Deterministic and Approximation Algorithms for Graph Theoretic Problems

Omer Wasim*
Department of Computer Science
The University of Hong Kong

April, 2018

A thesis submitted to the Department of Computer Science, HKU in partial fulfillment of the course COMP4801 and the degree of Bachelor of Engineering in Computer Science.

*email: omerwa90@connect.hku.hk

Abstract: Cut-based graph theoretic problems encompass some of the most seminal problems in theoretical computer science, having wide ranging applications within and beyond computing. The standard graph partitioning seeks to partition a graph $G = (V, E)$ into k pieces where $k \leq n = |V|$ while fulfilling a balance condition on the size of each piece. The goal is to minimize the edges that cross from one piece to another. Unfortunately, this problem and many of its variants are NP-complete. The set of closely related problems includes finding a balanced cut, maximum-cut and sparsest cut in a weighted undirected graph[1]. In this thesis, we study various NP-hard graph partitioning problems. The overarching goals of this thesis are: 1) to survey several breakthrough techniques in convex and continuous optimization (such as linear programming, semidefinite programming and eigenvalue optimization) that have been successfully applied to get provably good solutions for fundamental problems in combinatorial optimization and theoretical computer science and 2) to provide empirical evidence of the quality of approximation ratios obtained by employing such techniques to solve the maximum cut and sparsest cut problems in real world problem instances.

[1]the weighted undirected setting is sufficiently general to encapsulate others.

Contents

List of Figures

1 Introduction

In this thesis, we study a varied selection of topics in combinatorial optimization for graph theoretic problems which arise in a wide array of practical applications. The focus will be on studying several fundamental NP-hard problems and techniques that have been successfully employed in the past few years to affirmatively answer long standing open questions across complexity theory and the design of algorithms.

We aim to study NP-hard optimization problems in the broadest sense. We place special emphasis on cut-based graph theoretic problems. Such problems arise quite commonly in both theory and practice. For example consider a simple problem of image segmentation- i.e. to break an image into a number of parts so that it is easier to analyze the image on a macroscopic level. One needs to partition the image into clusters so that adjacent pixels which are similar are allocated to the same cluster, whereas dissimilar pixels are allocated to different clusters. This problem elegantly abstracts as a standard graph partitioning problem, where the goal is to partition the vertices (pixels) of the graph (image) into a number of pieces (partitions) such that the weight of the edges (dissimilarity between adjacent pixels) is minimized. One can think of each weighted edge as a connection between adjacent pixels where the weight is proportional to the pixel similarity.

Our *goals* for this project are as follows:

1. Studying extensively the techniques (such as convex programming) in the design and analysis of approximation algorithms.

2. Studying several important algorithmic results and breakthroughs over the past 2 decades in TCS.

3. Improving the state of the art results wrt approximation ratios/running times.

4. Rigorously analyze the performance of some well known algorithms for cut based problems (e.g max-cut, sparsest cut) on real world data sets.

Organization: Most sections of this thesis can be read independently of each other. Our organization is as follows: In the next subsection we briefly discuss a general framework in which NP-hard optimization problems are framed, and a canonical strategy to get reasonably good solutions. In section 2 we introduce the graph partitioning problem, its variants and other closely related problems. A brief overview of the results will also be provided. Section 3 contains a selection of problems we studied that we feel are most representative of the powerful algorithmic techniques and paradigms employed to tackle the curse of NP-completeness. Section 4 is intended to acquaint the reader with the theory of metric embeddings which is necessary to understand the sparsest cut problem. Section 5 includes the sparsest cut problem, and discusses various algorithms that have provably good guarantees. The final section contains empirical results obtained by running the (theoretically) state-of-the-art algorithms on real world data sets for the maximum cut and the sparsest cut problem respectively.

1.1 NP-completeness and Optimization

Algorithms can be broadly classified in two types-deterministic and non-deterministic respectively. For any given input problem, a deterministic algorithm runs exactly the same way-the order of the steps executed by the algorithm are identical for any two given runs on a certain underlying machine[2]. On the other hand, a non-deterministic algorithm is one for which the steps may be executed in possibly different orders and need not even be the same. A non-deterministic algorithm can be seen as one which makes certain choices during its execution so for two different runs on the same input, the steps taken and their order may not coincide. Non-determinism exploits the structure of the problem to yield provably good solutions (which may be randomized) to the intended problem.

The input to a general optimization problem includes an optimization objective, an underlying problem instance, and a set of constraints to be respected in a feasible solution. An algorithm is evaluated on the basis of how well it meets the optimization objective. For a wide variety of useful problems in both theory and practice, there is no hope to find the optimal solution in polynomial time unless the $P = NP$ conjecture holds true (see Figure 1.1). Thus, in these cases the goal of an algorithm designer is to design algorithms that find feasible solutions within some factor of the optimal solutions. Such an algorithm which is guaranteed to find a reasonably good solution for any input instance is known as an approximation algorithm. The quality of the solution it obtains is measured by its *approximation ratio*. Here is a formal definition:

Definition 1.1 (Approximation Algorithm [27]) *An α-approximation algorithm for an optimization problem is one which finds in polynomial time, a feasible solution whose cost is within an α factor of the optimal solution for any problem instance. Under the standard practice of defining an approximation ratio, it is easy to see that for a minimization objective $\alpha \geq 1$ and for a maximization objective, $\alpha \leq 1$.*

Current and past research has focused mainly on designing algorithms that have good approximation ratios and improving ones that have already been discovered. Recent work on the Unique Games Conjecture[3] has focused on the impossibility of achieving certain approximation ratios for a various problems under the assumption that $P \neq NP$ (see Figure 1.1). Although the conjecture has not been proven yet, it is widely believed to hold true and hence a major goal of researchers in this area is to close the gap between the current best ratio and that which is possible assuming the UGC holds true. Similar to showing that a certain problem is NP-complete via a reduction to a certain NP-complete problem already known, one can establish that no better approximation ratio is possible for a given problem via a reduction from Unique Games.

Techniques that are normally used to design approximate algorithms include linear and semidefinite programming [10] which harness results in probability theory and concentration of measure. Note that linear and semidefinite relaxations can be solved in polynomial time

[2]The Turing model of computation is normally assumed.

[3]Unique Games Conjecture[15] is also referred to as UGC hereafter

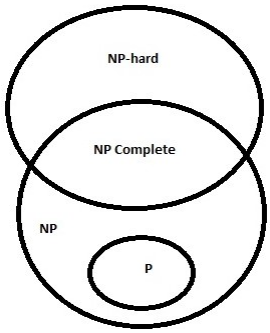

Figure 1.1: Basic complexity classes under the assumption $P \neq NP$. Note that $P \subseteq NP$

via the ellipsoid algorithm or other suitable interior point methods. Moreover, low distortion metric embeddings [3] have been used successfully in the past to approximate solutions to certain NP-hard problems. A general approximation scheme includes the following steps.

1. Model an optimization problem exactly as an integer linear program (ILP). Note that it is NP-hard to solve an ILP.
2. Relax the ILP. This is done by relaxing the constraints and/or variables.They are relaxed as linear constraints in the case of LP, as vectors to give a vector programming relaxation.
3. Solve the VP/LP.
4. Round the solution (using a rounding algorithm) to an integer solution. Bound the cost of the rounded solution using a clever analysis.

Mathematically, we can express the canonical process as follows, where P is a maximization problem. We let:

1. FRAC(P): The value of the relaxed(convex) solution.
2. OPT(P): The value of the true combinatorial optimum.
3. ROUND(P): The value of the solution obtained by the rounding algorithm.

Then, we have that

$$ROUND(P) \leq OPT(P) \leq FRAC(P)$$

The inequalities are reversed for a minimization problem. The goal is then to find an α s.t. ROUND(P) $\geq \alpha$FRAC(P). This establishes a rigorous guarantee on the quality of the approximation scheme.

Approximation algorithms are quite useful in practice and by banking on the fact that linear/semidefinite programming can be solved in polynomial time, one obtains an approximation scheme in polynomial time under some compromise on the optimality of the solution. This compromise has been shown in practice to be acceptable in many applications[25]. Note that the number of constraints of such a program could be exponential in input size but via a good separation oracle the number of constraints to be checked can be restricted to a polynomial number.

2 Background and Preliminaries

2.1 Motivation

Graph partitioning is a well studied problem in computer science. It has numerous practical applications in network design, image segmentation, task scheduling in multiprocessing environments and sparse gaussian elemination. There are various settings of the graph partitioning problem. One which generally captures the gist of the problem is: Given a graph (undirected) G, partition the graph into k pieces, such that each partition contains approximately the same number of vertices and the edge costs between those partitions are minimized. This is known to be an NP-hard problem, therefore only approximation algorithms exist currently. Related problems include the balanced cut and its specific case for k=2, the minimum bisection problem, both of which are NP-hard too. The motivation to study the problem comes from the pervasiveness of the problem in computer science and other fields. The usefulness of efficient algorithms can be realized from the fact that a lot of real world networks that are studied-such as say pathological networks, social networks, distribution of plant species, trade routes etc have relatively concentrated regions, and the problem of graph partitioning seeks to identify those regions.

In circuit design for example, the goal is to minimize the number of wires crossing each other and the cost of the components used, while being restricted to a certain board size. Graph partitioning in this case, is used to minimize the size of the interface by identifying a set of clusters containing endpoints which should be planted close to each other to minimize the number of crossing wires. A related problem we study extensively in this thesis is the sparsest cut problem-closely imitating the objective sought by efficient circuit design.

Graph partitioning is intimately connected to clustering and cut problems which will also form part of our study, and section 3 discusses some of them in detail.

2.2 Existing results on some cut-based problems

A prototypical graph partitioning, one which is called the k balanced partitioning on a graph $G = (V, E)$ is defined as follows:

Definition 2.1 (k-balanced partitioning problem) *Given a graph $G = (V, E)$ and a weight function $w : E \to \mathbb{R}^+$, partition G into k pieces of roughly equal size (i.e. n/k in the case when $k|n$), such that the total weight of edges connecting the pieces together is minimized (see Figure 2.1).*

Natural extensions include relaxing the balance condition, i.e. the number of vertices in any specific piece-this introduces a new parameter v called the balance parameter. The problem then is to minimize the total weight of cut edges (i.e. edges belonging to endpoints in distinct components) such that every piece contains no more than $(1 + v)\frac{n}{k}$ vertices, for $v > 0$. Note that if $v = 0$, then it is the standard k balanced partitioning problem.

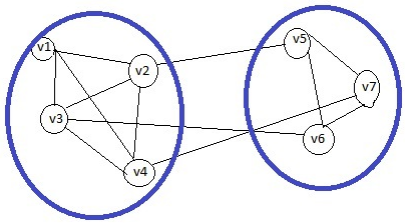

Figure 2.1: Graph partitioning problem: Note that the 2 pieces contain are roughly the same size such that the sum of the edge weights separating them is minimized. In this case, $k = 2$ and the problem is referred to as the minimum bisection problem which is also NP-complete.

Definition 2.2 (Bi-criteria approximation algorithm) *A bi-criteria approximation algorithm for the (k, v) balanced graph partitioning problem is one which outputs a solution in which each piece P_i, $i \in \{1, 2, ..., k\}$ has size at most $\frac{kv}{n}$ for $v > 1$ and whose value is at most $O(g(n))OPT$ where $g(n) \geq 1$ with OPT denoting the optimal value of the k balanced partitioning problem.*

It is quite surprising that even in the special case when $k = 2$ is also NP hard. To our knowledge, the best known bicriteria approximation of $O(\sqrt{\log n})$ is achieved by Arora, Rao and Vazirani [3] which uses semidefinite programming. Kruthgramer et al. [16] give a bi-criteria approximation algorithm for the general problem in which k is not restricted whose performance is $O(\sqrt{\log n \log k})$. They use a semidefinite relaxation which combines l_2^2 metrics with spreading metrics. For the rounding procedure, they use breakthrough techniques of Charikar et al. [8] and Makarychev et al. [9] for generating orthogonal separators, which they used to develop an efficient approximation algorithm for solving unique games.

Since the problem of graph partitioning has received so much attention over the past few decades, various heuristic results currently exist. Note that unlike the above results, heuristics do not offer any theoretical evidence that the algorithms perform better but provide empirical evidence to support claims. In particular for the k partitioning problem, a multi-level scheme for partitioning irregular graphs has been quite successful [13]. There are software packages available that use heuristic approaches and are useful in practice.

Clustering is a strongly related problem to graph partitioning and one which has been studied extensively under various objectives and settings. There has been recent work on hierarchical clustering,[7] and which uses SDP relaxations and metric embeddings, which might imply that similar techniques to related problems in graph partitioning might be applicable.

2.3 Problems studied

Our has had a predominantly theoretical focus and we studied a broad selection of topics. This has not been limited to papers in prominent TCS conferences (STOC, FOCS, SODA) but also surveys, books and courses. A partial list of the most representative problems we have studied so far is as follows:

1. Minimum Multiway Cut. ([5])
2. Maximum Cut. ([10])
3. Quadratic Programming. ([7], [20],)
4. Correlation Clustering. ([23])
5. Graph Coloring. ([26])
6. Unique Games. ([24] [8])
7. Graph partitioning ([16]) and its numerous variants.
8. Oblivious Routing and packing.
9. Sparsest Cut. ([3], [4])

In the next section, we present several useful techniques such as linear and semidefinite programming, which have been used in getting better approximations to these problems. Some of the most important techniques and results surveyed so far are presented. These techniques can be seen as our 'methodology' for this project.

3 Convex Relaxations: problems and techniques

There are two powerful convex optimization paradigms used in the design of approximation algorithms-namely linear and semidefinite programming. In the following subsection, linear programming and metric embedding techniques as applied to the minimum multiway cut problem are discussed and an algorithm due to Călinescu, Karloff and Rabani [5] is presented. Then, the seminal work of Goemans and Williamson [10] for the Max-Cut problem is presented. This was one of the first works which introduced semidefinite programming to design approximation algorithms. We then discuss quadratic programming and graph coloring-two problems in which semidefinite programming has been subsequently applied to get better approximation guarantees. The style of our presentation and proof ideas closely mirror that of [27].

3.1 The multiway cut problem

Problem 1 *Given an undirected graph $G = (V, E)$, costs $c_e \geq 0$ for all $e \in E$ and k distinguished vertices $s_1, s_2, ..., s_k$, one is required to output a set of minimum total cost edges, such that for any s_i, s_j, where $j \neq i$ are found in a different connected component of $(V, E \backslash F)$. See Figure 3.1.*

First, note the connection with the graph partitioning problem here: We find k connected components (pieces) similar to the graph partitioning problem with the same objective function namely to minimize total edge cost, but with the additional requirement that k specific vertices must lie in different components (pieces).

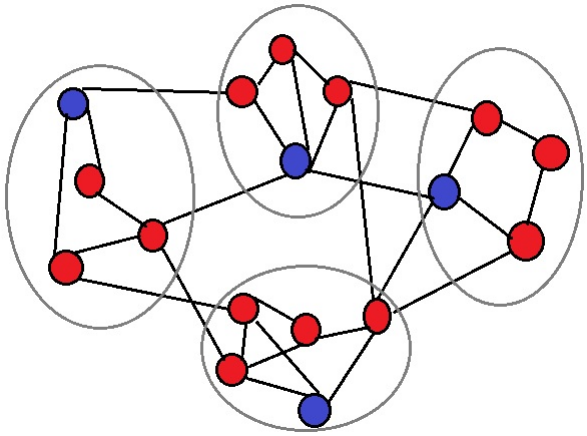

Figure 3.1: The multiway cut problem: The blue vertices denote the distinguished sources. The red vertices denote all the rest of the vertices. The goal is to separate those distinguished sources by computing a partition of V such that the weight of edges between individual partitions (circled in gray) are minimized.

Theorem 3.1 *The multiway cut problem is NP-complete. Thus, one cannot solve it in polynomial time unless P=NP.*

3.1.1 A simple 2-approximation algorithm

Note the structure of an optimal solution first. Let F denote an optimal solution. Removing F collapses G into k connected components. Let $V_1, V_2, ..., V_k$ denote the set of vertices in G reachable from $s_1, s_2, ..., s_k$. Note that for any $i \neq j$ $V_i \cap V_j = \emptyset$. Let $\delta(V_i)$ denote the set of removed edges which were incident to vertices in V_i. Now, note that $\bigcup_i \delta(V_i) = F$.

We now present a combinatorial 2-approximation algorithm:

Algorithm 1 2-approximation algorithm $(G, s_1, s_2, ..., s_k)$

1: $F^* = \emptyset$
2: **for** $i \in \{1, 2, ..k\}$ **do**
3: Add infinite capacity edges from all s_j, where $j \neq i$ to a terminal vertex t.
4: Compute the maximum s_i, t cut. Denote by E_i, the set of edges participating in the cut.
5: $F^* = F^* \bigcup E_i$
6: return F^*

Theorem 3.2 *Algorithm 1 is a 2-approximation algorithm for the multiway cut problem.*

Proof: Let $c(F^*)$ denote the cost of our solution-i.e, the sum of weights of all edges included in F^*, and similarly denote $c(F)$ denote the cost of the optimal solution. Note that in the optimal solution, any edge can be incident to at most 2 different connected components, i.e for $e = (u_i, u_j) \in F$, $u_i \in V_i$ and $u_j \in V_j$. Also, $c(E_i) \leq c(\delta(V_i))$, since E_i contains all those edges found in the minimum isolating cut for s_i. Hence we have that,

$$c(F^*) = \sum_{i=1}^{k} c(E_i) \leq \sum_{i=1}^{k} c(\delta(V_i)) \leq 2c(F) = 2OPT.$$

■

Hence our algorithm yields a 2 approximation, which can be trivially extended to yield a 2(1-$\frac{1}{k}$)-approximation. We now discuss how linear programming can be used to give us a $\frac{3}{2}$ approximation.

