Metric-Based Approximation Algorithms for Graph Cut Problems

— Thesis —

Zoltán Ádám Mann

Supervisor: András Benczúr Jr.

Department of Operations Research
Eötvös Loránd University
Faculty of Science

2004

Contents

Chapter 1

Introduction

Cut problems in directed and undirected graphs have been widely studied in combinatorial optimization. This is partly due to their connection to network flows, as shown in the fundamental work of Ford and Fulkerson [16]. There has been much work on generalizing the max-flow-min-cut theorem since then, for example towards multicommodity flows and multicuts [25, 19].

Finding minimum cuts has many direct applications: partitioning VLSI designs [3], assigning tasks to distributed processors [1], and hardware/software partitioning [35] are just a few examples. Furthermore, minimum cut algorithms can be used in the divide step of numerous divide-and-conquer algorithms, such as for linear arrangement, bandwidth minimization, minimum feedback arc set, etc. [42, 13, 32]. Moreover, since some cut problems can also be solved using divide-and-conquer, cut algorithms are also useful for devising other cut algorithms [15].

The most basic versions of the minimum cut problem, such as finding a minimum $s-t$ cut in a directed or undirected graph, or finding a globally minimum cut in a directed or undirected graph can be solved in polynomial time [2, 23, 37]. However, there are also two classes of NP-hard minimum cut problems that are of great theoretical and practical importance:

- Cutting a graph into more than two parts. Several formulations of the problem exist, the most important ones are the following:
  - Removing a minimum cardinality (cost) set of edges such that the remaining graph has a given number of components. This is a possible generalization of the globally minimum cut problem.
  - Removing a minimum cardinality (cost) set of edges such that some given vertices are in different components of the remaining graph. This is a possible generalization of the minimum $s-t$ cut problem, and is usually referred to as the minimum multiway cut problem.
  - Removing a minimum cardinality (cost) set of edges such that some given *pairs* of vertices are split in the remaining graph. This is another possible generalization of the minimum $s-t$ cut problem, and is usually referred to as the minimum multicut problem.
- Balanced cuts. In many applications (including divide-and-conquer schemes) it is desirable to split the graph into two parts of roughly equal size. Some widely used formulations:
  - The number of vertices in the two parts should be at most one apart. (Graph bisection problem.)
  - The number of vertices in each part should be at most βn, where $1 \geq \beta \geq 1/2$ is a given constant and n is the number of vertices of the graph. (β-balanced cut problem.)
  - No explicit limit on the size of the parts is specified; however, the objective function depends on the size of the parts. For example: minimize $c(A,B)/(|A||B|)$, where A and

B are the two parts and $c(A, B)$ is the total cost of edges between them. (Sparsest cut problem.)

Of course it is also possible to combine the two families of problems. This means cutting the graph into more than two parts, such that the parts have balanced sizes. This will be referred to as the γ-separator problem.

Since these problems are NP-hard, but important for several reasons, numerous approximation algorithms have been proposed for them. The first breakthrough was the work of Leighton and Rao in 1988 [31]: they presented an $O(\log n)$-approximation algorithm for the sparsest cut problem, and then using this result, showed polylogarithmic approximation algorithms for a wide variety of related problems. In particular, an $O(\log n)$-approximation result follows for the 2/3-balanced cut problem.

Since then, several improvements have been made. Garg, Vazirani and Yannakakis presented an $O(\log k)$-approximation algorithm for the minimum multicut problem, where k denotes the number of given source–sink pairs. Aumann and Rabani [5] and Linial, London and Rabinovich [34] have devised an $O(\log k)$ approximation algorithm for a generalization of the sparsest cut problem. These results also imply approximate max-flow-min-cut theorems for some multicommodity flow problems. It is also known that there is a worst-case gap of $\Omega(\log k)$ for these multicommodity flow problems and the corresponding cut capacities. Hence, these results are tight, and it is not possible to obtain better approximation algorithms for these cut problems using such methods.

For the graph bisection problem the best result is a polylogarithmic approximation algorithm by Feige and Krauthgamer [15], which uses different methods. Even, Naor, Rao and Schieber used spreading metrics to approximate a number of combinatorial optimization problems [13], including the γ-separator problem [12], and multicuts in directed graphs [14]. A slightly different problem, in which the number of parts is prescribed to be p was handled by Simon and Teng [43] by giving an $O(\log n \log k)$-approximation algorithm.

