaining a $2+\delta$ approximation

We used a $2+\delta$ approximation to the min-excess problem in order to obtain a 3-approximation to ORIENTEERING. The key idea behind obtaining this approximation was to show that for any integer r, there exist vertices u and v on P^*, such that

- the segment of P^* between u and v contains at least a $\frac{1}{r}$ fraction of the total value, and,
- the excess of P^* between u and v is at most a $\frac{1}{r}$ fraction of the excesses $\epsilon_{P^*}(s,u)+\epsilon_{P^*}(u,v)+\epsilon_{P^*}(v,t)$

A natural approach to improving the approximation factor achieved by our algorithm would be to show that the above statement holds for all real numbers r, and in particular for $r = 2+\delta$. Below we show that there are graphs for which this does not hold, and therefore our analysis cannot be improved in this direction.

Note that this statement does not preclude the possibility of obtaining a better approximation factor for the problem through a different analysis. For example, it may be possible to show that a different path between u and v, rather than a segment of P^*, has low excess and large reward.

Lemma 2.4. *For every $\delta > 0$, there is a graph $G(\delta)$ containing a path P^*, such that for every pair of points u and v with the property that P^* collects a $\frac{1}{2+\delta}$ fraction of its value between u and v, the excess $\epsilon_{P^*}(u,v) \geq \frac{1}{2}\epsilon_{P^*}(s,t) > \frac{1}{2+\delta}(\epsilon_{P^*}(s,u)+\epsilon_{P^*}(u,v)+\epsilon_{P^*}(v,t))$.*

Proof. The path P^* consists of $n = 3\left\lceil \frac{2}{\delta}+1 \right\rceil$ vertices $s = u_1 \ldots u_n = t$ (see Figure 2.3). The distance between u_i and u_{i+1} is 1 for all i. The distance between $u_{n/3-1}$ and $u_{n/3+1}$ is also 1, and between $u_{2n/3-1}$ and $u_{2n/3+1}$ is 1.

Note that the length of the path P^* is $n-1$, while $\mathrm{d}(s,t) = n-3$. So P^* has a total excess of 2. Any segment of P^* containing at least a $\frac{1}{2+\delta}$ fraction of its value contains either the nodes $u_{n/3-1}$, $u_{n/3}$ and $u_{n/3+1}$, or the nodes $u_{n/3-1}$, $u_{n/3}$ and $u_{n/3+1}$. It therefore has excess at least 1, and the lemma holds. ☐

2.4 Incorporating timing constraints

We now move on to max-value planning problems with more general timing constraints. In particular, in this section, we investigate a max-value problem where every location has a release-time and a deadline associated with it; The value at a location is obtained only if the location is visited after its release time and before its deadline. We call this the *Max-value Time-Windows TSP*, or the TIME-WINDOWS-TSP for short.

We also consider a special case of this problem where all the release times are zero. We call this the *Max-value Deadlines TSP*, or the DEADLINES-TSP. Note that the ORIENTEERING problem is a further special case of this problem in which all nodes $v \in V$ have $R(v) = 0$ and all have the same deadline $D(v) = D$.

In this section we give an algorithm for approximating the TIME-WINDOWS-TSP, that uses an ORIENTEERING approximation as a subroutine. An α_{Orient}-approximation for ORIENTEERING, when used in our algorithm, gives an $\alpha_{\text{Orient}} \log^2 n$ approximation to the TIME-WINDOWS-TSP. For the special case when all release times are zero, our algorithm gives an $\alpha_{\text{Orient}} \log n$ approximation to the DEADLINES-TSP. Using the 3-approximation for ORIENTEERING developed in Section 2.3, we obtain a $3 \log n$ and a $3 \log^2 n$ approximation for the DEADLINES-TSP and the TIME-WINDOWS-TSP respectively. We begin by describing the approximation for DEADLINES-TSP.

We begin by introducing some new notation. For a path P and set $S \subseteq V$, let $\Pi_P(S) = \sum_{v \in S: R(v) \le \ell_P(v) \le D(v)} \pi(v)$ and $\Pi_P = \Pi_P(V)$. Note that a path can visit a node multiple times, but the value at any node can be collected at most once. For any path P, the restriction of the path P to a set S of vertices, $P_{|S}$, denotes the path that visits nodes of S in the order that P visited them, and does not visit any other nodes. Note that, for all S and P, $\ell_{P_{|S}}(v) \le \ell_P(v)$ for all $v \in S$.

For the TIME-WINDOWS-TSP, our solution is allowed to "wait" at locations (or alternately, to take longer to traverse an edge than its specified length) as desired, in order to arrive at future locations after their release times.

2.4.1 Using ORIENTEERING to approximate DEADLINES-TSP

We first concentrate on the case where all release-times are zero. As mentioned earlier, our algorithm for the DEADLINES-TSP uses ORIENTEERING as a subroutine to find segments of a near optimal path. At a high level, our algorithm extracts subgraphs from the graph, such that solving instances of ORIENTEERING over each subgraph (with a single deadline) gives us a good approximation to the segment of P^* traversing the subgraph. We then combine these segments in the best possible manner using a dynamic program, and obtain an approximation to P^*. We say that a path visits a location, if the location is visited before its deadline.

More formally, we prove that there is a partition of the graph into sets $\{V_i\}$ with the following properties. The sets are characterized by deadlines $\{\delta_i\}$, such that $V_i = \{v \in V : D(v) \in (\delta_i, \delta_{i+1}]\}$. There is a path that for all pairs $i < j$ visits vertices in V_i before vertices in V_j, and obtains at least a $\frac{1}{\log n}$ fraction of the optimal reward. Furthermore, this path has the property, that among all vertices that it visits in the set V_i, the vertex with the smallest deadline is visited last. This allows us to use the ORIENTEERING subroutine to approximate the path in every set and stitch these together using dynamic programming.

In order to define the sets $\{V_i\}$, it is convenient to view the vertices in the optimal path P^* as lying on a two dimensional plane, with the horizontal and vertical axes corresponding to deadlines and time respectively. Let $s = u_0, u_1, \ldots, u_l = t$ denote the vertices visited by P^*. Then vertex u_i lies at the point $p_i = (D(u_i), \ell_{P^*}(u_i))$ (See Figure 2.4(a)).

2.4.2 Minimal vertices

We now define a special set of vertices visited by P^*, called *minimal* vertices, such that breaking P^* into segments at these vertices will allow us to transform the DEADLINES-TSP into multiple instances of ORIENTEERING.

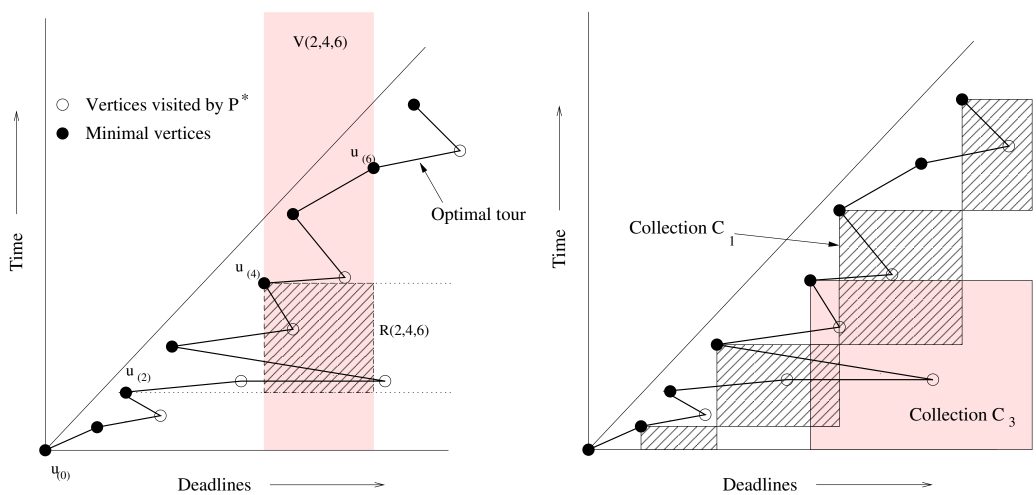
Without loss of generality, we assume that $D(s) = 0$. Let $u_{(0)} = s$ be the first minimal vertex. We define the ith minimal vertex $u_{(i)}$ to be the vertex with the smallest deadline visited by P^* after the $(i-1)$th minimal vertex. Formally, u_i is minimal if for any other vertex u_j, either $\ell_{P^*}(u_j) \leq \ell_{P^*}(u_i)$ or $D(u_j) \geq D(u_i)$. Pictorially, these are vertices that form the upper left envelope of the points p_i. Let $\mathcal{M} = \{u_{(0)}, \ldots, u_{(m)}\}$ denote the set of minimal vertices ordered in increasing order of deadlines, with $|\mathcal{M}| = m + 1$.

In order to understand the importance of minimal vertices, let us consider a simpler problem, where we know all the minimal vertices $u_{(i)}$ and the times $t_{(i)}$ at which P^* visits them. Here is how we can approximate P^* in this case. We construct approximations to segments of P^* between every consecutive pair of minimal vertices. Consider, for example the pair $u_{(i)}$ and $u_{(i+1)}$. Note that the vertices that P^* visits between $u_{(i)}$ and $u_{(i+1)}$ have deadlines larger than that of $u_{(i+1)}$, $D(u_{(i+1)})$. So we can simply remove all locations with deadlines smaller than $D(u_{(i+1)})$, and reduce the deadlines of the remaining ones to $D(u_{(i+1)})$. We now use ORIENTEERING with the time $t_{(i+1)} - t_{(i)}$ as the deadline to approximate this part.

One issue that we need to be careful about here is to avoid double-counting the value by visiting the same vertex multiple times in different segments. We can take care of this by first constructing the first segment from $u_{(0)}$ to $u_{(1)}$, removing all the vertices visited from the graph, constructing the second segment, removing all the vertices visited from the graph, and continuing in this fashion for the remaining segments. A simple covering argument shows that we only lose an additional additive factor of 1 in using this sequential construction. Therefore, if we know the minimal vertices and the times at which they were visited, this technique gives us a $\alpha_{\text{Orient}}+1$ approximation for DEADLINES-TSP.

Now let us consider the problem of not knowing the times at which P^* visits the minimal vertices. (We will deal with the issue of not know the minimal vertices later in Section 2.4.4.) One approach that we may take is to try out all possible values of these times. Each segment can have $\Theta(D_{\max})$ possible lengths, where $D_{\max}$ is the maximum deadline in the graph. Furthermore, the number of segments can be $\Theta(n)$. In this case, the number of trials in the sequential algorithm above would be $\Theta(D_{\max}^n)$. So this approach in infeasible.

Instead, if we could approximate each segment in parallel, then we would only require $\Theta(D_{\max})$ trials for each consecutive pair of minimal vertices, and therefore at most $\Theta(D_{\max} n)$ trials in all. We still need to deal with multiple-counting of value in this case. One way to avoid multiple-counting would be to partition the graph into sets V_i, such that while constructing a segment between $u_{(i)}$ and $u_{(i+1)}$, we can only visit vertices in the set V_i. All vertices in the set V_i would of course have deadlines larger than $D(u_{(i+1)})$. We may also restrict them to having deadlines less than $D(u_{(i+2)})$, so that V_i is disjoint from the set V_{i+1}. We must now show that restricting the constructed segments in this manner does not reduce the amount of reward collected by a large factor. We formalize this construction in the next subsection, and prove that we only lose a factor of $\log n$ in partitioning the graph in this manner. We will further show in Section 2.4.4 that we can approximate the best solution obeying this restriction to within a constant factor (namely, α_{Orient}). Together this would imply an $O(\log n)$ approximation.



(a) The 2-dimensional layout and minimal vertices

(b) Family of disjoint collections of rectangles

Figure 2.4: An example illustrating the construction of the rectangles $\mathcal{R}(x, y, z)$.

2.4.3 A partition into sets

Following the basic idea above, we now use sets of minimal vertices to define a partition of P^* into segments that also partitions the graph into sets based on deadlines.

For any three minimal vertices $u_{(x)}, u_{(y)}, u_{(z)} \in \mathcal{M}$ with $x \leq y \leq z$, define the set $V(x, y, z)$ to be the set of all vertices with deadlines between those of $u_{(y)}$ and $u_{(z)}$. Likewise, define the rectangle spanned by x, y, and z, denoted $\mathcal{R}(x, y, z)$, to be the set of all points in $V(x, y, z)$ that are visited by P^* between $u_{(x)}$ and $u_{(y)}$ (see Figure 2.4(a)). Formally, these are vertices u visited by P^* such that $D(u_{(y)}) \leq D(u) \leq D(u_{(z)})$ and $\ell_{P^*}(u_{(x)}) \leq \ell_{P^*}(u) \leq \ell_{P^*}(u_{(y)})$.

A crucial property of the sets $V(\cdot)$ that we will use in our analysis is that for any indices $w \leq x \leq y \leq z$, the sets $V(w,x,y)$ and $V(x,y,z)$ are disjoint. Likewise, the rectangles $\mathcal{R}(w,x,y)$ and $\mathcal{R}(x,y,z)$ are "component-wise" disjoint, that is, they are disjoint along *both* the time and deadline axes. Therefore, we can approximate the segments $P^*_{|\mathcal{R}(w,x,y)}$ and $P^*_{|\mathcal{R}(x,y,z)}$ in parallel without worrying about double-counting the value.

Extending this idea, given a subset $S = \{n_1, n_2, \cdots\}$ of indices of minimal vertices, we define a collection of rectangles: $\mathcal{C}(S) = \{\mathcal{R}(n_j, n_{j+1}, n_{j+2}) \,|\, 1 \leq j \leq |S| - 2\}$. Note that any two rectangles in $\mathcal{C}(S)$ are component-wise disjoint by definition.

We say that a vertex u visited by P^* is in a collection $\mathcal{C}(S)$ if it belongs to some rectangle $\mathcal{R} \in \mathcal{C}(S)$. We now construct $\log n$ collections of disjoint rectangles, $\mathcal{F} = \{\mathcal{C}_1, \ldots, \mathcal{C}_{\log n}\}$, with the property that each vertex visited by P^* lies in at least one of these collections. This implies that the total reward contained in this family of collections is at least Π_{P^*}. Therefore, there is some collection $\mathcal{C}_i$ that has at least a $1/\log n$ fraction of this reward. In the next subsection, we give a polynomial time procedure to compute a path that collects at least an $O(1)$ fraction of the reward contained in the best disjoint collection of rectangles. Together, these imply an $O(\log n)$ approximation.

The family $\mathcal{C}_1, \ldots, \mathcal{C}_{\log n}$ is defined as follows. Let

$$S_i = \{n_{i,j} = j2^i + 2^{i-1} : 0 \leq j \leq 2^{\log m - i} - 1\} \cup \{0, m\},$$

where $n_{i,-1} = 0$ and $n_{i,2^{\log m - i}} = m$, by definition. Now let $\mathcal{C}_i = \mathcal{C}(S)$. So, e.g.,

$$\begin{aligned} \mathcal{C}_1 &= \{\mathcal{R}(2j-1, 2j+1, 2j+3) : j = 0, \ldots, m/2 - 1\} \\ \mathcal{C}_2 &= \{\mathcal{R}(4j-2, 4j+2, 4j+6) : j = 0, \ldots, m/4 - 1\} \end{aligned}$$

and so on (see Figure 2.4 (b)). Thus we have a family $\mathcal{F} = \{\mathcal{C}_1, \ldots, \mathcal{C}_{\log m}\}$ consisting of $\log m \leq \log n$ collections. Note that for all $j \leq m$, at least one set S_i contains j.

Lemma 2.5. *For each vertex u visited in the optimum path P^*, there is at least one collection $\mathcal{C}_i \in \mathcal{F}$, such that u lies in some rectangle in $\mathcal{C}_i$.*

Proof. Consider a vertex $u \in P^*$ and let $u_{(i)}$ be the minimal vertex for which $D(u_{(i)}) \leq D(u) < D(u_{(i+1)})$. Likewise, let $u_{(j)}$ be the minimal vertex such that $\ell_{P^*}(u_{(j-1)}) < \ell_{P^*}(u) \leq \ell_{P^*}(u_{(j)})$. Observe that $i \geq j-1$. If $i = j-1$, then $D(u) < D(u_{(i+1)})$ and $\ell_{P^*}(u) > \ell_{P^*}(u_{(j-1)}) = \ell_{P^*}(u_{(i)})$ contradicts the fact that $u_{(i+1)}$ is the next minimal vertex after $u_{(i)}$. Therefore we have $i \geq j$.

Now observe that if for some b, the subset S_b contains exactly one number k with $j \leq k \leq i$, then the point u would lie in some rectangle in $\mathcal{C}_b$, in particular, the one corresponding to the number k. Specifically, let the indices preceding and following k in S_b be k_1 and k_2. Then we have $k_1 < j$ and $k_2 > i$. So, $\ell_{P^*}(u_{(k_1)}) \leq \ell_{P^*}(u_{(j-1)}) < \ell_{P^*}(u) \leq \ell_{P^*}(u_{(j)}) \leq \ell_{P^*}(u_{(k)})$. On the other hand, $D(u_{(k)}) \leq D(u_{(i)}) \leq D(u) < D(u_{(i+1)}) \leq D(u_{(k_2)})$. Therefore, $u \in \mathcal{R}(k_1, k, k_2)$.

It remains to show that there exists a subset S_b containing exactly one number k with $j \leq k \leq i$. Let b be such that $2^b \leq i - j < 2^{b+1}$. For this choice of b, either $|S_b \cap [i, j]| = 1$ or $|S_b \cap [i, j]| = 2$. In the first case, we are already done. In the second case, let x and y be the two points in $S_b \cap [i, j]$, then observe that $(x + y)/2 \in S_{b+1}$ and since $i - j < 2^{b+1}$, exactly one point from S_{b+1} lies in the range $[i, j]$. This concludes the proof of the lemma. □

2.4.4 Finding the best collection

We now show how we can find a path such that the total reward collected in that path is a constant factor of the reward in the best collection.

Recall that if we knew all the minimal vertices in P^*, then we could approximate the value collected by the best collection $\mathcal{C} \in \mathcal{F}$ in the following manner: Guess the amount of time that P^* spends in each rectangle in $\mathcal{C}$ (by trying all $D_{\max}$ values), approximate each rectangle $\mathcal{R}(x, y, z) \in \mathcal{C}$ using ORIENTEERING over the subgraph $G[V(x, y, z)]$, and then combine these to obtain an α_{Orient}-approximation to $\mathcal{C}$.

However, without knowing the minimal vertices, we cannot know which vertices lie in the sets $V(x, y, z)$, and therefore cannot obtain the above approximation. To get around this problem, we use dynamic programming to try all possible sets of minimal vertices. In particular, for every triple of vertices in the graph, assuming that they form a consecutive triple of minimal vertices in some collection $\mathcal{C}$, we find the approximately optimal path over the rectangle spanned by these vertices. We then use a dynamic program to combine a subset of these paths in the best possible way.

The algorithm is described in detail in Figure 2.5.

Lemma 2.6. *We can compute in polynomial time a feasible path that collects at least a third of the the reward collected by the best collection $\mathcal{C}_i$.*

Proof. Let $v_1, \ldots, v_n$ denote the vertices in G in the increasing order of their deadlines. We compute the following quantity: For every triple of vertices v_i, v_j, v_k such that $i \leq j \leq k$, and for all possible lengths ℓ, we find the best path that starts at v_i and ends at v_j, takes time ℓ, visits vertices with deadlines between $D(v_j)$ and $D(v_k)$, and obtains as much reward as possible. When v_i, v_j, and v_k are minimal vertices, this path approximates the value that the optimal path obtains in the rectangle $\mathcal{R}(i, j, k)$.

Having obtained these $O(n^3 D_{\max})$ quantities, the next step of our algorithm considers all possible ways of combining these approximate paths to get the maximum possible reward. The algorithm does this using a dynamic program that records for each vertex and each length, the best combination of the subpaths up to that vertex with that length (see Step 4 in Figure 2.5). Note that since the vertices are ordered by deadlines, no vertex is double counted in the reward. Secondly, observe that any path corresponding to a collection $\mathcal{C}_i$ would be considered by this dynamic program, as this corresponds to patching intervals corresponding to the disjoint rectangles in $\mathcal{C}_i$. Hence modulo the

---

Input: Graph $G = (V, E)$ with deadlines $D(v)$.
Output: Path P with $\Pi_P \geq \frac{1}{3 \log n} \Pi_{P^*}$.

---

1. Let $v_1, \ldots, v_n$ denote the vertices in G in the increasing order of their deadlines.
2. Let V_{jk} denote the set of vertices with deadlines between $D(v_j)$ and $D(v_k)$. For all $i \neq j \neq k \leq n$, for all $\ell \leq D(v_j)$, apply the ORIENTEERING approximation to the graph restricted to V_{jk} with distance bound ℓ, start v_i, and terminal vertex v_j, and let $\pi(i, j, k, \ell)$ denote the reward obtained.
3. Let $\pi'(j, k, \ell)$ with $\ell \leq D(v_k)$ denote the (approximately) maximum reward that can be obtained by a path of length ℓ covering nodes in disjoint rectangles having deadline at most $D(v_k)$ and terminating at vertex v_j.
4. For all $j \leq k \leq n$ and $\ell \leq D(v_k)$, compute $\pi'(j, k, \ell)$ and the corresponding path by the following recurrence
$$\pi'(j, k, \ell) = \max_{i \leq j, \text{ with } \ell' \leq D(v_j)} \{\pi'(i, j, \ell') + \pi(i, j, k, \ell - \ell')\}$$
5. Return the path corresponding to maximum reward $\max_j \pi'(j, n, D(v_n))$ computed in the last step.

Figure 2.5: Algorithm *DTSP*—$3 \log n$-approximation for DEADLINES-TSP

factor 3 that we lose in the point to point orienteering subroutine, we can obtain at least the reward contained in the optimum collection $\mathcal{C}_i$. □

The algorithm is given in Figure 2.5. By Lemma 2.5 and 2.6 we have that,

Theorem 2.7. *Algorithm* DTSP *described in Figure 2.5 is a* $3 \log n$ *approximation algorithm for the* DEADLINES-TSP *problem.*

2.4.5 From deadlines to time-windows

Note that Algorithm *DTSP* can be easily modified to return a path of length exactly T for some parameter T — Reduce the deadlines of all nodes with $D(v) > T$ to T and run the algorithm; If the resulting path is shorter than T, wait at the last node for an appropriate amount of time. Likewise, we can modify the algorithm so that the returned path must end at a pre-specified vertex t.

We now show how to use Algorithm *DTSP* to solve the TIME-WINDOWS-TSP problem in the special case when all the nodes have a deadline of ∞, but there is a fixed limit T on the length of the path and the path must start from node s and end at node t. (Note that the nodes have non-zero release times.) We call this problem ORIENTEERING *with Release Times*.

Given an instance G of ORIENTEERING with Release Times, we convert the graph G into its "complimentary" graph $G' = (V, E)$, with each $v \in V$ having a time window $[0, T - R(v)]$, where $R(v)$ is the release time of v in G. For any "legal" path P of length T in G that starts at s, ends at t

and visits vertices v within their time-windows $[R(v), \infty]$, we can define a legal path P^R in G' that starts at t, ends at s, follows path P in reverse, and visits nodes v within $[0, T-R(v)]$. Therefore, any solution to ORIENTEERING with Release Times on G can be converted into a solution with the same reward to the DEADLINES-TSP on G', and vice versa. Therefore, using an α_{DTSP}-approximation to the DEADLINES-TSP, we can obtain an α_{DTSP}-approximation to ORIENTEERING with Release Times. We get the following theorem:

Theorem 2.8. *Algorithm* DTSP *gives a* $3 \log n$ *approximation to* ORIENTEERING *with Release Times, when applied to the complimentary graph.*

Based on the above theorem, our algorithm for the TIME-WINDOWS-TSP problem is simple—run the algorithm *DTSP* (Figure 2.5), replacing the ORIENTEERING subroutine in step 1 by the algorithm for ORIENTEERING with Release Times. This gives a $3 \log^2 n$-approximation to the TIME-WINDOWS-TSP problem. In fact, an (α_{DTSP})-approximation to the DEADLINES-TSP can be converted to an $(\alpha_{\text{DTSP}} \log n)$-approximation to the TIME-WINDOWS-TSP problem using the same technique.

Theorem 2.9. *The algorithm described above gives a* $3 \log^2 n$ *approximation to the* TIME-WINDOWS-TSP *Problem.*

2.5 A bicriteria approximation for the TIME-WINDOWS-TSP

We now consider relaxing the timing constraints in the TIME-WINDOWS-TSP, so as to obtain larger value than that obtained by the algorithm in the previous section. In particular, we allow our solution to collect value at locations that are visited within their time-windows, as well as shortly after their deadlines have passed. Suppose that the optimal solution covering the maximum value while obeying time-windows exactly is P^*. We measure the quality of a solution by two criteria: (1) the ratio between the value obtained by our solution and the value obtained by P^*, and, (2) the maximum factor by which our solution exceeds the deadline of a location where it claims the value.

In this section, we describe a bicriteria algorithm for the TIME-WINDOWS-TSP that for any $\epsilon > 0$, obtains an $O(\log \frac{1}{\epsilon})$ fraction of the value obtained by P^*, if it is allowed to visit vertices in the window $[R(v), (1+\epsilon)D(v)]$. Equivalently, if we allow the path to exceed deadlines of vertices by a factor of $(1 + 2^{-k})$, for any $k > 0$, then we can obtain an $O(k)$ approximation to the value.

We begin with a few special purpose algorithms, and then describe how to combine them for the general case. We first consider the case when P^* visits a large fraction of the nodes very close to their deadlines (the *small margin* case). Then we consider the case when P^* visits many nodes much before their deadlines (the *large margin* case). Towards the end of this section, we show that the bicriteria result also implies a $O(\log D_{\max})$-approximation to the TIME-WINDOWS-TSP Problem.

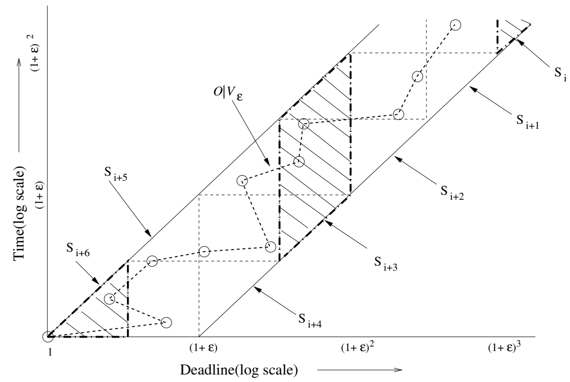


Figure 2.6: Division of set V_ϵ into segments S_j. Note that segments S_i, S_{i+3} and S_{i+6} are disjoint along the time axis.

2.5.1 The small margin case

At the heart of our bicriteria approximation is a procedure that obtains a constant fraction of the optimal reward while exceeding deadlines by a small factor, if P^* visits most nodes $v \in P^*$ very close to their deadlines (within some multiplicative factor).

Let ϵ be a fixed constant. Consider the set of nodes $V_\epsilon = \{v : D(v)/(1+\epsilon) \leq \ell_{P^*}(v) \leq D(v)\}$. We design an algorithm $\mathcal{A}$ that obtains a constant fraction of $\Pi_{P^*}(V_\epsilon)$ as reward, while exceeding deadlines by a factor of $(1+\epsilon)^2$. Note that only deadlines are violated — nodes on the path output by algorithm $\mathcal{A}$ are visited after their release times.

Intuitively, if we consider the nodes visited by P^* close to their deadlines, then P^* must visit them roughly in order of their deadlines. In fact, as we show below, we can divide these vertices into groups according to deadlines, such that if we consider every third group in the division, all the vertices in a group are visited by P^* before all the vertices in the third next group. This means that we can consider every third group separately, and approximate the subpath of P^* over it using ORIENTEERING, and patch the obtained paths to form a good solution. Note that in each group, the deadlines of nodes vary only by a small constant factor. Therefore, by applying ORIENTEERING to these groups with a single deadline picked appropriately, we only exceed the deadlines by a constant factor. We now describe the division into groups.

Let $f = \frac{1}{\sqrt{1+\epsilon}}$. We divide the nodes in the graph into segments as follows. Segment S_j consists of nodes that have deadlines in $(f^j D_{\max}, f^{j-1} D_{\max}]$ (Figure 2.6). The benefit of having small variation in the deadlines of vertices in each segment is that we can treat $f^{j-1} D_{\max}$ as a single deadline for all nodes in S_j, and would only be violating the deadlines by a factor of f. Similarly, we take care of all the release times, while losing an extra factor of f^3 in the deadlines, as described below.

However, note that in approximating $P^*_{|S_j}$ for some segment j, we may lose out on the value

Input: Graph $G = (V, E)$ with deadlines $D(v)$; Constant ϵ; $f = \frac{1}{\sqrt{1+\epsilon}}$. Let $S_j = \{v : D(v) \in (f^j D_{\max}, f^{j-1} D_{\max}]\}$, $\mathcal{S}_{i\text{mod}3} = \cup_{j=0}^{\infty} S_{3j+i}$.
Output: Path P with $\Pi_P \geq \frac{1}{9}\Pi_{P^*}(V_\epsilon)$ and $R(v) \leq \ell_P(v) \leq (1+\epsilon)^2 D(v)$ for all $v \in P$, where $V_\epsilon = \{v : D(v)/(1+\epsilon) \leq \ell_{P^*}(v) \leq D(v)\}$.

1. For all j, for all $u, v \in S_j$, and for all integers $\ell \leq f^{j-1}D_{\max} - f^j D_{\max}$, use the ORIENTEERING algorithm to find a path of length ℓ from u to v. Let the value collected by this path be $\pi(u, v, \ell)$.
2. For $k = 0, 1, 2$ do the following:
   (a) Let $\pi'(v, j, \ell)$ for $\ell \leq f^{j-1}D_{\max}$ denote the (approximately) maximum reward that can be collected from $\mathcal{S}_{k\text{mod}3}$ up to segment $S_j \subset \mathcal{S}_{k\text{mod}3}$ using a path of length ℓ that ends at $v \in S_j$.
   (b) Compute $\pi'(v, j, \ell)$ using the following recurrence.
   $$\pi'(v,j,\ell) = \max_{f^j D_{\max} \leq \ell' \leq \ell, u \in S_j, u' \in S_{j+3}} \{\pi'(u', j+3, \ell' - \ell(u', u)) + \pi(u, v, \ell - \ell')\}$$
3. Output the path corresponding to the maximum value collected until the last segment, slowed-down by a factor of f^3.

Figure 2.7: Algorithm $\mathcal{A}$—Bicriteria approximation for the small margin case

collected by P^* in neighboring segments, if P^* visits these nodes before visiting its last vertex in S_j. This issue can be dealt with by approximating only every third segment. In particular, we note that the definition of f and V_ϵ are such that for any j, all vertices in $S_j \cap V_\epsilon$ are visited by P^* after all vertices in $S_{j+3} \cap V_\epsilon$ (see Lemma 2.10 and Figure 2.6).

Lemma 2.10. *For any j and any nodes $u \in S_j \cap V_\epsilon$ and $v \in S_{j+3} \cap V_\epsilon$, $\ell_{P^*}(v) < \ell_{P^*}(u)$.*

Proof. We have $D(v) \leq f^{j+2}D_{\max}$. So, $\ell_{P^*}(v) \leq D(v) \leq f^{j+2}D_{\max}$. Likewise, $D_u > f^j D_{\max}$. So, by the definition of V_ϵ, $\ell_{P^*}(u) \geq f^2 D_u > f^{j+2}D_{\max} \geq \ell_{P^*}(v)$. □

The above lemma suggests a natural strategy for approximating the reward collected by P^* in V_ϵ. Let $\mathcal{S}_{i\text{mod}3} = \cup_{j=0}^{\infty} S_{3j+i}$ for $i = 0, 1, 2$. Then, $\cup_i \mathcal{S}_{i\text{mod}3} = V_\epsilon$. Then, one of the three sets contains at least a third of the total reward in V_ϵ. Let this be $\mathcal{S}_{k\text{mod}3}$. We approximate the value obtained by P^* in this set.

Let $P' = P^*_{|\mathcal{S}_{k\text{mod}3}}$. We approximate P' by constructing approximations to the optimal path in sets $S_{3j+k} \in \mathcal{S}_{k\text{mod}3}$, and join them by taking a shortcut across the intermediate sets S_{3j+k-1} and S_{3j+k-2}. In order to approximate the optimal path in some S_j, we guess the first and the last vertex that P' visits in this set, and the corresponding times at which it visits them. Then we use the ORIENTEERING algorithm to construct a path of the guessed length. We append these subpaths, with an appropriate amount of "waiting-time" between consecutive segments, so that the resulting path visits segments in the same time interval as P^*.

Since we do not actually know the times at which P^* visits these segments, our algorithm tries all possible values for the length of the subpath, and computes one subpath corresponding to each length. We then use a dynamic program to find the best division of length into different segments, in order to obtain a path with a given total length ending at a specific vertex (see Step 2(b) in Figure 2.7). The dynamic program takes polynomial time and considers all possible valid combinations of subpaths, returning one that is at least as good as the one corresponding to the optimal path P^*.

We now consider the issue of exceeding deadlines within each segment. Consider again the subpaths corresponding to P^*. Lemma 2.11 shows that the start time of such a path in segment S_j is at least f^3 times the release time of any node in S_j, whereas the end time time is at most f times the deadline of any node in S_j.

Lemma 2.11. *For any two nodes $v \in S_j \cap V_\epsilon$, $f^3 R(v) \leq \ell_{\mathcal{A}}(v) < fD(v)$.*

Proof. Note that by construction, both $\mathcal{A}$ and P^* visit vertices in $S_j \cap V_\epsilon$ after the time $f^{j+2}D_{\max}$ and before $f^{j-1}D_{\max}$. Therefore, for all $v \in S_j \cap V_\epsilon$, $R(v) \leq f^{j-1}D_{\max}$. This gives $\ell_{\mathcal{A}}(v) \geq f^{j+2}D_{\max} \geq f^3 R(v)$. Furthermore, by the definition of S_j, $D(v) < f^j D_{\max}$. Therefore, $\ell_{\mathcal{A}}(v) \leq f^{j-1}D_{\max} < fD(v)$. □

Finally, we "slow-down" this path by a factor of f^3, that is, if the path output by the algorithm visits a vertex at time t, we visit the vertex and then wait until time $\frac{t}{f^3}$ to move to the next vertex. In the worst case, the path constructed as above visits nodes in S_j at $\frac{1}{f^3} \times f^{j-1}D_{\max}$. By the definition of S_j, we know that this is at most $(1+\epsilon)^2$ times the deadline of any node in S_j. Furthermore, this slow-down ensures that the path visits every vertex after its release time. We lose a factor of 3 in reward by leaving out vertices in $\mathcal{S}_{i \bmod 3}, i \neq k$, and another factor of 3 by using algorithm *P2P*. Thus, we get the following theorem. Figure 2.7 describes the algorithm in detail.

Theorem 2.12. *Algorithm $\mathcal{A}$ returns a path P with $\Pi_P^\epsilon \geq \frac{1}{9}\Pi_{P^*}(V_\epsilon)$.*

2.5.2 The large margin case

In this section, we consider the second extreme case — approximating P^* on vertices that are visited much before their deadlines. Let $V_{\frac{1}{4}} = \{v : \ell_{P^*}(v) \leq \frac{D(v)}{4}\}$. We will describe an algorithm that collects a constant fraction of the reward $\Pi_{P^*}(V_{\frac{1}{4}})$.

For all nodes v, define new deadlines $D'(v) = \frac{D(v)}{4}$. Note that for all $v \in V_{\frac{1}{4}}$, P^* visits v before its new deadline $D'(v)$.

Our algorithm is allowed to violate these new deadlines $D'(v)$ by a factor of 4. Note that it is straightforward for us to obtain as much reward as P^* while increasing the total length of the path by a factor of $2+\epsilon$ (using the MIN-EXCESS-PATH algorithm of Section 2.2), or to obtain a fraction of that reward with the same total length as P^* using ORIENTEERING. However, we need a guarantee on the time taken to visit each node individually, rather than one on the total length of the path. A guarantee on the total length suffices when the deadlines of all the locations under consideration are within a constant factor of each other. This suggests the following approach—we divide the graph into subsets such that the deadlines of locations in each subset are within a constant factor of each other, and approximate each of these subsets.

We can approximate each of these subsets individually, while exceeding the new deadlines $D'(v)$ by a small constant factor. In fact, we can concatenate the approximations to some of these subsets while losing only another small constant factor in the time taken to reach each location. This is because suppose that we take time τ to approximate one of the subsets with small deadlines, and another subset has deadlines larger than $k\tau$, then by concatenating the two paths, we exceed the time taken to visit locations in the second subset by a factor of at most $1 + 1/k$. We can therefore approximate many of these subsets simultaneously, obtaining a good approximation.

We now give the details. Let $\alpha = 1.2$. For $i \geq 0$, the set S_i contains nodes v that have $D'(v) \in [\alpha^i, \alpha^{i+1})$. Let $\beta = 8$. We define "super-sets" containing every βth set S_i—for $j \in \{0, \cdots, \beta - 1\}$, define $\mathcal{S}_{j \bmod \beta} = \cup_{i \geq 0} S_{\beta i + j}$. Then, $\cup_{j \leq \beta - 1} \mathcal{S}_{j \bmod \beta} = V$. Our goal is to approximate P^* on the set $\mathcal{S}_{j \bmod \beta}$ where it collects the most value.

Let P_i be a path returned by the ORIENTEERING algorithm with parameter $D = \alpha^{i+1}$ when applied to the graph induced by $\{s\} \cup S_i$. Note that P_i collects at least $\frac{1}{3}\Pi_{P^*}(S_i)$ reward. Let T_i be a tour that starts at the root, follows path P_i and then returns back to the root. For some $j < \beta$, consider the path constructed by appending all $T_{\beta i + j}$ for $i \geq 0$. Assume that the length of T_i is exactly $2\alpha^{i+1}$ (we can ensure this by waiting for an appropriate amount of time at the root between consecutive tours). Let this path be called Q_j.

Input: Graph $G = (V, E)$ with deadlines $D(v)$.
Output: Path Q with $\Pi_Q \geq \frac{1}{24}\Pi_{P^*}(V_{\frac{1}{4}})$ and $R(v) \leq \ell_Q(v) \leq D(v)$ for all $v \in Q$, where $V_{\frac{1}{4}} = \{v : \ell_{P^*}(v) \leq \frac{D(v)}{4}\}$.

---

1. For all i, use the ORIENTEERING algorithm on graph $G(\{s\} \cup S_i)$ to construct a path P_i with length at most α^{i+1}. Let T_i be the corresponding tour of length $2\alpha^{i+1}$.
2. For all $j \in \{0, \cdots, \beta - 1\}$, let Q_j be the concatenation of $P_{\beta i + j}$ for all $i \geq 0$ and let $\pi_j = \sum_i \Pi_{P_{\beta i + j}}(S_{\beta i + j})$.
3. Return the path Q_j, slowed down by a factor of 2, corresponding to the maximum reward π_j over all j.