3.1.2 An LP-Rounding Algorithm: Beating factor 2

In this subsection, we describe how we can design a better algorithm using randomized rounding of a linear program. It also demonstrates the application of l_1 metrics.

We use a key insight from the above combinatorial algorithm. Indeed, to compute a minimum cost set of edges F, it suffices to compute a partition of V such that each distinguished

source is found in one of the k partitions and no two sources lie in the same partition. As before, let V_i denote the set of all vertices reachable from s_i, so that we have that $\delta(V_i)$ denotes the set of cut edges. Note that for every edge e in the optimal solution F, e is found in $\delta(V_i), \delta(V_j)$, such that $i \neq j$.

For each vertex $v \in V$ we have a variable k different variables $x_v^i, i \in [k]$, such that if v is found in the V_i, then $x_v^i = 1$ and is 0 otherwise. Note that for all $s_i, i \in [k]$, we have that $x_{s_i}^i = 1$, since s_i must surely be in V_i, the set of vertices reachable from s_i. Similarly, for every edge $e = (u, v)$, we create a variable z_e^i, such that $z_e^i = 1$ if $e \in \delta(C_i)$ and 0 otherwise. Note that any removed edge e must be found in $\delta(V_i), \delta(V_j)$, such that $i \neq j$. For such an edge e, we have that $z_e^i = 1, z_e^j = 1$.

Now note that for any edge $e = (u, v)$ we have that $z_e^i \geq x_u^i - x_v^i$ and $z_e^i \geq x_v^i - x_u^i$, so that $z_e^i = |x_u - x_v|$. Indeed if z is not a cut edge and is incident to vertices found in some C_i then we have that $z_e^i = 0$ and in fact z_e is the all-zeroes vector. If $e \in \delta(C_i), \delta(C_j)$ for some i, then clearly, $z_e^j = z_e^i = 1$. We give the following Integer Linear Program(ILP) as follows:

ILP:

$$
\begin{aligned}
\textit{minimize} \quad & \frac{1}{2} \sum_{e \in E} c_e \sum_{i=1}^{k} z_e^i \\
\textit{subject to} \quad & \sum_{i=1}^{k} x_u^i = 1 \quad \forall u \in V \\
& z_e^i \geq x_u^i - x_v^i \quad \forall e = (u, v) \in E \\
& z_e^i \geq x_v^i - x_u^i \quad \forall e = (u, v) \in E \\
& x_{s_i}^i = 1 \quad i \in 1, 2, ..., k \\
& x_u^i \in \{0, 1\} \quad \forall u \in V, i = 1, 2, ..., k
\end{aligned}
$$

LP-relaxation and connection with l_1 metrics

Definition 3.1 (Distance in the l_1 metric) *For any $u, v \in \mathbb{R}^d$, where u, v are vectors in the d-dimensional euclidean space the l_1 metric is a metric such that the distance between u, v is given as $\|u - v\|_1 = \sum_{i=1}^{d} |u_i - v_i|$, where u_i, v_i denote the i^{th} components of vectors u and v.*

Key technique: ILP's are NP-hard to solve. One way to cope with this is to relax the integrality constraints of the ILP yielding an LP relaxation which can be solved in polynomial time. The LP solution is then rounded to yield a feasible integral solution (not necessarily optimal) to the original problem.

Note that relaxing the integrality constraints of our ILP, and then using the above definition of the l_1 metric, we obtain a concise LP formulation to the multiway cut problem.

Whereas our original ILP contained k separate variables x_v^i, where $i \in [k]$, affter relaxing the integrality condition and thinking of each vector as being assigned a k dimensional vector, we have that each $x_v \in \mathbb{R}^k$ must lie in a k-dimensional simplex: this follows from the first constraint of our ILP, which enforces that the sum of entries of x_v is 1.

LP:

$$\begin{aligned} \textit{minimize} \quad & \frac{1}{2} \sum_{e=(u,v)\in V} c_e \|x_u - x_v\|_1 \\ \textit{subject to} \quad & x_{s_i} = 1 \quad \forall i \in [k] \\ & x_u \in \Delta_k \quad \forall u \in V \end{aligned} \tag{3.1}$$

Definition 3.2 (Open and Closed Balls in l_1) *Let the l_1 metric between any two points $x, y \in V \subseteq \mathbb{R}^n$ be defined as follows:*

$$\|x - y\|_1 = \sum_{i=1}^{n} |x_i - y_i|$$

The norm $\|x_1\|_1$ for a point $x \in V$ is defined similarly:

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|$$

We define the open and closed balls denoted by $B^o(x, r)$ and $B(x, r)$ in l_1 as follows:

$$B^o(x, r) = \{y \in V | \, \|x - y\|_1 < 2r\}, \text{ and } B(x, r) = \{y \in V | \, \|x - y\|_1 \leq 2r\}$$

The factor of 2 in the definition ensures that all points lie in a ball of radius 1 from any s_i where the radius, r is uniformly chosen from the open interval $(0, 1)$.

Algorithm 2 Approximation algorithm for minimum multiway cut

1: Solve the linear program (3.1) and let x be the solution.
2: Pick a random permutation π of the set $\{1, 2, ..., k\}$.
3: Pick the radius r uniformly at random from (0,1).
4: $A = \emptyset$, $C_i = \emptyset$ for all $i = 1, 2, .., k$ ▷ A denotes the set of assigned vertices while C_i denotes vertices assigned to s_i.
5: **for** $i = 1$ to k **do**
6: Let $C_{\pi(i)} = B(s_{\pi(i)}, r) \backslash A$.
7: $A = A \bigcup C_{\pi(i)}$.
8: $C_{\pi(i)} = V \backslash X$.
9: Return $\bigcup_{i=1}^{k} \delta(C_i)$.

Theorem 3.3 *The algorithm is a 3/2 approximation to the multiway cut problem.*

Proof: The proof of Theorem 3.3 relies on the following facts. The first states that the absolute difference along any dimension is bounded by at most half of the l_1 norm.

Fact 3.1 *For any $j \in \{1, 2, ..., k\}$ and $u, v \in V$, $|x_u^j - x_v^j| \leq \frac{1}{2} \|x_u - x_v\|_1$.*

Proof: WLOG, let $x_u^j \geq x_v^j$ so that

$$|x_u^j - x_v^j| = x_u^j - x_v^j = 1 - \sum_{i \neq j} x_u^i - (1 - \sum_{i \neq j} x_v^i) = \sum_{i \neq j} x_v^i - \sum_{i \neq j} x_u^i \leq \sum_{i \neq j} |x_v^i - x_u^i|.$$

Adding $|x_u^j - x_v^j|$ to both sides yields $2|x_u^j - x_v^j| \leq \|x_u - x_v\|$. ■

Fact 3.2 *A vertex $u \in B(s_j, r)$ if and only if $1 - x_u^j \leq r$.*

Proof: By definition, $u \in B(s_j, r)$ iff $\|e_j - x_u\|_1 \leq 2r$. The vertices s_j correspond to the k dimensional standard basis vectors e_j from the constraints in the LP. We have,

$$\begin{aligned} \|e_j - x_u\|_1 = \sum_{i=1}^{k} |e_j^i - x_u^i| \leq 2r \iff \\ \sum_{i \neq j} |x_u^i + 1 - x_u^j \leq 2r \iff \\ 1 - x_u^j + 1 - x_u^j \leq 2r \iff \\ 1 - x_u^j \leq 2r \end{aligned}$$

■

The remaining part of the proof now bounds the probability that edge $e = (u, v)$ is in the set of edges output by the algorithm. Let X_i denote the event that i is the first index such that at least one of u or v is assigned to $B(s_i, r)$. Let Y_i denote the event that exactly one of $u, v \in B(s_i, r)$. First note that Y_i does not depend on the random permutation and for any edge to be output by the algorithm, there must exist an i such that X_i and Y_i both occur.

From Fact 3.2, we have that

$$Pr[Y_i] = Pr[r \in (min\{1 - x_u^i, 1 - x_v^i\}, max\{1 - x_u^i, 1 - x_v^i\})] = |x_u^i - x_v^i|.$$

Let j be the index which minimizes over all $i = 1, 2, ..., k$, $\min\{1 - x_u^i, 1 - x_v^i\}$. The probability that i occurs after j is $1/2$ and hence for $i \neq j$ using conditioning we have:

$$\begin{aligned} Pr[X_i \wedge Y_i] =& Pr[X_i \wedge Y_i | \pi(s_i) < \pi(s_j)] Pr[\pi(s_i) < \pi(s_j)] + \\ & Pr[X_i \wedge Y_i | \pi(s_j) < \pi(s_i)] Pr[\pi(s_j) < \pi(s_i)] \\ & \leq Pr[Y_i | \pi(s_i) < \pi(s_j)].\frac{1}{2} \\ & = \frac{1}{2}|x_u^i - x_v^i| \end{aligned}$$

Moreover, $Pr[X_j \wedge Y_j] \leq Pr[X_j] \leq |x_u^j - x_v^j|$. Thus, the probability that (u, v) is in the set of edges returned is:

$$\begin{aligned} \sum_{i=1}^{k} Pr[X_i \wedge Y_i] \leq & |x_u^j - x_v^j| + \frac{1}{2} \sum_{i \neq j} |x_u^i - x_v^i| \\ & = \frac{1}{2}|x_u^j - x_v^j| + \frac{1}{2} \left\| x_u - x_v \right\|_1 \\ & \leq \frac{3}{4} \left\| x_u - x_v \right\|_1 . \end{aligned}$$

where the last inequality follows from Fact 3.1. The expected value of the solution is then given by:

$$\begin{aligned} & \sum_{e=(u,v) \in E} c_e.Pr[(u, v) \text{is returned by algorithm}] \\ & \leq \sum_{e=(u,v) \in E} c_e.\frac{3}{4} \left\| x_u - x_v \right\|_1 \\ & = \frac{3}{2}.\frac{1}{2} \sum_{e=(u,v) \in E} c_e \left\| x_u - x_v \right\|_1 \\ & \leq \frac{3}{2} OPT. \end{aligned}$$

■

We note that randomized rounding of linear programs is a powerful technique to design approximation algorithms for cut problems. It has been extensively applied to numerous variants of the graph partitioning problem.

The next subsection provides an introduction to semidefinite programming (SDP) in the context of the Maximum-Cut Problem. The results presented are due to Goemans et al. [10].

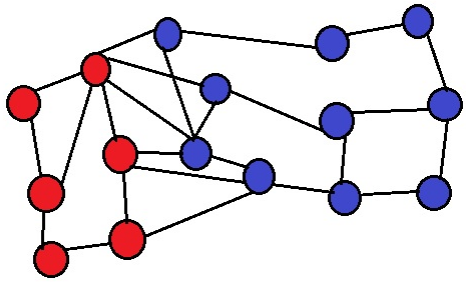

Figure 3.2: The max cut problem: The goal is to find a set of red and blue vertices whose union is V, such that the weight of the edges crossing between them is maximized. In this special case, we have all edges of unit cost, so the problem reduces to finding two sets between which there is a maximum number of edges.

3.2 The Maximum Cut Problem

Problem 2 *Given an undirected graph $G = (V, E)$, and a cost function $c : E \to \mathbb{R}^+$, partition V into 2 parts $S, \bar{S}$ such that the total cost of edges between those parts is* ***maximized****.*

See Figure 3.2 where a graph is partitioned into a set of red and blue vertices. To solve the Max-Cut problem, the best known algorithm [10] uses semidefinite programming.

Semidefinite programming is similar to linear programming in a lot of ways. For example, the objectives and constraints are linear in the variables used. We now present a few well known facts in linear algebra which SDP exploits.

Definition 3.3 (Positive semidefinite (psd) matrices) *A square matrix $X \in \mathbb{R}^{n \times n}$ is a positive semidefinite matrix if for all $y \in \mathbb{R}^n, y^T X y \geq 0$.*

Theorem 3.4 *Let $X \in \mathbb{R}^{n \times n}$ be a symmetric matrix. Then the following statements are equivalent:*
1) X is a psd matrix.
2) X has non negative eigenvalues.
3) $X = V^T V$, where $V \in \mathbb{R}^{m \times n}, m \leq n$
4) $X = \sum_{i=1}^{n} \lambda_i w_i w_i^T$ for some $\lambda_i \geq 0$ and orthonormal vectors $w_1, ..., w_n$.

The main difference between SDP's and LP's are that the set of variables in the constraints can be constrained to a positive semidefinite matrix.

The following is a canonical example of a semidefinite program:

SDP:

$$\begin{aligned} maximize/minimize & \sum_{i,j} c_{ij}x_{ij} \\ subject\ to & \sum_{i,j} a_{ij}x_{ij} = b_k \quad \forall k \\ & x_{ij} = x_{ji} \quad \forall i,j \\ & X = (x_{ij}) \succeq 0. \end{aligned}$$

From the third statement of Theorem 3.4, it follows that the entries of X can be viewed as dot products of some vectors. These vectors can be found via standard approaches using an eigenvalue computation or the LU decomposition when X is sparse. Consequently, a semidefinite program is linear in the variables used; these variables x_{ij} are inner products of some vectors v_i, v_j.

We first design an alternative formulation for the Maximum Cut problem, which is then relaxed into a semidefinite program. For each vertex i we associate a variable y_i such that $y_i = 1$ if $i \in S$ and -1 if $i \in \bar{S}$.

$$\begin{aligned} maximize & \quad \frac{1}{2}\sum_{(i,j)\in E} c_e(1 - y_i y_j) \\ subject\ to & \quad y_i \in \{-1, +1\} \quad \forall i = 1, 2, ..., n \end{aligned} \tag{3.2}$$

Theorem 3.5 *Program* (3.2) *models the Max-Cut problem.*

Proof: Consider the cut $(U, \bar{U})$ where $U = \{i | y_i = 1\}$ and $\bar{U} = \{i | y_i = -1\}$. Then for any edge (i, j) participating in the cut, we have $1 - y_i y_j = 2$ and $1 - y_i y_j = 0$ when both endpoints of (i, j) lie in one of U or $\bar{U}$. The factor $1/2$ prevents double counting of any edge. Maximizing the sum of the quantity $\frac{1}{2}c_e(1 - y_i y_j)$ over all edges $e = (i, j)$ then gives the maximum weighted sum of cut edges which by definition, is equal to the Maximum-Cut. ■

In the corresponding SDP relaxation, we have a vector v_i corresponding to all vertices $i \in V$. The variable y_i is essentially replaced by v_i.

SDP relaxation for Max-Cut:

$$\begin{aligned} maximize & \quad \frac{1}{2}\sum_{(i,j)\in E} c_e(1 - v_i.v_j) \\ subject\,to & \quad v_i.v_i = 1 \quad \forall i = 1, 2, ..., n. \end{aligned}$$

Theorem 3.6 *The above is a valid semidefinite relaxation of the Maximum-Cut problem.*

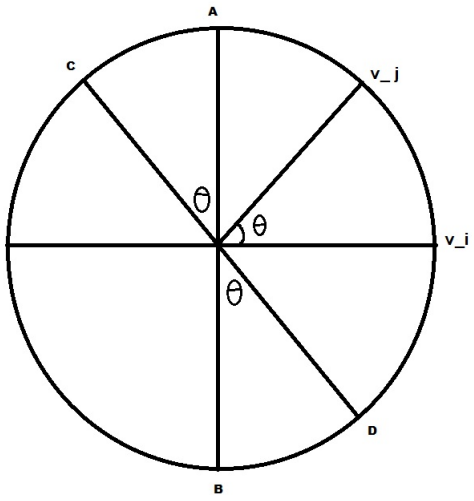

Figure 3.3: Randomized Rounding: AB denotes the line perpendicular to v_i and CD denotes the line perpendicular to v_j. The angle between v_i, v_j is θ.

Proof: First note that given any feasible solution y_i for the i^{th} vertex in V, one can set $v_i = (y_i, 0, ...0)$ and then, it follows that v_i is a unit vector and $v_i.v_j = y_i.y_j$. Note that if (i, j) participates in the cut $(S, \bar{S})$, then $1 - y_i.y_j = 2$. Hence in the objective function only the cost of the cut edges is accounted for. The factor $1/2$ prevents double counting of any edge as before. As a result, the SDP relaxation gives a valid formulation for the Max-Cut problem. Moreover, if VP, OPT denote the values of the SDP relaxation and the optimal value of Max-Cut respectively, then $VP \geq OPT$.

■

3.2.1 Randomized Hyperplane Rounding and Analysis

We now discuss a breakthrough technique due to Goemans et al. [10] to round the corresponding vector solution obtained by the above SDP relaxation to an integral value for each vertex. In other words, we want to transform an arbitrary unit vector in $\mathbb{R}^n$ while ensuring that the value of the solution obtained by the SDP relaxation doesn't change by much.
The rounding procedure is as follows:

Randomized Rounding for Max-Cut:

1) Choose a random vector r in $\mathbb{R}^n$, whose each component is drawn independently from a normal distribution with mean 0 and variance 1.
2) Compute the dot-product, $r.v_i$ for each v_i obtained via the above SDP relaxation.
3) If $r.v_i \geq 0$, set $y_i = 1$ otherwise set y_i to -1.