In the case of the minimum multiway cut, much better results have been achieved. The currently known best approximation factor is 1.3438, due to Karger, Klein, Stein, Thorup, and Young [27].

There are few results concerning the hardness of approximation to these problems. Bui and Jones proved that it is NP-hard to approximate the minimum balanced cut within an $n^{2-\varepsilon}$ additive term [7]; however, this does not give any information on multiplicative approximation factors. Dahlhaus, Johnson, Papadimitriou, Seymour, and Yannakakis proved that the minimum multiway cut problem is MAX-SNP-hard, and therefore it cannot have a polynomial approximation scheme, unless P=NP [11].

1.1 Aims of this work

The aim of this work is to present a family of approximation algorithms for cut problems: those based on graph metrics. The reasons for studying this class of algorithms are as follows:

- It is very interesting to observe how methods of completely different branches of mathematics (such as topology and geometry) can be used to construct efficient graph cut algorithms.
- For most of the above problem formulations, the best known approximation factors can be achieved using metric-based approaches.
- The metric-based approaches might yield general recipes for constructing approximation algorithms for graph cut problems, that can be transfered to other versions of the problem as well.

To our knowledge, this is the first work to systematically present the metric-based approximation algorithms for graph cut problems.

The metric-based approach was pioneered by Leighton and Rao [31]. Their algorithm starts with an optimal solution of the LP-relaxation of the sparsest cut problem (the dual of the respective multicommodity flow problem), which is an assignment of non-negative length values to the edges. The algorithm grows regions using the shortest-paths metric defined by this length assignment. It can be proven that the generated regions can be unified to form a low-cost cut. This approach was later refined by Garg, Vazirani, and Yannakakis [19], yielding a better approximation ratio and simpler proofs.

We will also review subsequent work of Even, Naor, Rao, and Schieber [12], which uses a special graph metric called spreading metric. Spreading metrics offer a general scheme for approximating graph problems [13]. For the case of graph cut problems, this technique is similar to the shortest-paths metric method mentioned above; however, it uses a more direct LP-relaxation of the problem. This way, we have better control on how regions grow, and the overall efficiency of the algorithm is increased.

Another metric-based method, pioneered by Aumann and Rabani [5] and Linial, London, and Rabinovich [34], will also be sketched. In this approach, the metric defined by the optimal solution of the LP-relaxation is embedded into a real space, in which standard geometric means (such as hyperplanes) can be used to find a low-cost cuts. It will be shown how this approach yields good approximation algorithms for the generalized sparsest cut problem [5, 34] as well as the minimum multiway cut problem. The research concerning the latter was initiated by Călinescu, Karloff, and Rabani [10], with subsequent results of Karger et al. [27].

Since the topic of approximating graph cuts is very wide, it cannot be the aim of this work to survey it totally. In particular, it is not the aim of this work to address the following related problems in detail: multicommodity flows [25, 41], disjoint paths problems [17], vertex separators [9, 18], efficiently solvable special cases (such as planar graphs [21, 29], trees [20], dense graphs [4] etc.), cuts in directed graphs [18, 14, 38], clustering heuristics [28], applications of cut algorithms [44, 45].

Most propositions and theorems are presented without proofs, with appropriate citations to the literature. Proofs are presented only where they illuminate some key concept.

1.2 Organization

Table 1.1: Summary of the presented results

	Problem	Approximation factor	Due to	Section
Methods without embedding	Minimum multicut	$O(\log k)$	Garg, Vazirani & Yannakakis, 1995 [19]	3.2
	Sparsest cut	$O(\log n)$	Leighton & Rao, 1988 [31]	3.3
	Minimum β-balanced cut	$O(\log n)$	Leighton & Rao, 1988 [31]	3.5
	Minimum γ-separator	$O(\log n)$	Even, Naor, Rao & Schieber, 1999 [12]	4.3
Methods with embedding	Generalized sparsest cut	$O(\log k)$	Aumann & Rabani, 1994 [5]; Linial, London & Rabinovich, 1994 [34]	5.2
	Minimum multiway cut	1.3438	Karger, Klein, Stein, Thorup & Young, 1999 [27]	5.3

Chapter 2 contains the basic definitions and facts. Moreover, it presents the different LP-formulations of cut problems that will be used throughout this work. The first metric-based cut algorithms are described in Section 3. Then in Chapter 4, the spreading-metric-based approach is detailed, while Chapter 5 presents the approximation algorithms using embeddings in real spaces. Chapter 6 summarizes the described methods and highlights open questions.