Figure 2.8: Algorithm $\mathcal{B}$—Approximation for the large margin case

Lemma 2.13. *For any i and j and $v \in S_{\beta i + j}$, $\frac{1}{2}R(v) \leq \ell_{Q_j}(v) \leq \frac{1}{2}D(v)$.*

Proof. The length of the path Q_j up to and including the set $S_{\beta i + j}$ is

$$\begin{aligned}\sum_{k<i} |T_{\beta k+j}| + |P_{\beta i+j}| &= \sum_{k<i} 2|P_{\beta k+j}| + |P_{\beta i+j}| \\ &= 2\sum_{k<i} \alpha^{\beta k+j+1} + \alpha^{\beta i+j+1} \\ &= 2\alpha^{j+1}\frac{\alpha^{\beta i}-1}{\alpha^{\beta}-1} + \alpha^{\beta i+j+1} \\ &\leq \alpha^{\beta i+j+1}\left(\frac{2}{\alpha^{\beta}-1}+1\right)\end{aligned}$$

The deadline of v is $D(v) = 4D'(v) \geq 4\alpha^{\beta i+j}$. Thus,

$$\ell_{Q_j}(v) \leq \frac{\alpha}{4}\left(\frac{2}{\alpha^{\beta}-1}+1\right)D(v)$$

Taking $\alpha = 1.2$ and $\beta = 8$, we get that $\ell_{Q_j}(v) \leq \frac{1}{2}D(v)$. Similarly,

$$\ell_{Q_j}(v) \geq \sum_{k<i} |T_{\beta k+j}| = \frac{2\alpha^{\beta i+j+1}}{\alpha^{\beta}-1}$$

Using $\beta = 8$, and $D'(v) \leq \alpha^{\beta i+j+1}$, we get $\ell_{Q_j}(v) \geq \frac{1}{2}D'(v)$. However we also have $R(v) \leq D'(v)$, because P^* collects vertex v before time $D'(v)$. Therefore we get $\ell_{Q_j}(v) \geq \frac{1}{2}R(v)$. □

Slowing down the path Q_j by a factor of 2 ensures that we cover all nodes within their time window $[R(v), D(v)]$. Note that the sets $\mathcal{S}_{j \bmod \beta}$ for $j \in \{0, \cdots, \beta-1\}$ together cover all the nodes in $V_{\frac{1}{4}}$. Furthermore, we have $\Pi_{Q_j} \geq \frac{1}{3}\sum_i \Pi_{P^*}(S_{\beta i+j})$. Thus, one of the paths Q_j gives a $3\beta = 24$ approximation to the reward collected by P^* in $V_{\frac{1}{4}}$. Our algorithm for finding the best Q_j is given in Figure 2.8.

Theorem 2.14. *Algorithm $\mathcal{B}$ returns a path P with $\Pi_P \geq \frac{1}{24}\Pi_{P^*}(V_{\frac{1}{4}})$.*

2.5.3 The general case

Now we will give an algorithm that produces a bicriteria approximation for the entire graph. In particular, given a parameter ϵ, we construct a path that obtains reward at least $\Omega(\log^{-1}\frac{1}{\epsilon})\Pi_{P^*}$ if it is allowed to exceed the deadlines by a factor of $1+\epsilon$.

The idea behind our algorithm is as follows. We consider a division of the graph into a small number of groups such that each group either falls into the small-margin case studied in Section 2.5.1, or the large-margin case studied in Section 2.5.2. We can approximate any of these groups using the algorithms $\mathcal{A}$ or $\mathcal{B}$ respectively, obtaining a large reward and exceeding deadlines by a small factor. The approximation corresponding to the largest group (in terms of total reward) forms our solution.

Input: Graph $G = (V, E)$ with deadlines $D(v)$; parameter ϵ.
Output: Path P with $\Pi_P \geq \Omega(\frac{1}{\log \frac{1}{\epsilon}})\Pi_{P^*}$ and $R(v) \leq \ell_P(v) \leq (1+\epsilon)D(v)$ for all $v \in P$.

---

1. Let $f = \frac{1}{\sqrt{1+\epsilon}}$ and s be the smallest integer for which $f^{(1.5)^s} \leq \frac{1}{4}$.
2. Apply algorithm $\mathcal{A}$ with parameter f to the graph, and let P_0 be the path obtained.
3. Apply algorithm $\mathcal{B}$ to the graph, and let P_{s+1} be the path obtained.
4. For all $i \in \{1, \cdots, s\}$, do the following:
   (a) For all $v \in V$, define $D'(v) = D(v)f^{(1.5)^{i-1}}$.
   (b) Apply algorithm $\mathcal{A}$ with parameter $f^{(1.5)^{i-1}}$ to the graph with the new deadlines D', and let P_i be the path obtained.
5. Among the paths constructed above, return the one with the maximum reward.

Figure 2.9: Algorithm $\mathcal{C}$—Bicriteria approximation for the general case

As before, let $f = \frac{1}{\sqrt{1+\epsilon}}$. Let s be defined as the smallest integer for which $f^{(1.5)^s} \leq \frac{1}{4}$. Then $s = O(\log \frac{1}{\epsilon})$. Divide all the nodes into $s+2$ groups as follows. Group i, $1 \leq i \leq s$, is given by $V_i = \{v : \ell_{P^*}(v) \in (f^{(1.5)^i}D(v), f^{(1.5)^{i-1}}D(v)]\}$. Group 0 is given by $V_0 = \{v : \ell_{P^*}(v) \in (fD(v), D(v)]\}$. Group $s+1$ is defined as $V_{s+1} = \{v : \ell_{P^*}(v) \in (0, D(v)/4]\}$. These groups together cover all the nodes in P^*. So, one of the groups contains at least a $\frac{1}{s+2}$ fraction of the total reward collected by P^*. Let V_i be such a group. We now show how we can approximate the value that P^* collects in any of these groups.

If $i = 0$, we can apply algorithm $\mathcal{A}$ right away and obtain a path P with $\Pi_P \geq \frac{1}{9}\Pi_{P^*}(V_0)$ that visits its nodes within a factor of $(1+\epsilon)$ of their deadlines.

Consider the case when $1 \leq i \leq s$. Scale all the deadlines down by a factor of $f^{(1.5)^{i-1}}$, that is, define $D'(v) = f^{(1.5)^{i-1}}D(v)$. Then the path P^* visits all nodes in V_i at time $\ell_{P'}(v) \in (f^{0.5(1.5)^{i-1}}D'(v), D'(v)]$. Now apply algorithm $\mathcal{A}$ with parameter $f^{0.5(1.5)^{i-1}}$. Then, the obtained path collects reward $\pi_P \geq \frac{1}{9}\Pi_{P^*}(V_i)$ and visits all nodes v before time $D'(v)f^{-(1.5)^{i-1}} = D(v)$.

Finally consider the case when $i = s+1$. Note that this is the large margin case considered in a previous subsection. So we can use algorithm $\mathcal{B}$ to obtain a 24-approximation in this case.

Putting everything together, we get the following theorem. The algorithm is described in Figure 2.9.

Theorem 2.15. *Algorithm $\mathcal{C}$ returns a path P with $\Pi_P^\epsilon \geq \frac{1}{24(s+2)}\Pi_{P^*} = \Omega(\frac{1}{\log \frac{1}{\epsilon}})\Pi_{P^*}$.*

As a simple corollary of Theorem 2.15, we get an $O(\log D_{\max})$-approximation to the TIME-WINDOWS-TSP problem without exceeding deadlines.

Corollary 2.16. *Algorithm $\mathcal{C}$ with $\epsilon = 1/D_{\max}$ gives an $O(\log D_{\max})$-approximation to the* TIME-WINDOWS-TSP *problem.*

Proof. From Theorem 2.15, we know that for $\epsilon = 1/D_{\max}$ the path P collects an $\Omega(1/\log D_{\max})$ fraction of the reward from nodes v visited before $(1+\epsilon)D(v)$. For $\epsilon = 1/D_{\max}$, $(1+\epsilon)D(v) < D(v)+1$. Since, all edge lengths and deadlines are integral by assumption, node v is visited by $D(v)$. □

2.6 Extensions

2.6.1 Max-Value Tree and Max-Value Cycle

In this section, we consider the tree and cycle variants of the ORIENTEERING problem. In Max-Value Tree, given a graph G with root r, value function Π and lengths d, we are required to output a tree $\mathcal{T}$ rooted at r with $d(\mathcal{T}) \leq \mathcal{D}$ and maximum possible reward $\Pi(\mathcal{T})$. This problem is also called the Budget Prize-Collecting Steiner Tree problem [89]. Although the unrooted version of the problem can be approximated to within a factor of $5+\epsilon$ via a 3-approximation for k-MST [89], the version of the problem in which a tree is required to contain a specified vertex has remained open until the initial publication of this work.

Let the optimal solution for this problem be a tree $\mathcal{T}^*$. Double the edges of this tree to obtain an Euler tour of length at most $2D$. Now, divide this tour into two paths, each starting from the root r and having length at most D. Among them, let P' be the path that has greater reward. Now consider the Max-Value Path problem on the same graph with distance limit D. Clearly the optimal solution P^* to this problem has $\Pi(P^*) \geq \Pi(P') \geq \frac{\Pi(\mathcal{T}^*)}{2}$. Thus, we can use the α_{Orient}-approximation for ORIENTEERING to get a $2\alpha_{\text{Orient}}$-approximation to T^*.

Finally we note that we can use our algorithm for the ORIENTEERING problem to approximate Max-Value Cycle. Namely, we can find an approximately maximum-value cycle of length at most D that contains a specified vertex s. To this end we apply our algorithm to an instance of the ORIENTEERING problem with the starting node s and the length constraint $D/2$. To obtain a cycle from the resulting path we connect its end-points by a shortest path. Clearly, the length of the resulting cycle is at most D. Now, notice that an optimal max-value cycle of length D can span at most twice the amount of value that an optimal max-value path of length $D/2$. Thus, using α_{Orient}-approximation to ORIENTEERING we get $2\alpha_{\text{Orient}}$-approximation to the Max-Value Cycle problem.

2.6.2 Multiple-Path ORIENTEERING

In this section we consider a variant of the ORIENTEERING in which we are allowed to construct up to k paths, each having length at most D.

We approximate this problem by applying the algorithm in Section 2.3 successively k times, to construct the k paths. At the i-th step, we set the values of all points visited in the first $i-1$ paths to 0, and constructed the i-th path on the new graph, using the ORIENTEERING algorithm in Section 2.3. Using a set-cover like argument, we get the following approximation guarantees for the cases when all paths have the same starting point and when different paths have different starts.

Theorem 2.17. *If all the paths have a common start node, the above algorithm gives a* $1/(1-e^{-1/\alpha_{Orient}})$ *approximation to Multiple-Path* ORIENTEERING. *If the paths have different given start nodes, the above algorithm gives a* $\alpha_{Orient}+1$ *approximation to Multiple-Path* ORIENTEERING. *Using* $\alpha_{Orient}=3$*, we get a* 3.53 *and* 4 *approximation respectively.*

Proof. Let $\alpha = \alpha_{\text{Orient}}$. Consider first the case when all the paths have the same starting point. Let the difference in the reward collected by the optimal solution and the reward collected by our solution up to stage i be Π_i. At the beginning, this is the total reward of the optimal solution. At step i, at least one of the paths in the optimal solution collects reward, not collected by the algorithm by stage i, of value at least $\frac{1}{k}\Pi_i$. Then, using the approximation guarantee of the algorithm for orienteering, our solution collects at least a $\frac{1}{k\alpha}$ fraction of Π_i. That is, $\Pi_{i+1} \leq (1-\frac{1}{k\alpha})\Pi_i$. By the end of k rounds, the total reward collected by optimal solution, but not collected by us, is at most $(1-\frac{1}{k\alpha})^k\Pi(P^*) \leq e^{-1/\alpha}\Pi(P^*)$, and the result follows.

Next consider the case when different paths have different starting locations. Let O_i be the set of points visited by the i-th path in the optimal solution, and A_i be the corresponding set of points visited by our algorithm. Let Δ_i be the set of points that are visited by the i-th path in the optimal solution and some other path in our solution. Let $O = \cup_i O_i$, $A = \cup_i A_i$ and $\Delta = \cup_i \Delta_i$. Now, in the i-th stage, there is a valid path starting at the i-th source, that visits all points in $O_i \setminus \Delta_i$. Thus we have $\Pi(A_i) \geq \frac{1}{\alpha}\left(\Pi(O_i) - \Pi(\Delta_i)\right)$. Summing over i, we get $\alpha\Pi(A) \geq (\Pi(O) - \Pi(\Delta))$. But $\Pi(\Delta) \leq \Pi(A)$. Thus $\Pi(A) \geq \frac{1}{\alpha+1}\Pi(O)$. □

2.6.3 ORIENTEERING on special metrics

Finally, we consider the case when the underlying graph on locations is a special metric, namely a tree metric, and show that in this case ORIENTEERING can be solved exactly in polynomial time.

In particular, our algorithm takes as input a tree T rooted at vertex s, a terminal node t, and a length bound D, and outputs the maximum reward that can be obtained by a path of length at most D traversing the tree starting at s and ending at t. Without loss of generality, we may assume that $t = s$ by adding $d(s,t)$ to the distance bound D. We may also assume that the tree is a binary tree, by adding zero-length edges.

Our algorithm proceeds by guessing the amount of time D_l that the optimal path spends in the left subtree of s; it recursively computes the reward obtained by a tour of length at most D_l in the left subtree of s, and the reward obtained by a tour of length at most $D - D_l - 2\ell_l - 2\ell_r$ in the right subtree of s. Here ℓ_l and ℓ_r are the lengths of the left and right edges incident on s respectively. The algorithm computes these values for all possible values of D_l, and picks the one for which the sum of the rewards obtained on either sides is maximized. The number of computations is at most D for each node in the tree, and therefore at most nD in all.

Theorem 2.18. *There exists a dynamic-programming-based algorithm that solves* ORIENTEERING *exactly on tree metrics and takes time $O(nD)$.*

2.7 Hardness of approximation

All the problems discussed in this paper are NP-hard, as they are generalizations of the TRAVELING SALESMAN PROBLEM. In this section we show that MIN-EXCESS-PATH, ORIENTEERING, DEADLINES-TSP and TIME-WINDOWS-TSP are APX-hard, that is, it is NP-hard to approximate these problems to within an arbitrary constant factor.

The hardness of approximating the MIN-EXCESS-PATH problem follows from the APX-hardness of TRAVELING SALESMAN PROBLEM[118]. In particular, we can approximate TRAVELING SALESMAN PROBLEM to within an α_{ex} factor by approximating the MIN-EXCESS-PATH problem to within an α_{ex} factor with a reward quota of n. We therefore get the following theorem:

Theorem 2.19. *The* MIN-EXCESS-PATH *problem is NP-hard to approximate to within a factor of* $\frac{220}{219}$.

Theorem 2.20. ORIENTEERING *is NP-hard to approximate to within a factor of* $\frac{1481}{1480}$.

Proof. We reduce the TRAVELING SALESMAN PROBLEM on $\{1, 2\}$-metrics to ORIENTEERING. In particular, let $G = (V, E)$ be a complete graph on n nodes, with edges lengths in the set $\{1, 2\}$. Engebretsen and Karpinski [60] show that the TRAVELING SALESMAN PROBLEM is NP-hard to approximate within a factor of $1 + \alpha = \frac{741}{740}$ on such graphs.

Our reduction is as follows. Let the length of the optimal TSP solution be $L = n + \delta n$. (We simply try all values of L between n and $2n$.) Suppose that there is an algorithm that approximates ORIENTEERING within a factor of $1 + \beta$, where $\beta \leq \frac{1}{1480}$. We apply this algorithm to the graph G with distance limit L. Note that the optimal solution (which is the optimal TSP path) collects $n - 1$ nodes within distance L (all nodes except the start, assuming a reward of 0 on the start node). Therefore, the solution returned by our algorithm collects $\frac{1}{1+\beta}(n - 1)$ nodes. We augment this solution to a tour containing all the nodes, by using $(1 - \frac{1}{1+\beta})(n - 1) + 1$ edges of length at most 2. Therefore, the length of our solution is at most

$$\begin{aligned} L + 2(1 - \tfrac{1}{1+\beta})(n-1) + 2 &= L + \tfrac{2\beta}{1+\beta}(n-1) + 2 \\ &< L + 2\beta n \\ &= L + \alpha n \leq (1+\alpha)L \end{aligned}$$

where the second inequality follows from assuming that $n > \frac{1}{\beta^2}$.

Therefore, we get a $(1+\alpha)$-approximation to TSP on G, contradicting the fact that the TRAVELING SALESMAN PROBLEM is NP-hard to approximate to within a $(1+\alpha)$ factor on $\{1,2\}$-metrics. $\square$

Using a similar argument as for ORIENTEERING, we get a $\frac{1481}{1480}$ hardness of approximation for the max-value cycle problem as well. The result trivially extends to the DEADLINES-TSP and TIME-WINDOWS-TSP problems as well.

2.8 Concluding remarks

In this chapter we develop new techniques for approximating budget versions of planning problems. We combine dynamic programming with LP-relaxations of the problems to give the first approximations to these problems.

Linear programming is one of the most widely used techniques for approximating NP-hard optimization problems; there is a large repertoire of techniques for rounding solutions to LP-relaxations of problems (see [138] for examples). Unfortunately, budget versions of planning problems do not seem to have good LP-relaxations. LP-based techniques appear to perform poorly specifically in instances where the "slack" in the solution is small (i.e. by reducing the budget by a small amount, the value obtained is reduced considerably). We formalize this notion of slack through a quantity that we call the excess of a path, and show how low-excess instances can be approximated via dynamic programming. It would be interesting to see if such a "slack"-based approach implies improved approximations to other problems that have poor LP-relaxations.

From the point of view of hardness of approximation, our current results imply the same hardness for the DEADLINES-TSP and TIME-WINDOWS-TSP problems, as for the ORIENTEERING problem. We strongly suspect that the former two problems are strictly harder to approximate than ORIENTEERING, and are in fact hard to approximate within $o(\log n)$ factors. The difficulty in proving such a result is to design a class of instances of the problems that have a good optimal solution, and for which the deadlines of vertices do not reveal the order in which the optimal solution visits them.

Two important classes of path-planning problems not considered in our work are pickup-and-delivery problems and problems dealing with the min-vehicle objective. For the latter, it is easy to obtain log-approximations using covering arguments. However we suspect that it may be possible to approximate these within constant factors. Another interesting class of problems is the online version of planning problems where requests for visiting sites arrive over time. It would be interesting to extend our algorithms for MIN-EXCESS-PATH and ORIENTEERING to this case.

Chapter 3

Planning under Uncertainty

3.1 Introduction

We continue our study of path-planning problems, albeit in a dynamic context, where the underlying map or constraints on the path change over time.

Consider a robot with a map of its environment, that needs to visit a number of sites in order to drop off packages, collect samples, search for a lost item, etc. In Chapter 2, we studied algorithms that allow the robot to perform its tasks as efficiently as possible while obeying deadlines on the tasks. However, what if the environment of the robot is changing over time, or alternately, the robot makes mistakes in following the algorithm precisely? For example, the algorithm may instruct the robot to take 10 steps forward, while the robot takes only 9. Furthermore, obstacles in the robot's environment may change unpredictably over time. Then the algorithm must take corrective measures, or may fail to achieve its goal. In this situation, it may not be possible to specify beforehand the course of actions taken by a robot following an *adaptive* algorithm, or quantify its performance before the actual execution. A natural question then is to ask for a *strategy* for the robot, that maximizes the number of tasks completed by a deadline *in expectation*, or minimizes the time taken to complete all tasks in expectation, etc. In a slightly different scenario, suppose that a rescue-robot is looking for a lost or trapped individual. Without knowledge of the exact location of the individual, the robot may rely on its beliefs about the individual's location (such as the relative likelihoods of the finding the person at different locations), in order to minimize the time taken to find the person, or maximize the probability that the person is found alive. We call such problems *stochastic path-planning* problems.

Stochasticity in robot navigation can arise both from the unreliability in the robot's behavior or external forces that might do something unpredictable to the robot's state. It is typically modeled using Markov decision processes (MDPs) [27, 28, 91, 120, 131]. We describe the MDP formalism below.

3.1.1 Markov decision processes

A Markov decision process (MDP) consists of a state space S, a set of actions A, a probabilistic transition function T, and a value (reward) function π. For this work, it is sufficient to consider discrete, finite S and A. At any given time step, an agent (such as a robot) acting in an MDP will be located at some state $s \in S$, where it can choose an action $a \in A$. The agent is subsequently relocated to a new state s' drawn from the transition probability distribution $T(s'|s,a) \equiv \mathbf{Pr}[q_{t+1} = s'|q_t = s, a]$, where q_t is a random variable indicating the agent's state at time step t. The transition function captures both the agent's stochastic dynamics (e.g., unreliable actuators) and structure and characteristics of the environment such as walls, pits, friction of the surface, etc.

As in the deterministic case, associated with each state is a (positive real) value, given by the function $\pi(s)$, which the agent receives each time it enters the state s.[1] For example, a package-delivery robot might get a reward every time it correctly delivers a package.

In certain planning problems, such as the TSP over an MDP, the robot does not obtain any additional benefit from visiting a state more than once, or may obtain lesser and lesser reward on subsequent visits. These problems can be formulated in the traditional MDP framework (where the robot gets the same reward every time it visits a state), by expanding the state space to represent not only the current position of the robot, but all the locations visited by it in the past. Unfortunately, this increases the state size by an exponential factor. So instead, we consider MDPs with reward functions that change over time. For example, the reward may decrease over time, or may become zero after the first visit to the state.

3.1.2 Strategies and policies

A solution to a Markov decision process is a strategy for the robot, that specifies the action that the robot should take given its current state and the history of its actions and outcomes. Formally, let $Q_t = \{q_i\}_{1 \le i < t}$ denote the set of states visited by the robot until time i. Then, the strategy of a robot is a function $\psi : S \times S^{t-1} \to A$ that maps a pair (q_t, Q_t) to an action a_t. More generally, we may allow the function ψ to specify a distribution over actions, rather than a single action: $\psi : S \times S^{t-1} \times A \to [0,1]$. The next state q_{t+1} is picked from the distribution $T(s|q_t, \psi(q_t, Q_t))$.

A strategy that picks actions based solely on the current state of the robot, and not on its history, is known as a *static policy*, or a policy for short. Strategies that depend on the history are known as *dynamic strategies*, or just strategies for short. In comparison to dynamic strategies, policies trade-off simplicity of expression for a degradation in performance. For MDPs with time-dependent rewards, taking history into account while picking an action can obtain much better performance than ignoring the history. We give some examples of this in Section 3.3.2.

[1] It is also possible to model rewards associated with actions or transitions by writing more general reward functions such as $\pi(s,a)$ or $\pi(s,a,s')$, but such extensions do not fundamentally change the nature of the MDP. Any such functions can be rewritten into a model of the form we give here with an appropriate modification to the state and action sets.

On the other hand, a strategy that depends on history may take super-polynomial space to specify completely, while a policy only takes linear space to express. Note also that, because a fixed (s, a) pair yields a fixed probability distribution over next states, the combination of an MDP with a fixed policy produces a Markov chain over S, making it easier to analyze its properties. Therefore, policies may be easier to pre-compute and implement in practice than strategies.

In this chapter, we consider optimization over the class of static as well as dynamic strategies. We also consider the question of how much we lose in terms of the value of the objective function by using a static policy instead of a strategy. This ratio of the performance of the optimal static policy to the performance of the optimal strategy is known in literature as the *Adaptivity Gap* [52, 53].

3.1.3 Objectives

The goal of planning in an MDP framework is to formulate a strategy that guides the agent to optimal long-term aggregate reward. This goal is formalized in the form of rewards at every state; Broadly, the objective is to maximize the average reward visited per time step, or the total reward visited until some terminating condition is achieved. Depending on whether or not the reward at a node stays the same independent of time, we can classify the corresponding objectives into those that are time-dependent and those that are not.

Time-independent objectives, that is, those where the reward at every state stays constant over time, have been studied extensively and can be solved optimally using static policies. There are well-known algorithms [27, 120, 131] for optimizing these objectives that take time polynomial in the cardinality of the state space. For example, consider the objective of reaching a specific state as fast as possible. Here we can assign a reward of 1 at the goal state, and 0 at any other state. In this case, it is easy to see that the best strategy for the robot is a history-independent policy. This *stochastic shortest path* problem can be solved easily using a linear program, or using faster techniques such as dynamic programming or value iteration (see [27] for a review). Likewise, consider a "reinforcement-learning" problem, where the goal of the robot is to learn how to perform "good actions" repeatedly, without performing "bad actions". In this case, the good actions are assigned reward 1, the remaining actions are assigned reward 0, and the goal is to maximize the average reward collected at every time step. Again in this case, the best strategy for the robot is a policy, and it can be found using techniques such as dynamic programming[2].

In the related *infinite horizon discounted reward* [91, 120, 131] objective, the reward at every location decreases exponentially with time; the goal is to obtain as much reward as possible in expectation. This optimality criterion guides the robot to accumulate as much reward as possible in

[2]The reinforcement-learning framework also deals with the problem of exploring an unknown environment. The goal is to reconstruct or "learn" the unknown MDP by trying different actions and observing the outcomes, while at the same time obtaining a large reward per time step. This can be done using algorithms such as "Q-learning" [91]; Finding the best strategy given a partial reconstruction of the MDP is a subroutine in such algorithms.

a timely manner, and produces what in practice turns out to be good behavior. Although outwardly this appears to be a time-dependent objective function, owing to the constant rate of decrease, the optimal strategy for this objective is a policy [24], and can be found via linear programming or through techniques such as value iteration or policy iteration [91].

In this chapter we consider a class of time-dependent objectives, in particular, those where the reward at a state disappears after the robot first visits the state. These objectives model one-time tasks such as delivering items or collecting samples. One way of optimizing these objectives while using existing techniques is to expand the state space to represent not only the current location of the robot, but also a record of all the sub-goals (reward to be collected, or packages to be delivered) that it has already covered. For example, one could write $S = L \times 2^d$, where L is a set of discrete locations that the robot could occupy, and the list of d bits tracks whether the agent has achieved each of d sub-goals. Then the new reward function can be $\pi(\langle l, b_1, \ldots, b_d \rangle) = 1$ iff l is a location containing sub-goal i and $b_i = 0$, or $R(s) = 0$ otherwise. When the robot reaches the location containing sub-goal i, b_i is set to 1 and remains so thereafter. This formulation yields an exponential increase in the size of the state space over the raw cardinality of L thereby preventing a fast, exact solution of the MDP. Instead it would be preferable to directly model the case of rewards that are given only the *first* time a state is visited.

As a first step towards tackling this general problem, we abandon the stochastic element and restrict the model to deterministic, reversible actions. Consider for example, the infinite horizon discounted reward objective, albeit with the difference that the reward at a state disappears the first time the state is visited. This problem, that we call the DISCOUNTED-REWARD-TSP, is interesting even in a deterministic context, where each action available to the robot is a deterministic action—it results in a specific state with probability 1. This model is a reasonable approximation to many robot-navigation style MDP domains, in which we can formulate sub-policies for navigating between pairs of locations in the environment. Often, such sub-policies, or macros, can be "nearly deterministic" (failing with probability $\leq \epsilon$) because they average out the stochasticity of atomic actions over many steps [106]. We can to a good approximation, therefore, treat such a domain as a deterministic planning problem over the set of sub-goal locations (nodes) and location-to-location macros (arcs).

Getting back to the original MDP model, we consider the objective of minimizing the expected time taken to visit all the states. We call this problem the STOCHASTIC-TSP. In the deterministic case, this simply becomes the ASYMMETRIC-TSP (ATSP). The ATSP is NP-hard and can be approximated within a factor of $O(\log n)$ [67]. In the stochastic case, we may consider finding an approximately-optimal dynamic strategy, or an approximately-optimal static policy for this problem. We show below (Section 3.3.2) that the expected time taken by the two optimal policies differs by a factor of at most poly(n).

Note that when the solution to the problem is a static policy, the expected length of the TSP tour is simply the cover-time of the corresponding Markov chain. The problem is, therefore, also related to the problem of estimating the cover-time of a Markov chain in directed graphs. Estimating the cover-time of a Markov chain, and more specifically that of the random walk on an undirected graph, has been a long-open problem. A simple way of doing this is to simulate the Markov chain several times and take the average of the cover times obtained in each execution. This algorithm is reasonably fast for undirected graphs, but can take exponentially long in the case of general Markov chains (as some chains have exponential cover times). In the case of undirected graphs, one can instead use quantities such as the *effective-resistance* of the graph to estimate the cover time to within some approximation. The best such approximation known to date is an $O((\log\log n)^2)$-approximation, and can be computed deterministically in polynomial time [93]. In the case of general Markov chains, an $O(\log n)$ approximation can be obtained via hitting times[3], and we are not aware of any better results. Another related problem is that of finding a static policy in a graph with the fastest mixing time[4]. This problem can be cast as a convex optimization problem and can be solved *exactly* using semi-definite programming [34]. We discuss the connection between mixing time and cover time in Section 3.3.3.

3.1.4 Stochastic optimization

Optimization problems involving stochasticity have been studied extensively in Operations Research as well as approximation algorithms (see, for example, [29] and [50]). One way of classifying these problems is by the levels of recourse available to the algorithm, that is, the sequence in which outcomes of stochastic processes is revealed and the algorithm is allowed to make choices.

A popular model of recourse studied extensively recently from the point of view of approximation is the fixed-stage recourse model [80, 82, 86, 123, 129]. In the 2-stage recourse model, the algorithm first makes some choices in the first stage; then the instantiations of the random variables are revealed and the algorithm is allowed to take *recourse actions* to fix the solution accordingly, albeit at a higher cost. This model can be extended to multiple (but fixed number of) stages, where in each stage the algorithm first makes some choices, then a few random variables are revealed. A recent result of Shmoys and Swamy [129] shows for a class of optimization problems (such as set cover or Steiner tree), that if a problem in the class has a β-approximation algorithm in the deterministic case, then in the 2-stage recourse model it can be approximated within a factor of $2\beta + \epsilon$. Hayrapetyan et al. [84] and Gupta et al. [81] independently extended this result to multiple stages for the Stochastic Steiner tree problem and given an $O(k)$-approximation for it in the k-stage recourse model.

Note that our problem can also be cast in the recourse model—at each step, the algorithm com-

[3]The hitting time from a state u to a state v is the expected time taken to go from u to v in a simulation of the Markov chain.

[4]Informally, the mixing time of a static policy, or a Markov chain, is the time taken by the policy to converge to its stationary distribution. See [117] for details.

mits to a part of the solution by picking an action, following which, an outcome is revealed. However, results analogous to that of Hayrapetyan et al. and Gupta et al. may not apply to our context, because the number of stages in our problems is variable, and may be as large as exponential in the number of states[5].

Among approximation algorithms for stochastic optimization problems, the work that relates most closely to ours is that of Dean, Goemans and Vondrak [52, 53], studying the stochastic knapsack problem. In the stochastic knapsack problem, one is given a number of item each with a random size and a value. The goal is to pack items into a knapsack of size 1, obtaining the maximum possible value. The size of an item gets instantiated when the algorithm decides to put that item in the knapsack. Dean et al. show that the adaptivity gap of this problem is a constant, and using this result they give a constant approximation to the optimal dynamic strategy for this problem. The stochastic knapsack problem can be modeled as a STOCHASTIC-ORIENTEERING problem, where the distance between the root and any node (representing an item) is a random variable, and the goal is to maximize the reward obtained in expectation by a fixed deadline. The results of Dean et al., however, do not generalize to STOCHASTIC-ORIENTEERING—the adaptivity gap for the latter can be as large as $\Omega(n)$.

3.1.5 Our Results

We study the DISCOUNTED-REWARD-TSP problem on deterministic undirected graphs, and present the first approximation—a 6.75-approximation—for it. Our algorithm is based on the observation that the optimal solution to this problem must collect a large fraction of its reward in a prefix that has constant excess. We then use the MIN-EXCESS-PATH algorithm developed in Chapter 2 as a subroutine to approximate this problem.

Next we study the STOCHASTIC-TSP and show that the adaptivity gap of this problem is bounded by $O(n^3 \log n)$. In the case of deterministic (directed or undirected) graphs, the adaptivity gap reduces to $\Theta(n)$. We then convert this gap into an $O(n^3 \log n)$-approximation algorithm that outputs a static policy for the problem. We also present a simple n-approximation for the problem that outputs a dynamic strategy for it.

3.2 Maximum Discounted-Reward Path

We now consider the infinite-horizon discounted-reward model in which the reward at every location decreases exponentially with time. One can motivate this exponential discounting by imagining that, at each time step, there is some fixed probability the game will end (the robot loses power, a catastrophic failure occurs, the objectives change, etc.) For another example, suppose we are

[5] In fact, our n-approximation for the STOCHASTIC-TSP is stronger than what may be obtained for the multi-stage recourse model in general, as the number of stages in our problem is at least n.

searching for a lost item, and at each time step there is some possibility the item will be taken (or, if we are searching for a trapped individual in a dangerous environment, and at each time step there is some probability γ the individual might die). Then, if the probability of finding the item at a location v at time 0 is $\pi(v)$, the probability of finding it at time t is $\pi(v)\gamma^t$. We can model this by placing rewards $\pi(v)$ at locations v, and *discounting* the reward collected for a site reached at time t by γ^t, where γ is a fixed and known discount factor.

Exponential discounting is often used in the MDP setting, as mentioned before. In this setting, it also has the nice mathematical property that it is *time-independent*, meaning that an optimal strategy is *stationary* and can be completely described by the mapping from states to actions given by ψ.[6] However, we will be considering the case that the reward at a node can be collected only the first time the robot visits the node (as in searching for a lost item, or a trapped individual). In this case, the optimal strategy may not be stationary policy.

As a first step towards solving this problem, we study it in the deterministic case. We now describe some notation. Let $\gamma \in (0, 1)$ denote the *discount factor* in the problem. Then the value or reward collected at a location at time t is discounted by a factor γ^t. Given a path P rooted at s, let the *discounted reward* collected at node v by path P be defined as $\rho_v^P = \pi(v)\gamma^{\ell_P(v)}$, where $\ell_P(v)$ is the first time at which P visits v. The *max-discounted-reward path* problem is to find a path P rooted at s, that maximizes $\rho^P = \sum_{v \in P} \rho_v^P$. We call this the DISCOUNTED-REWARD-TSP. Note that the length of the path is not specifically bounded in this problem, though of course shorter paths produce less discounting.

In this section we present an approximation algorithm for the DISCOUNTED-REWARD-TSP which builds upon our min-excess path algorithm. We assume without loss of generality that the discount factor is $\gamma = 1/2$—we simply rescale each length ℓ to ℓ' such that $\gamma^\ell = (\frac{1}{2})^{\ell'}$, i.e., $\ell' = \ell \log_2(1/\gamma)$.

We first establish a property of an optimal solution that we make use of in our algorithm: informally, we show that any optimal path must collect a large fraction of its discounted-reward in a prefix with low (constant) excess; we approximate this prefix using our MIN-EXCESS-PATH algorithm, losing only a constant in the length of the prefix, and therefore in the discount applied to the reward.

Define the *scaled value* π' of a node v to be the (discounted) reward that a path gets at node v if it follows a shortest path from the root to v. That is, $\pi'_v = \pi_v \gamma^{d_v}$. Let $\Pi'(P) = \sum_{v \in P} \pi'_v$. Note that for any path P, the discounted reward obtained by P is at most $\Pi'(P)$.

Now consider an optimal solution P^*. Let ϵ be a parameter that we will set later. Let t be the last node on the path P^* for which $\ell_{P^*}(t) - d_t \leq \epsilon$, i.e., the excess of path P^* at t is at most ϵ. Consider the portion of P^* from root s to t. Call this path P_t^*.

---

[6] Under other objective functions, an optimal policy could require dependence on the number of steps remaining in the game or other functions of the history of states encountered to date.

Input: Graph $G = (V, E)$; special start node s; discount factor γ.
Output: Path P starting from s with $\rho(P) \geq (6.75 + \delta)\rho(P^*)$.

1. Re-scale all edge lengths so that $\gamma = 1/2$.
2. Replace the value of each node with the value discounted by the shortest path to that node: $\pi'_v = \gamma^{d_v}\pi_v$. Call this modified graph G'.
3. Guess t—the last node on optimal path P^* with excess less than ϵ.
4. Guess k—the value of $\Pi'(P^*_t)$.
5. Apply our MIN-EXCESS-PATH approximation to find a path P collecting scaled value k with small excess.
6. Return this path as the solution.

Figure 3.1: A $(6.75 + \delta)$-approximation for DISCOUNTED-REWARD-TSP

Lemma 3.1. *Let P^*_t be the part of P^* from s to t. Then, $\rho(P^*_t) \geq \rho(P^*)(1 - \frac{1}{2^\epsilon})$.*

Proof. Assume otherwise. Suppose we shortcut P^* by taking a shortest path from s to the next node visited by P^* after t. This new path collects (discounted) rewards from the vertices of $P^* - P^*_t$, which form more than a $\frac{1}{2^\epsilon}$ fraction of the total discounted reward by assumption. The shortcutting procedure decreases the distance on each of these vertices by at least ϵ, meaning these rewards are "undiscounted" by a factor of at least 2^ϵ over what they would be in path P^*. Thus, the total reward on this path exceeds the optimum, a contradiction. □

It follows that we can approximate $\rho(P^*)$ by approximating $\rho(P^*_t)$. Based on the above observation, we give the algorithm of Figure 3.1 for finding an approximately optimal solution. Our algorithm uses the min-excess algorithm to approximate the path $\rho(P^*)$. Because the excess of this path is a constant, the extra discount on the reward, apart from the discount accounted for by the function Π', is at most a constant factor. Note that in the algorithm, "guess t" and "guess k" are implemented by exhausting all polynomially many possibilities.

Our analysis below proceeds in terms of $\alpha = \alpha_{\text{ex}}$, the approximation factor for our min-excess path algorithm.

Lemma 3.2. *Our approximation algorithm finds a path P that collects discounted reward $\rho(P) \geq \Pi'(P^*_t)/2^{\alpha\epsilon}$.*

Proof. The prefix P^*_t of the optimum path shows that it is possible to collect scaled value $k = \Pi'(P^*_t)$ on a path with excess ϵ. Thus, our approximation algorithm finds a path collecting the same scaled value with excess at most $\alpha\epsilon$. In particular, the excess of any vertex v in P is at most $\alpha\epsilon$.

Thus, the discounted reward collected at v is at least

$$\rho(v) \geq \pi_v \left(\frac{1}{2}\right)^{d_v+\alpha\epsilon} = \pi_v \left(\frac{1}{2}\right)^{d_v} \left(\frac{1}{2}\right)^{\alpha\epsilon} = \pi'_v \left(\frac{1}{2}\right)^{\alpha\epsilon}$$

Summing over all $v \in P$ and observing $\Pi'(P) \geq \Pi'(P^*)$ completes the proof. □

Combining Lemma 3.2 and Lemma 3.1, we get the following:

Theorem 3.3. *The solution returned by the above algorithm has* $\rho(P) \geq (1 - \frac{1}{2^\epsilon})\rho(P^*)/2^{\alpha\epsilon}$.