Theorem 3.7 *The randomized rounding procedure yields a 0.878-approximation for the Max-Cut problem, i.e. it finds a partition of V such that the cost of edges cut is at least $0.878.OPT$, where OPT is the value of the optimal solution.*

We first prove the following lemma which will lead to our theorem.

Lemma 3.1 *The probability that any edge (i, j) is in the cut is $\frac{1}{\pi} arccos(v_i.v_j)$.*

Proof: Consider a random vector r which is projected in the 2 dimensional plane spanned by v_i, v_j. Note that all vectors v_i lie in the unit sphere in $\mathbb{R}^n$. Let r_p denote the projected component of r. Then if $r = r_p + r'$, then r' is orthogonal to both v_i and v_j. Essentially one only needs to be concerned with the projected component r_p and consider the inner product with v_i, v_j since $r_p.v_i = r.v_i$, and similarly for v_j.

Now, refer to figure 3.3. Note that edge (i, j) is cut iff $r_p.v_i$ and $r_p.v_j$ have opposite signs. Let θ denote the angle between them and let AB, CD denote the perpendicular lines to v_i and v_j. Then, note that only if r_p lies in the sector BD or AB does the sign of 2 inner products $r_p.v_i$, $r_p.v_j$ is different. Since r is chosen uniformly at random, the probability that r_p lies in those sectors is $\frac{2\theta}{2\pi}$. Since $\theta = arccos(v_i.v_j)$, we get that $Pr[(i, j) \in (S, \bar{(S)})] = \frac{1}{\pi} arccos(v_i.v_j)$. ■

Denote by X_e the indicator random variable which is 1 if e is in the cut and 0 otherwise. Using the fact that $\frac{1}{\pi} arccos(x) \geq 0.878.\frac{1}{2}(1 - x)$, we have:

$$\begin{aligned} E[\sum_{e=(i,j)\in E} c_e X_e] &= \sum_{e=(i,j)\in E} c_e.Pr[X_e = 1] \\ &= \sum_{e=(i,j)\in E} c_e \frac{1}{\pi} arccos(v_i.v_j) \\ &\geq \sum_{e=(i,j)\in E} c_e.0.878.\frac{1}{2}(1 - v_i.v_j) \\ &= 0.878.VP \\ &\geq 0.878.OPT. \end{aligned}$$

The proof is now complete.

In the following section, we present a breakthrough result due to Charikar and Wirth [6], which exploits randomized rounding and concentration of measure to get a better approximation algorithm for the general quadratic programming problem.

3.3 Quadratic Programming

In this section, an $\Omega(1/\log n)$ approximation algorithm is presented, which generalizes Nesterov's [20] algorithm having an approximation ratio of $\frac{2}{\pi}$. In the latter result, the matrix was constrained to be positive semidefinite. We relax this condition and only require the diagonal entries of the matrix to be 0. Let us first define the quadratic programming problem, which we call IP.
IP:

$$\begin{aligned} &\text{maximize} \sum_{1 \le i,j \le n} a_{ij} x_i x_j \\ &\text{subject to } x_i \in \{-1, +1\} \end{aligned}$$

Why is a positive objective required? Note that in the general case if all diagonal entries are not necessarily zero, then the objective could be negative, and hence the notion of an *approximation* would not hold. Consider the case when all diagonal entries are negative and all non-diagonal entries are zero-then, since $x_i^2 = 1 \ \forall i$ it follows that the objective is negative. An α approximation algorithm ALG for a maximization problem implies that $\text{cost}(ALG) \ge \alpha OPT$, but in the case when $\text{cost}(ALG) < 0$, this implies that $\text{cost}(ALG) > OPT$ for $\alpha > 0$, clearly a contradiction.

Hence, due to the above argument, we require the matrix $A = (a_{ij})$ to have zero-valued diagonal entries, which results in the final objective being only changed by a constant.

Lemma 3.2 *If $a_{ii} = 0$ for all i, then $OPT \ge \frac{1}{n^2} \sum_{1 \le i,j \le n} |a_{ij} + a_{ji}|$. This implies that the maximum value of the objective is always positive.*

Proof: We construct a randomized solution. Let G be a complete graph on n vertices, with edge weights $w_{ij} = a_{ij} + a_{ji}$ for all $i, j \in \{1, 2, ..., n\}$. Then choose a matching M as follows. Pick an arbitrary edge (i, j), remove its endpoints and all edges incident to both i and j in the graph and continue until the graph contains at most one vertex-this happens when n is odd. Initially, the probability that edge (i, j) is chosen is equal to $\frac{2}{n(n-1)} \ge \frac{1}{n^2}$. This is a weak lower bound, but suffices for the proof. Given the matching M, the solution is constructed as follows: for each $(i, j) \in M$, set $x_i = 1$ with probabilty $1/2$ and $x_i = -1$ with probability $1/2$, set $x_j = x_i$ if $a_{ij} + a_{ji} < 0$ and $x_j = -x_i$ otherwise.

Note that if $(i, j) \in M$, then $E[(a_{ij} + a_{ji})x_i x_j] = |a_{ij} + a_{ji}|$ and 0 otherwise. Hence,

$$
\begin{aligned}
E[\sum_{1\le i,j\le n} a_{ij}x_ix_j] &= E[\sum_{1\le i,j\le n} (a_{ij}+a_{ji})x_ix_j] \\
&= \sum_{1\le i,j\le n} Pr[(i,j)\in M]E[(a_{ij}+a_{ji})x_ix_j|(i,j)\in M] \\
&+ \sum_{1\le i,j\le n} Pr[(i,j)\notin M]E[(a_{ij}+a_{ji})x_ix_j|(i,j)\notin M] \\
&\ge \frac{1}{n^2}|a_{ij}+a_{ji}|
\end{aligned}
$$

■

Since solving integer linear programs is NP=complete, we relax the constraints $x_i \in \{-1, 1\}$ into linear constraints $-1 \le y_i \le 1$, yielding a linear program, LIN.
LIN:

$$
\begin{aligned}
&\text{maximize} \sum_{1\le i,j\le n} a_{ij}y_iy_j \\
&\text{subject to } -1 \le y_i \le 1
\end{aligned}
$$

The claim is that given a solution to this LIN, one can find an integer solution with the same value.

Theorem 3.8 *Given an α-approximation algorithm for LIN, we can obtain a randomized α approximation algorithm for IP with integer constraints.*

Proof: Let $\bar{y}$ be any solution to the LIN program. Using randomized rounding, set $\bar{x_i} = 1$ with probability $\frac{1+\bar{y_i}}{2}$ and -1 with probability $\frac{1-\bar{y_i}}{2}$. Then, whenever $i \ne j$,

$$
\begin{aligned}
E[\bar{x_i}\bar{x_j}] &= Pr[\bar{x_i} = \bar{y_j}] - Pr[\bar{x_i} \ne \bar{x_j}] \\
&= \frac{1}{4}(2 + 2\bar{y_i}\bar{y_j}) - \frac{1}{4}(2 - 2\bar{y_i}\bar{y_j}) \\
&= \bar{y_i}\bar{y_j}
\end{aligned}
$$

This completes the proof. ■

Note that $OPT(LIN) \ge OPT$ since LIN maximizes over a larger set of constraints. Moreover, $OPT \ge OPT(LIN)$, by 3.8, so that $OPT(LIN) = OPT$. Therefore, solving LIN, the linear program shall yield us a solution to IP whose expected value is equal to OPT, the value of the optimal solution to the standard quadratic programming problem.

Given these results, we will be able to give a $\Omega(\frac{1}{\log n})$ approximation algorithm. Let OPT now denote $OPT(LIN)$, and $OPT(VP)$ denote the optimal value of VP given as follows:
VP:

$$\begin{aligned} \text{maximize} \quad & \sum_{1 \le i,j \le n} a_{ij}(v_i.v_j) \\ \text{subject to} \quad & v_i.v_i = 1, \quad \forall i \in \{1,..,n\} \\ & v_i \in \mathbb{R}^n \quad \forall i \in \{1,...,n\} \end{aligned}$$

The algorithm is as follows:
On a top-level, the algorithm chooses a random hyperplane, and computers computes z_i for

Algorithm 3 Quadratic Programming

```
Solve VP and obtain the vectors v_i
Draw a random vector r, s.t. r_i ~ N(0,1)
Set z_i = (v_i.r)/T
if |z_i| ≤ 1 then
    y_i = z_i
else if z_i < −1 then
    y_i = −1
else
    y_i = 1
return y
```

all i. The rounding procedure is different from Max-Cut in that the product $v_i.r$ is scaled down by T^2. This ensures that the probability that $|z_i| > 1$ is low, hence, allowing us to relate the value of OPT to $OPT(VP)$.

Lemma 3.3 $E[z_i z_j] = \frac{1}{T^2}(v_i.v_j)$

Proof: $E[z_i z_j] = \frac{1}{T^2}E[(v_i.r)(v_j.r)]$. We can consider the 2-dimensional plane spanned by v_i and v_j, where Wlog, $v_i = (1, 0, ..., 0)$, $v_j = (a, b, 0, ...0)$, since the solution is not affected by a rotation of the vectors and that r is uniformly distributed on the n dimensional unit sphere. Then, $v_i.r = r_1$, $v_j = ar_1 + br_2$, so that

$E[z_i z_j] = \frac{1}{T^2}E[ar_1^2 + br_1 r_2] = \frac{1}{T^2}aE[r_1^2] + bE[r_1 r_2]$

Now, $E[r_1^2] = 1$, since each component of r has a variance of 1, and $E[r_1 r_2] = 0$, since each component of r is drawn independently at random. Thus $E[z_i z_j] = \frac{1}{T^2}a = \frac{1}{T^2}(v_i.v_j)$. This concludes the proof. ■

From the above lemma, it follows that $E[\sum_{1 \le i,j \le n} a_{ij} z_i z_j] = \frac{1}{T^2}\sum_{1 \le i,j \le n} a_{ij}(v_i.v_j) = \frac{1}{T^2}OPT(VP)$. Furthermore, $E[\sum_{1 \le i,j \le n} a_{ij} y_i y_j] = \sum_{1 \le i,j \le n} a_{ij} E[y_i y_j]$. To relate the values of the 2 programs, we need to bound the quantity $|E[\Delta_{ij}]|$. This will help us bound the cost of the

solution y.

Theorem 3.9 (Core result on the error bound)
$|E[\Delta_{ij}]| \leq 8e^{-T^2}$, *where* $\Delta_{ij} = z_i z_j - y_i y_j$.

Proof: Let X_i denote the event that $y_i = z_i$. Let E_i be the expectation conditioned on the event X_i, and $E_{\neg i}$ as the expectation conditioned on $\bar{X}_i$. So for example $E_{\neg i,j}$ is the expectation conditioned on the event $\bar{X}_i \wedge X_j$. Then,

$$\begin{aligned}|E[\Delta_{ij}]| \leq |E_{i,j}[\Delta_{i,j}]Pr[X_i \wedge X_j]| + E_{\neg i,j}[\Delta_{i,j}]Pr[\bar{X}_i \wedge X_j] \\ + E_{i,\neg j}[\Delta_{i,j}]Pr[X_i \wedge \bar{X}_j] + E_{\wedge i,\wedge j}[\Delta_{i,j}]Pr[\bar{X}_i \wedge \bar{X}_j]\end{aligned}$$

We observe that

$$E_{\neg i}[|\Delta_{ij}|]Pr[\bar{X}_i] = E_{\neg i,j}[\Delta_{i,j}]Pr[\bar{X}_i \wedge X_j] + E_{\neg i,\neg j}[\Delta_{i,j}]Pr[\bar{X}_i \wedge \bar{X}_j].$$

This holds analogously for $E_{\neg j}[|\Delta_{ij}|]Pr[\bar{X}_j]$ by symmetry.

Thus the RHS of the above inequality is bounded by:

$|E_{i,j}[\Delta_{i,j}]Pr[X_i \wedge X_j]| + E_{\neg i}[|\Delta_{ij}|]Pr[\bar{X}_i] + E_{\neg j}[|\Delta_{ij}|]Pr[\bar{X}_j]$

For the sake of brevity, the remainder of the proof is omitted which involves upper-bounding the 3 terms above. The first term is simply 0, while the technique for bounding the rest of the 2 terms is symmetric. ■

Theorem 3.10 (Proof of approximation guarantee) *For large n, Algorithm 2 is a randomized $\Omega(1/log\, n)$ approximation algorithm for the quadratic programming problem.*

Proof: We have

$$\begin{aligned}E[\sum_{1\leq i,j\leq n} a_{ij}y_i y_j] &= \sum_{1\leq i,j\leq n} a_{ij}E[y_i y_j] \\ &= \sum_{1\leq i,j\leq n} a_{ij}E[z_i z_j] - \sum_{1\leq i,j\leq n} a_{ij}E[\Delta_{ij}] \\ &= \frac{1}{T^2}OPT(VP) - \sum_{1\leq i,j\leq n} a_{ij}E[\Delta_{ij}] \\ &\geq \frac{1}{T^2}OPT(VP) - |\sum_{1\leq i,j\leq n} a_{ij}E[\Delta_{ij}]| \\ &\geq \frac{1}{T^2}OPT(VP) - \sum_{1\leq i<j\leq n} |a_{ij}+a_{ji}|.|E[\Delta_{ij}]| \\ &\geq \frac{1}{T^2}OPT(VP) - 8e^{-T^2}\sum_{1\leq i<j\leq n} |a_{ij}+a_{ji}|\end{aligned}$$

Note that $\sum_{1\leq i<j\leq n}|a_{ij}+a_{ji}| \leq n^2.OPT$, so that the latter quantity is $\geq (\frac{1}{T^2}-8n^2e^{-T^2})OPT$. If $T=\sqrt{3ln\,n}$, a $\Omega(1/\log n)$ approximation results. ■

3.4 Coloring 3-colorable Graphs

Graph coloring lies at the heart of the hardest problems in the complexity class NP, i.e. it is NP-complete implying that any problem in NP can be reduced in polynomial time to the graph-coloring problem. We define the graph coloring problem as follows:

Problem 3 (Graph-coloring Problem) *Given an undirected graph $G=(V,E)onn=|V|$ vertices, assign a color to each vertex such that for all adjacent vertices in G, i.e. for any u,v st $(u,v)\in E$, the assignment of the colors to u and v is different. This assignment must be done in time polynomial in n and $m=|E|$.*

Definition 3.4 (3-colorable graphs) *In general a graph is n-colorable if the minimum number of colors required to color it is n. A 3-colorable graph is one for which exactly 3 colors suffice to color it.*

NP-completeness of 3-coloring The problem of determining a valid assignment to a 3-colorable graph is NP-complete. However, given an assignment of colors to the vertices it takes polynomial time to determine whether the assignment is feasible.

The goal of an approximation algorithm in this case is to find a feasible assignment to the vertices while minimizing the total number of colors used. As remarked earlier, finding an assignment using only 3 colors is hard in general. In this section, we present a $\tilde{O}(n^{0.387})$ approximation algorithm for the 3-coloring problem. A trivial coloring algorithm uses n colors-to color the n vertices with a different color. The algorithm in this section uses a vector-programming relaxation combined with a randomized hyperplane rounding technique as used in the previous subsections.

3.4.1 A combinatorial $O(\sqrt{n})$ approximation algorithm

In this section, we present a simple and elegant combinatorial algorithm due to Wigderson [26]. It works as follows: While the graph contains a vertex u of degree at most $\sqrt{n}$, we use 3 colors to color u and its neighbors. This can be done by assigning u a single color and coloring the neighbors using a polynomial time 2-coloring algorithm. Then, u and its neighbours are removed and the process continues until we have a vertex having maximum degree less than $\sqrt{n}$. Subsequently, we use $\sqrt{n}$ colors to color the remaining graph.

Lemma 3.4 *The algorithm uses $O(\sqrt{n})$ colors to color a 3-colorable graph.*

Proof: The algorithm uses at most $4\sqrt{n}$ colors. Note that each execution of the while loop takes uses 3 colors and is executed at most $\frac{n}{\sqrt{n}}=\sqrt{n}$ times. In the second step in which the maximum degree of any vertex in the modified graph is less than $\sqrt{n}$, at most $\sqrt{n}$ colors are used, yielding a total of at most $O(\sqrt{n})$ colors. ■

3.4.2 A better algorithm using SDP

We now present a better approximation algorithm which uses semidefinite programming. Let VP denote the vector program. Since the underlying assumption is that the graph requires a maximum of 3 colors in order to color it, an equilateral triangle in an n dimensional space with sides of length $\sqrt{3}$ whose vertices correspond to distinct colors is considered. A vector v_i corresponding to each vertex i of G is assigned a color corresponding to a vertex of the triangle. The goal is then to assign vectors v_i, v_j to distinct vertices i, j of the triangle whenever $(i, j) \in E$. We have the following program.