Table 1.1 shows the organization of this work as well as the presented results in terms of approximation factors.

Chapter 2

Preliminaries

This chapter presents the basics for the results of the later chapters. In particular, Section 2.1 defines the basic notions, and Section 2.2 defines the different problem formulations. Finally in Section 2.3, several LP formulations are given for the problems, which will be used later in the corresponding algorithms.

2.1 Notation and basic definitions

In all of the addressed problems, an undirected graph $G = (V, E)$, along with non-negative edge costs $c : E \to \mathbb{R}_+$ is given.

We do not allow negative edge costs because the minimum cut problem with general real edge costs also includes the maximum cut problem, which is very different (see [36] for more details). Also, we do not consider directed graphs because the methods for directed graphs differ substantially from those for undirected graphs [14].

The number of vertices in G is denoted by n, the number of edges by m. For simplicity of notation, we will use c to denote either the m-dimensional vector of edge costs (indexed by edges), or an $N = \binom{n}{2}$ dimensional vector indexed by node pairs such that $c_{u,v}$ is either the cost of edge (u, v) if $(u, v) \in E$ or 0 otherwise. It will always be clear from the context which vector is meant. The N-dimensional vector $(1, 1, \ldots, 1)$ will be denoted by 1_N.

For a set $S \subset V$, let $\delta(S) = \{e \in E : |e \cap S| = 1\}$ denote the edge set of the cut $(S, V \setminus S)$. The cost of the cut is $c(\delta(S)) = \sum_{e \in \delta(S)} c(e)$.

Let d be any N-dimensional vector indexed by node pairs. d will be called a *metric*, if it satisfies the following properties:

1. $d_{u,v} \geq 0$ for each $u, v \in V$
2. $d_{u,u} = 0$ for each $u \in V$
3. $d_{u,w} \leq d_{u,v} + d_{v,w}$ for each $u, v, w \in V$

Strictly speaking, these properties define only a semi-metric, because $d_{u,v} = 0$ is allowed for $u \neq v$ as well. However, it is common in the literature to use the term 'metric' in this setting.

The two most important graph metrics are the following:

Definition 2.1. Let $l : E \to \mathbb{R}_+$ be an assignment of length values to the edges. Then let $\mathsf{dist}_l(u, v)$ be the length of the shortest path between u and v. The vector dist_l is the *shortest paths metric* corresponding to the edge lengths l.

Definition 2.2. Let $\emptyset \neq S \subsetneq V$, and define d_S as follows: if $|S \cap \{u, v\}| = 1$ then $d_S(u, v) = 1$, otherwise $d_S(u, v) = 0$. The vector d_S is the *cut metric* corresponding to S.

It is easy to see that both shortest paths metrics and cut metrics are indeed metrics. Note also that the cost of the cut $(S, V \setminus S)$ can also be written as the scalar product $c \cdot d_S$.

We have to stress at this point that throughout this work, the notion $x \cdot y$, where x and y are two vectors of the same dimension, means scalar product. (For simplicity, we do not show explicitly that one of the vectors should be transposed.)

Throughout this work, the letter l will be used for an assignment of edge lengths (i.e. a non-negative vector of dimension m), whereas the letter d will be used for a graph metric (which is a non-negative vector of dimension N).

2.2 The problems

Definition 2.3. The *ratio* of a cut $(S, V \setminus S)$ is

$$\frac{c(\delta(S))}{|S| \cdot |V \setminus S|} = \frac{c \cdot d_S}{1_N \cdot d_S}$$

The cut $(S, V \setminus S)$ is called *sparsest cut*, if it has minimum ratio among all $\emptyset \neq S \subsetneq V$. The *sparsest cut problem* consists of finding a sparsest cut in a given graph.