Proof.

$$\begin{aligned} \rho(P) &\geq \Pi'(P^*)/2^{\alpha\epsilon} && \text{by Lemma 3.2} \\ &\geq \rho(P_t^*)/2^{\alpha\epsilon} && \text{by definition of } \pi' \\ &\geq \left(1 - \frac{1}{2^\epsilon}\right)\rho(P^*)/2^{\alpha\epsilon} && \text{by Lemma 3.1} \end{aligned}$$

□

We can now set ϵ as we like. Writing $x = 2^{-\epsilon}$ we optimize our approximation factor by maximizing $(1-x)x^\alpha$ to deduce $x = \alpha/(\alpha+1)$. Plugging in this x yields an approximation ratio of $(1+\alpha_{\text{ex}})(1+1/\alpha_{\text{ex}})^{\alpha_{\text{ex}}} \approx 6.75 + \delta$.

3.3 Stochastic TSP

In this section we consider the STOCHASTIC-TSP. In this problem, the goal is to produce a strategy that takes the (approximately) minimum time in expectation to visit all states in the MDP. Note that this problem generalizes the ASYMMETRIC-TSP (TRAVELING SALESMAN PROBLEM on directed graphs), and is therefore NP-hard. In this section we study the complexity as well as approximability of this problem. We consider approximating this problem via both static as well as dynamic strategies.

We also consider the question of how much worse the best static policy can be with respect to the best dynamic strategy. We bound this *adaptivity gap* in Section 3.3.2. Determining the adaptivity gap of STOCHASTIC-TSP is interesting even when we restrict the problem to deterministic graphs, that is, when every action in the underlying MDP is a deterministic action with a single outcome. In that case, as noted earlier, the STOCHASTIC-TSP simply becomes the TRAVELING SALESMAN PROBLEM. Note that an optimal solution to the TRAVELING SALESMAN PROBLEM may pass through some vertices more than once. In this case, when the solution is at some such vertex, the outgoing edge followed by it next depends on the number of times it has visited this node previously. In our terminology, it is therefore a dynamic (adaptive) solution. A static policy in this case is allowed to place probabilities on outgoing edges, such that the resulting Markov chain has a small cover-time.

We extend the Markov-chain terminology to dynamic strategies, and use the term "cover-time" to denote the expected time taken by a strategy to visit all the states. The cover-time of a strategy ψ is denoted $C(\psi)$.

We begin by examining the complexity and the adaptivity gap of the problem, following which we develop some simple approximations to it.

3.3.1 The complexity of computing the optimal strategy

In studying the complexity of the STOCHASTIC-TSP we are concerned with the following decision question — "Does there exist a strategy with cover-time at most t?" Note that t may be exponential in n. So in order to answer this question in polynomial time (or even for this question to be in NP), we may not be able to execute a candidate strategy and note its cover-time; Instead we must compute its cover-time analytically.

In some cases, it may take a long time to pre-compute and specify an optimal strategy or compute its cover-time before executing it, however, it may be easier to compute the next step in the strategy based on the current location and history. Suppose that for an approximation or exact solution, there is an algorithm that takes as input a state and history and outputs an action to be executed, and lies in complexity class $\mathcal{C}$. Then we say that the approximation or exact solution can be *i-computed*[7] *in class* $\mathcal{C}$.

For example, we show in Section 3.3.4 there exists a polynomial-time n-approximation to the STOCHASTIC-TSP. On the other hand, we show below that the optimal solution to STOCHASTIC-TSP can be i-computed in PSPACE. Note that the optimal strategy may take exponentially many bits to specify, so the problem may not lie in PSPACE.

Theorem 3.4. *The optimal solution to* STOCHASTIC-TSP *can be i-computed in PSPACE.*

Proof. Figure 3.2 gives an algorithm for computing a solution to STOCHASTIC-TSP iteratively. From the description of the algorithm, it is immediate that the algorithm lies in PSPACE. We now prove that the algorithm produces an optimal solution to STOCHASTIC-TSP.

Suppose that for all $s' \in S \setminus X$ the recursive calls to ITER$(X \cup \{s'\}, s')$ return the correct value $\tau_{s'}$ of the expected time taken to cover the remaining nodes. Then, a setting of the τ_u variables for $u \in X$, for which the following equations hold, must give the expected time to visit all the remaining states from the node u.

$$\tau_u = \min_{a \in A} \left\{ \ell(u,a) + \sum_{v \in S} \tau_v T(v|u,a) \right\} \quad \forall u \in X$$

[7] The "i" stands for iterative.

Input: An MDP on state space S and actions A with transition function T; a set of states $X \subset S$ already visited and a current state $s \in X$.
Output: Action $a \in A$ to be taken from s and an estimate of the expected time $\tau = \text{ITER}(X, s)$ to visit all nodes.

---

1. For all $s' \in S \setminus X$, compute $\tau_{s'} = \text{ITER}(X \cup \{s'\}, s')$.
2. For $s' \in X$, let $\tau_{s'}$ denote the expected time taken to visit all nodes given that the next action results in state s'. Solve the following linear program:

$$\max \quad \tau_s \quad \text{s.t.} \quad \tau_u \;\leq\; \ell(u,a) + \sum_{v \in S} \tau_v T(v|u,a) \quad \forall u \in X, a \in A$$

   Here $\ell(u, a)$ is the length (cost) of action a starting from state u.
3. Let $\tau = \text{ITER}(X, s)$ be the value of the above linear program, and a be the action for which $\tau_s = \ell(s,a) + \sum_{v \in S} \tau_v T(v|s,a)$. Output a and τ.

Figure 3.2: Exact iterative PSPACE algorithm ITER **for** STOCHASTIC-TSP

Suppose that in the solution to the above linear program, there is a variable τ_u for which the above equation does not hold, then we can increase the value of that variable. This increase propagates to other constraints in the LP and thus to other variables, and eventually we obtain a solution with a larger τ_s value. Then such a solution cannot be the optimal solution to the linear program. Therefore, the solution to the linear program returns correct values for the expected completion times. □

3.3.2 The adaptivity gap

The following example shows that the gap of adaptivity for STOCHASTIC-TSP can be as high as $\Omega(n)$ even for deterministic undirected graphs, where n is the number of states in the graph.

Example 3.5. Let G be a line graph containing n nodes $v_1, \cdots, v_n$. That is, for every $1 \leq i < n$, there is a unit-length edge between v_i and v_{i+1}. Then, the optimal TRAVELING SALESMAN PROBLEM tour on this graph has length $2(n-1)$. However, any Markov chain on this graph has cover time at least $\Omega(n^2)$, with the best one being a random walk on the graph [117]. Therefore, the adaptivity gap of TRAVELING SALESMAN PROBLEM on this graph is $\Omega(n)$.

A natural question to ask is whether the above example represents the worst case for the adaptivity gap. We now show that this is indeed the case for deterministic undirected graphs. We also show that the gap cannot be worse than $O(n \log n)$ for deterministic directed graphs, and $O(n^3 \log n)$ for general MDPs. However, we do not have any examples exhibiting a gap larger than $O(n)$ for the latter two cases.

Theorem 3.6. *The adaptivity gap of* STOCHASTIC-TSP *in any deterministic graph is at most $O(n \log n)$. In undirected graphs this gap is at most $O(n)$.*

Proof. Let $G(V, E)$ be the graph under consideration, and let T be the optimal TRAVELING SALESMAN PROBLEM tour in this graph. We assume for now that G is unweighted, that is, every edge in G has length 1. We later show how to remove this assumption. Let $G[T] = (V, T)$ denote the Eulerian graph defined by T—the edge set of this graph consists of all edges traversed by T. Note that the out-degree of every node in this graph is equal to its in-degree. Let R denote the random walk on this graph; this walk picks an out-going edge uniformly at random at every vertex. Standard results [117] now imply that if $G[T]$ is undirected, the cover time of R is at most $2n|T|$. In directed graphs, note that the hitting time from any node u to any node v is at most $n|T|$. Now, if we start at some node u and run the random walk R for $2n|T|(1 + \kappa) \log n$ steps, the probability that some node is not visited is at most $n^{-\kappa}$. Therefore, the cover time of the walk is at most $O(n|T| \log n)$. The result now follows from noting that the length of the optimal tour is $|T|$.

Finally, let us consider the case of weighted graphs. The above proof uses the fact that in the graph $G[T]$, for any two adjacent nodes u and v, the hitting time from u to v is at most $|T|$. We first note that this is (nearly) the case for weighted graphs as well. Suppose that each edge has an integral length, and imagine breaking each edge of the graph with length ℓ into ℓ pieces of length 1, and running a random walk on this new graph. Then the new graph is unweighted and has $|T|$ edges ($|T|$ now denotes the total length of all edges in T). This implies that for nodes u and v that are adjacent in the old graph $G[T]$, the hitting time from u to v is at most $|T| + \ell(u, v)$. For non-adjacent nodes u and v, note that there is a path from u to v containing at most $n - 2$ intermediate nodes. The hitting time to go from u to v is therefore at most $(n - 1)|T| + \ell(u, v) \leq n|T|$. The remaining analysis follows as before. □

We now show a weaker result for general MDPs.

Theorem 3.7. *For any strategy ψ in an MDP with n states, there exists a policy $\tilde{\psi}$ such that $C(\tilde{\psi}) = O(n^3 \log n) C(\psi)$.*

Proof. As in the proof of the previous theorem, we assume that the graph is unweighted, for simplicity. The weighted case can be proved analogously. We define a probability distribution p on state-action pairs (s, a) based on the strategy ψ, and consider the static policy $\tilde{\psi}$ that for every location s picks an action a with probability proportional to $p(s, a)$. The distribution p has the following property: if we extend it to state-outcome pairs (or "edges")—$p_e(s, s') = \sum_{a \in A} p(s, a) T(s'|s, a)$—then we have $\sum_s p_e(s, s') = \sum_s p_e(s', s)$. That is, p_e is the stationary distribution on edges (state-outcome pairs) defined by the policy $\tilde{\psi}$.

The distribution p is defined as follows. Consider the set $\mathbf{E}$ of all possible executions of ψ (note that these are all possible tours on S containing all the states—possibly multiple times), and let q_ψ

denote the probability distribution defined by ψ on this set. That is, for an execution $E \in \mathbf{E}$, $q_\psi(E)$ denotes the probability that executing ψ resulted in the tour E.

Let $\#_E(s,a)$ denote the number of times the action a was executed from state s in E, $\#_E(s,s')$ denote the number of times the state s' followed the state s in E, and $\#_E(s)$ denotes the number of times the state s was visited in E. We define p as

$$p(s,a) = \sum_{E \in \mathbf{E}} q_\psi(E) \frac{\#_E(s,a)}{C(\psi)}$$

Note that p is a probability distribution, as $\sum_{s,a} \sum_{E \in \mathbf{E}} q_\psi(E) \#_E(s,a) = \sum_{E \in \mathbf{E}} q_\psi(E)|E| = C(\psi)$, where $|E| = \sum_{s,a} \#_E(s,a)$ denotes the length of E. In fact, roughly speaking, $p(s,a)$ is the probability that the action a is taken from state s in a typical execution of ψ. Furthermore, it is immediate[8] that

$$p_e(s,s') = \sum_{E \in \mathbf{E}} q_\psi(E) \frac{\#_E(s,s')}{C(\psi)}$$

Finally, we can show that p_e is a stationary distribution, by noting that,

$$\begin{aligned}
\sum_s p_e(s,s') &= \sum_{E \in \mathbf{E}} q_\psi(E) \frac{\sum_s \#_E(s,s')}{C(\psi)} \\
&= \sum_{E \in \mathbf{E}} q_\psi(E) \frac{\#_E(s')}{C(\psi)} \\
&= \sum_{E \in \mathbf{E}} q_\psi(E) \frac{\sum_s \#_E(s',s)}{C(\psi)} = \sum_s p_e(s',s)
\end{aligned}$$

Having defined the distributions p and p_e, we now analyze the stationary policy $\tilde{\psi}$. We show that for any pair of states u and v, there is a path from u to v containing edges that have a high probability under the stationary distribution p_e of $\tilde{\psi}$. Informally, this means that the time taken by $\tilde{\psi}$ to follow this path starting from u is small. This gives us an upper bound on the hitting time $h_{\tilde{\psi}}(u,v)$, and allows us to derive a bound on the cover time using standard arguments [117].

Consider a directed graph G_ψ on states as follows. A pair (s,s') is a directed edge in G_ψ iff $p_e(s,s') \geq \frac{4}{C(\psi)n^2}$.

Claim 3.8. *G_ψ is strongly connected.*

[8] Another way of looking at the probability distributions p and p_e is to consider the computation tree of ψ with edges at alternate levels denoting actions and outcomes respectively. Suppose that each edge is labeled with the probability that it is encountered in an execution of ψ. Then the probability $p_e(s,s')$ is simply the sum over the labels of all edges of the form $a \to s'$ preceded by edges $s \to a$, divided by $C(\psi)$. Likewise, the probability $p(s,a)$ is the sum over all edges of the form $s \to a$, divided by $C(\psi)$. But the label of an edge $a \to s'$ is simply $T(s'|s,a)$ times the label of the preceding edge $s \to a$. Therefore the statement follows.

Proof. Suppose, for the sake of contradiction, that G_ψ is not strongly connected. Then there are sets X and $S \setminus X$ such that there is no edge in G_ψ from X to $S \setminus X$. Note that for all executions E, $\sum_{s \in X, s' \notin X} \#_E(s, s') \geq 1$, because E is a tour containing all the states. Then,

$$\begin{aligned}\sum_{s \in X, s' \notin X} p_e(s, s') &= \sum_{E \in \mathbf{E}} q_\psi(E) \frac{\sum_{s,s'} \#_E(s, s')}{C(\psi)} \\ &\geq \sum_{E \in \mathbf{E}} q_\psi(E) \frac{1}{C(\psi)} = \frac{1}{C(\psi)}\end{aligned}$$

This along with the fact that there are at most $n^2/4$ pairs (s, s') implies that for at least one pair (s, s'), $p_e(s, s') \geq \frac{4}{C(\psi)n^2}$. □

The above claim implies that for any two states u and v in S, there is a path from u to v such that each edge in the path has probability at least $\frac{4}{C(\psi)n^2}$ under the stationary distribution of $\tilde{\psi}$. Let this path be $u = s_1, s_2, \cdots, s_m = v$, where $m \leq n$. Note that, given that we are currently at state s_i, the expected time taken to go to the state s_{i+1} is at most the expected time taken to follow the edge (s_i, s_{i+1}), which is[9] at most $1/p_e(s_i, s_{i+1}) \leq \frac{1}{4}C(\psi)n^2$. Therefore, the expected time taken to reach v starting from u is at most the expected time taken to go to s_2, then to s_3, and so on to $s_m = v$, which is at most $\frac{1}{4}C(\psi)n^3$.

Now consider running the policy $\tilde{\psi}$ in phases, each of $C(\psi)n^3$ steps. In any phase, for any state s', the probability that we have not visited s' in this phase is at most $\frac{1}{4}$. Therefore, we expect to see at least $3/4$th of the states not yet visited, in this phase. Call a phase "bad" if we see fewer than $1/2$ of the remaining states in that phase, and "good" otherwise. Note that the probability that a phase is bad is at most $1/2$. Furthermore, we need at most $\log n$ "good" phases to see all the states in S. Therefore the expected number of phases, before we have seen all the states is $2 \log n$, and so we get $C(\tilde{\psi}) \leq 2C(\psi)n^3 \log n$.

This concludes the proof of the theorem. □

The theorem implies the following statement on the adaptivity gap of STOCHASTIC-TSP.

Corollary 3.9. *The adaptivity gap of* STOCHASTIC-TSP *in any MDP is at most* $2n^3 \log n$.

3.3.3 Approximation via static policies

We showed in the previous section that for any MDP, there exists a static policy with cover time within a factor of $O(n^3 \log n)$ of the cover time of the optimal strategy. We now show that we can in fact achieve this approximation in polynomial time. The proof of Theorem 3.7 suggests a natural

[9] To see this, suppose that the probability of state s_i under the stationary distribution defined by $\tilde{\psi}$ is $p_v(s_i) = \sum_{s'} p_e(s_i, s')$. Then the expected time taken to return to s_i, starting from s_i is $1/p_v(s_i)$, while the expected number of returns to s_i before we take the edge (s_i, s_{i+1}) is $p_e(s_i, s_{i+1})/(\sum_{s'} p_e(s_i, s')) = p_e(s_i, s_{i+1})/p_v(s_i)$.

way of achieving this. Recall that in the proof of the theorem, under the stationary distribution of the policy $\tilde{\psi}$, the total probability of edges from some set X to $S \setminus X$ is at least $1/C(\psi)$. We now construct an (exponential-sized) LP that finds such a distribution:

$$
\begin{array}{lll}
\max \quad P \quad \text{s.t.} & & \\
\sum_{s\in S, a\in A} p_{s,a} & = & 1 \\
p_{s,s'} & = & \sum_{a\in A} T(s'|s,a) p_{s,a} \quad \forall s,s' \in S \\
\sum_{s'\in S} p_{s,s'} & = & \sum_{s'\in S} p_{s',s} \quad \forall s \in S \\
\sum_{s\in X, s'\in S\setminus X} p_{s,s'} & \geq & P \quad \forall X \subset S \\
p_{s,a} & \geq & 0 \quad \forall s \in S, a \in A
\end{array}
$$

Note that the optimal solution P^* to the above linear program has value at least $1/C(\psi)$. Furthermore, the proof of Theorem 3.7 implies that the policy corresponding to the probability distribution found by the linear program has cover time at most $O(n^3 \log n \frac{1}{P^*}) = O(n^3 \log n\, C(\psi))$. Therefore, modulo solving the linear program, we can approximate the cover time of $C(\psi)$ to within a factor of $O(n^3 \log n)$ using this approach.

The linear program above has exponentially many constraints. However, note that given a potential solution, we can easily find a violated constraint in the LP. The first three sets of constraints can be checked in polynomial time. For the fourth set of constraints, we set up a complete directed graph on the set of states, and place a weight of $p_{s,s'}$ on edge (s, s'); we then find the minimum directed cut in this graph; if the value of this cut is less than C, then the corresponding constraint is a violated constraint. Given this separation oracle, we can use the ellipsoid algorithm to solve the linear program. We get the following theorem:

Theorem 3.10. *There is a polynomial-time algorithm that given an MDP finds a static strategy with cover-time at most $O(n^3 \log n)$ times that of the optimal strategy.*

The objective of the above linear program is to find a policy that maximizes the transition probability from each set to its complement. This seems very similar to the goal of finding a policy with a large *conductance*[10]. The conductance of a policy is closely related to its mixing time. So this suggests that finding a policy with a small mixing time may lead to a good approximation for the cover time. (As noted previously, this latter problem can be solved exactly via semi-definite programming [34].) We however do not know of any means of proving such a result. In particular, it is easy to see that a policy which has the uniform distribution as its stationary distribution and has a mixing time of T, has a cover time at most $O(Tn \log n)$. However, given a strategy with a small cover time, we do not know how to derive a policy with a small mixing time that has as its stationary distribution a close-to-uniform distribution.

[10]The conductance of a policy is the minimum over all subsets containing less than half the probability mass, of the transition probability from the subset to its complement, divided by the probability mass of the subset.

3.3.4 Approximation via dynamic strategies

Our approximation to the optimal strategy using static policies is limited by the adaptivity gap of the problem. We therefore investigate approximations via dynamic strategies. A natural place to start exploring good dynamic solutions is the special case of ASYMMETRIC-TSP. Recall that the ASYMMETRIC-TSP can be approximated to within an $O(\log n)$ factor [67]. A natural approach for extending an approximation to ASYMMETRIC-TSP to the stochastic case is the following:

- For every pair of states (s, s'), compute the static policy $\phi_{s,s'}$ with the smallest expected time for going from the state s to the state s'; let this expected time be denoted $|\phi_{s,s'}|$.
- Set up a complete directed graph on the set of states, with each edge (s, s') assigned the length $|\phi_{s,s'}|$.
- Solve ASYMMETRIC-TSP on the graph G; let the resulting tour be $s_1, s_2, \cdots, s_n, s_1$.
- Follow policy ϕ_{s_1,s_2} until arriving at s_2. Then follow policy ϕ_{s_2,s_3} until arriving at s_3, and so on.

The approximation factor achieved by the above algorithm is at least $\Omega(\log n)$, as this is the best approximation currently know for ASYMMETRIC-TSP. However, the following example shows that in fact this factor can be as large as $\frac{n-1}{\log n}$, even if we can solve the ASYMMETRIC-TSP exactly.

Example 3.11. Consider the following MDP on a state space S with n states. From each state s, there are n actions available. The first of these has length 1 and picks a state uniformly at random from S. The remaining $n - 1$ are deterministic actions and are defined as follows—for all states $s' \neq s$, there is an action that has length $n - 1$ and results in the state s' with probability 1.

Then, the optimal strategy for this MDP always follows the first (randomized) action, and has cover time $n \log n$. However, the best policy for going from a state s to a state s' always takes the deterministic action with length $n - 1$. Therefore, each edge in the graph G has length $n - 1$ and the resulting tour has length $n(n - 1)$.

We now give a simple algorithm that obtains an n approximation to the optimal strategy:

Algorithm ARBITRARY-ORDER

1. Fix an arbitrary order on the states—$s_1, s_2, \cdots, s_n$
2. For every $i \leq n$, compute the static policy ϕ_i with the smallest expected time for going from the state s_i to the state s_{i+1}; let this expected time be denoted $|\phi_i|$.
3. Follow strategy ϕ_1 until s_2 is seen, then follow ϕ_2 until s_3 is seen, and so on.

Theorem 3.12. *The above algorithm* ARBITRARY-ORDER *gives an n-approximation to the cover-time of the optimal strategy.*

Proof. The proof follows by noting that for all $i \leq n$, $|\phi_i| \leq C(\psi)$. ☐

3.4 Concluding remarks

In this chapter we consider problems where the robot collects a reward only the first time it visits a location. Solving such "time-dependent" MDPs is an important problem in AI. Unfortunately, the well-understood and widely employed techniques of linear programming and dynamic programming do not give good solutions to these problems — the size of the linear program solving the problem exactly may be exponential, and a dynamic program may require an exponential number of iterations to give an accurate solution. In fact we are not aware of any technique for solving these problems, other than simulating the MDP, or solving specific instances analytically.

We present the first approximation algorithms for a class of time-dependent MDPs. Although our approximation factors are polynomial in the size of the problem, they are interesting and non-trivial, as the cost of the optimal solution may in fact be exponential in the size of the problem. However, it would be interesting to obtain poly-log approximations to these problems. Another interesting question is to resolve whether STOCHASTIC-TSP and time-dependent MDPs in general are in PSPACE or not.

Finally, it would be interesting to develop general techniques for converting algorithms for deterministic problems into their stochastic counterparts. This is not entirely inconceivable – it can be done, for example, for some 2-stage stochastic problems (see, for example, [80]).

Chapter 4

Correlation Clustering

4.1 Introduction

Consider the following problem arising in Natural Language Processing — we are given a document that refers to several people and/or entities, and, we wish to determine which of these references correspond to the same entity. For example, a news article may contain references to Queen Elizabeth by different strings such as "her majesty", "the Queen of England", and "she". Resolving these references, or *Coreference Analysis* [46, 47, 139, 140], is essential to the understanding and automated processing of language and speech. Problems such as this arise in several fields such as natural language processing, computer vision and databases (for example, automated resolution of citations). Unfortunately, in most of these applications, the dependencies between references are ambiguous, and not always consistent. Moreover, the same string may be used to reference different entities in different contexts.

Cohen and Richman [46, 47] formulate the coreference problem as a graph problem in the following way — based on past data, they "learn" a classifier $f(A, B)$, that given two references A and B, outputs whether or not it believes A and B are similar to each other. Then they construct a graph on references, with the edge between references A and B weighted by the output of the classifier $f(A, B)$. Now the problem is to cluster the graph, such that pairs of references that have a heavy positive weight edge between them lie in the same cluster, whereas those with a heavy negative edge between them lie in different clusters. In particular, we want to maximize the total weight of positive edges with both end points in the same cluster, and negative edges with end points in different clusters. We call this the CORRELATION-CLUSTERING problem.

Specifically, we consider the following problem. Given a fully-connected graph G with edges labeled "$+$" (similar) or "$-$" (different), find a partition of the vertices into clusters that agrees as much as possible with the edge labels. In particular, we can look at this in terms of maximizing *agreements* (the number of $+$ edges inside clusters plus the number of $-$ edges between clusters) or in terms of minimizing *disagreements* (the number of $-$ edges inside clusters plus the number of $+$ edges between clusters). These two are equivalent at optimality but, as usual, differ from the point of view of approximation. In this paper we give a constant factor approximation to the problem of minimizing disagreements, and a PTAS[1] for maximizing agreements. We also extend some of our results to the case of real-valued edge weights.

In practice, the classifier f typically would output a probability, in which case the natural edge label is log(Pr(same)/Pr(different)). This is 0 if the classifier is unsure, positive if the classifier believes the items are more likely in the same cluster, and negative if the classifier believes they are more likely in different clusters. The case of $\{+, -\}$ labels corresponds to the setting in which the classifier has equal confidence about each of its decisions.

What is interesting about the clustering problem defined here is that unlike most clustering formulations, we do not need to specify the number of clusters k as a separate parameter. For example, in k-median [36, 88] or min-sum clustering [127] or min-max clustering [85], one can always get a perfect score by putting each node into its own cluster — the question is how well one can do with only k clusters. In our clustering formulation, there is just a single objective, and the optimal clustering might have few or many clusters: it all depends on the edge labels. Note also that many standard formulations assume the underlying graph is a metric; we do not need to assume that the similarity measure is somehow consistent[2].

To get a feel for this problem, notice that if there exists a perfect clustering, i.e., one that gets all the edges correct, then the optimal clustering is easy to find: just delete all "$-$" edges and output the connected components of the graph remaining. (In [47] this is called the "naive algorithm".) Thus, the interesting case is when no clustering is perfect. Also, notice that for any graph G, it is trivial to produce a clustering that agrees with at least *half* of the edge labels: if there are more $+$ edges than $-$ edges, then simply put all vertices into one big cluster; otherwise, put each vertex into its own cluster. This observation means that for maximizing agreements, getting a 2-approximation is easy (note: we will show a PTAS). In general, finding the optimal clustering is NP-hard [43].

Another simple fact to notice is that if the graph contains a triangle in which two edges are labeled $+$ and one is labeled $-$, then no clustering can be perfect. More generally, the number of edge-disjoint triangles of this form gives a lower bound on the number of disagreements of the optimal clustering. This fact is used in our constant-factor approximation algorithm.

For maximizing agreements, our PTAS is quite similar to the PTAS developed by de la Vega [51] for MAXCUT on dense graphs, and related to PTASs of Arora et al. [13, 10]. Notice that since there must exist a clustering with at least $n(n-1)/4$ agreements, this means it suffices to approximate agreements to within an additive factor of ϵn^2. This problem is also closely related to work on testing graph properties of [76, 119, 6]. In fact, we show how we can use the General Partition

---

[1] A PTAS (polynomial-time approximation scheme) is an algorithm that for any given *fixed* $\epsilon > 0$ runs in polynomial time and returns an approximation within a $(1 + \epsilon)$ factor. Running time may depend exponentially (or worse) on $1/\epsilon$, however.

[2] For example, if references A and B are similar, and B and C are similar, A and C are not necessarily similar.

Property Tester of Goldreich, Goldwasser, and Ron [76] as a subroutine to get a PTAS with running time $O(ne^{O((\frac{1}{\epsilon})^{\frac{1}{\epsilon}})})$. Unfortunately, this is doubly exponential in $\frac{1}{\epsilon}$, so we also present an alternative direct algorithm (based more closely on the approach of [51]) that takes only $O(n^2 e^{O(\frac{1}{\epsilon})})$ time.

Relation to agnostic learning: One way to view this clustering problem is that edges are "examples" (labeled as positive or negative) and we are trying to represent the target function f using a hypothesis class of vertex clusters. This hypothesis class has limited representational power: if we want to say (u, v) and (v, w) are positive in this language, then we have to say (u, w) is positive too. So, we might not be able to represent f perfectly. This sort of problem — trying to find the (nearly) best representation of some arbitrary target f in a given limited hypothesis language — is sometimes called *agnostic* learning [98, 25]. The observation that one can trivially agree with at least half the edge labels is equivalent to the standard machine learning fact that one can always achieve error at most $1/2$ using either the *all positive* or *all negative* hypothesis.

Our PTAS for approximating the number of agreements means that if the optimal clustering has error rate ν, then we can find one of error rate at most $\nu + \epsilon$. Our running time is exponential in $1/\epsilon$, but this means that we can achieve any constant error gap in polynomial time. What makes this interesting from the point of view of agnostic learning is that there are very few problems where agnostic learning can be done in polynomial time.[3] Even for simple classes such as conjunctions and disjunctions, no polynomial-time algorithms are known that give even an error gap of $1/2 - \epsilon$.

4.1.1 Related and subsequent work

The CORRELATION-CLUSTERING problem was first formulated by Cohen et al. [47] in the context of natural language processing. They proposed a greedy heuristic for solving this problem, but did not give any theoretical guarantees on the quality of the clustering produced. Studying this problem in the context of phylogeny reconstruction in computational biology, Chen et al. [43] proved, independently of us, that this problem is NP-hard even in unweighted graphs. For completeness, we present an alternative proof of NP-hardness in Section 4.3.

We gave the first approximation algorithms for minimizing disagreements and maximizing agreements for unweighted graphs. Wellner et al. [139, 140] implemented a variant of our algorithm for minimizing disagreements and used it for the problem of Proper Noun Coreference. Their results indicate that our algorithm outperforms all the previously known (graph based, as well as other machine learning based) techniques for this problem.

Following the initial publication of this work, several better approximations and lower bounds have been developed for minimizing disagreements and maximizing agreements on both unweighted

[3]Not counting trivial cases, like finding the best linear separator in a 2-dimensional space, that have only polynomially-many hypotheses to choose from. In these cases, agnostic learning is easy since one can just enumerate them all and choose the best.

and weighted graphs. The approximation for minimizing disagreements in the unweighted case was improved to a factor of 4 using an LP-relaxation by Charikar, Guruswami and Wirth [37]. Subsequently, Ailon, Charikar and Newman [4] developed a combinatorial 3-approximation for the problem. This is the currently best known algorithm for unweighted graphs.

On weighted graphs, Immorlica et. al. [54] and Emanuel et. al. [58] independently developed log-factor approximations for the problem of minimizing disagreements. The latter show that this problem is equivalent to the minimum multiway cut problem. Charikar, Guruswami and Wirth also give a 0.7664-approximation for maximizing agreements in a general weighted graph, which was recently improved to 0.7666 by Swamy [132]. Charikar et al. also improve our hardness of approximation result for minimizing disagreements to 29/28, and give a hardness of approximation of 115/116 for maximizing agreements.

CONSENSUS-CLUSTERING **– A special case:** Consider an experiment for producing a clustering of objects, that is influenced by environmental factors. One way of obtaining a reliable output from such an experiment is to run it multiple times, and then aggregate the clusterings output in different runs to produce a clustering that matches as closely as possible with all of them. More precisely, we say that the distance between two clusterings is the number of pairs of elements that they classify differently. Then, given several input clusterings, the problem is to produce a clustering that minimizes the sum of its distances from all the input clusterings. This CONSENSUS-CLUSTERING problem (see [64] and references therein) is known to be NP-hard even when the number of input clusterings is only 3. This problem reduces to the objective of minimizing disagreements, when the weight on every edge is given by the number of input clusterings that put both the end points of the edge in one cluster minus the number of clusterings that put the end points in different clusters. A 2-approximation for this problem is easy, and can be achieved by simply producing the best input clustering. This was recently improved by Ailon, Charikar and Newman [4] to an $11/7$ approximation.

4.2 Notation and definitions

Let $G = (V, E)$ be a complete graph on n vertices, and let $e(u, v)$ denote the label ($+$ or $-$) of the edge (u, v). Let $N^+(u) = \{u\} \cup \{v : e(u, v) = +\}$ and $N^-(u) = \{v : e(u, v) = -\}$ denote the positive and negative neighbors of u respectively.

We let OPT denote an optimal clustering on this graph. In general, for a clustering $\mathcal{C}$, let $\mathcal{C}(v)$ be the set of vertices in the same cluster as v. We will use A to denote the clustering produced by our algorithms.

In a clustering $\mathcal{C}$, we call an edge (u, v) a mistake if either $e(u, v) = +$ and yet $u \notin \mathcal{C}(v)$, or $e(u, v) = -$ and $u \in \mathcal{C}(v)$. When $e(u, v) = +$, we call the mistake a *positive mistake*, otherwise it is called a *negative mistake*. We denote the total number of mistakes made by a clustering $\mathcal{C}$ by $m_{\mathcal{C}}$, and use m_{OPT} to denote the number of mistakes made by OPT .

For positive real numbers x, y and z, we use $x \in y \pm z$ to denote $x \in [y - z, y + z]$. Finally, let $\overline{X}$ for $X \subseteq V$ denote the complement $(V \setminus X)$.

4.3 NP-completeness

In this section, we will prove that the problem of minimizing disagreements, or equivalently, maximizing agreements, is NP-complete. As mentioned earlier, the NP-hardness of this problem was known prior to our work [43] (although not to our knowledge). For completeness, we present a different proof below.

It is easy to see that the decision version of this problem (viz. Is there a clustering with at most z disagreements?) is in NP since we can easily check the number of disagreements given a clustering. Also, if we allow arbitrary weights on edges with the goal of minimizing *weighted* disagreements, then a simple reduction from the Multiway Cut problem proves NP-hardness – simply put a $-\infty$-weight edge between every pair of terminals, then the value of the multiway cut is equal to the value of weighted disagreements. We use this reduction to give a hardness of approximation result for the weighted case in Section 4.7.

We give a proof of NP hardness for the *unweighted* case by reducing the problem of Partition into Triangles (GT11 in [68]) to the problem of minimizing disagreements. The reader who is not especially interested in NP-completeness proofs should feel free to skip this section.

The Partition into Triangles problem is described as follows: Given a graph G with $n = 3k$ vertices, does there exist a partition of the vertices into k sets $V_1, \ldots, V_k$, such that for all i, $|V_i| = 3$ and the vertices in V_i form a triangle.

Given a graph $G = (V, E)$, we first transform it into a complete graph G' on the same vertex set V. An edge in G' is weighted +1 if it is an edge in G and -1 otherwise.

Let A be an algorithm that given a graph outputs a clustering that minimizes the number of mistakes. First notice that if we impose the additional constraint that all clusters produced by A should be of size at most 3, then given the graph G', the algorithm will produce a partition into triangles if the graph admits one. This is because if the graph admits a partition into triangles, then the clustering corresponding to this triangulation has no negative mistakes, and any other clustering with clusters of size at most 3 has more positive mistakes than this clustering. Thus we could use such an algorithm to solve the Partition into Triangles problem.

We will now design a gadget that forces the optimal clustering to contain at most 3 vertices in each cluster. In particular, we will augment the graph G' to a larger complete graph H, such that in the optimal clustering on H, each cluster contains at most 3 vertices from G'.

The construction of H is as follows: In addition to the vertices and edges of G', for every 3-tuple $\{u, v, w\} \subset G'$, H contains a clique $C_{u,v,w}$ containing n^6 vertices. All edges inside these cliques have weight $+1$. Edges between vertices belonging to two different cliques have weight -1. Furthermore, for all $u, v, w \in G'$ each vertex in $C_{u,v,w}$ has a positive edge to u, v and w, and a negative edge to all other vertices in G'.

Now assume that G admits a triangulation and let us examine the behavior of algorithm A on graph H. Let $N = n^6\binom{n}{3}$.

Lemma 4.1. *Given H as input, in any clustering that A outputs, every cluster contains at most three vertices of G'.*

Proof. First consider a clustering $\mathcal{C}$ of the following form:

1. There are $\binom{n}{3}$ clusters.
2. Each cluster contains exactly one clique $C_{u,v,w}$ and some vertices of G'.
3. Every vertex $u \in G'$ is in the same cluster as $C_{u,v,w}$ for some v and w.

In any such clustering, there are no mistakes among edges between cliques. The only mistakes are between vertices of G' and the cliques, and those between the vertices of G'. The number of mistakes of this clustering is at most $n^7(\binom{n}{2} - 1) + \binom{n}{2}$ because each vertex in G' has n^6 positive edges to $\binom{n}{2}$ cliques and is clustered with only one of them.

Now consider a clustering in which some cluster has four vertices in G', say, u, v, w and y. We show that this clustering has at least $n^7(\binom{n}{2} - 1) + \frac{n^6}{2}$ mistakes. Call this clustering X. Firstly, without loss of generality we can assume that each cluster in X has size at most $n^6 + n^4$, otherwise there are at least $\Omega(n^{10})$ negative mistakes within a cluster. This implies that each vertex in G' makes at least $\binom{n}{2}n^6 - (n^6 + n^4)$ positive mistakes. Hence the total number of positive mistakes is at least $n^7(\binom{n}{2} - 1) - n^5$. Let X_u be the cluster containing vertices $u, v, w, y \in G'$. Since X_u has at most $n^6 + n^4$ vertices, at least one of u, v, w, y will have at most n^4 positive edges inside X_u and hence will contribute at least an additional $n^6 - n^4$ negative mistakes to the clustering. Thus the total number of mistakes is at least $(\binom{n}{2} - 1)n^7 - n^5 + n^6 - n^4 \geq n^7(\binom{n}{2} - 1) + n^6/2$. Thus the result follows. $\square$

The above lemma shows that the clustering produced by A will have at most 3 vertices of G in each cluster. Thus we can use the algorithm A to solve the Partition into Triangles problem and the reduction is complete.

4.4 A constant factor approximation for minimizing disagreements

As a warm-up to the general case, we begin by giving a very simple 3-approximation to the best clustering containing two clusters. That is, if the best two-cluster partition of the graph has x mistakes, then the following algorithm will produce one with at most $3x$ mistakes.

Let $\mathrm{OPT}\ (2)$ be the best clustering containing two clusters, and let the corresponding clusters be $\mathcal{C}_1$ and $\mathcal{C}_2$. Our algorithm simply considers all clusters of the form $\{N^+(v), N^-(v)\}$ for $v \in V$. Of these, it outputs the one that minimizes the number of mistakes.

Theorem 4.2. *The number of mistakes of the clustering output by the algorithm stated above is at most* $m_A \leq 3m_{\mathrm{OPT}\ (2)}$.

Proof. Let's say an edge is "bad" if $\mathrm{OPT}\ (2)$ disagrees with it, and define the "bad degree" of a vertex to be the number of bad edges incident to it. Clearly, if there is a vertex that has no bad edges incident to it, the clustering produced by that vertex would be the same as $\{\mathcal{C}_1, \mathcal{C}_2\}$, and we are done with as many mistakes as $m_{\mathrm{OPT}\ (2)}$.

Otherwise, let v be a vertex with minimum bad degree d, and without loss of generality, let $v \in \mathcal{C}_1$. Consider the partition $\{N^+(v), N^-(v)\}$. Let X be the set of bad neighbors of v – the d vertices that are in the wrong set of the partition with respect to $\{\mathcal{C}_1, \mathcal{C}_2\}$. The total number of extra mistakes due to this set X (other than the mistakes already made by OPT) is at most dn. However, since all vertices have bad degree at least d, $m_{\mathrm{OPT}\ (2)} \geq nd/2$. So, the number of extra mistakes made by taking the partition $\{N^+(v), N^-(v)\}$ is at most $2m_{\mathrm{OPT}\ (2)}$. This proves the theorem. ☐

We now describe our main algorithm: a constant-factor approximation for minimizing the number of disagreements.