VP:

$$\begin{aligned} \min \ & \lambda \\ \text{st} \ & v_i.v_j \leq \lambda \ \ \forall (i,j) \in E \\ & v_i.v_i = 1 \ \ \forall i \in V \\ & v_i \in \mathbb{R}^n \ \forall i \in V \end{aligned}$$

Theorem 3.11 (Semicoloring) *A semicoloring of the vertices of G such that at most $n/4$ edges have endpoints of the same color. This implies that at least $n/2$ vertices are colored such that any edge between them have endpoints that are colored differently.*

Proof: Let the current graph be G, and let E' denote the set of edges sharing endpoints with the same color. Then, at most $n/2$ vertices are incident to edges in E'. Hence, there are at least $n/2$ vertices st any edge between them has endpoints that are colored *differently*. If a graph can be semicolored using k colors, then only $k\log(n)$ colors suffice to color the whole graph-since induced graph containing edges having endpoints colored by the same color can be semicolored recursively and removed from the graph to leave the part of the graph containing edges incident to vertices colored similarly. ■

In contrast to the Max-Cut problem in which one random hyperplane was chosen, a total of $t = 2 + \log_3\Delta$ (where Δ is the maximum degree of any vertex in the graph) hyperplanes $r_1, ..r_t$ are chosen; for each hyperplane, the components are chosen independently at random from $N(0, 1)$. We bound the probability in the case that v_i, v_j are not separated by any of the t hyperplanes when $(i, j) \in E$.

The t hyperplanes partition $\mathbb{R}^n$ into 2^t pieces. In the case of Max-Cut, where $t = 1$, the probability that v_i, v_j were separated was $\frac{1}{\pi}\arccos(v_i.v_j)$. Thus the probability that v_i and v_j are not separated (assigned the same color) is $1 - \frac{1}{\pi}\arccos(v_i.v_j)$. From the independence assumption,

$$Pr[i, j \text{ are assigned the same color}] = (1 - \frac{1}{\pi}\arccos(v_i.v_j))^t$$

Lemma 3.5 *If G is 3 colourable, then any feasible solution to **VP** satisfies $\lambda \leq 1/2$.*

Proof: Note that whenever $(i, j) \in E$, then $\cos(v_i.v_j) = -\frac{1}{2}$, since the angle between v_i, v_j is $2\pi/3$, whenever i and j are assigned different colors. Hence, in the optimal solution, $\lambda \leq -\frac{1}{2}$ ■

It follows that,

$$\begin{aligned} Pr[i,j \text{ are assigned the same color}] &= (1 - \frac{1}{\pi}\arccos(v_i.v_j))^t \\ &\leq (1 - \frac{1}{\pi}\arccos(\lambda))^t \\ &\leq (1 - \frac{1}{\pi}\arccos(-\frac{1}{2}))^t \\ &= (1 - \frac{1}{\pi}\frac{2\pi}{3})^t = (\frac{1}{3})^t \\ &= \frac{1}{3^2 3^{\log_3^\Delta}} = \frac{1}{9\Delta} \end{aligned}$$

Thus $Pr[i,j$ are assigned the same color$] \leq \frac{1}{9\Delta}$. Let $\mathbf{1}_{(u,v)}$ be the indicator random variable s.t. $\mathbf{1}_{(u,v)} = 1$ if (u,v) are assigned the same color and 0 otherwise. Let X be the random variable representing the number of edges whose endpoints are assigned the same color. Then,

$$E[X] = E[\textstyle\sum_{(u,v)\in E} \mathbf{1}_{(u,v)}] = mPr[\mathbf{1}_{(u,v)}] = \frac{m}{9\Delta} \leq \frac{n}{18}.$$

The last inequality follows from the fact that $m \leq \frac{n\Delta}{2}$.

Applying Markov's inequality, we have that

$$Pr[X \geq \frac{n}{4}] \leq \frac{E[X]}{n/4} \leq \frac{n/18}{n/4} \leq \frac{1}{2}$$

Lemma 3.6 *The algorithm uses $2^t = 4.2^{log_3\Delta} = 4\Delta^{log_3 2}$ colors to semicolor a graph with probability at least 1/2.*

If n is used as a bound on Δ, then we get a coloring of G that takes $O(n^{\log_3 2}.\log n) = \tilde{O}(n^{\log_3 2})$[4] in total.

However, this is worse than Wigderson's algorithm since $\log_3 2 \approx 0.631$. Using some ideas from Wigderson's algorithm, however, this guarantee can be improved to $\tilde{O}(n^{0.387})$.

Theorem 3.12 *There exists an algorithm which colors G using a total of $\tilde{O}(n^{0.387})$ colors.*

Proof: The algorithm works as follows: First, it chooses a parameter σ. Until there exists a vertex in the graph with degree σ, it chooses 3 colors to color a vertex and its neighbors having degree at least σ in the current graph. Subsequently, the vertices and its neighbors are removed and the procedure is repeated until there is no vertex with degree at least σ. After this step, a semicoloring algorithm is used to semicolor the remaining graph.

[4]The notation $\tilde{O}(.)$ hides polylogartihmic factors in n

Analysis: The first step uses at most $\frac{3n}{\sigma}$ colors while the second step uses $O(\sigma^{\log_3 2})$ colors. We set σ such that $\frac{n}{\sigma} = \sigma^{\log_3 2}$ in order to balance the colors used in each part. This yields $\sigma = n^{0.613}$. Then, both parts use $O(n^{0.387})$ colors. Thus, the overall algorithm takes at most $\tilde{O}(n^{0.387})$ colors. ■

Using more sophisticated ideas, it is possible to improve this result to color a 3-colorable graph with $\tilde{O}(\Delta^{1/3}\sqrt{ln\,\Delta})$ colors. We omit these results for the sake of brevity.

In the following section, we discus some notions which are cruicial to understanding bresome recent graph partitioning algorithms. They are essential prerequisites to understaning the ARV algorithm [3] and the Leighton-Rao algorithm for the uniform sparsest cut problem [16].

4 Metric Embeddings

4.1 Basic concepts

So far, we have focused on a variety of algorithms which exploit linear and semidefinite programming to give better approximation for combinatorial optimization problems. While some detail on l_1 metrics was given in the section on minimum multiway cut, general metrics were not studied. In the past few decades, low distortion metric embeddings have become a powerful tool in designing approximation algorithms. Two well known algorithms which we have studied and empirically evaluated-one due to Leighton and Rao [18], and the other due to Arora, Rao and Vazirani [3] utilize concepts in metric embeddings to give $O(log\, n)$, and $O(\sqrt{log\, n})$ approximations respectively for the sparsest cut problem.

We discuss the theory of metric embeddings briefly, which will allow us to formalize the discussion of the sparsest cut problem. The main goal of this section is to present Bourgain's theorem which concerns embedding of an arbitrary n-point metric into l_1. This greatly simplifies the analysis of the Leighton-Rao algorithm.

Definition 4.1 (General metrics) *A metric d defined on set V is a function $V \times V \rightarrow \mathbb{R}^+ \bigcup \{0\}$ such that:*

1. *(Non-negativity) $d(u,v) \geq 0$, for all $u, v \in V$.*
2. *(Symmetry) $d(u,v) = d(v,u)$ for all $u, v \in V$.*
3. *(Triangle inequality) $d(u,w) + d(w,v) \geq d(u,v)$ for all $u, v, w \in V$.*

A metric space is a set for which the distances between any 2 elements is well defined. A well-known metric space is the 3 dimensional Euclidean space, which is endowed with the metric d such that $d(u,v) = (\sum_{i=1}^{3} |u_i - v_i|^2)^{1/2}$, where u_i denotes the i^{th} component of u. A metric generalizes the notion of a distance in an arbitrary space. A semi-metric is one which does not necessarily satisfy the third condition, i.e. the triangle inequality.

Definition 4.2 (Cut-Metric) *Let V be a finite set. A cut metric, induced by $S \subseteq V$ where $S \neq \emptyset$ is defined as follows: i) $d(x,y) = 0$ whenever $x, y \in S$ and ii) $d(x,y) = 1$ whenever either x or y is in S but not both.*

Definition 4.3 (Shortest path metric) *Let $G = (V, E)$ be an undirected graph with edge weights(lengths) $w(u,v) \geq 0$ defined for all $e = (u,v) \in E$. The shortest path metric $d : V \times V \rightarrow \mathbb{R}$ is defined as follows: $d(u,v) = min_P \sum_{(i,j) \in P} w(i,j)$, where P is the set of all paths from u to v in G.*

Definition 4.4 (l_1 embeddable metrics[27]) *[27] A metric (V,d) is an l_1 embeddable metric (or embeds isometrically into l_1 if there exists a function $f : V \rightarrow \mathbb{R}^m$ for some m such that $d_{uv} = \|f(u) - f(v)\|_1$.*

Definition 4.5 (Distortion of an embedding [27]) *A metric (V, d) embeds into l_p, for $p \geq 1$ with distortion α if there exists a function $f : V \to \mathbb{R}^m$, for some m and r such that:*

$$r.d_{uv} \leq \|f(u) - f(v)\|_p \leq r\alpha.d_{uv}$$

Informally, in cut-based problems such as graph-partitioning defining low distortion embeddings allows us to solve the problem on a simplified space. The quality of this solution on the transformed metric can then be related to the quality on the original metric; this depends crucially on the the distortion of the embedding, α.

We now prove a key theorem about the embeddability of l_1 metrics which are intimately related to the cut-metrics.

Theorem 4.1 *Let (V, d) be an l_1 embeddable metric and let $f : V \to \mathbb{R}^m$ be the associated embedding. Then this metric can be expressed as a weighted sum of cuts of the vertex set V. The procedure takes polynomial time.*

Proof: Let $\Gamma_{\delta(S)}(u, v)$ be an indicator function which is 1 if (u, v) is in the cut defined by $(S, \bar{S})$ and 0 otherwise. We precisely show that there exist $S \subseteq V$ and $\mu_S \geq 0$ such that for all $u, v \in V$ the following holds:

$$\|f(u) - f(v)\|_1 = \sum_{S \subseteq V} \mu_S \Gamma_{\delta(S)}(u, v)$$

Consider the restricted case when f embeds V into $\mathbb{R}$, i.e. in one dimension. Let $V = \{1, 2, .., n\}$, so that $x_i = f(i) \in \mathbb{R}$, and assume Wlog that $x_1, x_2, ..., x_n$ is a non-decreasing sequence. Then, consider the cuts $\{1\}, \{1, 2\}, \{1, 2, 3\}, ..., \{1, 2, ..., n-1\}$, and define $\mu_{\{1,2,..,i\}} = x_{i+1} - x_i$ for $i \in [n-1]$. For any $i, j \in V$ where $i < j$,

$$|x_i - x_j| = x_j - x_i = \sum_{k=i}^{j-1} \mu_{\{1,..,k\}} = \sum_{S \subseteq V} \mu_S \Gamma_{\delta(S)}(i, j).$$

For general m we use the same procedure as above for each component of $f(x_i)$, i.e. sort the the points in increasing order of magnitude in a specific component and then considering the cuts. Then,

$$\|f(u) - f(v)\|_1 = \sum_{i=1}^{m} |x_u^i - x_v^i| = \sum_{i=1}^{m} \sum_{S \subseteq V} \mu_S^i \Gamma_{\delta(S)}(u, v) = \sum_{S \subseteq V} \mu_S \Gamma_{\delta(S)}(u, v)$$

■

It is important to note than not every metric is l_1 embeddable. Bourgain's theorem shows that the expected distortion incurred by embedding an arbitrary n point metric into l_1 is $O(\log n)$. Most of the presentation in the rest of the section is based on [27].

4.2 Bourgain's theorem: embeddability in l_1

Theorem 4.2 (Bourgain (1985)) *Let (X, d) be an arbitrary metric space on n points. Then, there exists a mapping $f : X \to \mathbb{R}^m$, where $m = O(log^2 n)$ such that the distortion of the embedding is $O(log n)$.*

We prove a generalized version of the theorem where there are k distinguished terminals given by s_i, t_i and an embedding f is constructed where $\|f(u) - f(v)\|_1 \leq O(\log^2 k) d_{uv}$ for all $u, v \in V$ and $\|f(s_i) - f(t_i)\|_1 \geq \Omega(\log k) d_{s_i t_i}$.

To see why this is a generalization, note that all $\binom{n}{2}$ pairs can be taken as s_i, t_i. Then, the generalized version reduces to 4.2.

One direction of this result can be easily proved using the concept of Fréchet embeddings, which are defined as follows.

Definition 4.6 (Fréchet Embedding) *Given a metric space (V, d) and t subsets of V given by $A_1, A_2, ..., A_D$, a Fréchet embedding $f : V \to \mathbb{R}^D$ is defined by*

$$f(v) = (d(v, A_1), d(v, A_2), ..., d(v, A_D)), \text{ where } d(v, A_i) = \min_{u \in A_i} d(v, u), \forall v \in V.$$

A nice property of the Fréchet embedding is that the l_1 distances don't get 'stretched' to much with respect to the original distance between any two points in V.

Theorem 4.3 (Upper bound on l_1 distance) *Given a metric space (V, d) and a Frèchet embedding $f : V \to \mathbb{R}^D$ the following holds:*

$$\|f(u) - f(v)\|_1 \leq D.d_{uv} \ \ \forall u, v \in V$$

Proof: We claim that for any i, $|d(u, A_i) - d(v, A_i)| \leq d_{uv}$. Then, it immediately follows that

$$\|f(u) - f(v)\|_1 = \textstyle\sum_{i=1}^{D} |d(u, A_i) - d(v, A_i)| \leq D.d_{uv}.$$

To see why the claim holds let p denote the nearest point to v in A_i. Then, $d(u, A_i) \leq d(u, p) \leq d(u, v) + d(v, p) = d(u, v) + d(v, A_i)$. Thus $d(u, A_i) - d(v, A_i) \leq d(u, v)$. Symmetrically, $d(v, A_i) - d(u, A_i) \leq d(u, v)$ too, from which the claim follows. ∎

Corollary 4.1 (Easier direction of generalized 4.2) *By picking $O(log^2 k)$ subsets A_i from V, we have that for any $u, v \in V$, $\|f(u) - f(v)\|_1 \leq O(log^2 k)\, d_{uv}$.*

What remains to prove now is that for any s_i, t_i, we have that $\|f(s_i) - f(t_i)\| \geq \Omega(\log k)\, d_{s_i t_i}$. To this end, we will pick $O(log^2 k)$ by randomly choosing elements from V and apply Chernoff bounds to obtain the result.

A constructive proof sketch of 4.2:
Note that we have k pairs of terminals s_i, t_i. Let $T = \bigcup_i = 1^k\{s_i, t_i\}$. Let δ denote the closest power of 2 to $2k$ such that $\delta = O(\log(\mathrm{k}))$. For $l = 1, 2, ..., L = O(log\, k)$ and $t = 1, 2, ..., \delta$, we pick sets A_{tl} by choosing $2^{\delta - t}$ vertices randomly from T with replacement. The goal is to show that for these $O(\log^2 k)$ sets, the Fréchet embedding has the desired properties.

The closed ball around $v \in V$ is defined as $B(v, r) = \{u \in T | d_{uv} \leq r\}$, and the open ball as $B^o(v, r) = \{u \in T | d_{uv} < r\}$. Then define $r_0 = 0$ and r_t be the minimum distance such that $|B(s_i, r)| \geq 2^t$ and $|B(t_i, r)| \geq 2^t$ for all $t = 1, ..., \delta$. Also we define μ to be the minimum index such that $r_\mu \geq \frac{1}{4} d_{s_i t_i}$, and redefine $r_\mu = \frac{1}{4} d_{s_i t_i}$. Note that $B(s_i, r_\mu) \cap B(t_i, r_\mu) = \emptyset$. Furthermore if $\mu = \delta$, then from the fact that $|B(s_i, r_{\mu-1})|, |B(t_i, r_{\mu-1})| \geq 2^{\mu-1}$ we have that for r_μ (which was redefined earlier to $\frac{1}{4} d_{s_i t_i}$), $|B(s_i, r_\mu)| = |B(t_i, r_\mu)| = 2^{\delta-1}$.

Using standard probability arguments, it can be shown that for any $l = 1, 2, ..., L$ and any $t = 1, ..., \mu$, A_{tl} has a constant probability of having an intersection with $B(s_i, r_{t-1})$ and having no intersection with $B(s_i, r_t)$, where the roles of s_i, t_i can be interchanged as necessary. Thus $|d(s_i, A_{tl} - d(t_i, A_{tl})| \geq r_t - r_{t-1}$. By applyting a Chernoff bound this would hold w.h.p for $l = 1, ..., L$, i.e. $\sum_{l=1}^{L} |d(s_i, A_{tl}) - d(t_i, A_{tl})| \geq \Omega(L(r_t - r_{t-1}))$. Then,

$$\begin{aligned}
\|f(s_i) - f(t_i)\| &\geq \sum_{t=1}^{\delta} \sum_{l=1}^{L} |d(s_i, A_{tl}) - d(t_i, A_{tl})| \\
&\geq \sum_{t=1}^{\delta} \Omega(L(r_t - r_{t-1})) \\
&= \Omega(L r_\delta) \\
&= \Omega(L d_{s_i t_i}) \\
&= \Omega(log\, k\, d_{s_i t_i})
\end{aligned}$$

We are now ready to discuss various relaxations which yield approximation algorithms for the uniform sparsest cut problem. These relaxations are detailed in the next section and include the spectral(eigenvalue), Leighton Rao [18] and the Arora, Rao and Vazirani [3] relaxations. Considered in this order, these relaxations give increasingly better approximations and can be seen as solving a more relaxed variant of the uniform sparsest cut problem. In a later section, we give empirical evidence of these algorithms on real world data sets.

5 The Sparsest Cut Problem

In this section, we discuss and formalize the sparsest cut problem. First, the definition and connection to edge expansion will be given. Then, we present the spectral relaxation of the problem and an algorithm due to Fiedler (closely related to Cheeger's inequalities). In the remaining subsections, we discuss better relaxations of the problem that can be solved in polynomial time to yield better approximations.

Problem 4 (Sparsest Cut) *Given an undirected graph $G = (V, E)$, costs $c_e \, \forall e \in E$, and k pairs of vertices s_i, t_i with positive integer demands d_i, find a set of vertices minimizing*

$$\rho(S) = \frac{\sum_{e \in \delta(S)} c_e}{\sum_{i:|S \cap \{s_i, t_i\}|=1} d_i}.$$

The uniform sparsest cut problem in an undirected graph is a special case, which we will focus on. Specifically, we consider all pairs of vertices u, v where $u, v \in E$ having unit demand between them.

Problem 5 (Uniform Sparsest Cut) *Given an undirected graph $G = (V, E)$, costs c_e for all $e \in E$, and a single unit demand between all $s, t \in V$, find a set S of vertices minimizing $\rho(S)$ as defined above. Furthermore, when there is a unit cost associated with all $e \in E$, $\rho(S) = \frac{|E(S,\bar{S})|}{|S||V-S|}$.*

In the case when G has unit cost for all edges, solving the uniform sparsest cut problem approximates the edge expansion of the graph. The edge expansion of a cut $S \subseteq V$, such that $|S| \leq \frac{n}{2}$ is given by $\phi(S) = \frac{\delta(S)}{|S|}$ and the *edge expansion* of the graph is given by $\phi(G) = \min_{S \subseteq V : |S| \leq |V|/2} \phi(S)$.

Note that since we only consider sets S where $|S| \leq |V|/2$, $|V - S| \geq |V|/2$. Thus the following inequalities hold (where $n = |V|$):

$$\frac{\phi(S)}{n} \leq \rho(S) \leq \frac{2}{n}\phi(S)$$

A well defined quantity in clustering and partitioning problems that commonly arises is the *graph conductance*, $\Phi(G)$ defined as:

$$\Phi(G) = \min_{S:|S| \leq |V|/2} \Phi(S) = \min_{S:vol(S) \leq vol(G)/2} \frac{|E(S, \bar{S})|}{vol(S)}$$

The quantity $vol(S) = \sum_{v \in S} d_v$, where d_v is the degree of v is the volume of S. Thus, conductance is a measure of the number of cut edges with respect to the trivial upper bound on the number of possible edges between S and $\bar{S}$. Another intuitive way to view conductance is as the probability that from any given vertex v in S, one can get to $\bar{S}$ by following any edge incident to v.

5.1 Spectral Graph Theory and Eigenvalue Relaxations

In this subsection, we discuss approximation schemes for finding the edge expansion of a graph via spectral graph theory-which relates the eigenvalues of the Laplacian matrix of a graph G to pure combinatorial properties such as connectivity. This is necessary to study here in order to compare and contrast (both theoretically and empirically) traditional but faster approaches using eigenvalue computations to recent but slower approaches which involves solving an LP or SDP. While the appeal of the latter is in the quality of approximation, we note that spectral partitioning algorithms have been found to be quite applicable in practice. Moreover, in the empirical evaluation of sparsest cut algorithms where computing the value of the sparsest cut is quite difficult in practice, such spectral methods would give us reasonably good bounds on the value and hence allow us to evaluate the quality of the corresponding solution using SDP.

Some definitions and notations from linear algebra would be useful and are presented here. The key theorem that will be utilized is the spectral theorem which relates the eigenvectors of a real symmetric matrix in $\mathbb{R}^n$ to a corresponding basis for the n dimensional Euclidean space. Proofs of basic facts and theorems are omitted.

Definition 5.1 (Eigenvalues and Eigenvectors) *Let $M \in \mathbb{C}^{n\times n}$ be a square matrix having complex valued entries. Note that $\mathbb{R} \subseteq \mathbb{C}$. Then λ is an eigenvalue of M iff $Mv = \lambda v$ for some $v \neq \mathbf{0}$. We say that v is an eigenvector of M corresponding to the eigenvalue λ.*

Definition 5.2 (Eigenvalues of a real symmetric matrix) *Let $M \in \mathbb{R}^{n\times n}$ be a symmetric matrix with all real-valued entries. Then, all eigenvalues of M are real. Furthermore, there exists a real eigenvector corresponding to each real eigenvalue of M.*

Definition 5.3 (Orthogonal and Orthonormal vectors) *Two vectors are $u, v \in \mathbb{R}^n$ are orthogonal iff $u.v = 0$, where $(.)$ is the standard dot product defined by $u.v = \sum_{i=1}^{n} u_i v_i$. An orthonormal basis for $\mathbb{R}^n$ is a set S of unit vectors (having magnitude 1) such that span(S)=$\mathbb{R}^n$, and for every $u, v \in S$, u, v are orthogonal.*

Given the above definitions, we can now state the spectral theorem, which relates the eigenvalues of a real symmetric matrix to the corresponding orthonormal basis of eigenvectors.

Theorem 5.1 *Let $M \in \mathbb{R}^{n\times n}$ be a real symmetric matrix. Then there are n real eigenvalues of M, $\lambda_1, \lambda_2, ..., \lambda_n$ not necessarily distinct and n orthonormal vectors $x_1, x_2, ..., x_n$ such that x_i is an eigenvector of λ_i.*

The algebraic multiplicity of an eigenvalue λ_i is the number of times the eigenvalue occurs in the characteristic polynomial of M. The geometric multiplicity on the other hand, is the dimension of the eigenspace spanned by the eigenvectors corresponding to λ_i. Note that for a real symmetric matrix, it follows from the spectral theorem that the algebraic multiplicity coincides with the geoemtric multiplicity.

5.1.1 Rayleigh Quotients and Eigenvalue Optimization

We will now discuss how the eigenvalues are related to the optimum value of a continuous optimization problem.

Definition 5.4 (Rayleigh Quotient) *Let $M \in \mathbb{R}^n$ be a symmetric real valued matrix. Then, the Rayleigh quotient of a vector $x \in \mathbb{R}^n$ with respect to M is defined as $R_M(x) = \frac{x^T M x}{x^T x}$.*

Quite interestingly, the eigenvalues of M are the optimum values of the problem of optimizing the Rayleigh quotient of M under certain constraints.

Theorem 5.2 (Eigenvalues as optimum values) *Let $M \in \mathbb{R}^n$ be a symmetric real valued matrix and $\lambda_1 \leq \lambda_2 \leq .... \leq \lambda_n$ be the eigenvalues of M in non-decreasing order. Then,*

$$\lambda_k = \min_{k-dim\, V} \max_{x \in V \setminus \{0\}} R_M(x)$$

Proof: Let $v_1, v_2, ..., v_n$ be the orthonormal eigenvectors of M corresponding to the eigenvalues $\lambda_1, \lambda_2, ..., \lambda_n$-the latter which are in non-decreasing order. We consider an arbitrary vector x in the k-dimensional subspace spanned by $v_1, ..., v_k$ and let $x = \sum_{i=1}^{k} a_i v_i$. Then, note that:

$$x^T M x = \sum_{i,j} a_i a_j v_i^T M v_j = \sum_{i,j} a_i a_j v_i^T \lambda_j v_j = \sum_{i=1}^{k} \lambda_i a_i^2 \leq \lambda_k \sum_{i=1}^{k} a_i^2.$$

The denominator, $x^T x = \sum_{i=1}^{k} a_i^2$ and hence we have that $R_M(x) \leq \lambda_k$.

We now show that $R_M(x) \geq \lambda_k$, where $x \in V$ for an arbitrary k dimensional subspace V. To this end, we let S be the subspace spanned by $v_k, .., v_n$. Clearly, the dimension of S is $n - k + 1$ and the dimension of V is k so there must exist a vector $x \in V \cap S$. Now, let $x = \sum_{i=k}^{n} a_i v_i$. Similar to the above argument, we have:

$$x^T M x = \sum_{i,j} a_i a_j v_i^T M v_j = \sum_{i,j} a_i a_j v_i^T \lambda_j v_j = \sum_{i=k}^{n} \lambda_i a_i^2 \geq \lambda_k \sum_{i=k}^{n} a_i^2.$$

The denominator is $\sum_{i=k}^{n} a_i^2$, so we have that $R_M(x) \geq \lambda_k$. This completes the proof.

■

We state a corollary without proof which directly follows from the above theorem.

Corollary 5.1 *Let λ_1 be the smallest eigenvalue of M. Then $\lambda_1 = \min_{x \neq 0} R_M(x)$. Furthermore $\lambda_2 = \min_{x \perp x_1, x \neq 0} R_M(x)$.*

5.1.2 Cheeger's Inequalities and Second Eigenvalue

In this section, we give a brief overview of the relationship between the eigenvalues of the Laplacian of a graph (tbd shortly) and the combinatorial properties of the graph such as the number of connected components and bipartiteness. In addition, the second largest eigenvalue will enable us to compute a good approximation to the conductance (and the edge expansion), which in turn as we saw earlier will yield an approximation to the value of the sparsest cut. The value obtained would roughly serve as a benchmark to compare spectral partitioning algorithms and those using LP/SDP's. Some of the proofs are omitted for the sake of brevity; however we fill in the most important ones.

Definition 5.5 (d regular graph) *A graph $G=(V,E)$ is regular (d-regular) if every vertex $v \in V$ has the same number of neighbors (d neighbors). Thus, $|E| = \frac{d|V|}{2}$. We let A be the adjacency matrix of G such that $A_{ij} = 1$ if $(i,j) \in E$ or $A(i,j) = w_{ij}$ if G is a weighted graph.*

Definition 5.6 (Normalized Laplacian of a regular-graph) *Let G be a d-regular graph. Define L, the normalized Laplacian of G to be the matrix:*

$$L = I - \frac{1}{d}A$$

where I is the n by n identity matrix.

Theorem 5.3 (On the combinatorial properties) *Let G be a d-regular graph and L be its normalized Laplacian. Let $\lambda_1 \leq \lambda_2 \leq .... \leq \lambda_n$ be the eigenvalues of L including multiplicities. Then,*

1. *$\lambda_1 \geq 0$ and $\lambda_n \leq 2$.*
2. *$\lambda_k = 0$ iff G has at least k connected components.*
3. *$\lambda_n = 2$ iff G has at least one bipartite connected component.*

Note that it is immediate from the above theorem that if λ_2 is strictly positive then G is connected. For irregular graphs, we define the Laplacian acccordingly such that similar properties hold. In this case L is defined to be:

$$L = I - D^{-1/2}AD^{-1/2}$$

where D is the n by n diagonal matrix having such that $(D)_{ii} = d_i$, where d_i is the degree of vertex $i \in V$.

We now state Cheeger's inequalities which hold for regular and irregular graphs. The key takeaway is that the second eigenvalue λ_2 is close to zero if there exists sparse cut or a 2 clustering of G.

Theorem 5.4 (Cheeger's Inequalities and Conductance) *Let G be any graph with normalized laplacian L such that $0 \leq \lambda_1, ..., \lambda_n \leq 2$. Then, the following hold:*

$$\frac{\lambda_2}{2} \leq \Phi(G) \leq \sqrt{2\lambda_2}$$

Tightness: Both sides of the inequalities are tight in the cases of a disconnected graph and a cycle.

Proof: $(\frac{\lambda_2}{2} \leq \Phi(G))$
Recall that from 5.1 we have that $\lambda_2 = \min_{x \perp x_1, x \neq 0} R_M(x)$. If we let $M = L$, the normalized Laplacian of G, then

$$\lambda_2 = \min_{x \perp x_1, x \neq 0} R_L(x) = \min_{x \perp x_1, x \neq 0} \frac{\sum_{(u,v) \in E} (x_u - x_v)^2}{\sum_v d_v x_v^2}.$$

If we let $x = \frac{\mathbf{1}_S}{|S|} - \frac{\mathbf{1}_{\bar{S}}}{|\bar{S}|}$, then note that every cut edge contributes to 4 in the numerator, and from the fact that $vol(\bar{S}) \geq vol(S)$, we have that

$$R_L(x) \leq \frac{4|E(S, \bar{S})|}{2vol(S)} \leq 2\Phi(S).$$

Then,

$$\Phi(G) = \min_{vol(S) \leq vol(G)/2} \geq \frac{1}{2} \min R(x) = \frac{\lambda_2}{2}$$

■

The proof of the other direction (which is algorithmic) is omitted. However, we give the following algorithm for spectral partitioning due to Fiedler. The running time of this algorithm is $O(|E| + |V| log |V|)$. The input is an eigenvector corresponding to λ_2. However, we note that an approximate eigenvector also suffices, and in fact just needs to be one whose Rayleigh quotient is small enough, i.e. $R_L(x_2) \leq O(\lambda_2)$.

Algorithm 4 Spectral Partitioning $(G = (V, E), x_2)$

1: Sort $v \in V$ according to the entries in x_2.
2: Output the cut which minimizes $\Phi(v_1, .., v_k)$ for $k = 1, .., n - 1$.

The algorithm finds a cut $(S, \bar{S})$ such that $\min\{\Phi(S), \Phi(\bar{S})\} \leq \sqrt{2\lambda_2} \leq 2\sqrt{\Phi(G)}$, where the latter inequality follows from Theorem 5.4.

In the next section we discuss an LP based relaxation which yields a $O(\log n)$ approximation to the uniform sparsest cut problem.

5.2 The Leighton-Rao(LR) relaxation [18]

Note that λ_2 is a continuous relaxation of the conductance $\Phi(G)$ of the graph-and Cheeger's inequalities state that λ_2 can not be arbitrarily smaller than the edge expansion. The conductance and the sparsity of the cut are related in the sense that given the value of one quantity one can (within some factor) compute the value of the other via a reduction. Recall that the uniform sparsest[5] cut problem is concerned with minimizing the quantity $\rho(S) = \frac{|E(S,\bar{S})|}{|S||\bar{S}|}$ across all sets $S \subseteq V$ such that $|S| \leq V/2$. The eigenvalue approach in the previous section can be seen as relaxing the indicator functions $\mathbf{1}_S$ for a set S, to arbitrary functions $f(u) \to \mathbb{R}, u \in V$.

The LR approach relaxes the indicator functions to semi-metrics. To formalize,

$$\rho(S) = \frac{|E(S,\bar{S})|}{|S||\bar{S}|} = \frac{\sum_{(u,v)\in E} |\mathbf{1}_S(u) - \mathbf{1}_S(v)|}{\sum_{u\in S, v\in\bar{S}} |\mathbf{1}_S(u) - \mathbf{1}_S(v)|}$$

If $d_S(u,v) := |\mathbf{1}_S(u) - \mathbf{1}_{\bar{S}}(v)|$ for a set S, then $d_S(.,.)$ defines a semimetric over V because $d_S(u,u) = 0$ for all $u \in V$, $d_S(u,v) = d_S(v,u)$ and fulfills the triangle inequality. The LR relaxation relaxes d_S to arbitrary semi-metrics $d : V \times V \to \mathbb{R}$.

Leighton-Rao(G):

$$\min_{d:d \text{ is a semimetric}} \frac{\sum_{(u,v)\in E} d(u,v)}{\sum_{u,v\in V} d(u,v)}$$

Note that the denominator denotes the sum of $d(u,v)$ for all $\binom{n}{2}$ unordered pairs $u, v \in V$. This is equivalent to solving the following linear program where semimetric conditions are imposed as constraints, and the denominator is normalized to one. The notation d_{uv} is used to denote the distance between u and v w.r.t semimetric d.

LR:

$$\begin{aligned} \textit{minimize} \quad & \sum_{(u,v)\in E} d_{uv} \\ \textit{subject to} \quad & \sum_{u,v\in S} d_{uv} = 1 \\ & d_{uv} \leq d_{uw} + d_{w,v} \ \ \forall u,v,w \in V \\ & d_{u,v} \geq 0 \ \ \forall u,v \in V \end{aligned}$$

[5]Referred to as USC hereafter.

Let $LR(G)$ be the optimum value of the linear program for an input graph G. From the above arguments, $LR(G) \leq \rho(G)$. Note that the semimetric considered is completely arbitrary. Moreover, the function $d_S(u,v) = |\mathbf{1}_S(u) - \mathbf{1}_S(v)|$ can be seen as mapping every vertex to the real line and then defining the distance between any two vertices u, v as their l_1 norm. We now show that relaxing the dimensionality condition i.e. considering arbitrary mappings in higher dimensions s.t. $f : V \to \mathbb{R}^m$ and minimizing the quantity $\frac{\sum_{(u,v)\in E}\|f(u)-f(v)\|_1}{\sum_{u,v\in V}\|f(u)-f(v)\|_1}$ over all f yields the optimum value of the uniform sparsest cut. This would allow us to use Theorem 4.2 and get an $O(log\, n)$ approximation guarantee.