The high-level idea of the algorithm is as follows. First, we show (Lemma 4.3 and 4.5) that if we can cluster a portion of the graph using clusters that each look sufficiently "clean" (Definition 4.4), then we can charge off the mistakes made within that portion to "erroneous triangles": triangles with two $+$ edges and one $-$ edge. Furthermore, we can do this in such a way that the triangles we charge are nearly edge-disjoint, allowing us to bound the number of these mistakes by a constant factor of OPT. Second, we show (Lemma 4.7) that there must exist a nearly optimal clustering $\mathrm{OPT}\,'$ in which all non-singleton clusters are "clean". Finally, we show (Theorem 4.8 and Lemma 4.12) that we can algorithmically produce a clustering of the entire graph containing only clean clusters and singleton clusters, such that mistakes that have an end-point in singleton clusters are bounded by $\mathrm{OPT}\,'$, and mistakes with both end-points in clean clusters are bounded using Lemma 4.5.

We begin by showing a lower bound for OPT. We call a triangle "erroneous" if it contains two positive edges and one negative edge. A fractional packing of erroneous triangles is a set of erroneous triangles $\{T_1, \cdots, T_m\}$ and positive real numbers r_i associated with each triangle T_i, such that for any edge $e \in E$, $\sum_{e \in T_i} r_i \leq 1$.

Lemma 4.3. *Given any fractional packing of erroneous triangles* $\{r_1, \cdots, r_m\}$*, we have* $\sum_i r_i \leq \mathrm{OPT}$.

Proof. Let M be the set of mistakes made by OPT. Then, $m_{\mathrm{OPT}} = \sum_{e \in M} 1 \geq \sum_{e \in M} \sum_{e \in T_i} r_i$, by the definition of a fractional packing. So we have $m_{\mathrm{OPT}} \geq \sum_i |M \cap T_i| r_i$. Now, for each T_i, we must have $|M \cap T_i| \geq 1$, because OPT must make at least one mistake on each erroneous triangle. This gives us the result. ☐

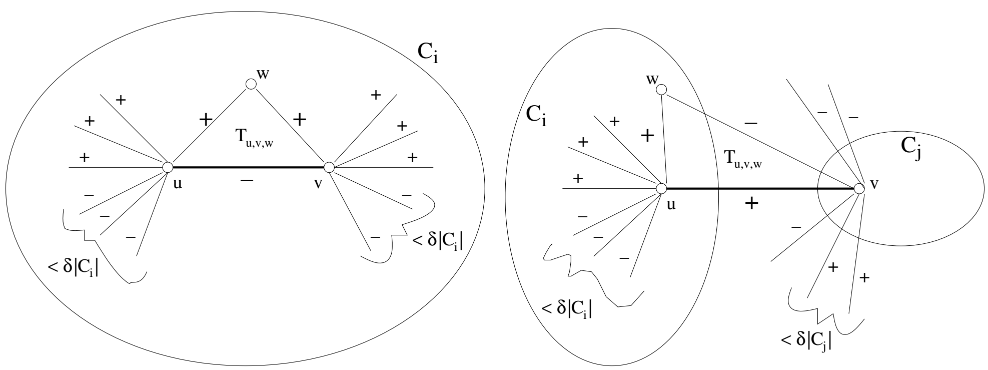
Next we give a definition of a "clean" cluster and a "good" vertex.

Definition 4.4. A vertex v is called δ**-good** with respect to C, where $C \subseteq V$, if it satisfies the following:

- $|N^+(v) \cap C| \geq (1-\delta)|C|$
- $|N^+(v) \cap (V \setminus C)| \leq \delta|C|$

If a vertex v is not δ-good with respect to (w.r.t.) C, then it is called δ**-bad** w.r.t. C. Finally, a set C is δ**-clean** if all $v \in C$ are δ-good w.r.t. C.

We now present two key lemmas.



(a) Erroneous triangles for negative mistakes

(b) Erroneous triangles for positive mistakes

Figure 4.1: Construction of a triangle packing for Lemma 4.5

Lemma 4.5. *Given a clustering of V in which all clusters are δ-clean for some $\delta \leq 1/4$, there exists a fractional packing $\{r_i, T_i\}_{i=1}^m$ such that the number of mistakes made by this clustering is at most $4\sum_i r_i$.*

Proof. Let the clustering on V be $(\mathcal{C}_1, \cdots, \mathcal{C}_k)$. First consider the case where the number of negative mistakes ($m_{\mathcal{C}}^-$) is at least half the total number of mistakes $m_{\mathcal{C}}$. We will construct a fractional packing of erroneous triangles with $\sum_i r_i \geq \frac{1}{2} m_{\mathcal{C}}^- \geq \frac{1}{4} m_{\mathcal{C}}$.

Pick a negative edge $(u, v) \in \mathcal{C}_i \times \mathcal{C}_i$ that has not been considered so far. We will pick a vertex $w \in \mathcal{C}_i$ such that both (u, w) and (v, w) are positive, and associate (u, v) with the erroneous triangle (u, v, w) (see Figure 1). We now show that for all (u, v), such a w can always be picked such that no other negative edges (u', v) or (u, v') (i.e. the ones sharing u or v) also pick w.

Since $\mathcal{C}_i$ is δ-clean, neither u nor v has more than $\delta|\mathcal{C}_i|$ negative neighbors inside $\mathcal{C}_i$. Thus (u, v) has at least $(1-2\delta)|\mathcal{C}_i|$ vertices w such that both (u, w) and (v, w) are positive. Moreover, at most $2\delta|\mathcal{C}_i| - 2$ of these could have already been chosen by other negative edges (u, v') or (u', v). Thus (u, v) has at least $(1-4\delta)|\mathcal{C}_i| + 2$ choices of w that satisfy the required condition. Since $\delta \leq 1/4$, (u, v) will always be able to pick such a w. Let T_{uvw} denote the erroneous triangle u, v, w.

Note that any positive edge (v, w) can be chosen at most 2 times by the above scheme, once for negative mistakes on v and possibly again for negative mistakes on w. Thus we can give a value of $r_{uvw} = 1/2$ to each erroneous triangle picked, ensuring that $\sum_{T_i \text{ contains } (u,v)} r_i \leq 1$. Now, since we pick a triangle for each negative mistake, we get that $\sum_{T_i} r_i = \frac{1}{2}\sum_{T_i} 1 \geq \frac{1}{2} m_{\mathcal{C}}^-$.

Next, consider the case when at least half the mistakes are positive mistakes. Just as above, we will associate mistakes with erroneous triangles. We will start afresh, without taking into account the labelings from the previous part.

Consider a positive edge between $u \in \mathcal{C}_i$ and $v \in \mathcal{C}_j$. Let $|\mathcal{C}_i| \geq |\mathcal{C}_j|$. Pick a $w \in \mathcal{C}_i$ such that (u, w) is positive and (v, w) is negative (see Figure 1). There will be at least $|\mathcal{C}_i| - \delta(|\mathcal{C}_i| + |\mathcal{C}_j|)$ such vertices as before and at most $\delta(|\mathcal{C}_i| + |\mathcal{C}_j|)$ of them will be already taken. Thus, there are at least $|\mathcal{C}_i| - 2\delta(|\mathcal{C}_i| + |\mathcal{C}_j|) \geq |\mathcal{C}_i|(1-4\delta) > 0$ choices for w. Moreover only the positive edge (u, w) can be chosen twice (once as (u, w) and once as (w, u)). Thus, as before, to obtain a packing, we can give a fractional value of $r_{uvw} = \frac{1}{2}$ to the triangle T_{uvw}. We get that $\sum_{T_i} r_i = \frac{1}{2}\sum_{T_i} 1 \geq \frac{1}{2} m_{\mathcal{C}}^+$.

Now depending on whether there are more negative mistakes or more positive mistakes, we can choose the triangles appropriately, and hence account for at least a quarter of the total mistakes in the clustering. □

Lemma 4.5 along with Lemma 4.3 gives us the following corollary.

Corollary 4.6. *Any clustering in which all clusters are δ-clean for some $\delta \leq \frac{1}{4}$ has at most $4m_{\mathrm{OPT}}$ mistakes.*

Lemma 4.7. *There exists a clustering OPT' in which each non-singleton cluster is δ-clean, and $m_{\mathrm{OPT}'} \leq (\frac{9}{\delta^2} + 1) m_{\mathrm{OPT}}$.*

Proof. Consider the following procedure applied to the clustering of OPT and call the resulting clustering OPT'.

Procedure δ-Clean-Up: Let $\mathcal{C}_1^{\mathrm{OPT}}, \mathcal{C}_2^{\mathrm{OPT}}, \ldots, \mathcal{C}_k^{\mathrm{OPT}}$ be the clusters in OPT.

1. Let $S = \emptyset$.

2. For $i = 1, \cdots, k$ do:

   (a) If the number of $\frac{\delta}{3}$-bad vertices in $\mathcal{C}_i^{\mathrm{OPT}}$ is more than $\frac{\delta}{3}|\mathcal{C}_i^{\mathrm{OPT}}|$, then, $S = S \cup \mathcal{C}_i^{\mathrm{OPT}}$, $\mathcal{C}_i' = \emptyset$. We call this "dissolving" the cluster.

(b) Else, let B_i denote the $\frac{\delta}{3}$-bad vertices in $\mathcal{C}_i^{\mathrm{OPT}}$. Then $S = S \cup B_i$ and $\mathcal{C}'_i = \mathcal{C}_i^{\mathrm{OPT}} \setminus B_i$.

3. Output the clustering OPT': $\mathcal{C}'_1, \mathcal{C}'_2, ..., \mathcal{C}'_k, \{x\}_{x \in S}$.

We will prove that m_{OPT} and $m_{\mathrm{OPT}'}$ are closely related.

We first show that each $\mathcal{C}'_i$ is δ clean. Clearly, this holds if $\mathcal{C}'_i = \emptyset$. Now if $\mathcal{C}'_i$ is non-empty, we know that $|\mathcal{C}_i^{\mathrm{OPT}}| \geq |\mathcal{C}'_i| \geq |\mathcal{C}_i^{\mathrm{OPT}}|(1 - \delta/3)$. For each point $v \in \mathcal{C}'_i$, we have:

$$\begin{aligned}
|N^+(v) \cap \mathcal{C}'_i| &\geq (1 - \frac{\delta}{3})|\mathcal{C}_i^{\mathrm{OPT}}| - (\frac{\delta}{3})|\mathcal{C}_i^{\mathrm{OPT}}| \\
&= (1 - 2\frac{\delta}{3})|\mathcal{C}_i^{\mathrm{OPT}}| \\
&> (1 - \delta)|\mathcal{C}'_i|
\end{aligned}$$

Similarly, counting positive neighbors of v in $\mathcal{C}_i^{\mathrm{OPT}} \cap \overline{\mathcal{C}'_i}$ and outside $\mathcal{C}_i^{\mathrm{OPT}}$, we get,

$$\begin{aligned}
|N^+(v) \cap \overline{\mathcal{C}'_i}| &\leq \frac{\delta}{3}|\mathcal{C}_i^{\mathrm{OPT}}| + \frac{\delta}{3}|\mathcal{C}_i^{\mathrm{OPT}}| \\
&\leq \frac{2\delta}{3}\frac{|\mathcal{C}'_i|}{(1 - \delta/3)} \\
&< \delta|\mathcal{C}'_i| \quad (\text{ as } \delta < 1)
\end{aligned}$$

Thus each $\mathcal{C}'_i$ is δ-clean.

We now account for the number of mistakes. If we dissolve some $\mathcal{C}_i^{\mathrm{OPT}}$, then clearly the number of mistakes associated with vertices in the original cluster $\mathcal{C}_i^{\mathrm{OPT}}$ is at least $(\delta/3)^2|\mathcal{C}_i^{\mathrm{OPT}}|^2/2$. The mistakes added due to dissolving clusters is at most $|\mathcal{C}_i^{\mathrm{OPT}}|^2/2$.

If $\mathcal{C}_i^{\mathrm{OPT}}$ was not dissolved, then, the original mistakes in $\mathcal{C}_i^{\mathrm{OPT}}$ were at least $\delta/3|\mathcal{C}_i^{\mathrm{OPT}}||B_i|/2$. The mistakes added by the procedure is at most $|B_i||\mathcal{C}_i^{\mathrm{OPT}}|$. Noting that $6/\delta < 9/\delta^2$, the lemma follows. $\square$

For the clustering OPT' given by the above lemma, we use $\mathcal{C}'_i$ to denote the non-singleton clusters and S to denote the set of singleton clusters. We will now describe Algorithm CAUTIOUS that tries to find clusters similar to OPT'. Throughout the rest of this section, we assume that $\delta = \frac{1}{44}$.

Algorithm CAUTIOUS **:**

1. Pick an arbitrary vertex v and do the following:

   (a) Let $A(v) = N^+(v)$.

   (b) (**Vertex Removal Step**): While $\exists x \in A(v)$ such that x is 3δ-bad w.r.t. $A(v)$, $A(v) = A(v) \setminus \{x\}$.

   (c) (**Vertex Addition Step**): Let $Y = \{y | y \in V, y$ is 7δ-good w.r.t. $A(v)\}$. Let $A(v) = A(v) \cup Y$.[4]

2. Delete $A(v)$ from the set of vertices and repeat until no vertices are left or until all the produced sets $A(v)$ are empty. In the latter case, output the remaining vertices as singleton nodes.

Call the clusters output by algorithm CAUTIOUS $A_1, A_2, \cdots$. Let Z be the set of singleton vertices created in the final step. Our main goal will be to show that the clusters output by our algorithm satisfy the property stated below.

Theorem 4.8. *$\forall j$, $\exists i$ such that $\mathcal{C}'_j \subseteq A_i$. Moreover, each A_i is 11δ-clean.*

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

Lemma 4.9. *If $v \in \mathcal{C}'_i$, where $\mathcal{C}'_i$ is a δ-clean cluster in OPT', then, any vertex $w \in \mathcal{C}'_i$ is 3δ-good w.r.t. $N^+(v)$.*

Proof. As $v \in \mathcal{C}_i$, $|N^+(v) \cap \mathcal{C}'_i| \geq (1-\delta)|\mathcal{C}'_i|$ and $|N^+(v) \cap \overline{\mathcal{C}'_i}| \leq \delta|\mathcal{C}'_i|$. So, $(1-\delta)|\mathcal{C}'_i| \leq |N^+(v)| \leq (1+\delta)|\mathcal{C}'_i|$. The same holds for w. Thus, we get the following two conditions.

$$|N^+(w) \cap N^+(v)| \geq (1-2\delta)|\mathcal{C}'_i| \geq (1-3\delta)|N^+(v)|$$

$$\begin{aligned} |N^+(w) \cap \overline{N^+(v)}| &\leq |N^+(w) \cap \overline{N^+(v)} \cap \mathcal{C}'_i| + |N^+(w) \cap \overline{N^+(v)} \cap \overline{\mathcal{C}'_i}| \\ &\leq 2\delta|\mathcal{C}'_i| \leq \tfrac{2\delta}{1-\delta}|N^+(v)| \leq 3\delta|N^+(v)| \end{aligned}$$

Thus, w is 3δ-good w.r.t. $N^+(v)$. □

Lemma 4.10. *Given an arbitrary set X, if $v_1 \in \mathcal{C}'_i$ and $v_2 \in \mathcal{C}'_j$, $i \neq j$, then v_1 and v_2 cannot both be 3δ-good w.r.t. X.*

Proof. Suppose that v_1 and v_2 are both 3δ-good with respect to X. Then, $|N^+(v_1) \cap X| \geq (1-3\delta)|X|$ and $|N^+(v_2) \cap X| \geq (1-3\delta)|X|$, hence $|N^+(v_1) \cap N^+(v_2) \cap X| \geq (1-6\delta)|X|$, which implies that

$$|N^+(v_1) \cap N^+(v_2)| \geq (1-6\delta)|X| \tag{4.1}$$

Also, since v_1 and v_2 lie in δ-clean clusters $\mathcal{C}'_i$ and $\mathcal{C}'_j$ in OPT' respectively, $|N^+(v_1) \setminus \mathcal{C}'_i| \leq \delta|\mathcal{C}'_i|$, $|N^+(v_2) \setminus \mathcal{C}'_j| \leq \delta|\mathcal{C}'_j|$ and $\mathcal{C}'_i \cap \mathcal{C}'_j = \emptyset$. It follows that

$$|N^+(v_1) \cap N^+(v_2)| \leq \delta(|\mathcal{C}'_i| + |\mathcal{C}'_j|) \tag{4.2}$$

Now notice that $|\mathcal{C}'_i| \leq |N^+(v_1) \cap \mathcal{C}'_i| + \delta|\mathcal{C}'_i| \leq |N^+(v_1) \cap X \cap \mathcal{C}'_i| + |N^+(v_1) \cap \overline{X} \cap \mathcal{C}'_i| + \delta|\mathcal{C}'_i| \leq |N^+(v_1) \cap X \cap \mathcal{C}'_i| + 3\delta|X| + \delta|\mathcal{C}'_i| \leq (1+3\delta)|X| + \delta|\mathcal{C}'_i|$. So, $|\mathcal{C}'_i| \leq \frac{1+3\delta}{1-\delta}|X|$. The same holds for $\mathcal{C}'_j$. Using Equation 4.2, $|N^+(v_1) \cap N^+(v_2)| \leq 2\delta\frac{1+3\delta}{1-\delta}|X|$.

However, since $\delta < 1/9$, we have $2\delta(1+3\delta) < (1-6\delta)(1-\delta)$. Thus the above equation along with Equation 4.1 gives a contradiction and the result follows. □

Corollary 4.11. ____________________

[4]Observe that in the vertex addition step, all vertices are added in one step as opposed to in the vertex removal step

This gives us the following important corollary.

After every application of the removal step 1b of the algorithm, no two vertices from distinct $\mathcal{C}'_i$ *and* $\mathcal{C}'_j$ *can be present in* $A(v)$.

Now we go on to prove Theorem 4.8.

Proof of Theorem 4.8: We will first show that each A_i is either a subset of S or contains exactly one of the clusters $\mathcal{C}'_j$. The first part of the theorem will follow.

We proceed by induction on i. Consider the inductive step. For a cluster A_i, let A'_i be the set produced after the vertex removal phase such the cluster A_i is obtained by applying the vertex addition phase to A'_i. We have two cases. First, we consider the case when $A'_i \subseteq S$. Now during the vertex addition step, no vertex $u \in \mathcal{C}'_j$ can enter A'_i for any j. This follows because, since $\mathcal{C}'_j$ is δ-clean and disjoint from A'_i, for u to enter we need that $\delta|\mathcal{C}'_j| \geq (1-7\delta)|A'_i|$ and $(1-\delta)|\mathcal{C}'_j| \leq 7\delta|A'_i|$, and these two conditions cannot be satisfied simultaneously. Thus $A_i \subseteq S$.

In the second case, some $u \in \mathcal{C}'_j$ is present in A'_i. However, in this case observe that from Corollary 4.11, no vertices from $\mathcal{C}'_k$ can be present in A'_i for any $k \neq j$. Also, by the same reasoning as for the case $A'_i \subseteq S$, no vertex from $\mathcal{C}'_k$ will enter A'_i in the vertex addition phase. Now it only remains to show that $\mathcal{C}'_j \subseteq A_i$. Note that all vertices of $\mathcal{C}'_j$ are still present in the remaining graph $G \setminus (\bigcup_{\ell<i} A_\ell)$.

Since u was not removed from A'_i it follows that many vertices from $\mathcal{C}'_j$ are present in A'_i. In particular, $|N^+(u) \cap A'_i| \geq (1-3\delta)|A'_i|$ and $|N^+(u) \cap \overline{A'_i}| \leq 3\delta|A'_i|$. Now $(1-\delta)|\mathcal{C}'_j| \leq |N^+(u)|$ implies that $|\mathcal{C}'_j| \leq \frac{1+3\delta}{1-\delta}|A'_i| < 2|A'_i|$. Also, $|A'_i \cap \mathcal{C}'_j| \geq |A'_i \cap N^+(u)| - |N^+(u) \cap \overline{\mathcal{C}'_j}| \geq |A'_i \cap N^+(u)| - \delta|\mathcal{C}'_j|$. So we have $|A'_i \cap \mathcal{C}'_j| \geq (1-5\delta)|A'_i|$.

We now show that all remaining vertices from $\mathcal{C}'_j$ will enter A_i during the vertex addition phase. For $w \in \mathcal{C}'_j$ such that $w \notin A'_i$, $|A'_i \cap \overline{\mathcal{C}'_j}| \leq 5\delta|A'_i|$ and $|\overline{N^+(w)} \cap \mathcal{C}'_j| \leq \delta|\mathcal{C}'_j|$ together imply that $|A'_i \cap \overline{N^+(w)}| \leq 5\delta|A'_i| + \delta|\mathcal{C}'_j| \leq 7\delta|A'_i|$. The same holds for $|\overline{A'_i} \cap N^+(w)|$. So w is 7δ-good w.r.t. A'_i and will be added in the Vertex Addition step. Thus we have shown that $A(v)$ can contain $\mathcal{C}'_j$ for at most one j and in fact will contain this set entirely.

Next, we will show that for every j, $\exists i$ s.t. $\mathcal{C}'_j \subseteq A_i$. Let v chosen in Step 1 of the algorithm be such that $v \in \mathcal{C}'_j$. We show that during the vertex removal step, no vertex from $N^+(v) \cap \mathcal{C}'_j$ is removed. The proof follows by an easy induction on the number of vertices removed so far (r) in the vertex removal step. The base case $(r = 0)$ follows from Lemma 4.9 since every vertex in $\mathcal{C}'_j$ is 3δ-good with respect to $N^+(v)$. For the induction step observe that since no vertex from $N^+(v) \cap \mathcal{C}'_j$ is removed thus far, every vertex in $\mathcal{C}'_j$ is still 3δ-good w.r.t. to the intermediate $A(v)$ (by mimicking the proof of Lemma 4.9 with $N^+(v)$ replaced by $A(v)$). Thus A'_i contains at least $(1-\delta)|\mathcal{C}'_j|$ vertices of $\mathcal{C}'_j$ at the end of the vertex removal phase, and hence by the second case above, $\mathcal{C}'_j \subseteq A_i$ after the vertex addition phase.

Finally we show that every non-singleton cluster A_i is 11δ-clean. We know that at the end of the vertex removal phase, $\forall x \in A'_i$, x is 3δ-good w.r.t. A'_i. Thus, $|N^+(x) \cap \overline{A'_i}| \leq 3\delta|A'_i|$. So the total number of positive edges leaving A'_i is at most $3\delta|A'_i|^2$. Since, in the vertex addition step, we add vertices that are 7δ-good w.r.t. A'_i, the number of these vertices can be at most $3\delta|A'_i|^2/(1-7\delta)|A'_i| < 4\delta|A'_i|$. Thus $|A_i| < (1+4\delta)|A'_i|$.

Since all vertices v in A_i are at least 7δ-good w.r.t. A'_i, $N^+(v) \cap A_i \geq (1-7\delta)|A'_i| \geq \frac{1-7\delta}{1+4\delta}|A_i| \geq (1-11\delta)|A_i|$. Similarly, $N^+(v) \cap \overline{A_i} \leq 7\delta|A'_i| \leq 11\delta|A_i|$. This gives us the result. □

Now we are ready to bound the mistakes of A in terms of OPT and OPT'. Call mistakes that have both end points in some clusters A_i and A_j as internal mistakes and those that have an end point in Z as external mistakes. Similarly in OPT', we call mistakes among the sets $\mathcal{C}'_i$ as internal mistakes and mistakes having one end point in S as external mistakes. We bound mistakes of CAUTIOUS in two steps: the following lemma bounds external mistakes.

Lemma 4.12. *The total number of external mistakes made by* CAUTIOUS *are less than the external mistakes made by* OPT'.

Proof. From Theorem 4.8, it follows that Z cannot contain any vertex v in some $\mathcal{C}'_i$. Thus, $Z \subseteq S$. Now, any external mistakes made by CAUTIOUS are positive edges adjacent to vertices in Z. These edges are also mistakes in OPT' since they are incident on singleton vertices in S. Hence the lemma follows. □

Now consider the internal mistakes of A. Notice that these could be many more than the internal mistakes of OPT'. However, we can at this point apply Lemma 4.6 on the graph induced by $V' = \bigcup_i A_i$. In particular, the bound on internal mistakes follows easily by observing that $11\delta \leq 1/4$, and that the mistakes of the optimal clustering on the graph induced by V' is no more than m_{OPT}. Thus,

Lemma 4.13. *The total number of internal mistakes of* CAUTIOUS *is* $\leq 4m_{\mathrm{OPT}}$.

Summing up results from the Lemmas 4.12 and 4.13, and using Lemma 4.7, we get the following theorem:

Theorem 4.14. $m_{\mathrm{CAUTIOUS}} \leq (\frac{9}{\delta^2}+5)m_{\mathrm{OPT}}$*, with* $\delta = \frac{1}{44}$.

4.5 A PTAS for maximizing agreements

In this section, we give a PTAS for maximizing agreements: the total number of positive edges inside clusters and negative edges between clusters.

As before, let OPT denote an optimal clustering and A denote our clustering. We will abuse notation and also use OPT to denote the number of agreements in the optimal solution. As noted in the introduction, $\mathrm{OPT} \geq n(n-1)/4$. So it suffices to produce a clustering that has at least $\mathrm{OPT} - \epsilon n^2$ agreements, which will be the goal of our algorithm. Let $\delta^+(V_1, V_2)$ denote the number of positive edges between sets $V_1, V_2 \subseteq V$. Similarly, let $\delta^-(V_1, V_2)$ denote the number of negative edges between the two. Let $\mathrm{OPT}\ (\epsilon)$ denote the optimal clustering that has all non-singleton clusters of size greater than ϵn.

Lemma 4.15. $\mathrm{OPT}\ (\epsilon) \geq \mathrm{OPT}\ - \epsilon n^2/2$.

Proof. Consider the clusters of OPT of size less than or equal to ϵn and break them apart into clusters of size 1. Breaking up a cluster of size s reduces our objective function by at most $\binom{s}{2}$, which can be viewed as $s/2$ per node in the cluster. Since there are at most n nodes in these clusters, and these clusters have size at most ϵn, the total loss is at most $\epsilon \frac{n^2}{2}$. □

The above lemma means that it suffices to produce a good approximation to $\mathrm{OPT}\ (\epsilon)$. Note that the number of non-singleton clusters in $\mathrm{OPT}\ (\epsilon)$ is less than $\frac{1}{\epsilon}$. Let $\mathcal{C}_1^{\mathrm{OPT}}, \ldots, \mathcal{C}_k^{\mathrm{OPT}}$ denote the non-singleton clusters of $\mathrm{OPT}\ (\epsilon)$ and let $\mathcal{C}_{k+1}^{\mathrm{OPT}}$ denote the set of points which correspond to singleton clusters.

4.5.1 A PTAS doubly-exponential in $1/\epsilon$

If we are willing to have a run time that is doubly-exponential in $1/\epsilon$, we can do this by reducing our problem to the General Partitioning problem of [76]. The idea is as follows.

Let G^+ denote the graph of only the $+$ edges in G. Then, notice that we can express the quality of $\mathrm{OPT}\ (\epsilon)$ in terms of just the sizes of the clusters, and the number of edges in G^+ between and inside each of $\mathcal{C}_1^{\mathrm{OPT}}, \ldots, \mathcal{C}_{k+1}^{\mathrm{OPT}}$. In particular, if $s_i = |\mathcal{C}_i^{\mathrm{OPT}}|$ and $e_{i,j} = \delta^+(\mathcal{C}_i^{\mathrm{OPT}}, \mathcal{C}_j^{\mathrm{OPT}})$, then the number of agreements in $\mathrm{OPT}\ (\epsilon)$ is:

$$\left[\sum_{i=1}^{k} e_{i,i}\right] + \left[\binom{s_{k+1}}{2} - e_{k+1,k+1}\right] + \left[\sum_{i \neq j} (s_i s_j - e_{i,j})\right].$$

The General Partitioning property tester of [76] allows us to specify values for the s_i and e_{ij}, and if a partition of G^+ exists satisfying these constraints, will produce a partition that satisfies these constraints approximately. We obtain a partition that has at least $\mathrm{OPT}\ (\epsilon) - \epsilon n^2$ agreements. The property tester runs in time exponential in $(\frac{1}{\epsilon})^{k+1}$ and polynomial in n.

Thus if we can guess the values of these sizes and number of edges accurately, we would be done. It suffices, in fact, to only guess the values up to an additive $\pm\epsilon^2 n$ for the s_i, and up to an additive $\pm\epsilon^3 n^2$ for the $e_{i,j}$, because this introduces an additional error of at most $O(\epsilon)$. So, at most $O((1/\epsilon^3)^{1/\epsilon^2})$ calls to the property tester need to be made. Our algorithm proceeds by finding a partition for each possible value of s_i and $e_{i,j}$ and returns the partition with the maximum number of agreements. We get the following result:

Theorem 4.16. *The General Partitioning algorithm returns a clustering of graph G which has more than* $\mathrm{OPT}\ - \epsilon n^2$ *agreements with probability at least* $1 - \delta$. *It runs in time* $e^{O((\frac{1}{\epsilon})^{1/\epsilon})} \times poly(n, \frac{1}{\delta})$.

4.5.2 A singly-exponential PTAS

We will now describe an algorithm that is based on the same basic idea of random sampling used by the General Partitioning algorithm. The idea behind our algorithm is as follows: Let $\{O_i\}$ be the clusters in OPT. We select a small random subset W of vertices and cluster them correctly into $\{W_i\}$ with $W_i \subset O_i\ \forall i$, by enumerating all possible clusterings of W. Since this subset is picked randomly, with a high probability, for all vertices v, the density of positive edges between v and W_i will be approximately equal to the density of positive edges between v and O_i. So we can decide which cluster to put v into, based on this information. However this is not sufficient to account for edges between two vertices v_1 and v_2, both of which do not belong to W. So, we consider a partition of the rest of the graph into subsets U_i of size m and try out all possible clusterings $\{U_{ij}\}$ of each subset, picking the one that maximizes agreements with respect to $\{W_i\}$. This gives us the PTAS.

Firstly note that if $|\mathcal{C}_{k+1}^{\mathrm{OPT}}| < \epsilon n$, then if we only consider the agreements in the graph $G \backslash \mathcal{C}_{k+1}^{\mathrm{OPT}}$, it affects the solution by at most ϵn^2. For now, we will assume that $|\mathcal{C}_{k+1}^{\mathrm{OPT}}| < \epsilon n$ and will present the algorithm and analysis based on this assumption. Later we will discuss the changes required to deal with the other case.

In the following algorithm ϵ is a performance parameter to be specified later. Let $m = \frac{88^3 \times 40}{\epsilon^{10}}(\log \frac{1}{\epsilon} + 2)$, $k = \frac{1}{\epsilon}$ and $\epsilon' = \frac{\epsilon^3}{88}$. Let p_i denote the density of positive edges inside the cluster $\mathcal{C}_i^{\mathrm{OPT}}$ and n_{ij} the density of negative edges between clusters $\mathcal{C}_i^{\mathrm{OPT}}$ and $\mathcal{C}_j^{\mathrm{OPT}}$. That is, $p_i = \delta^+(\mathcal{C}_i^{\mathrm{OPT}}, \mathcal{C}_i^{\mathrm{OPT}})/\binom{|\mathcal{C}_i^{\mathrm{OPT}}|}{2}$ and $n_{ij} = \delta^-(\mathcal{C}_i^{\mathrm{OPT}}, \mathcal{C}_j^{\mathrm{OPT}})/(|\mathcal{C}_i^{\mathrm{OPT}}||\mathcal{C}_j^{\mathrm{OPT}}|)$. Let $W \subset V$ be a random subset of size m.

We begin by defining a measure of goodness of a clustering $\{U_{ij}\}$ of some set U_i with respect to a fixed partition $\{W_i\}$, that will enable us to pick the right clustering of the set U_i. Let $\hat{p}_i$ and $\hat{n}_{ij}$ be estimates of p_i and n_{ij} respectively, based on $\{W_i\}$, to be defined later in the algorithm.

Definition 4.17. $U_{i1}, \ldots, U_{i(k+1)}$ is ϵ'**-good** w.r.t. $W_1, \ldots, W_{k+1}$ if it satisfies the following for all $1 \le j, \ell \le k$:

(1) $\delta^+(U_{ij}, W_j) \ge \hat{p}_j \binom{W_j}{2} - 18\epsilon' m^2$

(2) $\delta^-(U_{ij}, W_\ell) \ge \hat{n}_{j\ell}|W_j||W_\ell| - 6\epsilon' m^2$

and, for at least $(1-\epsilon')n$ of the vertices x and $\forall\, j$,

(3) $\delta^+(U_{ij}, x) \in \delta^+(W_j, x) \pm 2\epsilon' m$.

Our algorithm is as follows:

Algorithm DIVIDE&CHOOSE **:**

1. Pick a random subset $W \subset V$ of size m.

2. For all partitions $W_1, \ldots, W_{k+1}$ of W do
   (a) Let $\hat{p_i} = \delta^+(W_i, W_i)/\binom{|W_i|}{2}$, and $\hat{n}_{ij} = \delta^-(W_i, W_j)/|W_i||W_j|$.
   (b) Let $q = \frac{n}{m} - 1$. Consider a random partition of $V \setminus W$ into $U_1, \ldots, U_q$, such that $\forall i$, $|U_i| = m$.
   (c) For all i do:
   Consider all $(k+1)$-partitions of U_i and let $U_{i1}, \ldots, U_{i(k+1)}$ be a partition that is ϵ'-good w.r.t. $W_1, \ldots, W_{k+1}$ (by Definition 4.17 above). If there is no such partition, choose $U_{i1}, \ldots, U_{i(k+1)}$ arbitrarily.
   (d) Let $A_j = \bigcup_i U_{ij}$ for all i. Let $a(\{W_i\})$ be the number of agreements of this clustering.
3. Let $\{W_i\}$ be the partition of W that maximizes $a(\{W_i\})$. Return the clusters $\{A_i\}, \{x\}_{x \in A_{k+1}}$ corresponding to this partition of W.

We will concentrate on the "right" partition of W given by $W_i = W \cap \mathcal{C}_i^{\mathrm{OPT}}$, $\forall i$. We will show that the number of agreements of the clustering $A_1, \ldots, A_{k+1}$ corresponding to this partition $\{W_i\}$ is at least $\mathrm{OPT}\,(\epsilon) - 2\epsilon n^2$ with a high probability. Since we pick the best clustering, this gives us a PTAS.

We will begin by showing that with a high probability, for most values of i, the partition of U_is corresponding to the optimal partition is good with respect to $\{W_i\}$. Thus the algorithm will find at least one such partition. Next we will show that if the algorithm finds good partitions for most U_i, then it achieves at least $\mathrm{OPT}\; - O(\epsilon)n^2$ agreements.

We will need the following results from probability theory. Please refer to [7] for a proof.

FACT 1: Let $H(n, m, l)$ be the hypergeometric distribution with parameters n, m and l (choosing l samples from n points without replacement with the random variable taking a value of 1 on exactly m out of the n points). Let $0 \leq \epsilon \leq 1$. Then

$$Pr\left[\left|H(n, m, l) - \frac{lm}{n}\right| \geq \frac{\epsilon lm}{n}\right] \leq 2e^{-\frac{\epsilon^2 lm}{2n}}$$

FACT 2: Let $X_1, X_2, ..., X_n$ be mutually independent random variables such that $|X_i - E[X_i]| < m$ for all i. Let $S = \sum_{i=1}^{n} X_i$, then

$$Pr\left[|S - E[S]| \geq a\right] \leq 2e^{-\frac{a^2}{2nm^2}}$$

We will also need the following lemma:

Lemma 4.18. *Let Y and S be arbitrary disjoint sets and Z be a set picked from S at random. Then we have the following:*

$$Pr\left[\left|\delta^+(Y, Z) - \frac{|Z|}{|S|}\delta^+(Y, S)\right| > \epsilon'|Y||Z|\right] \leq 2e^{\frac{-\epsilon'^2|Z|}{2}}$$

Proof. $\delta^+(Y,Z)$ is a sum of $|Z|$ random variables $\delta^+(Y,v)$ ($v \in Z$), each bounded above by $|Y|$ and having expected value $\frac{\delta^+(Y,S)}{|S|}$.

Thus applying Fact 2, we get

$$Pr\left[\left|\delta^+(Y,Z) - |Z|\delta^+(Y,S)/|S|\right| > \epsilon'|Z||Y|\right] \le 2e^{-\epsilon'^2|Z|^2|Y|^2/2|Z||Y|^2} \le 2e^{-\epsilon'^2|Z|/2}$$

□

Now notice that since we picked W uniformly at random from V, with a high probability the sizes of W_is are in proportion to $|\mathcal{C}_i^{\mathrm{OPT}}|$. The following lemma formalizes this.

Lemma 4.19. *With probability at least* $1 - 2ke^{-\epsilon'^2\epsilon m/2}$ *over the choice of* W, $\forall i$, $|W_i| \in (1 \pm \epsilon')\frac{m}{n}|\mathcal{C}_i^{\mathrm{OPT}}|$.

Proof. For a given i, using Fact 1 and since $|\mathcal{C}_i^{\mathrm{OPT}}| \ge \epsilon n$,

$$Pr\left[\left||W_i| - \frac{m}{n}|\mathcal{C}_i^{\mathrm{OPT}}|\right| > \epsilon'\frac{m}{n}|\mathcal{C}_i^{\mathrm{OPT}}|\right] \le 2e^{-\epsilon'^2 m|\mathcal{C}_i^{\mathrm{OPT}}|/2n} \le 2e^{-\epsilon'^2\epsilon m/2}$$

Taking a union bound over the k values of i we get the result. □

Using Lemma 4.19, we show that the computed values of $\hat{p}_i$ and $\hat{n}_{ij}$ are close to the true values p_i and n_{ij} respectively. This gives us the following two lemmas.

Lemma 4.20. *If* $W_i \subset \mathcal{C}_i^{\mathrm{OPT}}$ *and* $W_j \subset \mathcal{C}_j^{\mathrm{OPT}}$, $i \ne j$, *then with probability at least* $1 - 4e^{-\epsilon'^2\epsilon m/4}$ *over the choice of* W, $\delta^+(W_i, W_j) \in \frac{m^2}{n^2}\delta^+(\mathcal{C}_i^{\mathrm{OPT}}, \mathcal{C}_j^{\mathrm{OPT}}) \pm 3\epsilon' m^2$.

Proof. We will apply Lemma 4.18 in two steps. First we will bound $\delta^+(W_i, W_j)$ in terms of $\delta^+(W_i, \mathcal{C}_j^{\mathrm{OPT}})$ by fixing W_i and considering the process of picking W_j from $\mathcal{C}_j^{\mathrm{OPT}}$.