Theorem 5.5 *For any graph G, the optimum value of the l_1 relaxation of the USC problem over arbitrary functions f for some $m > 0$ is equivalent to the optimum value of the USC of G, i.e.*

$$\min_{|S|\leq|V|/2} \frac{|E(S,\bar{S})|}{|S||\bar{S}|} = \min \frac{\sum_{(u,v)\in E} |\mathbf{1}_S(u) - \mathbf{1}_S(v)|}{\sum_{u\in S, v\in\bar{S}} |\mathbf{1}_S(u) - \mathbf{1}_S(v)|} = \inf_{f:V\to\mathbb{R}^m} \frac{\sum_{(u,v)\in E} \|f(u) - f(v)\|_1}{\sum_{u,v\in V} \|f(u) - f(v)\|_1}$$

Furthermore, given such a mapping a cut $(S, \bar{S})$ can be found in polynomial time.

Proof: Note first that for any non-negative reals $a_1, a_2, ..., a_n$ and positive reals $b_1, b_2, ..., b_n$, we have that $\min_i \frac{a_i}{b_i} \leq \frac{\sum_{i=1}^n a_i}{\sum_{i=1}^n b_i}$ since,

$$\sum_i^n a_i = \sum_i^n b_i \frac{a_i}{b_i} \geq \min_i \frac{a_i}{b_i} \sum_{i=1}^n b_i. \tag{5.1}$$

Denote by $f_i(v)$ the i^{th} coordinate of v in the mapping. Then,

$$\begin{aligned}\frac{\sum_{(u,v)\in E} \|f(u) - f(v)\|_1}{\sum_{u,v\in V} \|f(u) - f(v)\|_1} &= \frac{\sum_i \sum_{(u,v)\in E} \|f_i(u) - f_i(v)\|_1}{\sum_i \sum_{u,v\in V} \|f_i(u) - f_i(v)\|_1} \\ &\geq \min_i \frac{\sum_{(u,v)\in E} \|f_i(u) - f_i(v)\|_1}{\sum_{u,v\in V} \|f_i(u) - f_i(v)\|_1}\end{aligned}$$

Let j denote the coordinate of the mapping f which minimizes the above. Then, w.lo.g. consider g the normalized version of f_j-which is just the mapping f restricted to the j^{th} dimension. Note that $\max_v g(v) - \max_u g(u) = 1$. Picking a threshold $t \in (0, 1)$ uniformly at random and defining the set $S_t = \{v : g(v) \leq t\}$, we get that $\mathbb{E}[|\mathbf{1}_{S_t}(u) - \mathbf{1}_{S_t}(v)|] = |g(u) - g(v)|$. Hence,

$$\begin{aligned}\min_i \frac{\sum_{(u,v)\in E} \|f_i(u) - f_i(v)\|_1}{\sum_{u,v\in V} \|f_i(u) - f_i(v)\|_1} &= \frac{\sum_{(u,v)\in E} |g(u) - g(v)|}{\sum_{u,v\in V} |g(u) - g(v)|} \\ &\geq \frac{\mathbb{E}[\sum_{(u,v)\in E} |\mathbf{1}_{S_t}(u) - \mathbf{1}_{S_t}(v)|]}{\mathbb{E}[\sum_{u,v\in V} |\mathbf{1}_{S_t}(u) - \mathbf{1}_{S_t}(v)|]}\end{aligned}$$

Then, from equation 5.1 such a set S_t must exist and can be found by going through all the cuts $(\{v_1, ..., v_k\}$ for $k = 1$ to $n-1$ such that $g(v_j) \geq g(v_i)$ whenever $j > i$. ■

Theorem 5.6 ($O(\log n)$ guarantee for USC) *Given an optimal solution $OPT(LR)$ and distances d_{uv} for all $u, v \in V$, one can find an embedding in polynomial time $f : V \to \mathbb{R}^m$ such that, with high probability for all $u, v \in V$*

$$\|f(u) - f(v)\|_1 \leq d_{uv} \leq O(\log n) \|f(u) - f(v)\|_1$$

yielding an $O(\log n)$ approximation guarantee for the USC.

Proof: Note that the existence of f follows directly from 4.2. From the LP relaxation we have,

$$\begin{aligned} OPT(LR) &= \frac{\sum_{(u,v)\in E} d_{uv}}{\sum_{u,v\in S} d_{uv}} \\ &\geq \frac{\sum_{(u,v)\in E} \|f(u) - f(v)\|_1}{\sum_{u,v\in S} O(\log n) \|f(u) - f(v)\|_1} \\ &\geq \frac{\rho(S_t)}{O(\log n)} \end{aligned}$$

where the last inequality follows from theorem 5.5. S_t refers to the optimal cut found from an application of theorem 5.5. Since $OPT(LR) \leq \rho(S^*)$ where $(S^*, V - S^*)$ is the cut of minimum sparsity in G,

$$\rho(S_t) \leq O(\log n) OPT(LR) \leq O(\log n)\rho(S^*).$$

■

5.3 Goemans-Linial Relaxation and the ARV Algorithm [3]

We intend to give a brief overview of the breakthrough work of Arora, Rao and Vazirani [3] who gave a semidefinite programming relaxation of the uniform sparsest problem in undirected graphs to get a $O(\sqrt{log\, n})$ approximation algorithm. This result affirmatively answered several long standing open questions on embeddings of arbitrary metrics into l_1, embeddability of expander graphs in arbitrary graphs and mixing time of random walks.

Many of the proofs are quite long and are therefore omitted. However, we give key theorems and the main algorithm for solving that are sufficient to appreciate the complexity of solving Problem 5 and the power of the approximation ratio achieved.

The SDP relaxation:

$$\begin{aligned} &minimize\ \frac{1}{n^2}\sum_{e=(i,j)\in E} c_e \left\|v_i - v_j\right\|^2 \\ &subject\ to\ \sum_{i,j\in V: i\neq j} \left\|v_i - v_j\right\|^2 = n^2 \\ &\qquad \left\|v_i - v_j\right\|^2 \leq \left\|v_i - v_k\right\|^2 + \left\|v_k - v_j\right\|^2\ \ \forall i,j,k \in V \\ &\qquad v_i \in \mathbb{R}^n\ \forall i \in V. \end{aligned}$$

Note that the triangle inequality is required to hold for the l_2^2 norm instead of the usual l_2 norm. This ensures that the angle subtended by any 2 sides of a triangle formed by the corresponding vectors is not obtuse. Each vertex i of the graph is associated a vector v_i. The goal is to translate (roughly) the proximity of the nodes in the original graph to the proximity in the Euclidean space and to find a corresponding assignment of vectors to nodes such that the average distance between endpoints of edges is small while the average distance between any 2 vertices is large. Unlike previous SDP relaxations, the vectors are not constrained to lie in the unit sphere in $\mathbb{R}^n$. Before moving on, we give a few definitions.

Definition 5.7 (Closed ball around i) *Let $i \in V$. Then the ball of radius r around v is given by $B(i,r) = \{j \in V : d(i,j) \leq r\}$. Note that $d(i,j) = \left\|v_i - v_j\right\|^2$. Also known as the l_2^2 metric, the square of the usual Euclidean metric.*

Definition 5.8 (Δ-Separated Sets) *Let $d(i,S) = \min_{j\in S} d(i,j)$. Sets S,T are Δ-separated if $\forall i,j \in \left\|v_i - v_j\right\|^2 \geq \Delta$.*

Definition 5.9 ($\alpha-$large sets) *Sets L,R are $\alpha-$large if $|L| \geq \alpha.n$ and $|R| \geq \alpha.n$ for some $\alpha \in (0,1]$.*

The rounding algorithm works as follows: Given the vectors v_i, a fat-hyperplane rounding technique is used. A random vector in $\mathbb{R}^n$ is chosen-each of whose component is uniformly drawn at random from $N(0,1)$. Then, only vectors which have significant projection on

the random vector are considered. The rest of the vectors are discarded. It is then shown that this strategy yields sets L, R which are α-large and Δ-separated for some α and Δ. A sparsest cut of cost $O(\sqrt{log\, n})$ ratio of the optimal is then found.

The algorithm is as follows:

Algorithm 5 Sparsest Cut via Fat-Hyperplane Rounding

if there is an $i \in V$ st $|B(i, 1/4)| \geq n/4$ **then**
    $L' = B(i, 1/4)$
else
    Pick $o \in V$ which maximizes $|B(o, 4)|$
    Pick a random vector r.
    Let $L = \{i \in V : (v_i - v_o).r \geq \sigma\}$, $R = \{i \in V : (v_i - v_o).r \leq -\sigma\}$
    Let $L' = L, R' = R$
    while there exists $i \in L', j \in R'$ st $d(i, j) \leq \Delta$ **do**
        Remove i, j from L', R' resp.
Sort $i \in V$ in non-decreasing order of $d(i, L')$ to get $i_1, .., i_n$.
Return $S_k = \{i_1, i_2, .., i_k\}$ which minimizes $p(S_k)$, $1 \leq k \leq n$.

The proof of approximation ratio the algorithm achieves is based upon the following theorems. One concerns the existence of large enough sets L and R which are both $\Omega(n)$-large and the other establishes a good guarantee on the separation of those sets. We present these theorems without proof which will allow us to establish the $O(\sqrt{log\, n})$ guarantee.

Theorem 5.7 (Large-enough L, R) *If there is no $i \in V$ st $|B(i, 1/4)| \geq \frac{n}{4}$, then with constant probability, L, R are α-large for some constant α.*

Theorem 5.8 (Large-enough and well separated L', R') *If L, R are α-large, then with constant probability, L', R' are $\alpha/2$ large, and Δ-separated where $\Delta = C/\sqrt{log\, n}$ for some C.*

Theorem 5.9 (Existence of a Cut) *A cut S st $L \subseteq S \subseteq V - R$ can be found st. $\rho(S) \leq \frac{\sum_{e=(i,j)\in E} c_e \|v_i - v_j\|^2}{\sum_{i\in L, j\in R} \|v_i - v_j\|^2}$.*

Theorem 5.2 lies at the heart of the algorithmic guarantee. The original proof in [3] is for $\Delta = C/log^{2/3}\, n$. Using ideas from this proof, it is shown how to get a better separation of $\Delta = C/\sqrt{log\, n}$. The proof is via contradiction whose strategy is briefly described later. An important lemma (not provided here) on measure concentration is also used in the final analysis.

Theorem 5.1 mainly uses combinatorial arguments and is not the reason for the main difficulty of the overall proof. Theorem 3 uses standard arguments which are quite similar to those used for the LP-relaxation of Leighton and Rao to bound the cost of the cut output by the algorithm.

Lemma 5.1 *Algorithm 3 for the sparsest cut via the fat hyperplane rounding technique gives a $O(\sqrt{log\, n})$ approximation to the uniform sparsest cut problem.*

Proof: From theorem 17, once such a cut S has been found, we have

$$\sum_{i,j \in V: i \neq j} \|v_i - v_j\|^2 \geq \sum_{i \in L, j \in R} \|v_i - v_j\|^2$$
$$\geq \Omega(n^2/\sqrt{log\, n}).$$

From Theorem 5.3 we have

$$\rho(S) \leq O(\sqrt{log\, n}\, \frac{1}{n^2} \sum_{e=(i,j) \in E} \|v_i - v_j\|^2$$
$$\leq O(\sqrt{log\, n}).OPT.$$

The last inequality follows from the constraint in the SDP relaxation. ■

Proof ideas for theorem 5.2
The proof structure for theorem 5.2 when $\Delta = C/log^{2/3}\, n$ is now presented. This is due to Lee[17].

The proof is via contradiction so assume that L and R are α-large while L' and R' are not.

1. If L', R' are not α-large, then there must exist at least $|M(r)| = \Omega(n)$ 'matching' edges (i, j) which are removed.

2. A matching graph M is subsequently constructed. Its edges include all those incident pairs i, j which are removed by the algorithm for the distribution of random vectors r.

3. The probability that $d(i, j) \leq \Delta$ given that they are in L, R respectively is polynomially small (using Gaussian tail bounds).

4. Yet, such pairs exist for a constant fraction of the probability space of random vectors- this gives a contradiction.

In the next section, we present empirical results for the 3 approaches to achieving an approximation for the uniform sparsest cut problem. Additionally we evaluate the performance of the classical Goemans and Williamson[10] algorithm for Maximum Cut. All results are tested on real world graph data sets.

6 Empirical Results and Our Contribution

We now present our experimental results on evaluating the quality of approximation for various approximation algorithms for coping with NP-completeness in cut-based and partitioning problems. First, we provide experimental evidence of the approximation achieved by the SDP-based algorithm for Max-Cut due to Goemans et al. [10]. This is compared to the derandomized version of Max-Cut-one which can be seen as a greedy algorithm to maximize the number of cut edges at each step.

We then build upon the powerful techniques presented in Section 5 to approximate the uniform sparsest cut and present empirical evidence of the quality of cut (as measured by the value of USC) attained by the spectral, Leighton-Rao and ARV algorithms respectively. Although there has been quite a long line of work on the sparsest cut problem (and its dual, which is the maximum multicommodity flow problem), there has been relatively few instances where recent theoretical breakthroughs have been empirically supported. It is difficult to say whether the theoretical improvements are translated into improved performance on real-world data sets. We note that heuristics are still quite popular for solving many NP-hard graph theoretic problems. One of the major goals of this thesis has been to validate and align the elegant theoretical insight developed developed over the past 2 decades with concrete, workable implementations.

The work of Arora, Rao and Vazirani[3], for example, has been a major breakthrough over the past decade in TCS; in fact based on this result, there has been progress on other fundamental NP-hard problems (see [1], [16]). However, there have been few attempts to evaluate the algorithm to generate 'well-separated sets' on real world graph data sets. This work seeks to address the performance guarantee one can achieve in practical settings with the theoretically sound $O(\sqrt{log\, n})$ approximation. The algorithm (and its accompanying analysis) is quite complex and as a result, most practitioners have relied on traditional approaches.

Experimental Setup: All experiements were run on a standard Intel Core i5 Processor with dual processing capability (@3.2 GHz) and 4 GB of RAM. All development was done on MATLAB; To solve linear and semidefinite programs, we used CVX[11, 12] (which contained built-in solvers SDPT3, SeduMi) a package for specifying and solving convex programs. For practical and testing purposes we restricted the size of graphs as necessary so that any given run of a convex program on a specific graph data set could terminate with a feasible solution within 10 minutes. For the maximum-cut problem, we ran our algorithms on graphs having 200-300 nodes while for sparsest cut, the LP and SDP based approaches restricted us to about 60 nodes. However, the graphs we considered were typically dense and could accommodate about a quadratic number of edges. We note that the running time greatly increased as we scaled up the number of nodes for LP and SDP based approaches. All graph data sets we used in our algorithms were found from Network Repository (an interactive network data repository), see [22]. MATLAB codes and the data sets used for this project can be found at: http://i.cs.hku.hk/fyp/2017/fyp17002/.

6.1 Results for Max-Cut

The maximum cut problem seeks to find a cut which maximizes the number of cut edges (see section 3.4). The seminal result of Goemans et al. uses a semidefinite relaxation to get a 0.878 approximation factor with high probability in polynomial time. The random-hyperplane rounding introduces a degree of randomness-hence to be precise, the approximation guarantee holds only w.h.p. We note that a general framework to derandomize semidefinite programs, including Max-Cut was proposed by Mahajan et al.[19].

We now discuss the derandomized algorithm for Max-Cut which is guaranteed to give 0.5-approximation in polynomial time to Max-Cut. In fact it is stronger in the sense that it always outputs a cut having at least half of all edges in the graph. This can be discussed in terms of the general framework of derandomization-however, we provide the algorithm without proof for the sake of brevity.

Algorithm 6 Derandomized Max-Cut (G=(V, E))

1: $C = \{v_1\}$ ▷ C denotes the cut which maximizes $|E(C, \bar{C})|$
2: **for** $i = 2, ..., |V|$ **do**
3: Let cut-edges(C, v_i) denote the number of cut edges incident to v_i w.r.t C
4: **if** cut-edges$(C, v_i) \leq degree(v_i)$ **then**
5: $C = C \bigcup \{v_i\}$
6: Return C

If we simply assign each vertex with probability $1/2$ to C, then we get a randomized 0.5 approximation to Max-Cut. This follows from the linearity of expectation on the random variables X_e where $e = (u, v) \in E$ and $X_e = 1$ iff $C \cap \{u, v\} = 1$ and 0 otherwise. Therefore, $\sum_e E[X_e] = \frac{1}{2}|E|$.

Derandomized Max-Cut can be seen as a derandomized version of the randomized 0.5-approximation to Max-Cut due to the fact that in every iteration, it adds vertex v_i to C only if not assigning to C leads to a fewer number of cut-edges. That is to say, that v_i is added to C if it maximizes the expected number of cut edges with respect to the current assignment of vertices $v_1, v_2, ..., v_{i-1}$ to either C or $\bar{C}$.

Claim 6.1 *Algorithm Derandomized Max-Cut outputs C in time $O(mn)$ s.t $|E(C, \bar{C})| \geq \frac{|E|}{2}$.*

This derandomized version effectively provides a good benchmark to compare the performance of the SDP-based relaxation of Max-Cut. Since computing the exact value of cut can take a prohibitive amount of time, we feel it is most useful to compare the SDP-based algorithm[6] with other previously known polynomial time algorithms with reasonable approx-

[6]referred to as Max-CutSDP hereafter

imation ratios. The derandomized version is a fast algorithm providing a (reasonably) good certificate of the value of Max-Cut that can be achieved by any polynomial time algorithm.