Using W_i for Y, W_j for Z and $\mathcal{C}_j^{\mathrm{OPT}}$ for S in Lemma 4.18, we get the following[5].

$$Pr\left[\left|\delta^+(W_i, W_j) - \frac{m}{n}\delta^+(W_i, \mathcal{C}_j^{\mathrm{OPT}})\right| > \epsilon' m^2\right] \le 2e^{-\epsilon'^2\epsilon m/4}$$

We used the fact that $m \ge |W_j| \ge \epsilon m/2$ with high probability. Finally, we again apply Lemma 4.18 to bound $\delta^+(W_i, \mathcal{C}_j^{\mathrm{OPT}})$ in terms of $\delta^+(\mathcal{C}_i^{\mathrm{OPT}}, \mathcal{C}_j^{\mathrm{OPT}})$. Taking Y to be $\mathcal{C}_j^{\mathrm{OPT}}$, Z to be W_i and S to be $\mathcal{C}_i^{\mathrm{OPT}}$, we get

$$Pr\left[\left|\delta^+(W_i, \mathcal{C}_j^{\mathrm{OPT}}) - \frac{m}{n}\delta^+(\mathcal{C}_i^{\mathrm{OPT}}, \mathcal{C}_j^{\mathrm{OPT}})\right| > 2\epsilon'\frac{m}{n}|\mathcal{C}_i^{\mathrm{OPT}}||\mathcal{C}_j^{\mathrm{OPT}}|\right] \le 2e^{-\epsilon'^2\epsilon m/4}$$

Again we used the fact that $|W_i| < \frac{2m}{n}|\mathcal{C}_i^{\mathrm{OPT}}|$ with high probability. So, with probability at least $1 - 4e^{-\epsilon'^2\epsilon m/4}$, we have, $|\frac{m}{n}\delta^+(W_i, \mathcal{C}_j^{\mathrm{OPT}}) - \frac{m^2}{n^2}\delta^+(\mathcal{C}_i^{\mathrm{OPT}}, \mathcal{C}_j^{\mathrm{OPT}})| < 2\epsilon'\frac{m^2}{n^2}|\mathcal{C}_i^{\mathrm{OPT}}||\mathcal{C}_j^{\mathrm{OPT}}| < 2\epsilon' m^2$ and $|\delta^+(W_i, W_j) - \frac{m}{n}\delta^+(W_i, \mathcal{C}_j^{\mathrm{OPT}})| < \epsilon' m^2$. This gives us

[5] We are assuming that W is a set of size m chosen randomly from n with replacement, since m is a constant, we will have no ties with probability $1 - O(n^{-1})$.

$$Pr\left[\left|\delta^+(W_i, W_j) - \frac{m^2}{n^2}\delta^+(\mathcal{C}_i^{\mathrm{OPT}}, \mathcal{C}_j^{\mathrm{OPT}})\right| > 3\epsilon' m^2\right] \leq 4e^{-\epsilon'^2 \epsilon m/4}$$

□

Lemma 4.21. *With probability at least* $1 - \frac{8}{\epsilon'^2}e^{-\epsilon'^3\epsilon m/4}$ *over the choice of* W, $\hat{p}_i \geq p_i - 9\epsilon'$

Proof. Note that we cannot use an argument similar to the previous lemma directly here since we are dealing with edges inside the same set. Instead we use the following trick.

Consider an arbitrary partition of $\mathcal{C}_i^{\mathrm{OPT}}$ into $\frac{1}{\epsilon'}$ sets of size $\epsilon' n'$ each where $n' = |\mathcal{C}_i^{\mathrm{OPT}}|$. Let this partition be $\mathcal{C}_{i,1}^{\mathrm{OPT}}, \cdots, \mathcal{C}_{i,1/\epsilon'}^{\mathrm{OPT}}$ and let $W_{i,j} = W_i \cap \mathcal{C}_{i,j}^{\mathrm{OPT}}$. Let $m' = |W_i|$. Now consider $\delta^+(W_{i,j_1}, W_{i,j_2})$. Using an argument similar to the previous lemma, we get that with probability at least $1 - 4e^{-\epsilon'^3\epsilon m/4}$,

$$\delta^+(W_{i,j_1}, W_{i,j_2}) \in \frac{|W_{i,j_1}||W_{i,j_2}|}{|\mathcal{C}_{i,j_1}^{\mathrm{OPT}}||\mathcal{C}_{i,j_2}^{\mathrm{OPT}}|}\delta^+(\mathcal{C}_{i,j_1}^{\mathrm{OPT}}, \mathcal{C}_{i,j_2}^{\mathrm{OPT}}) \pm 2\epsilon'|W_{i,j_1}||W_{i,j_2}|$$

Noting that $\frac{|W_{i,j_1}||W_{i,j_2}|}{|\mathcal{C}_{i,j_1}^{\mathrm{OPT}}||\mathcal{C}_{i,j_2}^{\mathrm{OPT}}|} < (1+3\epsilon')\frac{m'^2}{n'^2}$, with probability at least $1 - 4e^{-\epsilon'^3\epsilon m/4}$, we get,

$$Pr\left[\left|\delta^+(W_{i,j_1}, W_{i,j_2}) - \frac{m'^2}{n'^2}\delta^+(\mathcal{C}_{i,j_1}^{\mathrm{OPT}}, \mathcal{C}_{i,j_2}^{\mathrm{OPT}})\right| < 8\epsilon'|W_{i,j_1}||W_{i,j_2}|\right] \geq 1 - 8e^{-\epsilon'^3\epsilon m/4}$$

This holds for every value of j_1 and j_2 with probability at least $1 - \frac{8}{\epsilon'^2}e^{-\epsilon'^3\epsilon m/4}$. Now,

$$\begin{aligned}
\delta^+(W_i, W_i) &\geq \sum_{j_1<j_2}\delta^+(W_{i,j_1}, W_{i,j_2}) \\
&\geq \frac{1}{1+8\epsilon'}\frac{m'^2}{n'^2}\sum_{j_1<j_2}\delta^+(\mathcal{C}_{i,j_1}^{\mathrm{OPT}}, \mathcal{C}_{i,j_2}^{\mathrm{OPT}}) \\
&\geq \frac{1}{1+8\epsilon'}\frac{m'^2}{n'^2}(p_i\frac{n'^2}{2} - \frac{1}{\epsilon'}\frac{\epsilon'^2 n'^2}{2}) \\
&\geq (p_i - 9\epsilon')\frac{|W_i|^2}{2}
\end{aligned}$$

□

Now let $U_{ij} = U_i \cap \mathcal{C}_j^{\mathrm{OPT}}$. The following lemma shows that for all i, with a high probability all U_{ij}s are ϵ'-good w.r.t. $\{W_i\}$. So we will be able to find ϵ'-good partitions for most U_is.

Lemma 4.22. *For a given* i, *let* $U_{ij} = U_i \cap \mathcal{C}_j^{\mathrm{OPT}}$, *then with probability at least* $1 - 32k\frac{1}{\epsilon'^2}e^{-\epsilon'^3\epsilon m/4}$ *over the choice of* U_i, $\forall j \leq k$, $\{U_{ij}\}$ *are* ϵ'*-good w.r.t.* $\{W_j\}$.

Proof. Consider the partition $\{U_{ij}\}$ of U_i. Using an argument similar to Lemma 4.20, we get $|\delta^+(U_{ij}, W_l) - \frac{m^2}{n^2}\delta^+(\mathcal{C}_j^{\mathrm{OPT}}, \mathcal{C}_l^{\mathrm{OPT}})| \leq 3\epsilon' m^2$ with probability at least $1 - 4e^{-\epsilon'^2 \epsilon m/4}$. Also, again from Lemma 4.20, $|\delta^+(W_j, W_l) - \frac{m^2}{n^2}\delta^+(\mathcal{C}_j^{\mathrm{OPT}}, \mathcal{C}_l^{\mathrm{OPT}})| \leq 3\epsilon' m^2$. So, $|\delta^+(U_{ij}, W_l) - \delta^+(W_j, W_l)| \leq 6\epsilon' m^2$ with probability at least $1 - 8e^{-\epsilon'^2 \epsilon m/4}$. This gives us the second condition of Definition 4.17.

Similarly, using Lemma 4.21, we obtain the first condition. The failure probability in this step is at most $16\frac{1}{\epsilon'^2}e^{-\epsilon'^3 \epsilon m/4}$.

Now, consider $\delta^+(x, U_{ij})$. This is a sum of m $\{0,1\}$ random variables (corresponding to picking U_i from V), each of which is 1 iff the picked vertex lies in $\mathcal{C}_j^{\mathrm{OPT}}$ and is adjacent to x. Applying Chernoff bound, we get,

$Pr\left[\left|\delta^+(x, U_{ij}) - \frac{m}{n}\delta^+(x, \mathcal{C}_j^{\mathrm{OPT}})\right| > \epsilon' m\right] \leq 2e^{-\epsilon'^2 m/2}$

Similarly we have, $Pr\left[\left|\delta^+(x, W_j) - \frac{m}{n}\delta^+(x, \mathcal{C}_j^{\mathrm{OPT}})\right| > \epsilon' m\right] \leq 2e^{-\epsilon'^2 m/2}$.

So we get, $Pr\left[|\delta^+(x, U_{ij}) - \delta^+(x, W_j)| > 2\epsilon' m\right] \leq 4e^{-\epsilon'^2 m/2}$.

Note that, here we are assuming that W and U_i are picked independently from V. However, picking U_i from $V \setminus W$ is similar to picking it from V since the collision probability is extremely small.

Now, the expected number of points that do not satisfy condition 3 for some U_{ij} is $4ne^{-\epsilon'^2 m/2}$. The probability that more than $\epsilon' n$ of the points fail to satisfy condition 3 for one of the U_{ij}s in U_i is at most $k\frac{1}{\epsilon' n}4ne^{-\epsilon'^2 m/2} \leq \frac{4k}{\epsilon'}e^{-\epsilon'^2 m/2}$. This gives us the third condition.

The total probability that some U_i does not satisfy the above conditions is at most

$$\begin{aligned}
&8e^{-\epsilon'^2 \epsilon m/4} + 16\tfrac{1}{\epsilon'^2}e^{-\epsilon'^3 \epsilon m/4} + \tfrac{4k}{\epsilon'}e^{-\epsilon'^2 m/2} \\
&\leq 32\tfrac{1}{\epsilon'^2}e^{-\epsilon'^3 \epsilon m/4}
\end{aligned}$$

□

Now we can bound the total number of agreements of $A_1, \ldots, A_k, \{x\}_{x \in A_{k+1}}$ in terms of OPT :

Theorem 4.23. *If* $|\mathcal{C}_{k+1}^{\mathrm{OPT}}| < \epsilon n$, *then* $A \geq \mathrm{OPT} - 3\epsilon n^2$ *with probability at least* $1 - \epsilon$.

Proof. From Lemma 4.22, the probability that we were not able to find an ϵ'-good partition of U_i w.r.t. $W_1, \cdots, W_k$ is at most $32\frac{1}{\epsilon'^2}e^{-\epsilon'^3 \epsilon m/4}$. By our choice of m, this is at most $\epsilon^2/4$. So, with probability at least $1 - \epsilon/2$, at most $\epsilon/2$ of the U_is do not have an ϵ'-good partition.

In the following calculation of the number of agreements, we assume that we are able to find good partitions of all U_is. We will only need to subtract at most $\epsilon n^2/2$ from this value to obtain the actual number of agreements, since each U_i can affect the number of agreements by at most mn.

We start by calculating the number of positive edges inside a cluster A_j. These are given by

$\sum_a \sum_{x\in A_j} \delta^+(U_{aj}, x)$. Using the fact that U_{aj} is good w.r.t. $\{W_i\}$ (condition (3)),

$$\begin{aligned}\textstyle\sum_{x\in A_j} \quad & \delta^+(U_{aj}, x)\\ & \geq \textstyle\sum_{x\in A_j}(\delta^+(W_j, x) - 2\epsilon' m) - \epsilon' n|U_{aj}|\\ & = \textstyle\sum_b \delta^+(W_j, U_{bj}) - 2\epsilon' m|A_j| - \epsilon' n|U_{aj}|\\ & \geq \textstyle\sum_b \{\hat{p_j}\frac{|W_j|^2}{2} - 18\epsilon' m^2\} - 2\epsilon' m|A_j| - \epsilon' n|U_{aj}|\end{aligned}$$

The last inequality follows from the fact that U_{bj} is good w.r.t. $\{W_i\}$ (condition (1)). From Lemma 4.19,

$$\begin{aligned}\textstyle\sum_{x\in A_j} \delta^+(U_{aj}, x) \quad & \geq \quad \textstyle\sum_b \{\frac{m^2}{n^2}\hat{p_j}(1-\epsilon')^2\frac{|\mathcal{C}_j^{\text{OPT}}|^2}{2} - 18\epsilon' m^2\} - 2\epsilon' m|A_j| - \epsilon' n|U_{aj}|\\ & \geq \quad \frac{m}{n}\hat{p_j}(1-\epsilon')^2\frac{|\mathcal{C}_j^{\text{OPT}}|^2}{2} - 18\epsilon' mn - 2\epsilon' m|A_j| - \epsilon' n|U_{aj}|\end{aligned}$$

Thus we bound $\sum_a \delta^+(A_j, U_{aj})$ as $\sum_a \delta^+(A_j, U_{aj}) \geq \hat{p_j}(1-\epsilon')^2\frac{|\mathcal{C}_j^{\text{OPT}}|^2}{2} - 18\epsilon' n^2 - 3\epsilon' n|A_j|$. Now using Lemma 4.21, the total number of agreements is at least

$$\begin{aligned}\textstyle\sum_j \left[\hat{p_j}(1-\epsilon')^2\frac{|\mathcal{C}_j^{\text{OPT}}|^2}{2}\right] \quad & -18\epsilon' n^2 k - 3\epsilon' n^2\\ & \geq \textstyle\sum_j \left[(p_j - 9\epsilon')(1-\epsilon')^2\frac{|\mathcal{C}_j^{\text{OPT}}|^2}{2}\right] - 18\epsilon' n^2 k - 3\epsilon' n^2\end{aligned}$$

Hence, $A^+ \geq \text{OPT}^+ - 11\epsilon' k n^2 - 21\epsilon' n^2 k \geq \text{OPT}^+ - 32\epsilon' n^2 k$.
Similarly, consider the negative edges in A. Using Lemma 4.20 to estimate $\delta^-(U_{ai}, U_{bj})$, we get,

$$\sum_{ab} \delta^-(U_{ai}, U_{bj}) \geq \delta^-(\mathcal{C}_i^{\text{OPT}}, \mathcal{C}_j^{\text{OPT}}) - 9\epsilon' n^2 - 2\epsilon' n|A_i| - \epsilon' n|A_j|$$

Summing over all $i < j$, we get the total number of negative agreements is at least $\text{OPT}^- - 12\epsilon' k^2 n^2$.

So we have, $A \geq \text{OPT} - 44\epsilon' k^2 n^2 = \text{OPT} - \epsilon n^2/2$. However, since we lose $\epsilon n^2/2$ for not finding ϵ'-good partitions of every U_i (as argued before), ϵn^2 due to $\mathcal{C}_{k+1}^{\text{OPT}}$, and $\epsilon n^2/2$ for using $k = \frac{1}{\epsilon}$ we obtain $A \geq \text{OPT} - 3\epsilon n^2$.

The algorithm can fail in four situations:

1. More than $\epsilon/2$ U_is do not have an ϵ'-good partition. However, this happens with probability at most $\epsilon/2$.

2. Lemma 4.19 does not hold for some W_i. This happens with probability at most $2ke^{-\epsilon'^2\epsilon m/2}$.

3. Lemma 4.21 does not hold for some i. This happens with probability at most $\frac{8k}{\epsilon'^2}e^{-\epsilon'^3\epsilon m/4}$

4. Lemma 4.20 does not hold for some pair i, j. This happens with probability at most $4k^2e^{-\epsilon'^2\epsilon m/4}$.

Observing that the latter three probabilities sum up to at most $\epsilon/2$ by our choice of m. So, the algorithm succeeds with probability greater than $1 - \epsilon$. □

Now we need to argue for the case when $|\mathcal{C}_{k+1}^{\mathrm{OPT}}| \geq \epsilon n$. Notice that in this case, using an argument similar to Lemma 4.19, we can show that $|W_{k+1}| \geq \frac{\epsilon m}{2}$ with a very high probability. This is good because, now with a high probability, $U_{i(k+1)}$ will also be ϵ'-good w.r.t. W_{k+1} for most values of i. We can now count the number of negative edges from these vertices and incorporate them in the proof of Theorem 4.23 just as we did for the other k clusters. So in this case, we can modify algorithm DIVIDE&CHOOSE to consider ϵ'-goodness of the $(k+1)$th partitions as well. This gives us the same guarantee as in Theorem 4.23. Thus our strategy will be to run Algorithm DIVIDE&CHOOSE once assuming that $|\mathcal{C}_{k+1}^{\mathrm{OPT}}| \geq \epsilon n$ and then again assuming that $|\mathcal{C}_{k+1}^{\mathrm{OPT}}| \leq \epsilon n$, and picking the better of the two outputs. One of the two cases will correspond to reality and will give us the desired approximation to OPT.

Now each U_i has $O(k^m)$ different partitions. Each iteration takes $O(nm)$ time. There are n/m U_is, so for each partition of W, the algorithm takes time $O(n^2 k^m)$. Since there are k^m different partitions of W, the total running time of the algorithm is $O(n^2 k^{2m}) = O(n^2 e^{O(\frac{1}{\epsilon^{10}} \log(\frac{1}{\epsilon}))})$. This gives us the following theorem:

Theorem 4.24. *For any $\delta \in [0,1]$, using $\epsilon = \frac{\delta}{3}$, Algorithm* DIVIDE&CHOOSE *runs in time $O(n^2 e^{O(\frac{1}{\delta^{10}} \log(\frac{1}{\delta}))})$ and with probability at least $1 - \frac{\delta}{3}$ produces a clustering with number of agreements at least* $\mathrm{OPT} - \delta n^2$.

4.6 Random noise

Going back to our original motivation, if we imagine there is some true correct clustering OPT of our n items, and that the only reason this clustering does not appear perfect is that our function $f(A,B)$ used to label the edges has some error, then it is natural to consider the case that the errors are random. That is, there is some constant noise rate $\nu < 1/2$ and each edge, independently, is mislabeled with respect to OPT with probability ν. In the machine learning context, this is called the problem of learning with random noise. As can be expected, this is much easier to handle than the worst-case problem. In fact, with very simple algorithms one can (w.h.p.) produce a clustering that is quite close to OPT, much closer than the number of disagreements between OPT and f. The analysis is fairly standard (much like the generic transformation of Kearns [97] in the machine learning context, and even closer to the analysis of Condon and Karp for graph partitioning [48]). In fact, this problem nearly matches a special case of the planted-partition problem of McSherry [116]. Shamir and Tsur [128] independently consider the random noise problem in a slightly more general framework—they consider different amounts of noise for positive and negative edges. Their results are similar in spirit as ours. We present our analysis anyway since the algorithms are so simple.

One-sided noise: As an easier special case, let us consider only one-sided noise in which each true "+" edge is flipped to "−" with probability ν. In that case, if u and v are in different clusters of OPT , then $|N^+(u) \cap N^+(v)| = 0$ for certain. But, if u and v are in the same cluster, then every other node in the cluster independently has probability $(1 - \nu)^2$ of being a neighbor to both. So, if the cluster is large, then $N^+(u)$ and $N^+(v)$ will have a non-empty intersection with high probability. So, consider clustering greedily: pick an arbitrary node v, produce a cluster $C_v = \{u : |N^+(u) \cap N^+(v)| > 0\}$, and then repeat on $V - C_v$. With high probability we will correctly cluster *all* nodes whose clusters in OPT are of size $\omega(\log n)$. The remaining nodes might be placed in clusters that are too small, but overall the number of edge-mistakes is only $\tilde{O}(n)$.

Two-sided noise: For the two-sided case, it is technically easier to consider the symmetric difference of $N^+(u)$ and $N^+(v)$. If u and v are in the same cluster of OPT , then every node $w \notin \{u, v\}$ has probability exactly $2\nu(1-\nu)$ of belonging to this symmetric difference. But, if u and v are in different clusters, then all nodes w in OPT $(u) \cup$ OPT (v) have probability $(1-\nu)^2 + \nu^2 = 1 - 2\nu(1 - \nu)$ of belonging to the symmetric difference. (For $w \notin$ OPT $(u) \cup$ OPT (v), the probability remains $2\nu(1 - \nu)$.) Since $2\nu(1 - \nu)$ is a constant less than $1/2$, this means we can confidently detect that u and v belong to different clusters so long as $|\text{OPT}(u) \cup \text{OPT}(v)| = \omega(\sqrt{n \log n})$. Furthermore, using just $|N^+(v)|$, we can approximately sort the vertices by cluster sizes. Combining these two facts, we can w.h.p. correctly cluster all vertices in large clusters, and then just place each of the others into a cluster by itself, making a total of $\tilde{O}(n^{3/2})$ edge mistakes.

4.7 Extensions

So far in the paper, we have only considered the case of edge weights in $\{+, -\}$. Now we consider real valued edge weights. To address this setting, we need to define a cost model – the penalty for placing an edge inside or between clusters.

One natural model is a linear cost function. Specifically, let us assume that all edge weights lie in $[-1, +1]$. Then, given a clustering, we assign a cost of $\frac{1-x}{2}$ if an edge of weight x is within a cluster and a cost of $\frac{1+x}{2}$ if it is placed between two clusters. For example, an edge weighing 0.5 incurs a cost of 0.25 if it lies inside a cluster and 0.75 otherwise. A $0-$weight edge, on the other hand, incurs a cost of $1/2$ no matter what.

Another natural model is to consider weighted disagreements. That is, a positive edge incurs a penalty equal to its weight if it lies between clusters, and zero penalty otherwise, and vice versa for negative edges. The objective in this case is to minimize the sum of weights of positive edges between clusters and negative edges inside clusters. A special case of this problem is edge weights lying in $\{-1, 0, +1\}$. Zero-weight edges incur no penalty, irrespective of the clustering, and thus can be thought of as missing edges.

In this section we show that our earlier results generalize to the case of linear cost functions for the problem of minimizing disagreements. However, we do not have similar results for the case of weighted disagreements or agreements. We give evidence that this latter case is hard to approximate.

Linear cost functions

First we consider the linear cost function on $[-1,+1]$ edges. It turns out, as we show in the following theorem, that any algorithm that finds a good clustering in a graph with $+1$ or -1 edges also works well in this case.

Theorem 4.25. *Let A be an algorithm that produces a clustering on a graph with $+1$ and -1 edges with approximation ratio ρ. Then, we can construct an algorithm A' that achieves a $(2\rho+1)$-approximation on a $[-1,1]-$graph, under a linear cost function.*

Proof. Let G be a $[-1,1]-$graph, and let G' be the graph with $+1$ and -1 edges obtained when we assign a weight of 1 to all positive edges in G and -1 to all the negative edges (0 cost edges are weighted arbitrarily). Let OPT be the optimal clustering on G and OPT' the optimal clustering on G'. Also, let m' be the measure of cost (on G') in the $\{+,-\}$ penalty model and m in the new $[-1,1]$ penalty model.

Then, $m'_{\mathrm{OPT}'} \leq m'_{\mathrm{OPT}} \leq 2m_{\mathrm{OPT}}$. The first inequality follows by design. The latter inequality holds because the edges on which OPT incurs a greater penalty according to m' in G' than according to m in G, are either the positive edges between clusters or negative edges inside a cluster. In both these situations, OPT incurs a cost of at least $1/2$ in m and at most 1 in m'.

Our algorithm A' simply runs A on the graph G' and outputs the resulting clustering A. So, we have, $m'_A \leq \rho m'_{\mathrm{OPT}'} \leq 2\rho m_{\mathrm{OPT}}$.

Now we need to bound m_A in terms of m'_A. Notice that, if a positive edge lies between two clusters in A, or a negative edge lies inside a cluster, then the cost incurred by A for these edges in m' is 1 while it is at most 1 in m. So, the total cost due to such mistakes is at most m'_A. On the other hand, if we consider cost due to positive edges inside clusters, and negative edges between clusters, then OPT also incurs at least this cost on those edges (because cost due to these edges can only increase if they are clustered differently). So cost due to these mistakes is at most m_{OPT}.

So we have,

$$\begin{aligned} m_A &\leq m'_A + m_{\mathrm{OPT}} \leq 2\rho m_{\mathrm{OPT}} + m_{\mathrm{OPT}} \\ &= (2\rho+1)m_{\mathrm{OPT}} \end{aligned}$$

□

Interestingly, the above theorem holds generally for a class of cost functions that we call *unbiased*. An unbiased cost function assigns a cost of at least $\frac{1}{2}$ to positive edges lying between clusters and negative edges inside clusters, and a cost of at most $\frac{1}{2}$ otherwise. A $0-$weight edge always incurs a cost of $\frac{1}{2}$ as before. For example, one such function is $\frac{1+x^3}{2}$ if an edge of weight x lies between clusters and $\frac{1-x^3}{2}$ otherwise.

Weighted agreements/disagreements

Next we consider minimizing weighted disagreements or maximizing weighted agreements. Consider first, the special case of edge weights lying in $\{-1, 0, +1\}$. Notice that, as before, if a perfect clustering exists, then it is easy to find it, by simply removing all the $-$ edges and producing each connected component of the resulting graph as a cluster. The random case is also easy if defined appropriately. However, our approximation techniques do not appear to go through. We do not know how to achieve a constant-factor, or even logarithmic factor, approximation for minimizing disagreements. Note that we can still use our DIVIDE&CHOOSE algorithm to achieve an additive approximation of ϵn^2 for agreements. However, this does not imply a PTAS in this variant, because OPT might be $o(n^2)$.

Now, suppose we allow arbitrary real-valued edge weights, lying in $[-\infty, +\infty]$. For example, the edge weights might correspond to the log odds[6] of two documents belonging to the same cluster. It is easy to see that the problem of minimizing disagreements for this variant is APX-hard, by reducing the problem of minimum multiway cut to it. Specifically, let G be a weighted graph with special nodes $v_1, \cdots, v_k$. The problem of minimum multiway cut is that of finding the smallest cut that separates these special nodes. This problem is known to be APX-hard [68]. We convert this problem into a disagreement minimization problem as follows: among each pair of special nodes v_i and v_j, we put an edge of weight $-\infty$. Then, notice that any clustering algorithm will definitely put each of $v_1, \cdots, v_k$ into separate clusters. The number (or total weight) of disagreements is equal to the value of the cut separating the special nodes. Thus, any algorithm that achieves an approximation ratio of ρ for minimizing disagreements, would achieve an approximation ratio of ρ for minimum multiway cut problem. We get the following:

Theorem 4.26. *The problem of minimizing disagreements on weighted graphs with unbounded weights is APX-hard.*

Note that the above result is pretty weak. It does not preclude the possibility of achieving a constant approximation, similar to the one for $\{+, -\}$-weighted graphs. However we have reason to believe that unlike before, we cannot obtain a PTAS for maximizing agreements in this case. We show that a PTAS for maximizing agreements gives a polynomial time procedure for $O(n^\epsilon)$ coloring a $3-colorable$ graph. While it is unknown whether this problem is NP-Hard, the problem is well-studied and the best known result is due to Blum and Karger [31], who give a polynomial time algorithm to $\tilde{O}(n^{3/14})$ color a 3-colorable graph.

Theorem 4.27. *Given a PTAS for the problem of maximizing agreements, we can use the algorithm to obtain an algorithm for $O(n^\epsilon)$ coloring a $3-colorable$ graph, for any $\epsilon > 0$.*

[6]For example, if the classifier assigns a probability p to two documents being the same, the log odds could be defined as $\log \frac{p}{1-p}$.

Proof. Let $G = (V, E)$ be a 3 colorable graph, and let $m = |E|$ and $n = |V|$. Let K be an n vertex complete graph obtained from G as follows: an edge e of K has weight -1 if e is an edge in G, and has a positive weight of $\delta m/\binom{n}{2}$ otherwise. Here δ is a parameter to be specified later.

If we choose each color class as a cluster, it is easy to see that the resulting clustering agrees on the m negative weight edges and on at least $3\binom{n/3}{2}$ positive weight edges. Thus the total weight of agreements in the optimal clustering is at least $m(1+\delta/3)$. Let us invoke the PTAS for maximizing agreements with $\epsilon' = \delta/30$, then we obtain a clustering which has cost of agreements at least $m(1+\delta/3)/(1+\delta/30) \geq m(1+\delta/5)$.

We now claim that the size of largest cluster is at least $n/5$. Suppose not. Then the weight of positive agreements can be at most $\delta \frac{m}{\binom{n}{2}} \cdot 5 \cdot \binom{n/5}{2}$ which is about $\delta m/5$. Since the total weight of negative edges is m, the total weight of agreements for the clustering cannot be more than $m(1+\delta/5)$, violating the guarantee given by the PTAS. Hence, there exists a cluster of size at least $n/5$ in this clustering. Call this cluster C.

Now observe that since the PTAS returns a clustering with at least $(1+\delta/5)m$ agreements, and the total weight of all positive edges is at most δm, the total weight of negative agreements is at least $(1 - \frac{4\delta}{5})m$. This implies that C contains at most $\frac{4\delta}{5}m$ negative weight edges. Thus the density of negative weight edges in C is at most $\frac{4\delta m}{5}/\binom{n/5}{2} \approx 20\delta \cdot \frac{m}{\binom{n}{2}}$. That is, the cluster C has an edge density of at most about 20δ times that of G and size at least $n/5$.

We can now apply this procedure recursively to C (since C is also 3-colorable). After $2\log_b n$ such recursive steps, where $b = \frac{1}{20\delta}$, we obtain a set of density at most $1/n^2$ times that of C (and hence independent). Call this independent set I. Note that the size of I is at least $n/(5^{2\log_b n})$. Choosing δ such that $b = 5^{2/\epsilon}$, it is easy to verify that I has size at least $n^{1-\epsilon}$.

Now we can remove I from G and iterate on $G - I$ (since $G - I$ is also 3-colorable). It is easy to see that this procedure gives an $O(n^\epsilon)$ coloring of G. $\square$

4.8 Concluding remarks

In this chapter we consider clustering problems with pairwise constraints on vertices. It would be interesting to extend our work to the problems where we are given constraints on three or more vertices. For example, one constraint on three specific vertices may require putting all the three vertices into the same cluster or all three into different clusters. Such constraints arise in the context of Markov Random Fields, where the goal is to find the most likely (energy minimizing) global configuration, given the effect of local structure on the overall energy. More generally, given a relationship between the energy of a configuration and the probability of its occurence, and therefore a distribution on all configurations, it would be desirable to develop an algorithm that outputs a random configuration drawn from this distribution. This turns out to be an important problem in statistical physics and also has applications in statistical machine learning. However in general this problem is #-P hard. It would be interesting to develop algorithms for special classes of Markov Random Fields.

Another interesting extension of our work would be to allow each vertex in the graph to lie in multiple, but few, clusters. This would be useful, for example, in a document classification context where a document may contain information on two different topics. The goal would then be to correctly classify a large fraction (or all) of the edges, while minimizing the number of distinct clusters each document belongs to, or minimizing the total number of clusters. Additionally, in the case that the clustering disagrees with some edges, it would be interesting to also output a confidence level for each edge in the graph specifying how confident the algorithm is in its classification of the edge. This feature may be useful in machine learning applications.

Chapter 5

Min-Ratio Partitioning

5.1 Introduction

In the GENERALIZED-SPARSEST-CUT problem, we are given an undirected graph $G = (V, E)$ with edge capacities c_e. We are also given a subset $\mathbb{D} \subseteq V \times V$, containing $|\mathbb{D}| = k$ source-sink pairs $\{s_i, t_i\}$, called *demand pairs*, with each pair having an associated demand D_i. The sparsity of a set (cut) $S \subseteq V$ is defined to be the factor by which the demand going across the cut exceeds the capacity of the cut:

$$\Phi(S) = \frac{C(S, \bar{S})}{D(S, \bar{S})}$$

where $D(S, \bar{S})$ is the net demand going from the terminals in S to those outside S, and $C(S, \bar{S})$ is the total capacity of edges exiting S. The sparsest cut is defined to be the cut with the minimum sparsity, and its sparsity is denoted by $\Phi = \min_{S \subseteq V} \Phi(S)$.

The GENERALIZED-SPARSEST-CUT problem is NP-hard even when there is a unit demand between all pairs of vertices. In this latter case, the problem is simply known as the SPARSEST-CUT problem. There is much work on approximating this fundamental problem (see, e.g., [109, 130]), and $O(\log k)$ approximations were previously known [111, 18]. In a recent breakthrough, Arora, Rao and Vazirani [17] developed an improved $O(\sqrt{\log n})$ approximation for the uniform demands SPARSEST-CUT based on a semidefinite programming relaxation of the problem. In this chapter, we extend the work of Arora et al. to the general case, obtaining a weaker $O(\log^{3/4} n)$ approximation.

The closely related MULTICUT problem has the same input as GENERALIZED-SPARSEST-CUT, but the goal is to find a smallest (or least weight) subset of the edges $M \subseteq E$ whose removal disconnects all the demand pairs, i.e., in the subgraph $(V, E \setminus M)$ every s_i is disconnected from its corresponding vertex t_i. This natural partitioning problem has also been extensively studied, and is known to have an $O(\log n)$ approximation [70] via linear programming. Unfortunately, the semidefinite programming relaxation that leads to an improved approximation for SPARSEST-CUT does not give a similar improvement for the MULTICUT problem, and was in fact recently shown to have an integrality gap of $\Omega(\log n)$ [3].

In this chapter, we also study the inapproximability of the MULTICUT and SPARSEST-CUT problems. We show that assuming the *Unique Games Conjecture* of Khot [101], these problems are NP-hard to approximation within any constant factor. If a stronger version of the conjecture is true, then our result implies an $\Omega(\sqrt{\log \log n})$ hardness for the two problems. This is the first hardness of approximation result for SPARSEST-CUT, and improves on a previous APX-hardness result for MULTICUT. Our result also extends to the MIN-2CNF≡ DELETION problem, which is a special case of MULTICUT. In the MIN-2CNF≡ DELETION problem the input is a weighted set of clauses on n variables, each clause of the form $x = y$, where x and y are literals, and the goal is to find an assignment to the variables minimizing the total weight of unsatisfied clauses. We elaborate on this problem towards the end of this chapter.

5.1.1 SPARSEST-CUT and low-distortion metric embeddings

The Sparsest Cut problem is closely linked to the problem of embedding general metrics into ℓ_1 metrics with a low distortion. In fact, one of the first major applications of metric embeddings in Computer Science was an $O(\log k)$ approximation to the SPARSEST-CUT problem with non-uniform demands [111, 18]. This result was based on a fundamental theorem of Bourgain [33] in the local theory of Banach spaces, which showed that any finite n-point metric could be embedded into ℓ_1 space (and indeed, into any of the ℓ_p spaces) with distortion $O(\log n)$. The connection between these results uses the fact that the Generalized SPARSEST-CUT problem seeks to minimize a linear function over all cuts of the graph, which is equivalent to optimizing over all n-point ℓ_1 metrics:

$$\Phi = \min_{S \subseteq V} \frac{C(S,\bar{S})}{D(S,\bar{S})} = \min_{\text{cut metrics } \delta_S} \frac{\sum_{(u,v)\in E} c_{uv}\ \delta_S(u,v)}{\sum_{ij} D_i\ \delta_S(s_i,t_i)} = \min_{\mathrm{d}\in \ell_1} \frac{\sum_{(u,v)\in E} c_{uv}\ \mathrm{d}(u,v)}{\sum_{ij} D_i\ \mathrm{d}(s_i,t_i)} \tag{5.1}$$

Since the problem of minimizing over all ℓ_1 metrics is NP-hard, we can optimize over all n-point metrics instead, and then use an algorithmic version of Bourgain's embedding to embed into ℓ_1 with only an $O(\log n)$ loss in performance.

A natural extension of this idea is to optimize over a smaller class of metrics that contains ℓ_1; a natural candidate for this class is NEG, the class of n-point *metrics of negative type*[1]. These are just the metrics obtained by squaring an Euclidean metric, and hence are often called "ℓ_2-squared" metrics. It is known that the following relationships hold:

$$\ell_2 \text{ metrics } \subseteq \ell_1 \text{ metrics } \subseteq \mathsf{NEG} \text{ metrics.} \tag{5.2}$$

Since it is possible to optimize over NEG via semidefinite programming, this gives us a semidefinite relaxation for the GENERALIZED-SPARSEST-CUT problem [73]. Now if we could prove that

[1]Note that NEG usually refers to all *distances* of negative-type, even those that do not obey the triangle inequality. In this thesis, we will use NEG only to refer to negative-type *metrics*.

n-point metrics in NEG embed into ℓ_1 with distortion D, we would get a D-approximation for SPARSEST-CUT; while this D has been conjectured to be $O(\sqrt{\log n})$ or even $O(1)$, no bounds better than the $O(\log n)$ were known prior to this work. (See the Section 5.3 for subsequent progress towards the resolution of this conjecture.)

In a recent breakthrough, Arora, Rao, and Vazirani [17] showed that every n-point metric in NEG has a contracting embedding into ℓ_1 such that the sum of the distances decreases by only $O(\sqrt{\log n})$. Formally, they showed that the SDP relaxation had an integrality gap of $O(\sqrt{\log n})$ for the case of uniform demand SPARSEST-CUT; however, this is equivalent to the above statement by the results of Rabinovich [121]. We extend the techniques of Arora, Rao, and Vazirani to give embeddings for n-point metrics in NEG into ℓ_2 with distortion $O(\log^{3/4} n)$, thereby implying an $O(\log^{3/4} n)$-approximation algorithm for the GENERALIZED-SPARSEST-CUT problem.

5.1.2 The Unique Games Conjecture

Our hardness results are based on a conjecture of Khot [101] that states that it is NP-hard to determine whether the value of a certain kind of 2-prover game, in particular a *Unique 2-prover game*, is close to 0 or close to 1.

Unique 2-prover game is the following problem. The input is a bipartite graph $G = (Q, E_Q)$, where each side $p = 1, 2$ contains $n = |Q|/2$ vertices denoted $q_1^p, \cdots, q_n^p$, and represents n possible questions to prover p. In addition, the input contains for each edge $(q_i^1, q_j^2) \in E_Q$ a non-negative weight $w(q_i^1, q_j^2)$. These edges will be called *question edges*, to distinguish them from edges in the MULTICUT instance. Each question is associated with a set of d distinct answers, denoted by $[m] = \{1, \ldots, m\}$. The input also contains, for every edge $(q_i^1, q_j^2) \in E_Q$, a bijection $b_{ij} : [m] \to [m]$, which maps every answer of question q_i^1 to a distinct answer for q_j^2.

A *solution* A to the 2-prover game consists of an answer $A_i^p \in [m]$ for each question q_i^p (i.e., a sequence $\{A_i^p\}$ over all $p \in [2]$ and $i \in [n]$). The solution is said to satisfy an edge $(q_i^1, q_j^2) \in E_Q$ if the answers A_i^1 and A_j^2 agree, i.e., $A_j^2 = b_{ij}(A_i^1)$. We assume that the total weight of all the edges in E_Q is 1 (by normalization). The value of a solution is the total weight of all the edges satisfied by the solution. The value of the game is the maximum value achievable by any solution to the game.

Conjecture 5.1 (Unique Games [101]). For every fixed $\eta, \delta > 0$ there exists $m = m(\eta, \delta)$ such that it is NP-hard to determine whether a unique 2-prover game with answer set size m has value at least $(1 - \eta)$ or at most δ.