Due to memory constraints our experimental setup handles graphs containing up to 230 nodes. For each graph data set we run Max-CutSDP and Derandomized Max-Cut simultaneously and obtain the value of the maximum cut. For almost all data sets, we note that the Max-CutSDP runs in comparable time and outputs a cut of value which is very close to the derandomized version. Note that if we were to derandomize Max-CutSDP based on the technique of Mahajan et al.[19], chances are that it could even beat Derandomized Max-Cut on most real world instances.

For each algorithm A, on an input I (I is the adjacency matrix of the graph) we compute the following ratio, β_A defined as follows:

$$\beta_A^I = \frac{\#\text{cut edges output by A on } I}{|E|}$$

Note that β does not give us an approximation ratio for the maximum cut but simply allows the performance of Max-CutSDP and Derandomized Max-Cut to be compared by normalizing the value of the number of cut edges returned. Moreover β_{DR} where DR corresponds to Derandomized Max-Cut lies in the interval $[0.5, 1]$. Moreover, suppose $OPT(MC)$ is an algorithm to solve Max-Cut exactly. Then,

$$0.5 \leq \beta_{DR}^I \leq \beta_{OPT(MC)}^I \leq 1 \quad \forall I.$$

Intuitively, if B_{SDP} (where SDP denotes the Max-CutSDP algorithm) has value close to B_{DR} it is a good approximation. Since we are not concerned with the exact value of the optimal solution, we treat B_{DR} as best possible on most instances where $B_{SDP} < B_{DR}$. It is also unrealistic to expect B_{SDP} to fare better than B_{DR} on most instances, since Max-CutSDP makes a randomized choice for hyperplane rounding while Derandomized Max-Cut essentially makes deterministic choices yielding a cut whose value is at least $|E|/2$. It is also quite possible for most instances that $|B_{OPT(MC)} - B_{DR}|$ is quite small. In fact this gap is bounded by $|1 - B_{DR}|$.

As expected, the results prove to be quite convincing, with the average difference $B_{DR} - B_{SDP}$ being about 0.17 across all data set instances we ran the algorithms on. In fact, in most cases B_{SDP} is just slightly less than B_{DR} and on one instance B_{SDP} turned out to be strictly greater than B_{DR}. This shows that deterministic choices might not always result in the best algorithms in the case of an adversarial input. However, we also note that for some input instances B_{SDP} was much smaller than B_{DR} emphasizing the crucial need to derandomize SDP-based approaches to be acceptable in practice.

The results are summarized in Figures 6.1 and 6.2.

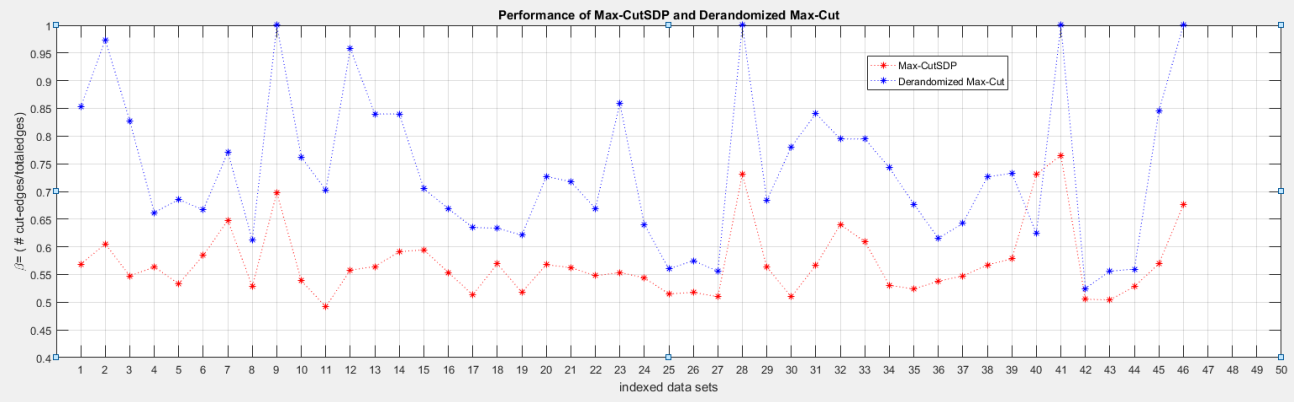

Figure 6.1: As can be seen, Derandomized Max-Cut fares well on nearly all instances. However, for most inputs the difference between the values is small. As claimed $B_{DR} \geq 0.5$.

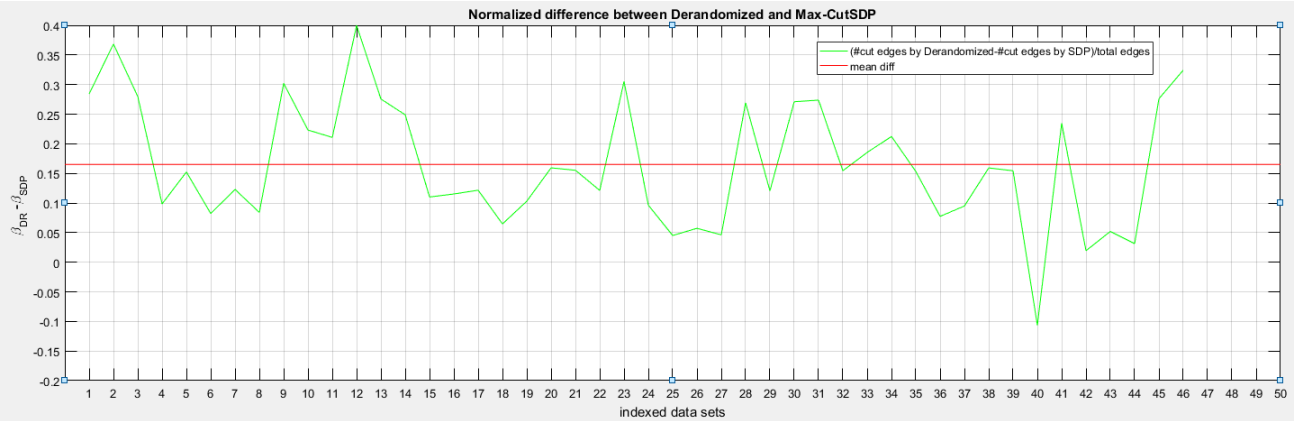

Figure 6.2: The mean difference is about 0.17 shown as the horizontal line, while the highest difference is about 0.4. The important thing to note is that Max-CutSDP outperforms Derandomized Max-Cut on one instance. It is interesting since Max-CutSDP might discover some large cut owing to randomization in the case when such large cuts are rare in a graph, while Derandomized Max-Cut might overlook such a cut.

6.2 Results for Uniform Sparsest Cut

We now discuss the results we obtained for the three algorithms (Spectral, Leighton-Rao and the Arora, Rao and Vazirani algorithm)[7] for the USC problem. These form the significant part of our experimentation step due to the difficulty and effort involved to implement and evaluate the performance of those algorithms on real world data sets. Significant time was expended to make sure that the code was highly robust and optimized to handle large enough instances. Rounding steps for LR and ARV were quite nontrivial and to the best of our knowledge, there is little or no evidence of how the ARV algorithm as presented in [3] fares on real world data sets. Our most significant contribution include fully functional MATLAB functions which implement the three algorithms for the USC problem. They take an adjacency matrix of a graph as input and return the value of the uniform sparsest cut. Given that the adjacency matrix is well defined (i.e. symmetric with ones on the diagonal) the functions return a feasible value for USC, which is between 0 and 1. This follows from the definition of the USC, since

$$\rho(S) = \frac{|E(S, \bar{S})|}{|S||\bar{S}|}$$

where the denominator is an upper-bound for the number of edges that can participate in the cut.

Most of the data sets in this section are different from the ones used in the previous section to evaluate the performance of algorithms for Max-Cut. One of the reasons was that the convex programs for solving the USC problem took significant amount of time to test graphs having more than 60 vertices and since we wanted to test on many different data sets to get an overview of the overall performance of the 3 algorithms, the maximum instance size was restricted. We also wanted to ensure that such graphs represented practical instances for which a common objective is to find well-connected clusters and sub-graphs.

Nevertheless, all algorithms can evaluate larger graphs (200-300 vertices) with the current implementation given sufficient time-which can be a few hours for the largest instances. The time the largest graph (on 51 nodes) took was about 10 minutes for both the LR and ARV algorithm to return the value of USC. The running time for the LR and ARV algorithm was about the same. The spectral algorithm (which relies on an eigenvalue computation of the Laplacian matrix) was significantly faster than both LR and ARV algorithm and took only a few seconds.

A total of 54 data sets were used in the experiments and chosen from a wide variety of networks including (but not limited to) biological, infrastructure, transportation, social, ecological, web, dynamic and brain networks. This ensured that the performance of any particular algorithm could be evaluated on almost all types of networks arising in practice.

[7] Throughout this section, they are abbreviated as SP, LR and ARV and used interchangeably.

Comment on exact value and approximation ratio:
We note that computing the 'exact' value of the uniform sparsest cut is not possible in feasible time since this involves enumerating an exponential number of cuts. The next most accurate measure to *certify* the quality of the approximation ratio attained by various algorithms is to compare their values against the best performing algorithm. Moreover, since our motivation was to compare and contrast the performance differences between successive theoretical improvements for the USC problem, we decided not to compare against heuristic approaches. The main goal was to see whether $O(\sqrt{log\, n})$ approximation was observable in practice and yielded measurably different results against the an older $O(log\, n)$ approximation and the more classical spectral approach.

Discussion of Results:

Throughout the remainder of this section, let $\text{usc}_{SP}, \text{usc}_{LR}$ and usc_{ARV} denote the value of the uniform sparsest cut returned by the algorithms SP, LR and ARV. Since $O(\sqrt{log\, n})$ is still small for our problem instances and the value of the constants are not accounted for, there should be little expectation for the performance of ARV to be much different from LR. Interestingly, the results confirmed the intuition. ARV performed better on only some instances while on most others, it either output the same value(of USC) as LR or slightly higher.

In terms of running times SP was the fastest-with the value returned in a matter of a few seconds. For modest sized inputs LR and ARV took a few minutes while on the largest inputs they took up to 10 minutes. Since our main goal is to compare the quality of the approximation attained we do not compare the exact running times of each algorithm. Essentially, we only compare the values $\text{usc}_{SP}, \text{usc}_{LR}$ and usc_{ARV}. This is in line with the observation that researchers in approximation algorithms are most concerned about attaining better approximation ratios instead of running time improvements.

The results are summarized in Figures 6.3, 6.4, 6.6 and 6.5. Figure 6.3 is a plot of the value of the uniform sparsest cut against the indexed data sets, for SP, LR and ARV respectively. As can be seen, the error between all 3 algorithms is small and in general each of them tends to *'follow'* the others.

Data sets for which only two points are plotted correspond to duplicate values of USC attained by any 2 algorithms. In cases where two colors are missing, all 3 algorithms attain the same value-hence MATLAB only shows one point. This is also the reason why any single plot (corresponding to a specific algorithm) is sometimes 'disconnected' due to superimposition by another plot-corresponding to another algorithm achieving similar value. Thus, we conclude that for most real world instances, all algorithms perform satisfactorily. More importantly, the spectral partitioning algorithm is useful where the graph has expansion bounded by a constant, i.e $\Phi(G) = \Theta(1)$. This is the case for most real world data sets where the second eigenvalue, λ_2 is a good approximation to $\Phi(G)$.

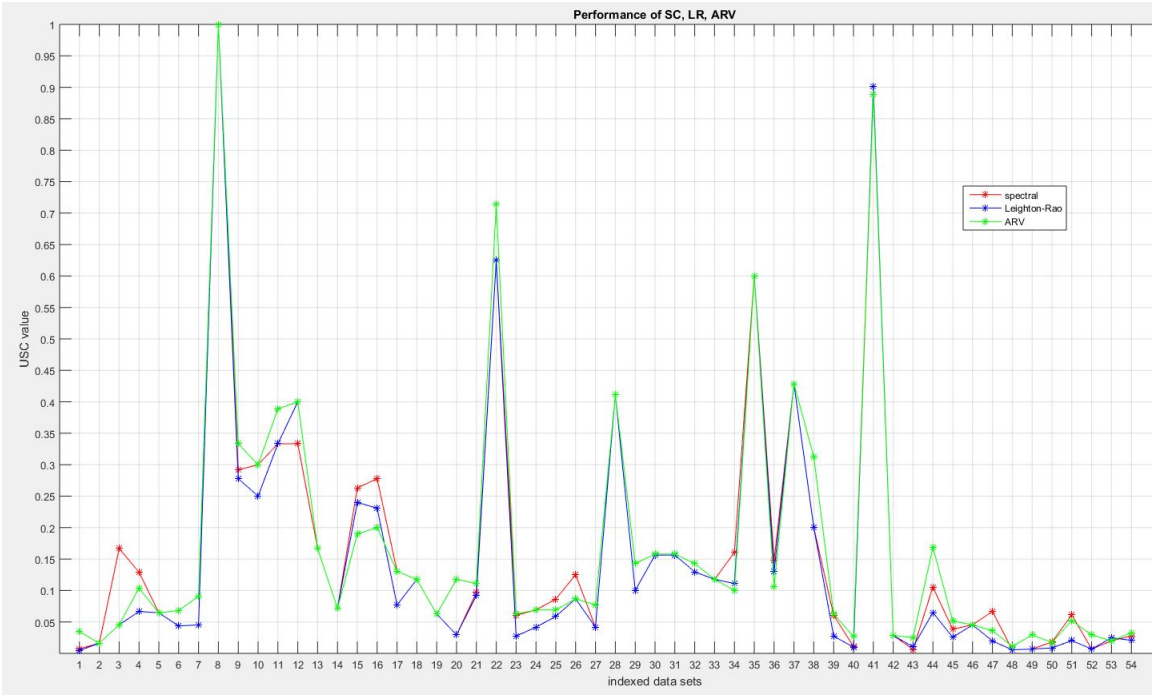

Figure 6.3: The value of the uniform sparsest cut against inputs to the algorithm. Note in particular how each of the 3 algorithms do not perform much worse than the rest. The differences are small and whenever the values output are the same for any 2 algorithms, only one is shown.

It can be observed from 6.3 that on most instances, $\text{usc}_{LR} \leq \min\{\text{usc}_{SP}, \text{usc}_{ARV}\}$. More interestingly, $\text{usc}_{ARV} \leq \text{usc}_{SP}$ for more than 60% for the instances. SP does better than LR on very few inputs while ARV is worse than LR on roughly 60% of the instances. The average differences are as follows:

$$\frac{\text{usc}_{SP} - \text{usc}_{LR}}{\#ofinstances} = 0.0136 \; ; \; \frac{\text{usc}_{ARV} - \text{usc}_{LR}}{\#ofinstances} = 0.0187; \; \frac{\text{usc}_{ARV} - \text{usc}_{SP}}{\#ofinstances} = 0.0051.$$

These are plotted against the actual differences in Figures 6.4, 6.5 and 6.6 and shown as a horizontal black line.

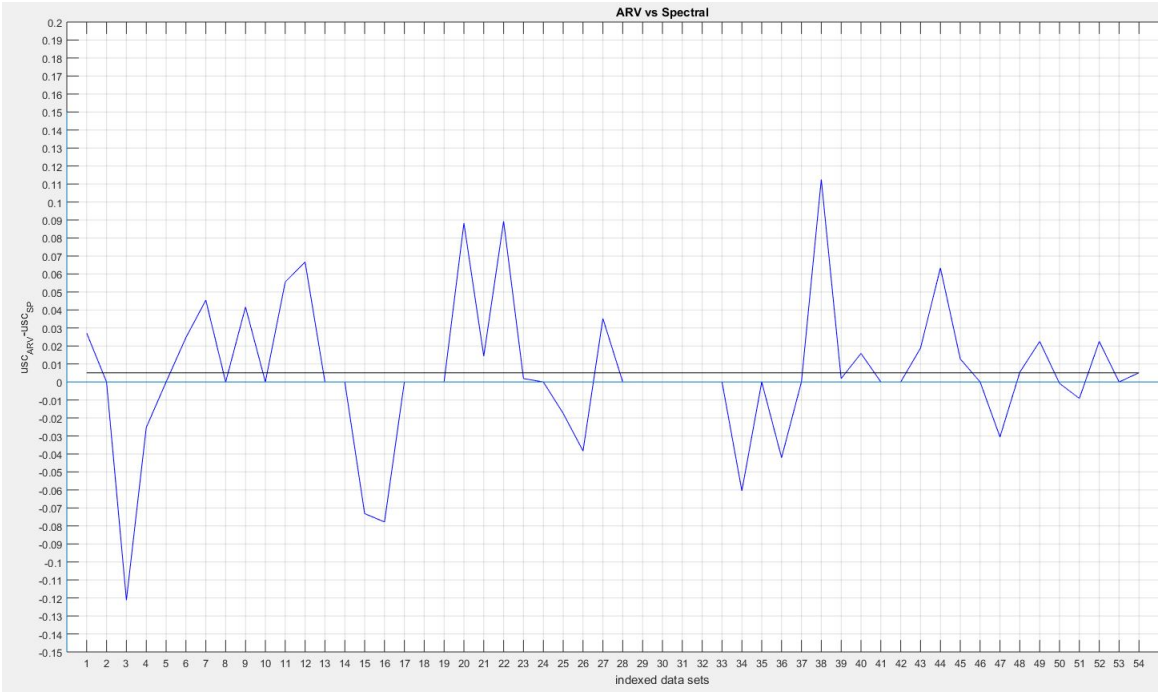

Figure 6.4: Plot of $\text{usc}_{ARV} - \text{usc}_{SP}$: ARV does better than SP on most instances. However, the average difference is positive but small.