We will also consider stronger versions of the Unique Games Conjecture in which η, δ, and m are functions of n. Specifically, we will consider versions with $\eta \leq O(1/\sqrt{\log \log n})$, $\delta \leq 1/(\log n)^{\Omega(\eta)}$, and $m = m(\eta, \delta) \leq O(\log n)$. We denote the size of an input instance by N. Notice that $N = (nm)^{\Theta(1)}$, and is thus polynomial in n as long as $m \leq n$, and in particular for $m \leq O(\log n)$.

Plausibility of the conjecture and its stronger version.

The Unique Games Conjecture has been used to show optimal inapproximability results for VERTEX COVER [100] and MAX-CUT [99]. Proving the conjecture using current techniques appears quite hard. In particular, the asserted NP-hardness is much stronger than what we can obtain via standard constructions using the PCP theorem [15, 14] and the parallel repetition theorem [124], two deep results in computational complexity.

Although the conjecture seems difficult to prove in general, some special cases are well-understood. In particular, if at all the Unique Games Conjecture is true, then necessarily $m \geq \max\{1/\eta^{1/10}, 1/\delta\}$. This follows from a semidefinite programming algorithm presented by Khot in [101]. Trevisan [136] recently improved the algorithm of Khot to obtain the following guarantee: given an instance of Unique Games with value $1 - \frac{\epsilon}{\log n}$ for some $\epsilon > 0$, the algorithm obtains a solution with value $1 - O(\epsilon^{1/3})$. This result implies that the conjecture is false when $\eta = O(1/\log n)$ (assuming $P \neq NP$). Very recently, we were informed [38] of a different lower bound on η expressed in terms of δ and m; We do not know the precise values for the three parameters obtained in this result, however the values that we use in our construction are not excluded by this result. Feige and Reichman [63] recently showed that for every constant $L > 0$ there exists a constant $\delta > 0$, such that it is NP-hard to distinguish whether a unique 2-prover game (with $m = m(L, \delta)$) has value at least $L\delta$ or at most δ; this result falls short of the Unique Games Conjecture in that $L\delta$ is bounded away from 1.

We obtain a $\Omega(\min\{\frac{1}{\eta}, \log\frac{1}{\delta}\})$-hardness for MULTICUT and SPARSEST-CUT (see Theorem 5.5 below). Assuming the Unique Games Conjecture with $\eta \leq O(1/\sqrt{\log\log n})$, $\delta \leq 1/(\log n)^{\Omega(\eta)}$, and $m = m(\eta, \delta) \leq O(\log n)$, which is not excluded by the above results, this implies an $\Omega(\sqrt{\log\log n})$ hardness (Corollary 5.7). Further results on the conjecture may imply that the hardness obtained by us is weaker than reported in Corollary 5.7.

5.2 Our results

An improved approximation for GENERALIZED-SPARSEST-CUT

We extend the techniques of Arora, Rao, and Vazirani to give embeddings for n-point metrics in NEG into ℓ_2 with distortion $O(\log^{3/4} n)$. More generally, we obtain the following theorem.

Theorem 5.2. *Given (V, d), a negative-type metric, and a set of terminal-pairs $\mathbb{D} \subseteq V \times V$ with $|\mathbb{D}| = k$, there is a contracting embedding $\varphi : V \to \ell_2$ such that for all pairs $(x, y) \in \mathbb{D}$,*

$$\|\varphi(x) - \varphi(y)\|_2 \geq \frac{1}{O(\log^{3/4} k)}\, \mathrm{d}(x, y).$$

Note that the above theorem requires the embedding to be contracting for all node pairs, but the resulting contraction needs to be small only for the terminal pairs. In particular, when $\mathbb{D} = V \times V$, the embedding is an $O(\log^{3/4} n)$-distortion embedding into ℓ_2. Though we also give a randomized polynomial-time algorithm to find this embedding, let us point out that optimal embeddings into ℓ_2 can be found using semidefinite programming [111, Thm. 3.2(2)].

Let us now note some simple corollaries.

Theorem 5.3. *Every n-point metric in* NEG *embeds into ℓ_1 with $O(\log^{3/4} n)$ distortion, and every n-point metric in ℓ_1 embeds into Euclidean space ℓ_2 with $O(\log^{3/4} n)$ distortion. These embeddings can be found in polynomial time.*

The existence of both embeddings follows immediately from equation (5.2). To find the map $\mathsf{NEG} \to \ell_1$ in polynomial time, we can use the fact that every ℓ_2 metric can be embedded into ℓ_1 isometrically; if we so prefer, we can find a distortion-$\sqrt{3}$ embedding into ℓ_1 in deterministic polynomial time using families of 4-wise independent random variables [111, Lemma 3.3].

Theorem 5.4. *There is a randomized polynomial-time $O(\log^{3/4} k)$-approximation algorithm for the* SPARSEST-CUT *problem with non-uniform demands.*

Theorem 5.4 thus extends the results of Arora et al. [17] to the case of non-uniform demands, albeit it proves a weaker result than the $O(\sqrt{\log k})$ approximation that they achieve for uniform demands.

The proof of Theorem 5.4 follows from the fact that the existence of distortion-D embeddings of negative-type metrics into ℓ_1 implies an integrality gap of at most D for the semidefinite programming relaxation of the GENERALIZED-SPARSEST-CUT problem. Furthermore the embedding can be used to find such a cut as well. (For more details about this connection of embeddings to the GENERALIZED-SPARSEST-CUT problem, see the survey by Shmoys [130, Sec. 5.3]; the semidefinite programming relaxation can be found in the survey by Goemans [73, Sec. 6]).

The hardness of approximating SPARSEST-CUT and MULTICUT

We prove that if a strong version of the Unique Games Conjecture of Khot [101] is true, then MULTICUT is NP-hard to approximate to within a factor of $\Omega(\sqrt{\log\log n})$.[2] Under the original version of this conjecture, our reduction shows that for every constant $L > 0$, it is NP-hard to approximate MULTICUT to within factor L. Our methods yield similar bounds for SPARSEST-CUT and MIN-2CNF≡ DELETION. The results also extend to the CORRELATION CLUSTERING problem of minimizing disagreements in a weighted graph (see Chapter 4), as this problem is known to be

[2] The conference version of our paper [40] presented a weaker bound than that of Theorem 5.5, with the dependence on η being $\Omega(\log 1/\eta)$. The reduction is exactly the same, but the analysis is different (and perhaps simpler), as the current version uses Friedgut's Junta Theorem [66] rather than a theorem of Kahn, Kalai and Linial [92]. This improvement was motivated, in part, by the integrality ratio of [102] for unique games, which suggests a significant asymmetry between $\eta(n)$ and $\delta(n)$.

equivalent to the MULTICUT problem on weighted graphs. In particular, we prove the following hardness result:

Theorem 5.5. *Suppose that for $\eta = \eta(n)$, $\delta = \delta(n)$, and $m = m(\eta, \delta) \leq O(\log n)$, it is NP-hard to determine whether a unique 2-prover game with $|Q| = 2n$ vertices and answer set size m has value at least $1 - \eta(n)$ or at most $\delta(n)$. Then there exists $L(n) = \Omega\left(\min\left\{\frac{1}{\eta(n^{\Omega(1)})}, \log\frac{1}{\delta(n^{\Omega(1)})}\right\}\right)$ such that it is NP-hard to approximate* MULTICUT, SPARSEST-CUT, *and* MIN-2CNF≡ DELETION *to within factor $L(n)$.*

This theorem immediately implies the following two specific hardness results.

Corollary 5.6. *The Unique Games Conjecture implies that, for every constant $L > 0$, it is NP-hard to approximate* MULTICUT, SPARSEST-CUT, *and* MIN-2CNF≡ DELETION *to within factor L.*

Corollary 5.7. *If a stronger version of the Unique Games Conjecture with $\eta \leq O(1/\sqrt{\log\log n})$, $\delta \leq 1/(\log n)^{\Omega(\eta)}$, and $m = m(\eta, \delta) \leq O(\log n)$ is true, then for some fixed $c > 0$, it is NP-hard to approximate* MULTICUT, SPARSEST-CUT, *and* MIN-2CNF≡ DELETION *to within factor $\Omega(\sqrt{\log\log n})$.*

For SPARSEST-CUT our hardness results hold only for the search version (in which the algorithm needs to produce a cutset and not only its value), since our proof employs a Cook reduction.

5.3 Related and subsequent work

MULTICUT and SPARSEST-CUT are fundamental combinatorial problems, with connections to multicommodity flow, expansion, and metric embeddings. Both problems can be approximated to within an $O(\log k)$ factor through linear programming relaxations [109, 70, 18, 111]. These bounds match the lower bounds on the integrality gaps up to constant factors [109, 70].

The decade-old $O(\log k)$-approximation for the uniform-demands version of SPARSEST-CUT was recently improved by Arora, Rao and Vazirani [17] to a $O(\sqrt{\log n})$-approximation. In their ground-breaking work, Arora, Rao and Vazirani use a semi-definite programming relaxation with triangle-inequality constraints to give a low-average distortion embedding from negative-type metrics into ℓ_1. The obvious modification of the semidefinite program used for SPARSEST-CUT to solve MULTICUT was recently shown to have an integrality ratio of $\Omega(\log k)$ [3], which matches the approximation factor and integrality gap of previously analyzed linear programming relaxations for this problem.

Our work adopts and adapts techniques of Arora, Rao and Vazirani [17], who gave an $O(\sqrt{\log n})$-approximation for the uniform demand case of SPARSEST-CUT. In fact, using their results about the behavior of projections of negative-type metrics almost as a black-box, we obtain an $O(\log^{5/6} n)$-approximation for GENERALIZED-SPARSEST-CUT. Our approximation factor is further improved to $O(\log^{3/4} n)$ by the results of Lee [107] showing that the hyperplane separator algorithm of Arora et al. [17, Section 3] itself gives an $O(\sqrt{\log n})$ approximation for the uniform demand case.

Over the past decade, there has been a large body of work on low-distortion embeddings of finite metrics; see, e.g., [22, 33, 42, 61, 78, 79, 105, 111, 115, 113, 122], motivated in part by the fact that it has proved invaluable in many algorithmic applications. Other references and examples can be found in the surveys by Indyk [87] and Linial [110], or in the chapter by Matoušek [114]. Our work stems in spirit from many of these papers. However, it draws most directly on the technique of *measured descent* developed by Krauthgamer et. al. [105].

Independently of our work, Lee [107] has used so-called "scale-based" embeddings to give low-distortion embeddings from ℓ_p $(1 < p < 2)$ into ℓ_2. The paper gives a "Gluing Lemma" of the following form: if for every distance scale i, we are given a contracting embedding ϕ_i such that each pair x, y with $\mathrm{d}(x,y) \in [2^i, 2^{i+1})$ has $\|\phi_i(x) - \phi_i(y)\| \geq \frac{\mathrm{d}(x,y)}{K}$, one can glue them together to get an embedding $\phi : \mathrm{d} \to \ell_2$ with distortion $O(\sqrt{K \log n})$. His result is a generalization of [105], and of our Lemma 5.21; using this gluing lemma, one can derive an ℓ_2 embedding from the decomposition bundles of Theorem 5.19 without using any of the ideas in Section 5.4.5.

Following the initial publication of this work, Arora, Lee, and Naor [16] built upon our techniques to obtain an embedding from negative-type metrics into ℓ_2 with distortion $O(\sqrt{\log n} \log \log n)$, implying an approximation to GENERALIZED-SPARSEST-CUT with the same factor of approximation. Their improvement lies in a stronger gluing lemma. This result is essentially tight, as it is known that embedding negative-type metrics into ℓ_2 requires $\Omega(\sqrt{\log n})$ distortion [59].

On the hardness side, it is known that MULTICUT is APX-hard [49], i.e., there exists a constant $c > 1$, such that it is NP-hard to approximate MULTICUT to within a factor smaller than c. It should be noted that this hardness of approximation holds even for $k = 3$, and that the value of c is not specified therein, but it is certainly much smaller than 2. The MIN-2CNF$\equiv$ DELETION problem is also known to be APX-hard, as follows, e.g., from [83].

Assuming his Unique Games Conjecture, Khot [101, Theorem 3] essentially obtained an arbitrarily large constant-factor hardness for MIN-2CNF$\equiv$ DELETION, and this implies, using the aforementioned reduction of [103], a similar hardness factor for MULTICUT. These results are not noted in [101], and are weaker than our results in several respects. First, our quantitative bounds are better; thus if a stronger, yet almost as plausible, version of this conjecture is true, then our lower bound on the approximation factor improves to $L = \Omega(\sqrt{\log \log n})$, compared with the roughly $\Omega((\log \log n)^{1/4})$ hardness that can be inferred from [101]; this can be viewed as progress towards proving tight inapproximability results for MULTICUT. Second, by qualitatively strengthening our MULTICUT result to a bicriteria version of the problem, we extend our hardness results to the SPARSEST-CUT problem. It is unclear whether Khot's reduction similarly leads to a hardness result for SPARSEST-CUT. Finally, our proof is simpler (both the reduction and its analysis), and makes direct connections to cuts (in a hypercube), and thus may prove useful in further investigation of such questions.

For SPARSEST-CUT, no hardness of approximation result was previously known. Independent of our work, Khot and Vishnoi [102] have recently used a different construction to show an arbitrarily large constant factor hardness for SPARSEST-CUT assuming the Unique Games Conjecture; their hardness factor improves to $\Omega(\log\log n)^{1/6-\epsilon}$, under a stronger quantitative version of the conjecture similar to the one in Corollary 5.7. Additionally, they show the existence of a negative-type metric that must incur a distortion of $\Omega(\log\log n)^{1/4-\epsilon}$ when embedded into ℓ_1. This result in turn implies an integrality gap of $\Omega((\log\log)^{1/4-\epsilon})$, for the semidefinite programming relaxation of the SPARSEST-CUT problem with triangle inequalities.

5.4 An improved approximation for SPARSEST-CUT

5.4.1 A high-level overview

The proof of the Main Theorem 5.2 proceeds thus: we first classify the terminal pairs in $\mathbb{D}$ by distance scales. We define the scale-i set $\mathbb{D}_i$ to be the set of all pairs $(x, y) \in \mathbb{D}$ with $\mathrm{d}(x, y) \approx 2^i$. For each scale i, we find a partition of V into components such that for a constant fraction of the terminal pairs $(x, y) \in \mathbb{D}_i$, the following two "good" events happens: (1) x and y lie in different components of the partition, and (2) the distance from x to any other component is at least $\eta\, 2^i$, and the same for y. Here $\eta = 1/O(\sqrt{\log k})$. Informally, both x and y lie deep within their *distinct* components, and this happens for a constant fraction of the pairs $(x, y) \in \mathbb{D}_i$. This partition defines a contracting embedding of the points into a one-dimensional ℓ_1 metric (a line) such that every pair (x, y), for which the above "good" events happen, has low distortion. (The details of this process are given in Section 5.4.3; the proofs use ideas from the paper by Arora, Rao, and Vazirani [17] and the subsequent improvements by Lee [107].)

Note that the good event happens for only a constant fraction of the pairs in $\mathbb{D}_i$, and we have little control over which of the pairs will be the lucky ones. However, to obtain low distortion for every terminal pair, we want a partitioning scheme that separates a random constant fraction of the pairs in $\mathbb{D}_i$. To this end, we employ a simple reweighting scheme (reminiscent of the Weighted Majority algorithm [112] and many other applications). We just duplicate each unlucky pair and repeat the above process $O(\log k)$ times. Since each pair that is unlucky gets a higher weight in the subsequent runs, a simple argument given in Section 5.4.4 shows that each pair in $\mathbb{D}_i$ will be separated in at least $\log k$ of these $O(\log k)$ partitions. (Picking one of these partitions uniformly at random would now ensure that each vertex is separated with constant probability.)

We therefore obtain a good partition for each distance scale individually. We could now use these $O(\log k)$ partitions naïvely, by concatenating the corresponding "line-embeddings", to construct an embedding where the contraction for the pairs in $\mathbb{D}$ would be bounded by $\sqrt{\log k}/\eta = O(\log k)$. However, this would be no better than the previous bounds, and hence we have to be more careful. We slightly adapt the *measured descent* embeddings of Krauthgamer et al. [105] to combine the $O(\log k)$ partitions for the various distance scales to get a distortion-$O(\sqrt{\log k/\eta}) = O(\log^{3/4} k)$ embedding. The details of the embedding are given in Section 5.4.5.

5.4.2 Notation and definitions

The SPARSEST-CUT SDP

Recall that in the SPARSEST-CUT problem, our goal is to find a cut S that minimizes the function $\Phi(S) = C(S, \bar{S})/D(S, \bar{(S)})$. This problem is equivalent to the following program:

$$\begin{aligned} \min \sum_{(u,v)\in E} c_{uv}\, \mathrm{d}(u, v) \qquad & \text{s.t.} \\ \sum_{ij} D_i\, \mathrm{d}(s_i, t_i) &= 1 \\ \mathrm{d} &\in \ell_1 \end{aligned}$$

This problem is NP-hard [96], so we relax the last constraint to $\mathrm{d} \in \mathsf{NEG}$. This gives us the following semi-definite program, with $\mathrm{d}(u, v) = \|x_u - x_v\|^2 = 2(1 - x_u \cdot x_v)$:

$$\begin{aligned} \min \sum_{(u,v)\in E} c_{uv}\, 2(1 - x_u \cdot x_v) \qquad & \text{s.t.} \\ \sum_{ij} D_i\, (1 - x_{s_i} \cdot x_{t_i}) &= 1/2 \\ \|x_u\|^2 &= 1 \qquad \forall u \end{aligned}$$

Let Φ_{NEG} denote the value of the semidefinite program.

$$\Phi_{\mathsf{NEG}} = \min_{\mathrm{d}\in\mathsf{NEG}} \frac{\sum_{(u,v)\in E} c_{uv}\, \mathrm{d}(u, v)}{\sum_i D_i\, \mathrm{d}(s_i, t_i)} \tag{5.3}$$

This quantity can be approximated well in polynomial time (see, e.g., [73]). Since $\ell_1 \subseteq \mathsf{NEG}$, it follows that $\Phi_{\mathsf{NEG}} \leq \Phi$. On the other hand, if we can embed n-point metrics in NEG into ℓ_1 with distortion at most D, we can obtain a solution of value at most $D \times \Phi_{\mathsf{NEG}}$. It follows that $\Phi \leq D \times \Phi_{\mathsf{NEG}}$, and the solution is a D approximation to GENERALIZED-SPARSEST-CUT.

Metrics

The input to our embedding procedure is a *negative-type metric* (V, d) with $|V| = n$. We can, and indeed will, use the following standard correspondence between finite metrics and graphs: we set V to the node set of the graph $G = (V, E = V \times V)$, where the length of an edge (x, y) is set to $\mathrm{d}(x, y)$. This correspondence allows us to perform operations like deleting edges to partition the graph. By scaling, we can assume that the smallest distance in (V, d) is 1, and the maximum distance is some value $\Delta(\mathrm{d})$, the *diameter* of the graph.

It is well-known that any *negative-type* distance space admits a geometric representation as the *square of a Euclidean metric*; i.e., there is a map $\psi : V \to \mathbb{R}^n$ such that $\|\psi(x) - \psi(y)\|_2^2 = \mathrm{d}(x, y)$ for every $x, y \in V$ [57, Thm. 6.2.2]. Furthermore, the fact that d is a metric implies that the angle subtended by any two points at a third point is non-obtuse. Since this map can be found in polynomial time using semidefinite programming, we will assume that we are also given such a map ψ. For any node $x \in V$, we use $\vec{x}$ to denote the point $\psi(x) \in \mathbb{R}^n$.

Terminal pairs

We are also given a set of *terminal pairs* $\mathbb{D} \subseteq V \times V$; these are the pairs of nodes for which we need to ensure a small contraction. In the sequel, we will assume that each node in V takes part in *at most one* terminal-pair in $\mathbb{D}$. This is without loss of generality: if a node x belongs to several terminal pairs, we add new vertices x_i to the graph at distance 0 from x, and replace x in the i-th terminal pair with x_i. (Since this transformation adds at most $O(|\mathbb{D}|)$ nodes, it does not asymptotically affect our results.) Note that a result of this is that $\mathbb{D}$ may have two terminal pairs (x, y) and (x', y') such that $\mathrm{d}(x, x') = \mathrm{d}(y, y') = 0$.

A node $x \in V$ is a *terminal* if there is a (unique) y such that $(x, y) \in \mathbb{D}$; call this node y the *partner* of x. Define $\mathbb{D}_i$ to be the set of node-pairs whose distance according to d is approximately 2^i.

$$\mathbb{D}_i = \{(x, y) \in \mathbb{D} \mid 2^i \le \mathrm{d}(x, y) < 2^{i+1}\} \tag{5.4}$$

We use the phrase *scale-i* to denote the distances in the interval $[2^i, 2^{i+1})$, and hence $\mathbb{D}_i$ is merely the set of terminal pairs that are at distance scale i. If $(x, y) \in \mathbb{D}_i$, then x and y are called *scale-i terminals*. Let $\mathcal{D}$ be the set of all terminal nodes, and $\mathcal{D}_i$ be the set of scale-i terminals.

The radius r *ball* around $x \in V$ is naturally defined to be $\mathbf{B}(x, r) = \{z \in V \mid \mathrm{d}(x, z) \le r\}$. Given a set $S \subseteq V$, the ball $\mathbf{B}(S, r) = \cup_{x \in S} \mathbf{B}(x, r)$.

Metric decompositions: suites and bundles

Much of this section will deal with finding decompositions of metrics (and of the underlying graph) with specific properties; let us define these here. Given a distance scale i and a partition P_i of the graph, let $C_i(v)$ denote the component containing a vertex $v \in V$. We say that a pair $(x, y) \in \mathbb{D}_i$ is *δ-separated* by the partition P_i if

- the vertices x and y lie in different components; i.e., $C_i(x) \neq C_i(y)$, and
- both x and y are "far from the boundary of their components", i.e., $\mathrm{d}(x, V \setminus C_i(x)) \ge \delta\, \mathrm{d}(x, y)$ and $\mathrm{d}(y, V \setminus C_i(y)) \ge \delta\, \mathrm{d}(x, y)$.

A *decomposition suite* Π is a collection $\{P_i\}$ of partitions, one for each distance scale i between 1 and $\lfloor \log \Delta(\mathrm{d}) \rfloor$. Given a *separation function* $\delta(x, y) : V \times V \to [0, 1]$, the decomposition suite

Π is said to $\delta(x,y)$-*separate* $(x,y) \in \mathbb{D}$ if for the distance scale i such that $(x,y) \in \mathbb{D}_i$, (x,y) is $\delta(x,y)$-separated by the corresponding partition $P_i \in \Pi$.

Finally, a $\delta(x,y)$-*decomposition bundle* is a collection $\{\Pi_j\}$ of decomposition suites such that for each $(x,y) \in \mathbb{D}$, at least a constant fraction of the Π_j $\delta(x,y)$-separate the pair (x,y).

In Section 5.4.3, we show how to create a decomposition suite that $\Omega(1/\sqrt{\log k})$-separates a constant fraction of the pairs $(x,y) \in \mathbb{D}_i$, for all distance scales i. Using this procedure and a simple reweighting argument, we construct a $\Omega(1/\sqrt{\log k})$-decomposition bundle with $O(\log k)$ suites. Finally, in Section 5.4.5, we show how decomposition bundles give us embeddings of the metric d into ℓ_2.

5.4.3 Creating decomposition suites

In this section, we will give the procedure Project-&-Prune that takes a distance scale i, and constructs a partition P_i of V that η-separates at least a constant fraction of the pairs in $\mathbb{D}_i$. Here we use $\eta = \frac{1}{4c\sqrt{\log k}}$, where c is a constant to be defined later; let us also define $f = \frac{1}{4\eta} = c\sqrt{\log k}$.

Procedure Project-&-Prune**:**

Input: The metric (V, d), its geometric representation where $x \in V$ is mapped to $\vec{x} \in \mathbb{R}^n$, and, a distance scale i.

1. **Project.** In this step, we pick a random direction and project the points in V on the line in this direction. Formally, we pick a random unit vector u. Let $p_x = \sqrt{n}\ \langle \vec{x}, u \rangle$ be the *normalized projection* of the point $\vec{x}$ on u.

2. **Bucket.** Let $\ell = 2^{i/2}$, and set $\beta = \ell/6$. Informally, we will form *buckets* by dividing the line into intervals of length β. We then group the terminals in $\mathcal{D}_i$ according to which interval $(\bmod\ 4)$ they lie in. (See Figure 5.1.) Formally, for each $a = 0, 1, 2, 3$, define

$$A_a = \{\ x \in \mathcal{D}_i \mid p_x \in \cup_{m \in \mathbb{Z}}\big((4m+a)\beta, (4m+1+a)\beta\big)\ \}$$

   A terminal pair $(x,y) \in \mathbb{D}_i$ is *split* by A_a if $x \in A_a$ and $y \in A_{(a+2) \bmod 4}$. If the pair (x,y) is not split by any A_a, we remove both x and y from the sets A_a. For $a \in \{0,1\}$, let $\mathbb{B}_a \subseteq \mathbb{D}_i$ be the set of terminal pairs split by A_a or A_{a+2}.

3. **Prune.** If there exist terminals $x \in A_a$ and $y \in A_{(a+2) \bmod 4}$ for some $a \in \{0,1\}$ (not necessarily belonging to the same terminal pair) with $\mathrm{d}(x,y) < \ell^2/f$, we remove x and y and their partners from the sets $\{A_a\}$. We repeat until no such pairs remain.

4. **Cleanup.** For each a, if $(x,y) \in \mathbb{B}_a$ and the above pruning step has removed either of x or y, then we remove the other one as well, and remove (x,y) from $\mathbb{B}_a$. Once this is done, $\mathbb{B}_a = \mathbb{D}_i \cap (A_a \times A_{(a+2) \bmod 4})$ is once again the set of terminal pairs split by A_a or A_{a+2}.

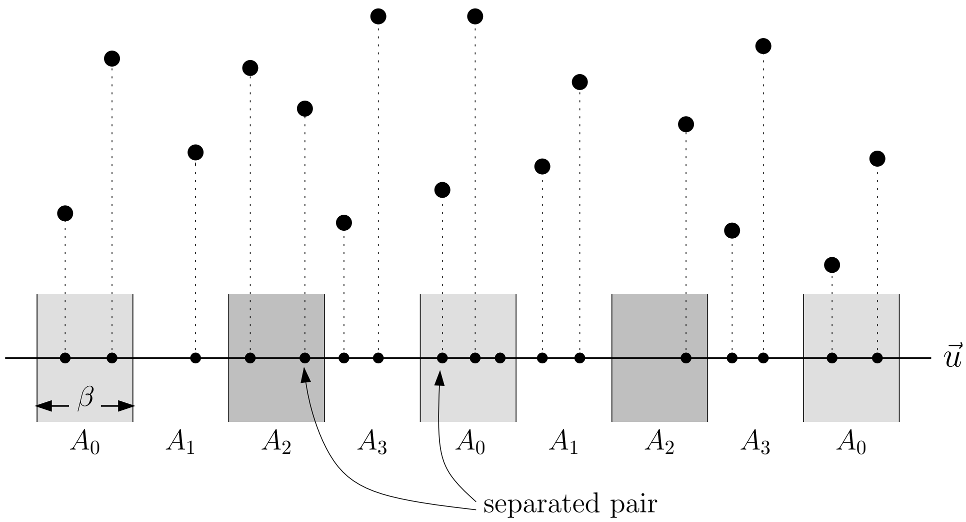


Figure 5.1: Projection and Bucketing

5. If $\max\{|\mathbb{B}_0|, |\mathbb{B}_1|\} \leq \frac{1}{64}|\mathbb{D}_i|$, go back to Step 1, else go to Step 6.

6. Say the set $\mathbb{B}_a$ has more pairs than $\mathbb{B}_{(1-a) \bmod 2}$. Define the partition P_i by deleting all the edges at distance $\ell^2/2f$ from the set A_a. (This step can be thought of as taking $C = \mathbf{B}(A_a, \ell^2/2f)$, and defining the partition P_i to be $G[C]$ and $G[V \setminus C]$, the components induced by C and $V \setminus C$.)

Note the procedure above ensures that for *any pair of terminals* $(x, y) \in A_a \times A_{(a+2) \bmod 4}$, the distance $\mathrm{d}(x, y)$ is at least $\ell^2/f = 2^i/f$, even if $(x, y) \notin \mathbb{D}_i$. Why do we care about these pairs? It is because the separation of ℓ^2/f between the sets A_a and $A_{(a+2) \bmod 4}$ ensures that the balls of radius $\frac{\ell^2}{2f}$ around these sets are disjoint.

This in turn implies that terminal pairs $(x, y) \in \mathbb{D}_i \cap (A_a \times A_{(a+2) \bmod 4})$ are η-separated upon deleting the edges in Step 6. Indeed, for such a pair (x, y), the components $C_i(x)$ and $C_i(y)$, obtained upon deleting the edges at distance $\frac{\ell^2}{2f}$ from the set A_a, are distinct, and both $\mathrm{d}(x, V \setminus C_i(x))$ and $\mathrm{d}(y, V \setminus C_i(y))$ are at least $\frac{\ell^2}{2f} \geq \frac{d(x,y)}{4f}$.

The following theorem now shows that the procedure Project-&-Prune terminates quickly.

Theorem 5.8. *For any distance scale i, the procedure* Project-&-Prune *terminates in a constant number of iterations. This gives us a random polynomial-time algorithm that outputs a partition P_i which η-separates at least $\frac{1}{64}|\mathbb{D}_i|$ pairs of $\mathbb{D}_i$.*

The proof of this theorem has two parts, which we will prove in the next two subsections. We first show that the sets $\mathbb{B}_0$ and $\mathbb{B}_1$ contain most of $\mathbb{D}_i$ before the pruning step (with a high probability over the random direction u). We then show that the pruning procedure removes only a constant fraction of the pairs from these sets $\mathbb{B}_0$ and $\mathbb{B}_1$ with a constant probability. In fact, the size of $\mathbb{B}_0 \cup \mathbb{B}_1$

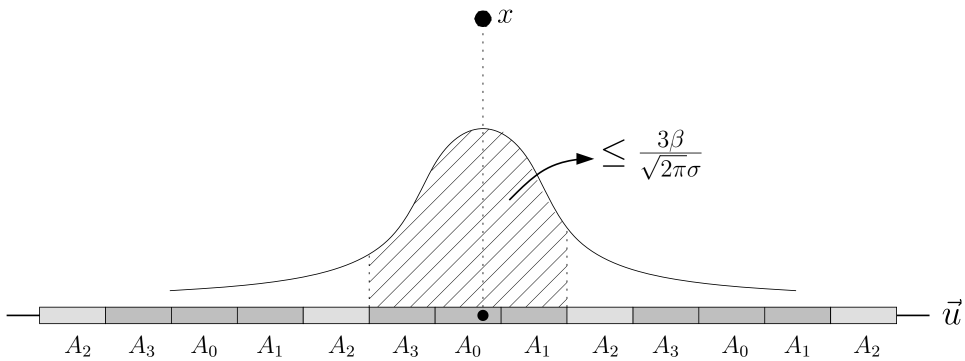


Figure 5.2: The distribution of projected edge lengths in the proof of Lemma 5.9. If y falls into a light-shaded interval, the pair (x, y) is split.

remains at least $|D_i|/32$ even after the pruning, and then it follows that the larger of these sets must have half of the terminal pairs, proving the theorem.

The projection step

Lemma 5.9. *Fix a distance scale i. At the end of the bucketing stage, the set $\mathbb{B}_0 \cup \mathbb{B}_1$ contains at least $\frac{1}{16}|\mathbb{D}_i|$ terminal pairs w.p. $\frac{1}{15}$.*

Proof. Recall that a terminal pair $(x, y) \in \mathbb{D}_i$ is split if x lies in the set A_a and y lies in $A_{(a+2) \bmod 4}$ for some $a \in \{0, 1, 2, 3\}$. Also, we defined $\ell^2 = 2^i$, and hence $(x, y) \in \mathbb{D}_i$ implies that $\|\vec{x} - \vec{y}\|^2 = \mathrm{d}(x, y) \in [\ell^2, 2\,\ell^2)$. Consider the normalized projections p_x and p_y of the vectors $\vec{x}, \vec{y} \in \mathbb{R}^n$ on the random direction u, and note that $p_y - p_x$ is distributed as a Gaussian random variable $Z_u \sim N(0, \sigma^2)$ with a standard deviation $\sigma \in [\ell, \sqrt{2}\ell)$ (see Figure 5.2.)

Now consider the bucket of width β in which p_x lies. The pair (x, y) will not be separated if p_y lies in either the same bucket, or in either of the adjoining buckets. (The probability of each of these three events is at most $\frac{1}{\sqrt{2\pi}\,\sigma} \times \beta$.) Also, at least $\frac{1}{4}$ of the remainder of the distribution causes (x, y) to be split, since each good interval is followed by three bad intervals with less measure.

Putting this together gives us that the probability of (x, y) being split is at least

$$\frac{1}{4}\left(1 - 3\beta\,\frac{1}{\sqrt{2\pi}\,\sigma}\right) \ge \frac{1}{4}\left(1 - \frac{3(\ell/6)}{\sqrt{2\pi}\,\ell}\right) \ge \frac{1}{8}$$

Since each pair $(x, y) \in \mathbb{D}_i$ is separated with probability $1/8$, the linearity of expectations and Markov's inequality implies that at least one-sixteenth of $\mathbb{D}_i$ must be split at the end of the bucketing stage with probability $\frac{1}{15}$. $\square$

The pruning step

We now show that a constant fraction of the terminal pairs in $\mathbb{D}_i$ also survive the pruning phase. This is proved by contradiction, and follows the argument of Arora et al. [17].

Assume that, with a large probability (over the choice of the random direction u), a large fraction of the terminal pairs in $\mathbb{D}_i$ (say $\frac{1}{64}|\mathbb{D}_i|$) get removed in the pruning phase. By the definition of the pruning step, the projection of $\vec{x}-\vec{y}$ on u must have been large for such a removed pair (x, y). In our algorithm, this happens when $\mathrm{d}(x, y) < \ell^2/f$, or equivalently when $\beta > \sqrt{\mathrm{d}(x,y)} \times \sqrt{f}/6$. Since the expected value of $|p_x - p_y|$ is exactly $\sqrt{\mathrm{d}(x,y)}$, while p_x and p_y are separated by at least one bucket of width β, this implies that the expectation is exceeded by a factor of at least $\sqrt{f}/6 = \Omega(\log^{1/4} k)$. Setting $t = \sqrt{f}/6$, we can say that such a pair (x, y) is "stretched by a factor t in the direction u". For any given direction u, the stretched pairs removed in the pruning step are disjoint, and hence form a matching M_u.

Arora et al. showed the following geometric property—for a given set W and some constant C, the number of disjoint t-stretched pairs in $W \times W$ cannot be more than $C|W|$ with constant probability (over the choice of u); however, their proof only proved this for stretch $t = \Omega(\log^{1/3}|W|)$. The dependence on t was improved subsequently by Lee [107] to $t = \Omega(\log^{1/4}|W|)$.

In order to make the above discussion more precise, let us first recall the definition of a stretched set of points.

Definition 5.10 ([17], Defn. 4). A set of n points $\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n$ in $\mathbb{R}^n$ is said to be (t, γ, β)-*stretched at scale* l, if for at least a γ fraction of the directions u, there is a partial matching $M_u = \{(x_i, y_i)\}_i$, with $|M_u| \geq \beta n$, such that for all $(x, y) \in M_u$, $\mathrm{d}(x, y) \leq l^2$ and $\langle u, \vec{x} - \vec{y}\rangle \geq tl/\sqrt{n}$. That is, the pair (x, y) is *stretched by a factor of* t *in direction* u.

Theorem 5.11 ([17], Thm. 5). *For any $\gamma, \beta > 0$, there is a $C = C(\gamma, \beta)$ such that if $t > C\log^{1/3} n$, then no set of n points in $\mathbb{R}^n$ can be (t, γ, β)-stretched for any scale l.*

The above theorem has been subsequently improved by Lee to the following (as implied by [107, Thm. 4.1]).

Theorem 5.12. *For any $\gamma, \beta > 0$, there is a $C = C(\gamma, \beta)$ such that if $t > C\log^{1/4} n$, then no set of n points in $\mathbb{R}^n$ can be (t, γ, β)-stretched for any scale l.*

Summarizing the implication of Theorem 5.12 in our setting, we get the following corollary.

Corollary 5.13. *Let W be a set of vectors corresponding to some subset of terminals satisfying the following property: with probability $\Theta(1)$ over the choice of a random unit vector u, there exist subsets $S_u, T_u \subseteq W$ and a constant ρ such that $|S_u| \geq \rho|W|$ and $|T_u| \geq \rho|W|$, and the length of the projection $|\langle u, \vec{x} - \vec{y}\rangle| \geq \ell/(6\sqrt{n})$ for all $\vec{x} \in S_u$ and $\vec{y} \in T_u$. Then with probability $\Theta(1)$ over the choice of u, the pruning procedure applied to sets S_u and T_u returns sets S'_u and T'_u with $|S'_u| \geq \frac{3}{4}|S_u|$ and $|T'_u| \geq \frac{3}{4}|T_u|$, such that for all $\vec{x} \in S'_u$ and $\vec{y} \in T'_u$, $\mathrm{d}(x, y) \geq \ell^2/f$.*

Proof. For a unit vector u, let $M(u)$ denote the matching obtained by taking the pairs (x, y) of terminals that are deleted by the pruning procedure when given the vector u. Note that pairs $(x, y) \in M(u)$ have the property that $\mathrm{d}(x, y) < \ell^2/f$ and $|p_x - p_y| > \ell/6$. For the sake of contradiction, suppose there is a constant γ such that the matchings $M(u)$ are larger than $\rho/4|W|$ with probability at least $1 - \gamma$ over the choice of u.

Using Definition 5.10 above, we get that the vectors in W form an $(6\sqrt{f}, \gamma, \rho/4)$-stretched set at scale $\ell/\sqrt{f}$. Theorem 5.12 now implies that $6\sqrt{f} = 6\sqrt{c}(\log k)^{1/4}$ must be at most $C \log^{1/4} |W|$. However, since $|W| \leq 2k$, setting the parameter c suitably large compared to C would give us the contradiction. □

Finally, we are in a position to prove Theorem 5.8 using Lemma 5.9 and Corollary 5.13.

Proof of Theorem 5.8. Define W to be $\mathcal{D}_i$, the set of all terminals that belong to some terminal pair in $\mathbb{D}_i$. Let a be the index corresponding to the larger of $\mathbb{B}_0$ and $\mathbb{B}_1$ before the pruning step, and set $S_u = A_a$ and $T_u = A_{(a+2) \bmod 4}$ for this value of a. Lemma 5.9 assures us that $|S_u| = |T_u| \geq \frac{1}{32}|\mathbb{D}_i| = \frac{1}{16}|W|$ with probability $\frac{1}{15}$ (over the random choice of the vector $u \in \mathbb{R}^n$). Furthermore, for each $\vec{x} \in S_u$ and $\vec{y} \in T_u$, the fact that $|p_x - p_y| \geq \beta$ translates to the statement that $\langle \vec{x} - \vec{y}, u \rangle \geq \ell/(6\sqrt{n})$.

These vectors satisfy the conditions of Corollary 5.13, and hence we can infer that with a constant probability, the pruning procedure removes at most $\frac{1}{4}|S_u|$ and $\frac{1}{4}|T_u|$ vertices from S_u and T_u respectively. Their partners may be pruned in the cleanup step as well, and hence the total number of terminal pairs pruned is at most $\frac{1}{2}|S_u|$. Thus the number of terminal pairs remaining in $\mathbb{D}_i \cap (S'_u \times T'_u)$ is at least $\frac{1}{2}|S_u| \geq \frac{1}{64}|\mathbb{D}_i|$ pairs.

Since this happens with a constant probability, we will need to repeat Steps 1-3 of the procedure (each time with a new unit vector u) only a constant number of times until we find a partition that η-separates at least $\frac{1}{64}|\mathbb{D}_i|$ of the terminal pairs; this proves the result. □

Running the procedure Project-&-Prune for each distance scale i between 1 and $\lfloor \log \Delta(\mathrm{d}) \rfloor$, we can get the following result with $\gamma = \frac{1}{64}$.

Theorem 5.14. *Given a negative-type metric* d, *we can find in randomized polynomial time a decomposition suite* $\Pi = \{P_i\}$ *that* η*-separates a constant fraction* γ *of the terminal pairs at each distance scale* i.

In the next section, we will extend this result to get a set of $O(\log k)$ decomposition suites $\{\Pi_j\}$ so that each terminal pair $(x, y) \in \mathbb{D}$ is separated in a constant fraction of the Π_j's.

5.4.4 Obtaining decomposition bundles: weighting and watching

To start off, let us observe that the result in Theorem 5.14 can be generalized to the case where terminal pairs have an associated weight $w_{xy} \in \{0, 1, 2, \ldots, k\}$.

Lemma 5.15. *Given terminal pairs $(x, y) \in \mathbb{D}$ with weights w_{xy}, there is a randomized polynomial time algorithm that outputs a decomposition suite Π which, for each distance scale i, $\Omega(1/\sqrt{\log k})$-separates terminals with total weight at least $\gamma \sum_{(x,y) \in \mathbb{D}_i} w_{xy}$.*

Proof. The proof is almost immediate: we replace each terminal pair $(x, y) \in \mathbb{D}_i$ having weight $w_{xy} > 0$ with w_{xy} new terminal pairs (x^j, y^j), where the points $\{x^j\}$ and $\{y^j\}$ are placed at distance 0 to x and y respectively. Doing this reduction for all weighted pairs gives us an unweighted instance with a set $\mathbb{D}'_i$ of terminal pairs. Now Theorem 5.14 gives us a decomposition suite η-separating at least $\frac{1}{64} |\mathbb{D}'_i|$ of the new terminal pairs at distance scale i, where $\eta = 1/O(\sqrt{\log \mathbb{D}'_i}) = 1/O(\sqrt{\log k})$. Finally, observing that the separated terminal pairs at scale i contribute at least $\frac{1}{64} \sum_{(x,y) \in \mathbb{D}_i} w_{xy}$ completes the claim. □

In the sequel, we will associate weights with the terminal pairs in $\mathbb{D}$ and run the procedure from Lemma 5.15 repeatedly. The weights start off at k, and the weight of a pair that is separated in some iteration is halved in the subsequent iteration; this reweighting ensures that all pairs are separated in significantly many rounds. (Note: this weighting argument is fairly standard and has been used, e.g., in geometric algorithms [45], machine learning [112], and many other areas; see Welzl [141] for a survey.)

The Algorithm:

1. Initialize $w^{(0)}(x, y) = 2^{\lceil \log k \rceil}$ for all terminal pairs $(x, y) \in \mathbb{D}$. Set $j = 0$.
2. Use the algorithm from Lemma 5.15 to obtain a decomposition suite Π_j. Let T_j be the set of terminal pairs η-separated by this decomposition.
3. For all $(x, y) \in T_j$, set $w^{(j+1)}(x, y) \leftarrow w^{(j)}(x, y)/2$; set $w^{(j+1)}(x, y) \leftarrow w^{(j)}(x, y)$ for the others. If $w^{(j+1)}(x, y) < 1$ then $w^{(j+1)}(x, y) \leftarrow 0$.
4. Increment $j \leftarrow j + 1$. If $\sum_{(x,y) \in \mathbb{D}_i} w^{(j)}(x, y) \geq 1$ for some i, go to step 2, else halt.

Note that the distance function d in each iteration of the algorithm remains the same.

Lemma 5.16. *In each iteration j of the above algorithm the following holds*

$$\sum_{(x,y) \in \mathbb{D}_i} w^{(j+1)}(x, y) \leq (1 - \frac{\gamma}{2}) \sum_{(x,y) \in \mathbb{D}_i} w^{(j)}(x, y)$$

Proof. In each iteration, the algorithm of Lemma 5.15 separates at least a γ fraction of the weight $\sum_{(x,y) \in \mathbb{D}_i} w^j(x, y)$, and hence the total weight in the next round drops by at least half this amount. □

Noting that initially we have $\sum_{(x,y) \in \mathbb{D}_i} w^{(0)}(x, y) \leq 2k^2$, one derives the following simple corollary:

Corollary 5.17. *The above algorithm has at most $\frac{4}{\gamma}\log k$ iterations.*

Lemma 5.18. *Every pair $(x, y) \in \mathbb{D}_i$ is η-separated in at least $\log k$ iterations.*

Proof. Since we start off with $w^{(0)}(x, y) \geq k$ and end with $w^{(j)}(x, y) < 1$, the weight $w^{(j)}(x, y)$ must have been decremented at least $\log k$ times. Each such reduction corresponds to a round j in which (x, y) was η-separated by Π_j. □

Theorem 5.19. *The above procedure outputs an η-decomposition bundle with at most $\frac{4}{\gamma}\log k$ decomposition suites, such that each terminal pair (x, y) is η-separated in at least $\log k$ of these suites.*

5.4.5 Embedding via decomposition bundles

In the previous sections we have constructed a decomposition bundle with a large separation between terminal pairs. Now, we show how to obtain a small distortion ℓ_2-embedding from this. The proof mainly follows the lines of the results in Krauthgamer et al. [105].

Theorem 5.20. *Given an $\alpha(x, y)$-decomposition bundle for the metric d and a set $\mathbb{D}$, there exists a randomized contracting embedding $\varphi : V \longrightarrow \ell_2$, such that for each pair $(x, y) \in \mathbb{D}$,*

$$||\varphi(x) - \varphi(y)||_2 \geq \Omega\left(\sqrt{\frac{\alpha(x, y)}{\log k}}\right) \cdot \mathrm{d}(x, y)$$

Note that for $\alpha(x, y) = \Omega(1/\sqrt{\log k})$ this theorem implies Theorem 5.2.

Along the lines of the reasoning in [105], we define a measure of "local expansion". Let

$$V(x, y) = \max\left\{\log\frac{|B(x, 2\mathrm{d}(x, y))|}{|B(x, \mathrm{d}(x, y)/8)|}, \log\frac{|B(y, 2\mathrm{d}(x, y))|}{|B(y, \mathrm{d}(x, y)/8)|}\right\}$$

where $B(x, r)$ denotes the set of *terminal nodes* within the ball of radius r around x. We derive Theorem 5.20 from the following lemma.

Lemma 5.21. *Given an $\alpha(x, y)$-decomposition bundle, there is a randomized contracting embedding $\varphi : V \longrightarrow \ell_2$ such that for every pair (x, y) with constant probability*

$$||\varphi(x) - \varphi(y)||_2 \geq \Omega\left(\sqrt{\frac{V(x, y)}{\log k}} \cdot \alpha(x, y)\right) \cdot \mathrm{d}(x, y) \ .$$

By repeatedly applying Lemma 5.21, we obtain the following guarantee:

Corollary 5.22. *Given an $\alpha(x, y)$-decomposition bundle, there is a randomized contracting embedding $\varphi : V \longrightarrow \ell_2$ such that for every pair (x, y),*

$$||\varphi(x) - \varphi(y)||_2 \geq \Omega\left(\sqrt{\frac{V(x, y)}{\log k}} \cdot \alpha(x, y)\right) \cdot \mathrm{d}(x, y) \ .$$

Proof. The corollary follows by applying Lemma 5.21 repeatedly and independently for each decomposition suite several times. Then concatenating and rescaling the resulting maps gives with high probability an embedding that fulfills the corollary. In passing, we note that this algorithm (using independent repetitions) may result in an embedding with a large number of dimensions, which may not be algorithmically desirable. However, it shows the *existence* of such an embedding, and we can then use semidefinite programming followed by random projections to obtain a nearly-optimal embedding of the metric into ℓ_2 with $O(\log n)$ dimensions in randomized polynomial time. □

To see that the above corollary implies Theorem 5.20, we use a decomposition due to Fakcharoenphol et al. [61] (and its extension to general measures, as observed by Lee and Naor [108] and by Krauthgamer et al. [105]) that has the property that with probability at least $1/2$, a pair (x, y) is $\Omega(1/V(x, y))$-separated in this decomposition. Applying the corollary to this decomposition bundle, we get an embedding φ_1, such that

$$||\varphi_1(x) - \varphi_1(y)||_2 \geq \Omega\left(\frac{1}{\sqrt{V(x, y) \cdot \log k}}\right) \cdot \mathrm{d}(x, y) \ .$$

Applying the corollary to the decomposition bundle assumed by the theorem gives an embedding φ_2 with

$$||\varphi_2(x) - \varphi_2(y)||_2 \geq \Omega\left(\sqrt{\frac{V(x, y)}{\log k}} \cdot \alpha(x, y)\right) \cdot \mathrm{d}(x, y) \ .$$

Concatenating the two mappings and rescaling, we get a contracting embedding $\varphi = \frac{1}{\sqrt{2}}(\varphi_1 \otimes \varphi_2)$, with

$$\begin{aligned} &||\varphi(x) - \varphi(y)||_2 \\ &\geq \Omega\left(\frac{1}{\sqrt{\log k}} \cdot (\frac{1}{V(x, y)^{\frac{1}{2}}} + V(x, y)^{\frac{1}{2}} \ \alpha(x, y))\right) \cdot \mathrm{d}(x, y) \\ &\geq \Omega\left(\sqrt{\frac{\alpha(x, y)}{\log k}}\right) \cdot \mathrm{d}(x, y) \end{aligned}$$

as desired. Now it remains to prove Lemma 5.21.

The embedding

Let $T = \{1, \dots, \log k\}$ and $Q = \{0, \dots, h - 1\}$, for some suitably chosen constant h. In the following we define an embedding into $|T| \cdot |Q|$ dimensions. For $t \in T$, let $\mathrm{r}_t(x)$ denote the minimum radius r such that the ball $B(x, r)$ contains at least 2^t terminal nodes. We call $\mathrm{r}_t(x)$ the *t-radius of x*. Further, let $\ell_t(x) \in \mathbb{N}$ denote the distance class this radius belongs to (i.e., $2^{\ell_t(x)-1} \leq \mathrm{r}_t(x) \leq 2^{\ell_t(x)}$).

Pick a decomposition suite $\Pi = \{P_s\}$ from the decomposition bundle at random. In the following $\delta(x, y)$ denotes the separation-factor between x and y in this suite, i.e., $\delta(x, y) = \frac{1}{\mathrm{d}(x,y)} \min\{\mathrm{d}(x, V\setminus$

$C_s(x)), \mathrm{d}(y, V \setminus C_s(y))\}$ if $C_s(y) \neq C_s(x)$ and 0, otherwise. Observe that with constant probability we have $\delta(x, y) \geq \alpha(x, y)$.

The standard way to obtain an embedding from a decomposition suite is to create a coordinate for every distance scale and embed points in this coordinate with respect to the partitioning for this scale. For example, one could assign a random color, 0 or 1, to each cluster $C \in P_i$. Let W_i denote the set of nodes contained in clusters with color 0 in partitioning P_i. By setting the i-th coordinate of the image $\varphi(x)$ of a point x to $\mathrm{d}(x, W_0^i)$, a pair (x, y) gets a distance $\Omega(\delta(x, y)\mathrm{d}(x, y))$ with probability $1/2$, because this is the probability that the clusters $C_i(x)$ and $C_i(y)$ get different colors (in this case the distance is $\Omega(\delta(x, y)\mathrm{d}(x, y))$ since both nodes are at least that far away from the boundary of their cluster). Overall this approach gives an embedding into ℓ_2 with contraction $O(\sqrt{\log k}/\delta(x, y))$, and has e.g. been used by Rao [122] for getting a $\sqrt{\log n}$ embedding of planar metrics into ℓ_2.

In order to improve this, along the lines of the measured descent idea from [105], the goal is to construct an embedding in which the distance between $(\varphi(x), \varphi(y))$ increases as the local expansion $V(x, y)$ increases. This can be achieved by constructing a coordinate for every $t \in T$ and then embed points in this coordinate according to the partitioning for the corresponding distance scale $\ell_t(x)$ (i.e., different points use different distance scales depending on their local expansion). Thereby, for a pair with a high $V(x, y)$-value the nodes will often ($\approx V(x, y)$ times) be embedded according to the partitioning for distance scale $i = \lfloor \log \mathrm{d}(x, y) \rfloor$ that corresponds to $\mathrm{d}(x, y)$. Therefore, the pair (x, y) gets a larger distance (by a factor of roughly $\sqrt{V(x, y)}$) in this embedding than in the standard approach.

However, transferring the rest of the standard analysis to this new idea has some difficulties. If we define the set W_t as the nodes x that are colored 0 in the partitioning for scale $\ell_t(x)$ we cannot argue that for a pair (x, y) either $\mathrm{d}(x, W_t)$ or $\mathrm{d}(y, W_t)$ is large, because nodes u very close to x or y may have distance scales $\ell_t(u)$ that are different from $\ell_t(x)$ or $\ell_t(y)$. In order to ensure local consistency such that all nodes close to x obtain their color from the same partitioning, we construct several coordinates in the embedding for every t, such that for each distance scale $\ell_t(x)$ there is a coordinate in which all nodes close to x derive their color from the partitioning for scale $\ell_t(x)$. The details are as follows.

Let $Q = \{0, \cdots, h-1\}$ denote the set of indices of coordinates corresponding to each value of t. For each $q \in Q$, we partition the distance scales into *groups* g_q of size h each, and let the median scale in each group represent that group for the coordinate corresponding to q. In the $(q, t)^{\text{th}}$ coordinate, the color of a node is picked according to the median distance scale in the group g_q to which $\ell_t(x)$ belongs.

In particular, let $g_q(\ell) := \lceil \frac{\ell - q}{h} \rceil$. Note that each distance group contains (at most) h consecutive distance classes which means that distances within a group differ at most by a constant factor – all distances in group g are in $\Theta(2^{h \cdot g})$. We define a mapping π_q between distance classes that maps all classes of a group to the median distance class in this group (the value of π_q for the first and last distance group is rounded off appropriately; we omit a precise definition for the sake of clarity).

$$\pi_q(\ell) := i + h \cdot g_q(\ell) - \lfloor \frac{h}{2} \rfloor$$

Observe that this partitioning satisfies the key property that for each distance class i, there exists a q such that $\pi_q(i) = i$.

Based on this mapping we define a set W_t^q for each choice of $t \in T$ and $q \in Q$ by $W_t^q = \{x \in V : \mathrm{color}_{\pi_q(\ell_t(x))}(x) = 0\}$, where $\mathrm{color}_i(x)$ denotes the color of the cluster that contains x in partitioning P_i. Note that all nodes whose t-radii fall into the same distance group (w.r.t. parameter q) derive their color (and hence whether they belong to W_t^q) from the same partitioning.

Based on the sets W_t^q we define an embedding $\varphi_{t,q} : V \longrightarrow \mathbb{R}$ for each coordinate (t,q) — $\varphi_{t,q}(x) = \mathrm{d}(x, W_t^q)$. The embedding $\varphi : V \longrightarrow \mathbb{R}^{|T||Q|}$ is defined by

$$\varphi(x) := \otimes_{t,q}\, \varphi_{t,q}(x). \tag{5.5}$$

Next we analyse the distortion of the map φ.

The analysis

Since each coordinate of φ maps point x to its distance from some subset of points, it follows that each coordinate of this embedding is contracting. Therefore, we have for all $x, y \in V$

$$\begin{aligned} ||\varphi(x) - \varphi(y)||_2 &\leq \sqrt{|T| \cdot |Q| \cdot \mathrm{d}(x,y)^2} \\ &\leq O(\sqrt{\log k}) \cdot \mathrm{d}(x,y) \end{aligned}$$

Now, we show that for a pair x, y that is $\delta(x,y)$-separated in the partitioning corresponding to its distance scale $\lfloor \log(\mathrm{d}(x,y)) \rfloor$, with a constant probability, we get

$$||\varphi(x) - \varphi(y)||_2 \geq \Omega(\delta(x,y) \cdot \mathrm{d}(x,y)) \cdot \sqrt{V(x,y)} \tag{5.6}$$

This gives Lemma 5.21 since $\delta(x,y) > \alpha(x,y)$ with constant probability.

Fix a pair (x,y) that is $\delta(x,y)$-separated in the partitioning for distance scale $\lfloor \log(\mathrm{d}(x,y)) \rfloor$. Without loss of generality assume that the maximum in the definition of $V(x,y)$ is attained by the first term, i.e. $\frac{|B(x,2\mathrm{d}(x,y))|}{|B(x,\mathrm{d}(x,y)/8)|} \geq \frac{|B(y,2\mathrm{d}(x,y))|}{|B(y,\mathrm{d}(x,y)/8)|}$. We show that for any t with $|B(x, \mathrm{d}(x,y)/8)| \leq 2^t \leq |B(x, 2\mathrm{d}(x,y))|$, there is a $q \in Q$ such that the coordinate (t,q) gives a large contribution, i.e., $|\varphi_{t,q}(x) - \varphi_{t,q}(y)| \geq \Omega(\delta(x,y) \cdot \mathrm{d}(x,y))$. Equation 5.6 then follows.

We fix an integer t with $\log(|B(x, \mathrm{d}(x,y)/8)|) \leq t \leq \log(|B(x, 2\mathrm{d}(x,y))|)$, and we use $i = \lfloor \log \mathrm{d}(x,y) \rfloor$ to denote the distance class of $\mathrm{d}(x,y)$. Clearly, the distance class $\ell_t(x)$ of the t-radius of x is in $\{i-4, \ldots, i+2\}$, because $\mathrm{d}(x,y)/8 \leq \mathrm{r}_t(x) \leq 2\mathrm{d}(x,y)$. The following claim gives a similar bound on the t-radius for nodes that are close to x.

Claim 5.23. *Let* $z \in B(x, \frac{1}{16}\mathrm{d}(x,y))$. *Then* $\ell_t(z) \in \{i-5, i+3\}$.

Proof. For the t-radius $\mathrm{r}_t(z)$ around z we have $\mathrm{r}_t(x) - \mathrm{d}(x,y)/16 \leq \mathrm{r}_t(z) \leq \mathrm{r}_t(x) + \mathrm{d}(x,y)/16$. Since $\mathrm{d}(x,y)/8 \leq \mathrm{r}_t(x) \leq 2\mathrm{d}(x,y)$ we get $\frac{1}{16}\mathrm{d}(x,y) \leq \mathrm{r}_t(z) \leq \frac{33}{16}\mathrm{d}(x,y)$, which yields the claim. □

In the following we choose h (the number of distances classes within a group) as 10, and q such that $\pi_q(i) = i$, i.e., i is the median of its distance group. Then the above claim ensures that for all nodes $z \in B(x, \frac{1}{16}\mathrm{d}(x,y))$, the distance class $\ell_t(z)$ is in the same distance group as i. Furthermore, these nodes choose their color (that decides whether they belong to W_t^q) according to the partitioning for distance scale i. Recall that x is $\delta(x,y)$-separated in this partitioning. Therefore, we can make the following claim.

Claim 5.24. *If x does not belong to the set W_t^q, then,*

$$\mathrm{d}(x, W_t^q) \geq \min\{\tfrac{1}{16}, \delta(x,y)\}\, \mathrm{d}(x,y) \geq \frac{1}{16}\,\delta(x,y)\,\mathrm{d}(x,y)$$

Now, we consider the following events concerning the distances of x and y from W_t^q, respectively.

- $X_0 \quad = \{\mathrm{d}(x, W_t^q) = 0\}$, i.e., $x \in W_t^q$
- $X_{\text{far}} \quad = \{\mathrm{d}(x, W_t^q) > \frac{1}{16}\delta(x,y)\mathrm{d}(x,y)\}$
- $Y_{\text{close}} = \{\mathrm{d}(y, W_t^q) \leq \frac{1}{32}\delta(x,y)\mathrm{d}(x,y)\}$
- $Y_{\text{far}} \quad = \{\mathrm{d}(y, W_t^q) > \frac{1}{32}\delta(x,y)\mathrm{d}(x,y)\}$

These events only depend on the random colorings chosen for the partitionings in different distance classes. First we claim that the events X_0 and X_{far} are independent of events Y_{close} and Y_{far}. To see this, note that X_0 and X_{far} only depend on colors chosen for nodes in $B(x, \frac{1}{16}\delta(x,y)\mathrm{d}(x,y))$. Our choice of q ensures that these colors are derived from the partitioning for distance class i, and Claim 5.23 implies that all nodes in $B(x, \frac{1}{16}\delta(x,y)\mathrm{d}(x,y))$ get the color assigned to the cluster $C_i(x)$.

The events Y_{close} and Y_{far}, however, depend on colors chosen for nodes that lie in the ball $B(y, \delta(x,y)\,\frac{1}{32}\mathrm{d}(x,y))$. Such a color is either derived from a partitioning for a distance class different from i (in this case independence is immediate), or it is equal to the color assigned to the cluster $C_i(y)$, using the fact that $\mathrm{d}(y, V \setminus C_i(y)) \geq \delta(x,y)\mathrm{d}(x,y)$. In the latter case the independence follows, since x and y lie in different clusters in this partitioning as they are separated by it.

If $X_0 \cap Y_{\text{far}}$ or $X_{\text{far}} \cap Y_{\text{close}}$ happens, then the dimension (t,q) gives a contribution of $\Omega(\delta(x,y)\mathrm{d}(x,y))$. This happens with probability

$$\begin{aligned}
&\mathbf{Pr}[X_0 \cap Y_{\text{far}} \uplus X_{\text{far}} \cap Y_{\text{close}}] \\
&\qquad = \mathbf{Pr}[X_0 \cap Y_{\text{far}}] + \mathbf{Pr}[X_{\text{far}} \cap Y_{\text{close}}] \\
&\qquad = \mathbf{Pr}[X_0] \cdot \mathbf{Pr}[Y_{\text{far}}] + \mathbf{Pr}[X_{\text{far}}] \cdot \mathbf{Pr}[Y_{\text{close}}] \\
&\qquad = \mathbf{Pr}[X_0] \cdot \mathbf{Pr}[Y_{\text{far}}] + \mathbf{Pr}[X_{\text{far}}] \cdot (1 - \mathbf{Pr}[Y_{\text{far}}]) \\
&\qquad = 1/2\ .
\end{aligned}$$

Here we used the fact that $\mathbf{Pr}[X_0] = \mathbf{Pr}[X_{\mathrm{far}}] = 1/2$ which holds due to Claim 5.24. This completes the proof of Lemma 5.21.

5.5 Hardness of approximating SPARSEST-CUT and MULTICUT

5.5.1 Preliminaries

Regular Unique Games

A unique 2-prover game is called *regular* if the total weight of question edges incident at any single vertex is the same, i.e., $1/n$, for every vertex in Q. We now show that we can assume without loss of generality that the graph in the Unique Games Conjecture is regular. For simplicity, we state this only for fixed η and δ. A similar result holds when they depend on n, because we increase the input size by no more than a polynomial factor, and increase η and δ by no more than a constant factor.

Lemma 5.25. *The Unique Games Conjecture implies that for every fixed $\eta, \delta > 0$, there exists $m = m(\eta, \delta)$ such that it is NP-hard to decide if a regular unique 2-prover game has value at least $(1 - \eta)$ or at most δ.*

The proof is based on an argument of Khot and Regev [100, Lemma 3.3].

Proof. Given a unique 2-prover game Q, we describe how to convert it to a regular game while preserving its completeness and soundness. First we claim that we can assume that the ratio between the max weight $\max_e w_e$ and the min weight $\min_e w_e$ is bounded by n^3. This is because we can remove all edges with weight less than $\frac{1}{n^3} \max_e w_e$ from the graph, changing the soundness and completeness parameters by at most $\frac{1}{n}$. By a similar argument, we can assume that all weights in the graph are integral multiples of $t = \frac{1}{n^2} \min_e w_e$.

Now we convert Q to a regular graph Q' as follows. For each prover $p \in \{1, 2\}$ and question q_i^p, form $W(p,i)/t$ vertices $q_i^p(1), \cdots, q_i^p(W(p,i)/t)$, where $W(p,i)$ is the total weight of all the edges incident on q_i^p. For every pair of vertices (q_i^1, q_j^2), connected by an edge e in Q, we form an edge between $q_i^1(x)$ and $q_j^2(y)$, for all possible values of x and y, with weight $w_e \frac{t}{W(1,i)} \frac{t}{W(2,j)}$.

Note that the total weight of all the edges remains the same as before. Each new node $q_i^1(x)$ has total weight $\sum_e w_e \frac{t}{W(1,i)} \frac{t}{W(2,j)} \frac{W(2,j)}{t} = t$, where the sum is over all edges e incident on q_i^1. Therefore, the graph is regular. Furthermore, the number of vertices increases by a factor of at most n^5.

It only remains to show that the soundness and completeness parameters are preserved. To see this, note that any solution on the original graph Q can be transformed to a solution of the same value on Q', by picking the same answer for every node $q_i^p(x)$ in Q' as the answer picked for q_i^p in Q. Likewise, consider a solution in Q'. Note that the answers for the questions $q_i^p(x)$ with different values of x must all be the same, because all these questions are connected to identical sets of vertices, with the same weights. Therefore, the solution in Q that picks the same answer for q_i^p as the answer for $q_i^p(x)$ in Q' has the same weight as the given solution in Q'.

Thus for every solution in Q, there is a solution of the same weight in Q' and vice versa. This proves that the two games have exactly the same soundness and completeness parameters. □

Bicriteria MULTICUT

Our proof for the hardness of approximating SPARSEST-CUT relies on a generalization of MULTICUT, where the solution M is required to cut only a certain fraction of the demand pairs. For a given graph $G = (V, E)$, a subset of the edges $M \subseteq E$ will be called throughout a *cutset* of the graph. A cutset whose removal disconnects all the demand pairs is called a *multicut*.

An algorithm is called an (α, β)-bicriteria approximation for MULTICUT if, for every input instance, the algorithm outputs a cutset M that disconnects at least an α fraction of the demands and has cost at most β times the weight of the optimum multicut. In other words, if M^* is the least cost cutset that disconnects all the k demand pairs, then M disconnects at least αk demand pairs and $c(M) \leq \beta \cdot c(M^*)$.

Hypercubes, dimension cuts, and antipodal vertices

As usual, the *m-dimensional hypercube* (in short an *m-cube*) is the graph $C = (V_C, E_C)$ which has the vertex set $V_C = \{0, 1\}^m$, and an edge $(u, v) \in E_C$ for every two vertices $u, v \in \{0, 1\}^m$ which differ in exactly one dimension (coordinate). An edge (u, v) is called a *dimension-a edge*, for $a \in [m]$, if u and v differ in dimension a, i.e., $u \oplus v = 1_a$ where 1_a is a unit vector along dimension a. The set of all the dimension-a edges in a hypercube is called the *dimension-a cut* in the hypercube. The *antipode* of a vertex u is the (unique) vertex $\overline{u}$ all of whose coordinates are different from those of u, i.e., the vertex $u \oplus \underline{1}$ where $\underline{1}$ is the vector with 1 in every coordinate. Notice that v is the antipode of u if and only if u is the antipode of v; thus, $\langle u, \overline{u} \rangle$ form an *antipodal pair*. The following simple fact will be key in our proof.

Fact 5.26. *In every hypercube, a single dimension cut disconnects every antipodal pair.*

5.5.2 **Hardness of bicriteria approximation for** MULTICUT

In this section we prove the part of Theorem 5.5 regarding the MULTICUT problem, namely, that the Unique Games Conjecture implies that it is NP-hard to approximate MULTICUT within a certain factor L. Our proof will actually show a stronger result—for every $\alpha \geq 7/8$ it is NP-hard to distinguish between whether there is a multicut of cost less than $n2^{m+1}$ (the YES instance) or whether every cutset that disconnects at least αk demand pairs has cost at least $n2^{m+1}L$ (the NO instance). This implies that it is NP-hard to obtain an (α, L)-bicriteria approximation for MULTICUT.

We start by describing a reduction from unique 2-prover game to MULTICUT (Section 5.5.2), and then proceed to analyze the YES instance (Section 22) and the NO instance (Sections 22 and 22). Finally, we discuss the gap that is created for a bicriteria approximation of MULTICUT (Section 22).

The reduction

Given a unique 2-prover game instance $G_Q = (Q, E_Q)$ with $n = |Q|/2$ and the corresponding edge weights $w(e)$ and bijections $b_{ij} : [m] \to [m]$, we construct a MULTICUT instance $G = (V, E)$ with demand pairs, as follows. For every vertex (i.e., question) $q_i^p \in Q$, construct a m-dimensional hypercube C_j^p. The dimensions in this cube correspond to answers for the question q_j^p.[3] We let the edges insides these $2n$ cubes have cost 1, and call them *hypercube edges*.

For each question edge $(q_i^1, q_j^2) \in E_Q$, we extend b_{ij} to a bijection from the vertices of C_i^1 (subsets of the answers for q_i^1) to the vertices of C_j^2 (subsets of the answers for q_j^2), and denote the resulting bijection by $b'_{ij} : \{0,1\}^m \to \{0,1\}^m$. Formally, for every $u \in \{0,1\}^m$ (vertex in C_i^1) and every $a \in [m]$, the a-th coordinate of $b'_{ij}(u)$ is given by $(b'_{ij}(u))_a = u_{b_{ij}^{-1}(a)}$. Then, we connect every vertex $v \in C_i^1$ to the corresponding vertex $b'_{ij}(u) \in C_j^2$ using an edge of cost $w_{ij}\Lambda$, where $\Lambda = \frac{n}{\eta}$ is a scaling factor. These edges are called *cross edges*.

Denote the resulting graph by $G = (V, E)$. Notice that V is simply the union of the vertex sets of the hypercubes C_i^p, for all $p \in [2]$ and $i \in [n]$, and that the edge set E contains two types of edges, hypercube edges and cross edges.

To complete the reduction, it remains to define the demand pairs. For a vertex $u \in V$, the *antipode* of u in G, denoted $\overline{u}$, is defined to be the antipodal vertex of u in the hypercube C_i^p that contains u. The set $\mathbb{D}$ of demand pairs then contains every pair of antipodal vertices in G, and hence $k = |\mathbb{D}| = n2^{m-1}$. Note that every vertex of G belong to exactly one demand pair.

The YES instance

Lemma 5.27. *If there is a solution A for the unique 2-prover game G_Q such that the total weight of the satisfied questions is at least $1 - \eta$, then there exists a multicut $M \subseteq E$ for the* MULTICUT *instance G such that $c(M) \le 2^{m+1}n$.*

Proof. Let A be such a solution for G_Q. Construct M by taking the following edges. For every question $q_i^p \in Q$ and the corresponding answer A_i^p (of prover p), take the dimension-A_i^p cut in cube C_i^p. In addition, for every edge $(q_i^1, q_j^2) \in E_Q$ that the solution A does not satisfy, take all the cross edges between the corresponding cubes C_i^1 and C_j^2.

[3] This is a standard technique in PCP constructions for graph optimization problems. A hypercube can be interpreted as a "long code" [23], and a dimension cut is the encoding of an answer in the 2-prover game.

We first claim that removing M from G disconnects all the demand pairs. To see this, we define a Boolean function $f : V \to \{0, 1\}$ on the graph vertices. For every cube C_i^p, consider the dimension-A_i^p cut; it disconnects the cube into two connected components, one containing the all zeros vector $\underline{0}$ and one containing the all ones vector $\underline{1}$. For every $v \in C_i^p$, let $f(v) = 0$ if v is in the same side as $\underline{0}$, and $f(v) = 1$ otherwise. This is exactly the A_i^p-th bit in v, i.e., $f(v) = v_{A_i^p}$. Now consider any demand pair $(v, \overline{v})$, and note that $f(v) = 1 - f(\overline{v})$. We will show below that every edge $(u, v) \notin M$ satisfies the property $f(u) = f(v)$. This clearly proves the claim.

Consider first a hypercube edge (u, v) in C_i^p that is not a dimension-A_i^p edge. Then $f(u) = u_{A_i^p} = v_{A_i^p} = f(v)$, by the definition of f. Next consider a cross edge $(u, v) \notin M$. Then this edge lies between cubes C_i^1 and C_j^2, such that the question edge (q_i^1, q_j^2) satisfied by the unique 2-prover game solution A. Therefore, $b_{ij}(A_i^1) = A_j^2$. Then, $f(u) = u_{A_i^1} = v_{b_{ij}(A_i^1)} = v_{A_j^2} = f(v)$.

Finally, we bound the cost of the solution. Let S be the set of question edges not satisfied by the solution A. The total cost of the multicut solution is thus $c(M) = 2n\, 2^{m-1} + 2^m \Lambda \sum_{(Q_i^1, Q_j^2) \in S} w_{ij} \leq 2^m n + 2^m \frac{n}{\eta} \eta = 2^{m+1} n$. □

Hypercube cuts, Boolean functions, influences and juntas

We will analyze the NO instance shortly, but first we set up some notation and present a few technical lemmas regarding cuts in hypercubes. In particular, we present Theorem 5.28, which will have a crucial role in the sequel.

Recall that the dimensions of the hypercubes in the multicut instance corresponds to answers to the 2-prover game. Therefore, we would like to determine which dimensions are the most significant participants in a cut on the cube, as follows. Let $C = (V_C, E_C)$ be an m-dimensional hypercube. It will be useful to represent cuts on the hypercube C as functions $f : V_C \to \mathbb{Z}$. Such a function f corresponds to a partition of V_C into sets $\{f^{-1}(r) \ : \ r \in f(V_C)\}$, which in turn corresponds to the cutset $\{(u, v) \in E_C : f(u) \neq f(v)\}$. Notice that f can be described as a function on m Boolean variables (corresponding to the dimensions of the hypercube), where the dimension $a \in [m]$ corresponds to the a-th variable. The *influence of a dimension (variable)* $a \in [m]$ on the function f, denoted I_a^f, is defined as the fraction of the dimension a-edges $(u, v) \in E_C$ for which $f(u) \neq f(v)$. In other words, $I_a^f = \mathbf{Pr}_{u \in V_C}[f(u) \neq f(u \oplus 1_a)]$ where 1_a is a unit vector along dimension a. The *total influence* (also called average sensitivity) of f is $\sum_{a \in [m]} I_a^f$. We say that the function f is a *k-junta* if there exists a subset $J \subseteq [m]$, $|J| \leq k$, such that for every variable (dimension) $a \notin J$ and for every $u \in V_C$, we have $f(u) = f(u \oplus 1_a)$. In other words, f depends on at most k variables, and the remaining variables have zero influence. Two functions f and f' are said to be ε-close if $\mathbf{Pr}_{u \in V_C}[f(u) \neq f'(u)] \leq \varepsilon$.

An important special case is that of Boolean functions, i.e., $g : V_C \to \{0, 1\}$, which corresponds to a bipartition of V_C. The *balance* of a Boolean function g is defined as $\min\{\frac{|g^{-1}(0)|}{|V_C|}, \frac{|g^{-1}(1)|}{|V_C|}\}$, i.e., the minimum between $\mathbf{Pr}_{u \in V_C}[g(u) = 0]$ and $\mathbf{Pr}_{u \in V_C}[g(u) = 1]$.

The next theorem, due to Friedgut [66], asserts that every function of low total influence is close to a junta. We will later use it to determine a set of dimensions that are the most significant participants in a low-cost cutset.

Theorem 5.28 (Friedgut's Junta Theorem). *Let g be a Boolean function defined on a hypercube and fix $\varepsilon > 0$. Then g is ε-close to a Boolean function h defined on the same cube and depending on only $2^{O(T/\varepsilon)}$ variables, where $T = \sum_{a\in[m]} I^g_a$ is the total influence of g.*

The NO instance

Lemma 5.29. *There exists $L = \Omega(\min\{\eta^{-1}, \log \delta^{-1}\})$ such that if the* MULTICUT *instance G has a cutset of cost at most $2^m nL$ whose removal disconnects $\alpha \geq 7/8$ fraction of the demand pairs, then there is a solution A for the unique 2-prover game G_Q whose value is larger than δ.*

Proof. Let $L = \min\{c/\eta, c\log(1/\delta)\}$ for a constant $c > 0$ to be determined later, and let $M \subseteq E$ be a cutset of cost $c(M) \leq 2^m nL$ whose removal disconnects $\alpha \geq 7/8$ fraction of the demand pairs. Using M, we will construct for the unique 2-prover game G_Q a randomized solution A whose expected value is larger than δ, thereby proving the existence of a solution of value larger than δ. The randomized solution A (i.e., a strategy for the two provers) is defined as follows. Label each connected component of $G \setminus M$ with either 0 or 1 independently at random with equal probabilities, and define a Boolean function $f : V \to \{0, 1\}$ by letting $f(v)$ be the label of the connected component of $v \in V$. Next, fix $\varepsilon = 1/64$ and for each vertex (question) $q^p_i \in Q$, consider the restriction of f to the hypercube $C^p_i \subset V$, denoted $f_{|C^p_i}$, and apply to it Theorem 5.28 (Friedgut's Junta Theorem) to obtain a subset of the variables (dimensions) $J^p_i \subseteq [m]$ with $|J^p_i| \leq 2^{O(L/\varepsilon)}$. Finally, choose the answer A^p_i uniformly at random from J^p_i, independently of all other events.

We proceed to analyze the expected value of this randomized solution A. Recall that the value of a solution is equal to the probability that, for a question edge (q^1_i, q^2_j) chosen at random with probability proportional to its weight, we have $a^2_j = b_{ij}(a^1_i)$. Notice that although q^1_i and q^2_j are correlated, each one is uniformly distributed because Q is regular. Without loss of generality, we assume removing M disconnects at least as many demand pairs inside the cubes $\{C^1_l\}_{l\in[n]}$ as inside the cubes $\{C^2_l\}_{l\in[n]}$. Now we claim that with a constant probability over the choice of a question edge, the cut M has a low cost over edges incident on the corresponding hypercubes, and disconnects many demand pairs in the hypercubes. In other words, the quality of the cut locally is nearly as good as the quality of the cut globally. In particular, we upper bound the probability of the following four "bad" events (for a random question edge (q^1_i, q^2_j)):

$\mathbf{E}_1$ = fewer than $1/8$-fraction of the vertices $v \in C^1_i$ satisfy $f(v) \neq f(\overline{v})$.

$\mathbf{E}_2$ = M contains more than $2^{m+4}L$ hypercube edges in C^1_i.