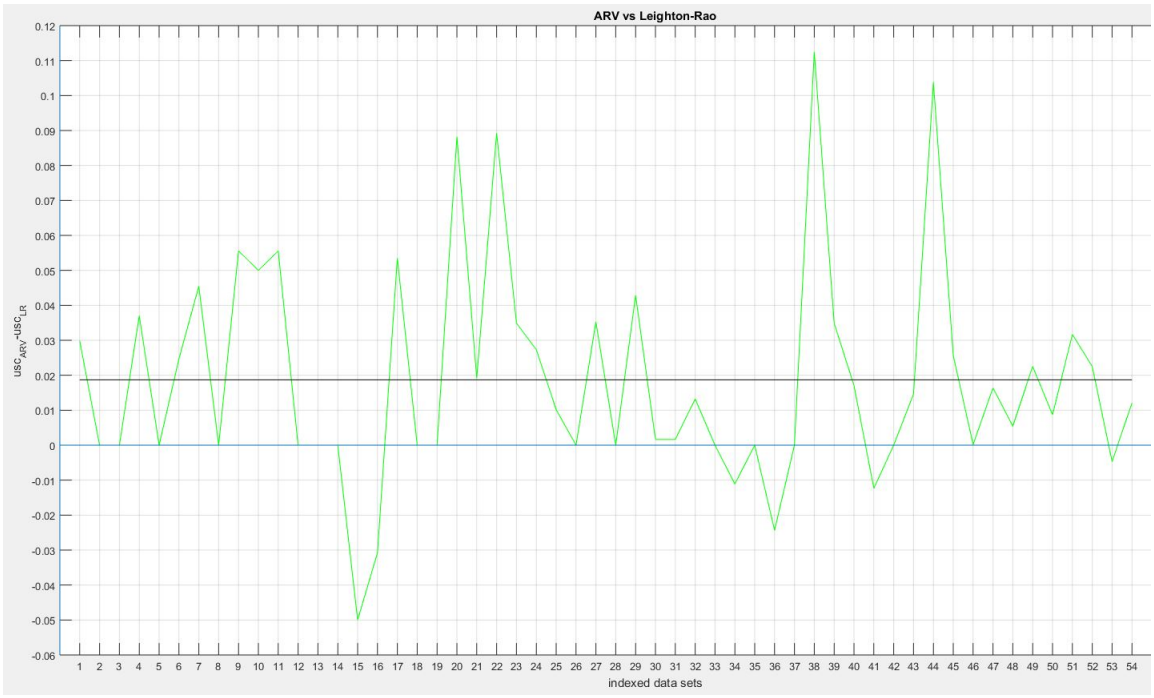

Figure 6.5: Plot of $\text{usc}_{ARV} - \text{usc}_{LR}$: LR clearly does better than ARV in most instances. This is also reflected by the greater average difference than in Figure 6.4. The maximum difference is bounded by about 0.12.

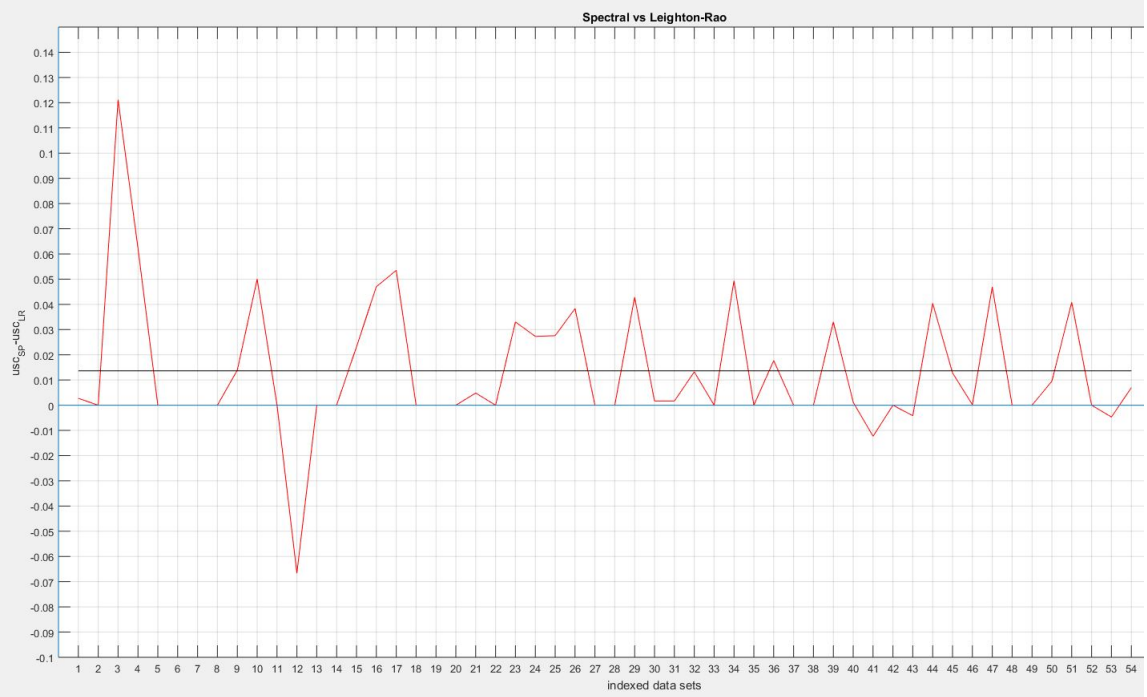

Figure 6.6: Plot of $\text{usc}_{SP} - \text{usc}_{LR}$: SP does strictly better than LR for exactly 4 instances. The average difference is small while the maximum difference is bounded by roughly 0.12.

It would be difficult to say which algorithm is truly the best among all. If running time is the concern, then the spectral algorithm performs quite well and outputs a cut of value which is close to both the LR and the ARV algorithm. However, if one wants the best cut and is willing to settle for a trade-off on running time, then either LR or ARV should be good choices. It must be remarked that as the input size increases, both LR and ARV would take quite a significant amount of time with the current implementation. Thus, an interesting future research direction would be to reduce the size of the constants in the running time, or reduce the width of constraints of convex programs. Both LR and ARV in this work involve $O(n^3)$ constraints corresponding to the triangle inequalities in the convex relaxations.

We remark that recently, there has been an interesting line of work on optimizing the running time of the semidefinite program in the seminal work of Arora et al. [3]. The multiplicative weights update method has been proposed as an alternate way to approximately solve linear and semidefinite programs with high accuracy in feasible time [2]. Khandekar et al. [14] also propose solving the sparsest cut problem by computing single commodity flows attaining a $O(log\, n)$ approximation. Orecchia et al. [21] further generalized the single commodity flow framework to get $O(\sqrt{log\, n})$ approximation to the uniform sparsest cut. It would be interesting to see whether this new framework gives rise to better algorithms that can accommodate bigger inputs in practice; and whether, such improvements are worth the difference in performance with respect to traditional approaches and heuristics such as METIS [13].

7 Conclusion

One of the overarching goals of this thesis included studying approximation algorithms in the context of recent techniques and methods developed over the past two decades. To this end, we surveyed various papers in top TCS conferences and journals containing both old and new results. We learnt that it was impossible to appreciate modern developments without appreciating limits of old ones. For example, while the ARV algorithm was contained in one paper (albeit long and challenging), it would have been pointless to study it without covering linear programming algorithms and spectral approaches in sufficient detail for sparsest cut and other related problems. Moreover, to fully understand the theoretical results we continuously branched into mastering several other topics deemed as prerequisites.

Another major goal was to provide empirical evidence of the usefulness of approximation algorithms. The implementation stage posed quite a number of problems. It was initially unclear how to translate the theoretical components of various algorithms into running code, let alone expect them to perform well on real world instances. At many points throughout the year, it seemed an ambitious and relentless task to contribute on evaluating these algorithms given the numerical inaccuracies associated with convex programs and rounding algorithms. This work affirmatively answered the question of whether algorithms based on solving convex relaxations of NP-hard problems were 'competitive' enough in practice. Through extensive experimentation, we demonstrated how these methods fare better if one can afford to slightly relax the constraint on running time. We also believe that the theoretical insights gained via this project would enable us to work on various directions in algorithms and contribute to advancing the state of knowledge in theoretical computer science in the future.

8 References

[1] Amit Agarwal et al. "$O(\sqrt{log\, n})$ Approximation Algorithms for Min UnCut, Min 2CNF Deletion, and Directed Cut Problems". In: *Proceedings of the Thirty-seventh Annual ACM Symposium on Theory of Computing*. STOC '05. Baltimore, MD, USA: ACM, 2005, pp. 573–581. ISBN: 1-58113-960-8. DOI: 10.1145/1060590.1060675. URL: http://doi.acm.org/10.1145/1060590.1060675.

[2] Sanjeev Arora, Elad Hazan, and Satyen Kale. "The Multiplicative Weights Update Method: a Meta-Algorithm and Applications". In: *Theory of Computing* 8.6 (2012), pp. 121–164. DOI: 10.4086/toc.2012.v008a006. URL: http://www.theoryofcomputing.org/articles/v008a006.

[3] Sanjeev Arora, Satish Rao, and Umesh V. Vazirani. "Expander flows, geometric embeddings and graph partitioning". In: *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*. Ed. by László Babai. ACM, 2004, pp. 222–231. ISBN: 1-58113-852-0. DOI: 10.1145/1007352.1007355. URL: http://doi.acm.org/10.1145/1007352.1007355.

[4] Florian Bourse, Marc Lelarge, and Milan Vojnovic. "Balanced Graph Edge Partition". In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '14. New York, New York, USA: ACM, 2014, pp. 1456–1465. ISBN: 978-1-4503-2956-9. DOI: 10.1145/2623330.2623660. URL: http://doi.acm.org/10.1145/2623330.2623660.

[5] Gruia Călinescu, Howard Karloff, and Yuval Rabani. "An Improved Approximation Algorithm for Multiway Cut". In: *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*. STOC '98. Dallas, Texas, USA: ACM, 1998, pp. 48–52. ISBN: 0-89791-962-9. DOI: 10.1145/276698.276711. URL: http://doi.acm.org/10.1145/276698.276711.

[6] M. Charikar and A. Wirth. "Maximizing quadratic programs: extending Grothendieck's inequality". In: *45th Annual IEEE Symposium on Foundations of Computer Science*. Oct. 2004, pp. 54–60. DOI: 10.1109/FOCS.2004.39.

[7] Moses Charikar and Vaggos Chatziafratis. "Approximate Hierarchical Clustering via Sparsest Cut and Spreading Metrics". In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*. Ed. by Philip N. Klein. SIAM, 2017, pp. 841–854. ISBN: 978-1-61197-478-2. DOI: 10.1137/1.9781611974782.53. URL: https://doi.org/10.1137/1.9781611974782.53.

[8] Moses Charikar, Konstantin Makarychev, and Yury Makarychev. "Near-optimal Algorithms for Unique Games". In: *Proceedings of the Thirty-eighth Annual ACM Symposium on Theory of Computing*. STOC '06. Seattle, WA, USA: ACM, 2006, pp. 205–214. ISBN: 1-59593-134-1. DOI: 10.1145/1132516.1132547. URL: http://doi.acm.org/10.1145/1132516.1132547.

[9] Eden Chlamtac, Konstantin Makarychev, and Yury Makarychev. "How to Play Unique Games Using Embeddings". In: *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*. FOCS '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 687–696. ISBN: 0-7695-2720-5. DOI: 10.1109/FOCS.2006.36. URL: http://dx.doi.org/10.1109/FOCS.2006.36.

[10] Michel X. Goemans and David P. Williamson. "Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming". In: *J. ACM* 42.6 (Nov. 1995), pp. 1115–1145. ISSN: 0004-5411. DOI: 10.1145/227683.227684. URL: http://doi.acm.org/10.1145/227683.227684.

[11] Michael Grant and Stephen Boyd. *CVX: Matlab Software for Disciplined Convex Programming, version 2.1*. http://cvxr.com/cvx. Mar. 2014.

[12] Michael Grant and Stephen Boyd. "Graph implementations for nonsmooth convex programs". In: *Recent Advances in Learning and Control*. Ed. by V. Blondel, S. Boyd, and H. Kimura. Lecture Notes in Control and Information Sciences. http://stanford.edu/~boyd/graph_dcp.html. Springer-Verlag Limited, 2008, pp. 95–110.

[13] George Karypis and Vipin Kumar. "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs". In: *SIAM J. Sci. Comput.* 20.1 (Dec. 1998), pp. 359–392. ISSN: 1064-8275. DOI: 10.1137/S1064827595287997. URL: http://dx.doi.org/10.1137/S1064827595287997.

[14] Rohit Khandekar, Satish Rao, and Umesh Vazirani. "Graph Partitioning Using Single Commodity Flows". In: *J. ACM* 56.4 (July 2009), 19:1–19:15. ISSN: 0004-5411. DOI: 10.1145/1538902.1538903. URL: http://doi.acm.org/10.1145/1538902.1538903.

[15] Subhash Khot. "On the Power of Unique 2-prover 1-round Games". In: *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*. STOC '02. Montreal, Quebec, Canada: ACM, 2002, pp. 767–775. ISBN: 1-58113-495-9. DOI: 10.1145/509907.510017. URL: http://doi.acm.org/10.1145/509907.510017.

[16] Robert Krauthgamer, Joseph (Seffi) Naor, and Roy Schwartz. "Partitioning Graphs into Balanced Components". In: *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '09. New York, New York: Society for Industrial and Applied Mathematics, 2009, pp. 942–949. URL: http://dl.acm.org/citation.cfm?id=1496770.1496872.

[17] James R. Lee. "On Distance Scales, Embeddings, and Efficient Relaxations of the Cut Cone". In: *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '05. Vancouver, British Columbia: Society for Industrial and Applied Mathematics, 2005, pp. 92–101. ISBN: 0-89871-585-7. URL: http://dl.acm.org/citation.cfm?id=1070432.1070446.

[18] Tom Leighton and Satish Rao. "Multicommodity Max-flow Min-cut Theorems and Their Use in Designing Approximation Algorithms". In: *J. ACM* 46.6 (Nov. 1999), pp. 787–832. ISSN: 0004-5411. DOI: 10.1145/331524.331526. URL: http://doi.acm.org/10.1145/331524.331526.

[19] Sanjeev Mahajan and H. Ramesh. "Derandomizing Approximation Algorithms Based on Semidefinite Programming". In: *SIAM Journal on Computing* 28.5 (1999), pp. 1641–1663. DOI: 10.1137/S0097539796309326. eprint: https://doi.org/10.1137/S0097539796309326. URL: https://doi.org/10.1137/S0097539796309326.

[20] Yu Nesterov. "Semidefinite relaxation and nonconvex quadratic optimization". In: *Optimization Methods and Software* 9.1-3 (1998), pp. 141–160. DOI: 10.1080/10556789808805690. eprint: https://doi.org/10.1080/10556789808805690. URL: https://doi.org/10.1080/10556789808805690.

[21] Lorenzo Orecchia et al. "On Partitioning Graphs via Single Commodity Flows". In: *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*. STOC '08. Victoria, British Columbia, Canada: ACM, 2008, pp. 461–470. ISBN: 978-1-60558-047-0. DOI: 10.1145/1374376.1374442. URL: http://doi.acm.org/10.1145/1374376.1374442.

[22] Ryan A. Rossi and Nesreen K. Ahmed. "The Network Data Repository with Interactive Graph Analytics and Visualization". In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. 2015. URL: http://networkrepository.com.

[23] Chaitanya Swamy. "Correlation Clustering: Maximizing Agreements via Semidefinite Programming". In: *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '04. New Orleans, Louisiana: Society for Industrial and Applied Mathematics, 2004, pp. 526–527. ISBN: 0-89871-558-X. URL: http://dl.acm.org/citation.cfm?id=982792.982866.

[24] Luca Trevisan. "Approximation Algorithms for Unique Games". In: *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*. FOCS '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 197–205. ISBN: 0-7695-2468-0. DOI: 10.1109/SFCS.2005.22. URL: https://doi.org/10.1109/SFCS.2005.22.

[25] Vijay V. Vazirani. *Approximation Algorithms*. New York, NY, USA: Springer-Verlag New York, Inc., 2001. ISBN: 3-540-65367-8.

[26] Avi Wigderson. "Improving the Performance Guarantee for Approximate Graph Coloring". In: *J. ACM* 30.4 (Oct. 1983), pp. 729–735. ISSN: 0004-5411. DOI: 10.1145/2157.2158. URL: http://doi.acm.org/10.1145/2157.2158.

[27] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. 1st. New York, NY, USA: Cambridge University Press, 2011. ISBN: 0521195276, 9780521195270.