$\mathbf{E}_3$ = M contains more than $2^{m+4}L$ hypercube edges in C^2_j.

$\mathbf{E}_4$ = M contains more than $2^{m+4}\eta L$ cross edges between C_i^1 and C_j^2.

First, by our assumption above, removing M disconnects at least $\alpha \geq 7/8$ fraction of the demand pairs inside the cubes $\{C_l^1\}_{l\in[n]}$. Thus, the expected fraction of demand pairs $(v, \overline{v})$ in C_i^1 that are not disconnected in $G \setminus M$ (and thus clearly $f(v) = f(\overline{v})$) is at most $1/8$. In addition, the expected fraction of demand pairs $(v, \overline{v})$ in C_i^1 that are disconnected in $G \setminus M$ and satisfy $f(v) = f(\overline{v})$, is at most $1/2$, because different connected components of $G \setminus M$ are labeled independently. Thus, the expected fraction of vertices $v \in C_i^1$ for which $f(v) = f(\overline{v})$ is at most $5/8$, and by Markov's inequality, $\mathbf{Pr}[\mathbf{E}_1] \leq 5/7$. Next, the cutset M contains at most $2^m nL$ hypercube edges, thus the expected number of edges in $C_i^1 \cup C_j^2$ that are contained in M is at most $2^m L$, and by Markov's inequality $\mathbf{Pr}[\mathbf{E}_2 \cup \mathbf{E}_3] \leq 1/16$. Finally, $\mathbf{Pr}[\mathcal{E}_4] \leq 1/16$, as otherwise the total cost of M along the cross edges corresponding to this event is more than $1/16 \cdot (2^{m+4}\eta L) \cdot \Lambda = 2^m nL \geq c(M)$. Taking a union bound, we upper bound the probability that any of the bad events occurs by

$$\mathbf{Pr}[\mathbf{E}_1 \cup \mathbf{E}_2 \cup \mathbf{E}_3 \cup \mathbf{E}_4] \leq \frac{5}{7} + \frac{2}{16} < \frac{6}{7}.$$

In order to lower bound the expected value of the randomized solution A, we would like to show that if none of the above bad events happens, then the two sets of dimensions J_i^1 and J_j^2 obtained using Friedgut's Junta Theorem are relatively small, and further they are in "weak agreement", and these two properties will immediately imply that the randomized solution A satisfies $\mathbf{Pr}[b_{ij}(A_i^1) = A_j^2] > \delta$. Observe that if $(u, v) \in E$ and $f(u) \neq f(v)$, then $(u, v) \in M$. If the event $\mathbf{E}_2$ does not occur, then the total influence of $f_{|C_i^1}$ is at most $8L$, and thus $|J_i^1| \leq 2^{O(L/\varepsilon)}$. Similarly, if the event $\mathbf{E}_3$ does not occur, then the total influence of $f_{|C_j^2}$ is at most $8L$, and thus $|J_j^2| \leq 2^{O(L/\varepsilon)}$. In addition, if the event $\mathbf{E}_1$ does not occur, then the balance of $f_{|C_i^1}$ is at least $1/16$.

We now claim that if none of the above bad events happens then there is $a \in J_i^1$ for which $b_{ij}(a) \in J_j^2$. Indeed, assume towards contradiction that $J_i^1 \cap b_{ij}^{-1}(J_j^2) = \emptyset$. Then by construction there is a Boolean function $g_i^1 : C_i^1 \to \{0, 1\}$ that is ε-close to $f_{|C_i^1}$ and depends on only variables in J_i^1. Note that the balance of g_i^1 is at least $1/16 - \varepsilon$. Similarly, there is a Boolean function $g_j^2 : C_j^2 \to \{0, 1\}$ that is ε-close to $f_{|C_j^2}$ and depends on only variables in J_j^2. We can relate these two functions via $b'_{ij} : C_1^i \to C_j^2$, namely by considering $h : C_i^1 \to \{0, 1\}$ given by $h(v) = g_j^2(b'_{ij}(v))$.

Notice that h is ε-close to $f_{|C_j^2} \circ b'_{ij}$, and that it depends only on variables in $b_{ij}^{-1}(J_j^2)$. Therefore g_i^1 and h depend on disjoint sets of variables. It follows that $\mathbf{Pr}_{v\in C_i^1}[g_i^1(v) \neq h(v)] \geq 1/16 - \varepsilon$, because if we condition on the value of the variables in $b_{ij}^{-1}(J_j^2)$ we get that $h(v)$ is determined, but this does not affect the distribution of $g_i^1(v)$, which still attains each value (0 or 1) with probability at least $1/16 - \varepsilon$. Consequently, g_i^1 and $h = g_j^2 \circ b_{ij}$ are not $(1/16 - \varepsilon)$-close.

On the other hand, the event $\mathbf{E}_4$ not occuring implies that at most $16\eta L$ vertices $v \in C_i^1$ satisfy $f(v) \neq f(b_{ij}(v))$. In other words, $f_{|C_i^1}$ is $(16\eta L)$-close to $f_{|C_j^2} \circ b_{ij}$. The former is ε-close to g_i^1 while the latter is ε-close to $g_j^2 \circ b_{ij}$ (because b_{ij} is a bijection on the variables), and by the triangle inequality we get that g_i^1 and $h = g_j^2 \circ b_{ij}$ are $2\varepsilon + 16\eta L$ close. If $c > 0$ is sufficiently small, $2\varepsilon + 16\eta L < 1/16 - \varepsilon$, which yields a contradiction.

Using the above claim we get that for a random question edge,

$$\begin{aligned}
\mathbf{Pr}[A_j^2 = b_{ij}(A_i^1)] &\geq \mathbf{Pr}[A_i^1 \in J_i^1 \cap b_{ij}^{-1}(J_j^2), A_j^2 = b_{ij}(A_i^1)] \\
&\geq \frac{1}{7} \cdot 2^{-O(L/\varepsilon)} \cdot 2^{-O(L/\varepsilon)} \\
&= 2^{-O(L)}.
\end{aligned}$$

We conclude that the expected value of the randomized solution A is $\mathbf{Pr}[A_j^2 = b_{ij}(A_i^1)] \geq 2^{-O(L)} > \delta$, where the last inequality holds if $c > 0$ is sufficiently small, and this completes the proof of Lemma 5.29. □

Putting it all together

The above reduction from unique 2-prover game to MULTICUT produces a gap of $L(n) = \Omega(\min\{\frac{1}{\eta(n)}, \log\frac{1}{\delta(n)}\})$. We assumed $m(\eta, \delta) \leq O(\log n)$, and thus the resulting MULTICUT instance G has size $N = (n2^d)^{O(1)} = n^{\Theta(1)}$. It follows that in terms of the instance size N, the gap is

$$L(N) = \Omega(\min\{\tfrac{1}{\eta(N^{\Theta(1)})}, \log\tfrac{1}{\delta(N^{\Theta(1)})}\}).$$

This completes the proof of the part of Theorem 5.5 regarding the MULTICUT problem, namely, that the Unique Games Conjecture implies that it is NP-hard to approximate MULTICUT within the above factor $L(N)$. In fact, the above proof shows that it is even NP-hard to obtain a $(7/8, L(N))$-bicriteria approximation.

Note that the number of demand pairs is $k = n2^{m-1} = n^{\Theta(1)}$, and thus the hardness of approximation factor is similar when expressed in terms of k as well. Note also that all edge weights in the MULTICUT instance constructed above are bounded by a polynomial in the size of the graph. Therefore, via a standard reduction, a similar hardness result holds for the unweighted MULTICUT problem as well.

5.5.3 Hardness of approximating SPARSEST-CUT

In this section we prove the part of Theorem 5.5 regarding the SPARSEST-CUT problem. The proof follows immediately from the next lemma in conjunction with the hardness of bicriteria approximation of MULTICUT (from the previous section).

Lemma 5.30. *Let $0 < \alpha < 1$ be a constant. If there exists an algorithm for* SPARSEST-CUT *that produces in polynomial time a cut whose value is within factor $\rho \geq 1$ of the minimum, then there is a polynomial time algorithm that computes an $(\alpha, \frac{2\rho}{1-\alpha})$-bicriteria approximation for* MULTICUT.

Proof. Fix $0 < \alpha < 1$, and suppose $\mathcal{A}$ is an algorithm for SPARSEST-CUT that produces in polynomial time a cut whose value is within factor $\rho \geq 1$ of the minimum. Now suppose we are given an input graph $G = (V, E)$ and k demand pairs $\{s_i, t_i\}_{i=1}^p$. We may assume without loss of generality that every s_i is connected (in G) to its corresponding t_i. Let $c_{\min}$ and $c_{\max}$ be the smallest and largest edge costs in G, and let $n = |V|$.

We now describe the bicriteria approximation algorithm for MULTICUT. For every value $C \in [c_{\min}, n^2 c_{\max}]$ which is a power of 2, execute a procedure that we will describe momentarily to compute a cutset $M_C \subseteq E$, and report, from all these cutsets M_C whose removal disconnects at least αk demand pairs, the one of least cost. For a given value $C > 0$, the procedure starts with $M_C = \emptyset$, and then iteratively M_C is "augmented" as follows: Take a connected component S of $G \setminus M_C$, apply algorithm $\mathcal{A}$ to $G[S]$ (the subgraph induced on S and all the demand pairs that lie inside S), and if the resulting cutset E_S has value (in $G[S]$) at most $\frac{\rho}{1-\alpha} \cdot \frac{C}{k}$, then add the edges E_S to M_C. Here, the value (ratio of cost to demands cut) of E_S is defined as $b_S = c(E_S)/|\mathbb{D}_S|$, where $\mathbb{D}_S$ is the collection of demand pairs that lie in $G[S]$ and get disconnected (in $G[S]$) when E_S is removed. Proceed with the iterations until for every connected component S in $G \setminus M_C$ we have $b_S > \frac{\rho}{1-\alpha} \frac{C}{k}$, at which point the procedure returns the cutset M_C.

This algorithm clearly runs in polynomial time, so let us analyze its performance. We first claim that for every value C, the cutset M_C returned by the above procedure has sparsity at most $\frac{\rho}{1-\alpha} \frac{C}{k}$. Indeed, suppose the procedure performs t augmentation iterations. Denote by S_i the connected component S that is cut at iteration $i \in [t]$, by E_{S_i} the corresponding cutset output by $\mathcal{A}$, and by $\mathbb{D}_{S_i}$ the corresponding set of demand pairs that get disconnected. Clearly, M_C is the disjoint union $E_1 \cup \cdots \cup E_t$, and it is easy to verify that the collection $\mathbb{D}_C$ of demand pairs cut by the cutset M_C is the disjoint union $\mathbb{D}_{S_1} \cup \cdots \cup \mathbb{D}_{S_t}$. Thus,

$$\begin{aligned} c(M_C) = \sum_{i=1}^{t} c(E_{S_i}) \quad &\leq \quad \frac{\rho}{1-\alpha} \cdot \frac{C}{k} \sum_{i=1}^{t} |\mathbb{D}_{S_i}| \\ &= \quad \frac{\rho}{1-\alpha} \cdot \frac{C}{k} |\mathbb{D}_C|, \end{aligned}$$

which proves the claim.

For the sake of analysis, fix an optimal multicut $M^* \subseteq E$, i.e., a cutset of G whose removal disconnects all the demand pairs and has the least cost. The sparsity of M^* is $b^* = c(M^*)/k$. We will show that if $C \in [c(M^*), 2c(M^*)]$, then the above procedure produces a cutset M_C whose removal disconnects a collection $\mathbb{D}_C$ containing $|\mathbb{D}_C| \geq \alpha k$ demand pairs; this will complete the proof of the lemma, because it immediately follows that

$$c(M_C) \leq \frac{\rho}{1-\alpha} \cdot \frac{C}{k} |\mathbb{D}_C| \leq \frac{\rho}{1-\alpha} \cdot 2c(M^*),$$

and clearly $c(M^*) \in [c_{\min}, \binom{n}{2} \cdot c_{\max}]$. So suppose now $C \in [c(M^*), 2c(M^*)]$ and assume for contradiction that $|\mathbb{D}_C| < \alpha k$. Denote by $V_1, \ldots, V_p \subseteq V$ the connected components of $G \setminus M_C$, and let $\mathbb{D}_j$ contain the demand pairs that lie inside V_j. It is easy to see that $\sum_{j=1}^{p} |\mathbb{D}_j| = k - |\mathbb{D}_C| > (1-\alpha)k$. Similarly, let M_j^* be the collection of edges in M^* that lie inside V_j. Then $c(M^*) \geq \sum_{j=1}^{p} c(M_j^*)$. Notice that, in every induced graph $G[V_j]$, the edges of M_j^* form a cutset (of $G[V_j]$) that cuts all the demand pairs in $\mathbb{D}_j$. Using the stopping condition of the procedure, and since $\mathcal{A}$ provides an approximation within factor ρ, we have $c(M_j^*) \geq \frac{1}{1-\alpha}\frac{C}{k}|\mathbb{D}_j|$ (the inequality is not strict because $\mathbb{D}_j$ might be empty). We thus derive the contradiction

$$c(M^*) \geq \sum_{j=1}^{p} c(M_j^*) \geq \frac{1}{1-\alpha} \cdot \frac{C}{k} \sum_{j=1}^{p} |\mathbb{D}_j| > c(M^*).$$

This shows that when $C \in [c(M^*), 2c(M^*)]$, the procedure stops with a cutset M_C whose removal disconnects $|\mathbb{D}_C| \geq \alpha k$ demand pairs, and concludes the proof of the lemma. □

5.5.4 Hardness of approximating MIN-2CNF≡ DELETION

In this section, we modify the reduction in Section 5.5.2 to obtain a hardness of approximation for MIN-2CNF≡ DELETION. In particular, we reduce the MULTICUT instance obtained in Section 5.5.2 to MIN-2CNF≡ DELETION, such that a solution to the latter gives a MULTICUT of the same cost in the former.

The MIN-2CNF≡ DELETION instance contains $2^{m-1}n$ variables, one for each demand pair $(u, \overline{u})$. In particular, for every demand pair $(u, \overline{u}) \in \mathbb{D}$, we associate the literal x_u with u and the literal $x_{\overline{u}} = \neg x_u$ with $\overline{u}$. There is a clause for every edge (u, v) in the graph G—$(x_u = x_v)$—with weight equal to w_e.

The following lemma is immediate from the construction and implies an analogue of Lemma 5.29 for MIN-2CNF≡ DELETION.

Lemma 5.31. *Given an assignment S of cost W to the above instance of* MIN-2CNF≡ DELETION*, we can construct a solution of cost W to the* MULTICUT *instance G.*

Proof. Let M be the set of edges (u, v) for which $S(x_u) \neq S(x_v)$. Then M corresponds to the clauses that are not satisfied by S and has weight W. The lemma follows from observing that M is indeed a multicut—S is constant over connected components in $G \setminus M$, and for any demand pair $(u, \overline{u})$, $S(x_u) \neq S(x_{\overline{u}})$. □

We now give an analog of Lemma 5.27.

Lemma 5.32. *If there is a solution A for the unique 2-prover game G_Q such that the total weight of the satisfied questions is at least $1-\eta$, then there exists an assignment S for the above* MIN-2CNF≡ DELETION *instance such that $c(S) \leq 2^{m+1}n$.*

Proof. Given the solution A for G_Q, we construct an assignment S as follows. For every question q_i^p and for every vertex u in the corresponding hypercube C_i^p, define $S(x_u)$ to be the A_i^p-th bit of u, i.e., $S(x_u) = u_{A_i^p}$. Note that this is a valid assignment, i.e., $S(x_u) = 1 - S(x_{\overline{u}})$ for all vertices u, as $u_{A_i^p} = 1 - \overline{u}_{A_i^p}$.

We bound the cost of the solution by first analyzing the clauses corresponding to hypercube edges in the corresponding MULTICUT instance. Consider unsatisfied clauses containing both variables in the same hypercube C_i^p, and note that the hypercube edges corresponding to these clauses form a dimension-A_i^p cut in the cube C_i^p. Therefore, the total weight of these clauses is at most $(2^{m-1})(2n) = 2^m n$.

Finally, consider an unsatisfied clause $x_u = x_v$ corresponding to vertices in different hypercubes C_i^1 and C_j^2. Then $S(x_u) \neq S(x_v)$ implies that $u_{A_i^1} = v_{b_{ij}(A_i^1)} \neq v_{A_j^2}$, or, $b_{ij}(A_i^1) \neq A_j^2$. There are at most 2^m such clauses for each question pair not satisfied by the solution A. Therefore, the total weight of such clauses is at most $2^m \frac{n}{\eta}\eta = 2^m n$.

The lemma follows from adding the two costs. □

Lemmas 5.31 and 5.32 along with Lemma 5.29 imply the part of Theorem 5.5 regarding MIN-2CNF≡ DELETION.

5.5.5 Limitations of our approach

The main bottleneck to improving the hardness factor lies in Friedgut's Junta Theorem. These bounds are tight in general, as shown by the tribes function [26] (see also [66, Section 2]).

Our reduction does not extend to the uniform-demands case of the SPARSEST-CUT problem or the BALANCED-CUT problem. In particular, if a 2-prover system has a low-cost balanced cut, then the corresponding graph on hypercubes would have a low-cost balanced cut even if the 2-prover system does not have a high value solution.

5.6 Concluding remarks

Our work reduces the gap between the known approximability and hardness of approximation for the SPARSEST-CUT and MULTICUT problems. It would be nice to reduce this gap further. For the MULTICUT problem, an approximation better than $O(\log n)$ seems unlikely. Even for the SPARSEST-CUT, an improvement over the currently best $O(\sqrt{\log n})$ approximation appears unlikely. On the other hand, there is a vast scope for improving the hardness of approximation results. As a related problem, it is important to resolve the Unique Games Conjecture. This would lead to new hardness results for several problems. Finally, it would be interesting to obtain hardness results for other partitioning problems such as min-bisection and minimum balanced cut. Although good "pseudo-approximations" are known for these problems[4], the best true approximation known is a weak $O(\log^2 n)$ approximation [62]. On the hardness side, even a PTAS is not ruled out for these problems.

[4]In a pseudo-approximation, the algorithm is allowed to output a cut with balance slightly worse than that of the optimal cut.

Bibliography

[1] Bibliography on vehicle routing, http://people.freenet.de/Emden-Weinert/VRP.html.

[2] Traveling salesman problem web, http://www.tsp.gatech.edu/.

[3] A. Agarwal, M. Charikar, K. Makarychev, and Y. Makarychev. $O(\sqrt{\log n})$ approximation algorithms for Min UnCut, Min 2CNF Deletion, and directed cut problems. To appear in 37th STOC, 2005.

[4] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: Ranking and clustering. To appear in 37th STOC, 2005.

[5] A. Allahverdi, J. Gupta, and T. Aldowaisan. A review of scheduling research involving setup considerations. *Omega, Int. Journal of Management Science*, 27:219–239, 1999.

[6] N. Alon, E. Fischer, M. Krivelevich, and M. Szegedy. Efficient testing of large graphs. *Combinatorica*, 20(4):451–476, 2000.

[7] N. Alon and J. H. Spencer. *The Probabilistic Method*. John Wiley and Sons, 1992.

[8] E. M. Arkin, J. S. B. Mitchell, and G. Narasimhan. Resource-constrained geometric network optimization. In *Symposium on Computational Geometry*, pages 307–316, 1998.

[9] S. Arora. Polynomial-time approximation schemes for euclidean TSP and other geometric problems. *JACM*, 45(5):753–782, 1998.

[10] S. Arora, A. Frieze, and H. Kaplan. A new rounding procedure for the assignment problem with applications to dense graph arrangements. *Mathematical Programming*, 92(1):1–36, 2002.

[11] S. Arora, M. Grigni, D. Karger, P. Klein, and A. Woloszyn. Polynomial time approximation scheme for weighted planar graph TSP. In *Symposium on Discrete Algorithms*, pages 33–41, 1998.

[12] S. Arora and G. Karakostas. A $2 + \epsilon$ approximation algorithm for the k -MST problem. In *Symposium on Discrete Algorithms*, pages 754–759, 2000.

[13] S. Arora, D. Karger, and M. Karpinski. Polynomial time approximation schemes for dense instances of np-hard problems. *JCSS*, 58(1):193–210, 1999.

[14] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.

[15] S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45(1):70–122, 1998.

[16] Sanjeev Arora, James Lee, and Assaf Naor. Euclidean distortion and the sparsest cut. In *Proceedings of the 37th ACM Symposium on Theory of Computing (STOC)*, 2005. To appear.

[17] Sanjeev Arora, Satish Rao, and Umesh Vazirani. Expander flows, geometric embeddings, and graph partitionings. In *Proceedings of the 36th ACM Symposium on Theory of Computing (STOC)*, pages 222–231, 2004.

[18] Yonatan Aumann and Yuval Rabani. An $O(\log k)$ approximate min-cut max-flow theorem and approximation algorithm. *SIAM Journal on Computing*, 27(1):291–301, 1998.

[19] B. Awerbuch, Y. Azar, A. Blum, and S. Vempala. Improved approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. *Siam J. Computing*, 28(1):254–262, 1999.

[20] E. Balas. The prize collecting traveling salesman problem. *Networks*, 19:621–636, 1989.

[21] R. Bar-Yehuda, G. Even, and S. Shahar. On approximating a geometric prize-collecting traveling salesman. In *Proceedings of the European Symposium on Algorithms*, 2003.

[22] Yair Bartal. On approximating arbitrary metrics by tree metrics. In *Proceedings of the 30th ACM Symposium on Theory of Computing (STOC)*, pages 161–168, 1998.

[23] M. Bellare, O. Goldreich, and M. Sudan. Free bits, PCP's and non-approximability – towards tight results. *SIAM Journal on Computing*, 27(3):804–915, 1998.

[24] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[25] S. Ben-David, P. M. Long, and Y. Mansour. Agnostic boosting. In *Proceedings of the 2001 Conference on Computational Learning Theory*, pages 507–516, 2001.

[26] M. Ben-Or and N. Linial. Collective coin flipping. In *Randomness and Computation*, pages 91–115, 1990.

[27] D. P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 1995.

[28] D. P. Bertsekas and J. N. Tsitsiklis. *Neural Dynamic Programming*. Athena Scientific, 1996.

[29] John R. Birge and Francois Louveaux. *Introduction to stochastic programming*. Springer Series in Operations Research. Springer-Verlag, New York, 1997.

[30] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan. The minimum latency problem. In *Proceedings of the 26th Annual ACM Symposium on Theory of Computing*, pages 163–171, 1994.

[31] A. Blum and D. Karger. An $\tilde{O}(n^{3/14})$-coloring algorithm for 3-colorable graphs. *IPL: Information Processing Letters*, 61, 1997.

[32] A. Blum, R. Ravi, and S. Vempala. A constant-factor approximation algorithm for the k-MST problem. *JCSS*, 58:101–108, 1999.

[33] Jean Bourgain. On Lipschitz embeddings of finite metric spaces in Hilbert space. *Israel Journal of Mathematics*, 52(1-2):46–52, 1985.

[34] S. Boyd, P. Diaconis, and L. Xiao. Fastest mixing Markov chain on a graph. *SIAM Review*, 46(4):667–689, 2003.

[35] J. Bruno and P. Downey. Complexity of task sequencing with deadlines, set-up times and changeover costs. *SIAM Journal on Computing*, 7(4):393–404, 1978.

[36] M. Charikar and S. Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, 1999.

[37] M. Charikar, V. Guruswami, and A. Wirth. Clustering with qualitative information. In *Proceedings of the 44th Annual Symposium on Foundations of Computer Science*, pages 524–533, 2003.

[38] M. Charikar, K. Makarychev, and Y. Makarychev. Private Communication, September 2005.

[39] K. Chaudhuri, B. Godfrey, S. Rao, and K. Talwar. Paths, trees, and minimum latency tours. In *Proceedings of the 44th Annual Symposium on Foundations of Computer Science*, Cambridge, Massachusetts, 2003.

[40] Shuchi Chawla, Robert Krauthgamer, Ravi Kumar, Yuval Rabani, and D. Sivakumar. On the hardness of approximating sparsest cut and multicut. In *Proceedings of the 20th IEEE Conference on Computational Complexity (CCC)*, 2005. To appear.

[41] C. Chekuri and A. Kumar. Maximum coverage problem with group budget constraints and applications. In *Proceedings of the International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, 2004.

[42] Chandra Chekuri, Anupam Gupta, Ilan Newman, Yuri Rabinovich, and Alistair Sinclair. Embedding k-outerplanar graphs into ℓ_1. In *Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 527–536, 2003.

[43] Z. Chen, T. Jiang, and G. Lin. Computing phylogenetic roots with bounded degrees and errors. *SIAM J. on Computing*, 32(4):864–879, 2003.

[44] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. *GSIA Technical Report, Carnegie Mellon University*, 1976.

[45] Kenneth L. Clarkson. Las Vegas algorithms for linear and integer programming when the dimension is small. *Journal of the ACM*, 42(2):488–499, 1995.

[46] W. Cohen and J. Richman. Learning to match and cluster entity names. In *ACM SIGIR'01 workshop on Mathematical/Formal Methods in IR*, 2001.

[47] W. Cohen and J. Richman. Learning to match and cluster large high-dimensional data sets for data integration. In *Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2002.

[48] A. Condon and R. Karp. Algorithms for graph partitioning on the planted partition model. *Random Structures and Algorithms*, 18(2):116–140, 1999.

[49] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM J. Comput.*, 23(4):864–894, 1994.

[50] George B. Dantzig. Linear programming under uncertainty. *Management Science*, 1:197–206, 1955.

[51] F. de la Vega. Max-cut has a randomized approximation scheme in dense graphs. *Random Structures and Algorithms*, 8(3):187–198, 1996.

[52] Brian Dean, Michel Goemans, and Jan Vondrak. The benefit of adaptivity: approximating the stochastic knapsack problem. In *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 208–217, 2004.

[53] Brian Dean, Michel Goemans, and Jan Vondrak. Adaptivity and approximation for stochastic packing problems. In *Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 395–404, 2005.

[54] E. Demaine and N. Immorlica. Correlation clustering with partial information. In *Proceedings of APPROX*, pages 1–13, 2003.

[55] M. Desrochers, J. Desrosiers, and M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40:342–354, 1992.

[56] M. Desrochers, J.K. Lenstra, M.W.P. Savelsbergh, and F. Soumis. Vehicle routing with time windows: optimization and approximation. In *B.L. Golden, A.A. Assad (eds.). Vehicle Routing: Methods and Studies, North-Holland, Amsterdam*, pages 65–84, 1988.

[57] Michel Marie Deza and Monique Laurent. *Geometry of cuts and metrics*, volume 15 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, 1997.

[58] D. Emanuel and A. Fiat. Correlation clustering—minimizing disagreements on arbitrary weighted graphs. In *Proceedings of ESA*, pages 208–220, 2003.

[59] Per Enflo. On the nonexistence of uniform homeomorphisms between L_p-spaces. *Arkiv för matematik*, 8:103–105, 1969.

[60] Lars Engebretsen and Marek Karpinski. Approximation hardness of tsp with bounded metrics. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming,*, pages 201–212. Springer-Verlag, 2001.

[61] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *Proceedings of the 35th ACM Symposium on Theory of Computing (STOC)*, pages 448–455, 2003.

[62] U. Feige and R. Krauthgamer. A polylogarithmic approximation of the minimum bisection. *SIAM Journal on Computing*, 31(4):1090–1118, 2002.

[63] U. Feige and D. Reichman. On systems of linear equations with two variables per equation. In *Proceedings of APPROX*, 2004.

[64] V. Filkov and S. Skiena. Integrating microarray data by consensus clustering. In *15th IEEE International Conference on Tools with Artificial Intelligence*, pages 418–427, 2003.

[65] F. Fomin and A. Lilngas. Approximation algorithms for time-dependent orienteering. *Information Processing Letters*, 83(2):57–62, 2002.

[66] E. Friedgut. Boolean functions with low average sensitivity depend on few coordinates. *Combinatorica*, 18(1):27–35, 1998.

[67] A. Frieze, G. Galbiati, and M. Maffioli. On the worst-case performance of some algorithms for the asymmetric traveling salesman problem. *Networks*, 12:23–39, 1982.

[68] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 2000.

[69] N. Garg. Saving an ϵ: A 2-approximation for the k-MST problem in graphs. To appear in 37th STOC, 2005.

[70] N. Garg, V. V. Vazirani, and M. Yannakakis. Approximate max-flow min-(multi)cut theorems and their applications. *SIAM J. Comput.*, 25(2):235–251, 1996.

[71] M. Goemans and J. Kleinberg. An improved approximation ratio for the minimum latency problem. In *Proc. 10th Ann. ACM-SIAM Symp. on Discrete Algorithms*, 1996.

[72] M. Goemans and D. Williamson. A general approximation technique for constrained forest problems. In *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 307–315, 1992.

[73] Michel X. Goemans. Semidefinite programming and combinatorial optimization. *Mathematical Programming*, 49:143–161, 1997.

[74] M.X. Goemans and D.P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24:296–317, 1995.

[75] B.L. Golden, L. Levy, and R. Vohra. The orienteering problem. *Naval Research Logistics*, 34:307–318, 1987.

[76] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *JACM*, 45(4):653–750, 1998.

[77] M. Gravel, W. Price, and C. Gagn. Scheduling jobs in a alcan aluminium factory using a genetic algorithm. *International Journal of Production Research*, 38(13):3031–3041, 2000.

[78] Anupam Gupta, Robert Krauthgamer, and James R. Lee. Bounded geometries, fractals, and low–distortion embeddings. In *Proceedings of the 44th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 534–543, 2003.

[79] Anupam Gupta, Ilan Newman, Yuri Rabinovich, and Alistair Sinclair. Cuts, trees and ℓ_1-embeddings of graphs. *Combinatorica*, 24(2):233–269, 2004. Also in *Proc. 35th FOCS*, 1999, pp. 399–409.

[80] Anupam Gupta, Martin Pal, R. Ravi, and Amitabh Sinha. Boosted sampling: Approximation algorithms for stochastic optimization problems. In *Proceedings of the 36th ACM Symposium on Theory of Computing (STOC)*, pages 417–425, 2004.

[81] Anupam Gupta, Martin Pal, R. Ravi, and Amitabh Sinha. What about wednesday? approximation algorithms for multistage stochastic optimization. In *Proceedings of the International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, page to appear, 2005.

[82] Anupam Gupta, R. Ravi, and Amitabh Sinha. An edge in time saves nine: Lp rounding approximation algorithms for stochastic network design. In *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 218–227, 2004.

[83] J. Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001.

[84] Ara Hayrapetyan, Chaitanya Swamy, and Eva Tardos. network design for information networks. In *Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 933–942, 2005.

[85] D. Hochbaum and D. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *JACM*, 33:533–550, 1986.

[86] Nicole Immorlica, David Karger, Maria Minkoff, and Vahab Mirrokni. On the costs and benefits of procrastination: Approximation algorithms for stochastic combinatorial optimization problems. In *Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 684–693, 2004.

[87] Piotr Indyk. Algorithmic aspects of geometric embeddings. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 10–33, 2001.

[88] K. Jain and V.V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *JACM*, 48(2):274–296, 2001.

[89] D. Johnson, M. Minkoff, and S. Phillips. The prize collecting steiner tree problem: Theory and practice. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 760–769, 2000.

[90] C. Jordan. A two-phase genetic algorithm to solve variants of the batch sequencing problem. *International Journal of Production Research (UK)*, 36(3):745–760, 1998.

[91] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4, 1996.

[92] J. Kahn, G. Kalai, and N. Linial. The influence of variables on boolean functions. In *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 68–80, 1998.

[93] J. Kahn, J. H. Kim, L. Lovasz, and V. H. Vu. The cover time, the blanket time, and the matthews bound. In *Proceedings of the 41th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 467–475, 2000.

[94] M. Kantor and M. Rosenwein. The orienteering problem with time windows. *Journal of the Operational Research Society*, 43:629–635, 1992.

[95] Y. Karuno and H. Nagamochi. A 2-approximation algorithm for the multi-vehicle scheduling problem on a path with release and handling times. In *Proceedings of the European Symposium on Algorithms*, pages 218–229, 2001.

[96] Alexander V. Karzanov. Metrics and undirected cuts. *Mathematical Programming*, 32(2):183–198, 1985.

[97] M. Kearns. Efficient noise-tolerant learning from statistical queries. *JACM*, 45(6):983–1006, 1998.

[98] Michael J. Kearns, Robert E. Schapire, and Linda M. Sellie. Toward efficient agnostic learning. *Machine Learning*, 17(2/3):115–142, 1994.

[99] S. Khot, G. Kindler, E. Mossel, and R. O'Donnell. Optimal inapproximability results for MAX-CUT and other 2-variable CSPs. In *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science (FOCS)*, 2004.

[100] S. Khot and O. Regev. Vertex cover might be hard to approximate to within $2 - \epsilon$. In *IEEE Conference on Computational Complexity*, pages 379–386, 2003.

[101] Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the 34th ACM Symposium on Theory of Computing (STOC)*, pages 767–775, 2002.

[102] Subhash Khot and Nisheeth Vishnoi. The unique games conjecture, integrality gap for cut problems and embeddability of negative-type metrics into ℓ_1, 2005. Manuscript.

[103] P. Klein, A. Agrawal, R. Ravi, and S. Rao. Approximation through multicommodity flow. In *31st Annual Symposium on Foundations of Computer Science, Vol. I, II (St. Louis, MO, 1990)*, pages 726–737, 1990.

[104] A. Kolen, A. Rinnooy Kan, and H. Trienekens. Vehicle routing with time windows. *Operations Research*, 35:266–273, 1987.

[105] Robert Krauthgamer, James Lee, Manor Mendel, and Assaf Naor. Measured descent: A new embedding method for finite metrics. In *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 434–443, 2004.

[106] T. Lane and L. P. Kaelbling. Nearly deterministic abstractions of markov decision processes. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, Edmonton, 2002.

[107] James R. Lee. On distance scales, embeddings, and efficient relaxations of the cut cone. In *Proceedings of the 16th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 92–101, 2005.

[108] James R. Lee and Assaf Naor. Extending lipschitz functions via random metric partitions. *Inventiones Mathematicae*, 160(1):59–95, 2005.

[109] Frank Thomson Leighton and Satish B. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 422–431, 1988.

[110] Nathan Linial. Finite metric-spaces – combinatorics, geometry and algorithms. In *Proceedings of the International Congress of Mathematicians, Vol. III*, pages 573–586, Beijing, 2002. Higher Ed. Press.

[111] Nathan Linial, Eran London, and Yuri Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995. Also in *Proc. 35th FOCS*, 1994, pp. 577–591.

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

[113] Jiří Matoušek. On embedding trees into uniformly convex Banach spaces. *Israel Journal of Mathematics*, 114:221–237, 1999. (Czech version in: *Lipschitz distance of metric spaces*, C. Sc. degree thesis, Charles University, 1990).

[114] Jiří Matoušek. *Lectures on discrete geometry*, volume 212 of *Graduate Texts in Mathematics*. Springer, New York, 2002.

[115] Jiří Matoušek. On the distortion required for embedding finite metric spaces into normed spaces. *Israel Journal of Mathematics*, 93:333–344, 1996.

[116] F. McSherry. Spectral partitioning of random graphs. In *Proceedings of the 42th Annual Symposium on Foundations of Computer Science*, pages 529–537, 2001.

[117] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge, UK, 1995.

[118] Christos Papadimitriou and Santosh Vempala. On the approximability of the traveling salesman problem. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, 2000.

[119] M. Parnas and D. Ron. Testing the diameter of graphs. *Random Structures and Algorithms*, 20(2):165–183, 2002.

[120] M. L. Puterman. *Markov Decision Processes*. Wiley, 1994.

[121] Yuri Rabinovich. On average distortion of embedding metrics into the line and into ℓ_1. In *Proceedings of the 35th ACM Symposium on Theory of Computing (STOC)*, pages 456–462, 2003.

[122] Satish B. Rao. Small distortion and volume preserving embeddings for planar and Euclidean metrics. In *Proceedings of the 15th ACM Symposium on Computational Geometry*, pages 300–306, 1999.

[123] R. Ravi and Amitabh Sinha. Hedging uncertainty: Approximation algorithms for stochastic optimization problems. In *Proceedings of the 10thInteger Programming and Combinatorial Optimization Conference*, pages 101–115, 2004.

[124] R. Raz. A parallel repetition theorem. *SIAM Journal on Computing*, 27(3):763–803, 1998.

[125] G. Robins and A. Zelikovsky. Improved steiner tree approximation in graphs. In *Proc. 10th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 770–779, 2000.

[126] M. Savelsbergh. Local search for routing problems with time windows. *Annals of Operations Research*, 4:285–305, 1985.

[127] L. Schulman. Clustering for edge-cost minimization. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pages 547–555, 2000.

[128] R. Shamir and D. Tsur. Improved algorithms for the random cluster graph model. In *Proceedings of the Scandinavian Workshop on Algorithmic Theory*, pages 230–239, 2002.

[129] David Shmoys and Chaitanya Swamy. Stochastic optimization is (almost) as easy as deterministic optimization. In *Proceedings of the 45th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 228–237, 2004.

[130] David B. Shmoys. Cut problems and their application to divide-and-conquer. In Dorit S. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*, pages 192–235. PWS Publishing, 1997.

[131] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[132] C. Swamy. Correlation clustering: Maximizing agreements via semidefinite programming. In *Proceedings of the Symposium on Discrete Algorithms*, 2004.

[133] K.C. Tan, L.H. Lee, and K.Q. Zhu. Heuristic methods for vehicle routing problem with time windows. In *Proceedings of the 6th AI and Math*, 2000.

[134] S. Thangiah. *Vehicle Routing with Time Windows using Genetic Algorithms, Application Handbook of Genetic Algorithms: New Frontiers, Volume II. Lance Chambers (Ed.).* CRC Press, 1995.

[135] S. R. Thangiah, I. H. Osman, R. Vinayagamoorthy, and T. Sun. Algorithms for the vehicle routing problems with time deadlines. *American Journal of Mathematical and Management Sciences*, 13(3& 4):323–355, 1993.

[136] Luca Trevisan. Approximation algorithms for unique games. *ECCC Report TR05-034*, April 2005.

[137] J. Tsitsiklis. Special cases of traveling salesman and repairman problems with time windows. *Networks*, 22:263–282, 1992.

[138] V. Vazirani. *Approximation Algorithms*. Springer-Verlag, Berlin, 2001.

[139] B. Wellner and A. McCallum. Object consolidation by graph partitioning with a conditionally-trained distance metric. In *KDD Workshop on Data Cleaning, Record Linkage and Object Consolidation*, 2003.

[140] B. Wellner and A. McCallum. Towards conditional models of identity uncertainty with application to proper noun coreference. In *IJCAI Workshop on Information Integration and the Web*, 2003.

[141] Emo Welzl. Suchen und Konstruieren durch Verdoppeln. In Ingo Wegener, editor, *Highlights der Informatik*, pages 221–228. Springer, Berlin, 1996.

[142] J. Wisner and S. Siferd. A survey of u.s. manufacturing practices in make-to-order machine shops. *Production and Inventory Management Journal*, 1:1–7, 1995.

[143] W. Yang and C. Liao. Survey of scheduling research involving setup times. *International Journal of Systems Science*, 30(2):143–155, 1999.