A related quantity is the *flux*[1]:

Definition 2.4. The *flux* of a cut $(S, V \setminus S)$ is

$$\frac{c(\delta(S))}{\min(|S|, |V \setminus S|)}$$

The *minimum flux cut problem* consists of finding a cut with minimum flux.

The connection between the sparsest cut and the minimum flux cut is formalized by the following:

Proposition 2.1. *Let s be the ratio of the sparsest cut, and f be the minimum flux. Then $\frac{n}{2}s \leq f \leq ns$.*

Thus, it suffices to develop approximation algorithms for the sparsest cut problem, and we get automatically analogous approximation results for the minimum flux cut problem.

A generalization of the sparsest cut problem arises in the context of multicommodity flows. Now, beside the graph G and the cost function c (which is called capacity function in this context), k pairs of vertices are also given: s_i is the source of commodity i, and t_i is the sink of commodity i ($i = 1, \ldots, k$). Note that a vertex can act as the source or destination of multiple commodities. Furthermore, a demand $\mathsf{dem}_i \in \mathbb{R}_+$ is given for all commodities. Just as above, dem will denote both a k-dimensional vector, indexed by commodities, and an N-dimensional vector, indexed by node pairs, padded with zeros accordingly.

Definition 2.5. The *maximum concurrent flow problem* consists of finding the highest $f \in \mathbb{R}_+$ such that for each commodity i an amount of $\mathsf{dem}_i \cdot f$ of that commodity can be simultaneously routed from s_i to t_i, subject to conservation and capacity constraints.

Note that throughout this work in all formulations of multicommodity flow problems, the aim is to find an optimal *fractional flow*, i.e. integrality of the flow is not required.

The dual of the maximum concurrent flow is a relaxation of the following problem:

Definition 2.6. The demand of a cut $(S, V \setminus S)$ is $\mathsf{dem}(S) = \sum_{i:|S \cap \{s_i, t_i\}|=1} \mathsf{dem}_i = \mathsf{dem} \cdot d_S$. The generalized ratio of the cut is

$$\frac{c(\delta(S))}{\mathsf{dem}(S)} = \frac{c \cdot d_S}{\mathsf{dem} \cdot d_S}$$

The *generalized sparsest cut problem* consists of finding a cut for which the generalized ratio is minimal among all $\emptyset \neq S \subsetneq V$.

[1]There is much confusion with the names of these quantities. The flux is sometimes called expansion or quotient cost or sometimes even ratio. Some authors call the minimum flux cut sparsest cut.

It can be seen that the generalized sparsest cut problem contains the sparsest cut problem as a special case, in which all pairs of vertices define a commodity with demand 1. In this special case, the maximum concurrent flow problem is called the *maximum uniform concurrent flow problem.*

It can also be easily seen that the generalized sparsest cut is an upper bound on the maximum concurrent flow. (And consequently, the sparsest cut is an upper bound on the maximum uniform concurrent flow.) Moreover, the special case in which there is only one commodity corresponds to the well-known maximum flow and minimum $s-t$-cut, and the two quantities are equal. However, equality does not hold in general.

Another possible generalization of the maximum flow problem is the following:

Definition 2.7. Given a graph G with edge costs c and k source-sink pairs, the *maximum throughput problem* consists of finding a multi-commodity flow – subject to conservation and capacity constraints – that maximizes the sum of the routed amounts of commodities.

The corresponding cut problem:

Definition 2.8. Given a graph G with edge costs c and the pairs (s_i, t_i) for $i = 1, \ldots, k$, the *minimum multicut problem* consists of finding a minimum cost set of edges that splits each s_i from its corresponding t_i.

Again, it can be easily seen that the value of the minimum multicut is an upper bound on the maximum throughput, and for $k = 1$ equality holds, but not for $k > 1$.

A related problem is the multiway cut:

Definition 2.9. Given a graph G with edge costs c and k terminals $t_1, \ldots, t_k$, the *minimum multiway cut problem* consists of finding a minimum cost set of edges that splits each terminal from all the others.

As mentioned in Chapter 1, many applications require cuts in which the number of vertices in the resulting components is roughly the same. The objective function of the sparsest cut problem includes a bias into this direction; however, such a balance criterion can also be explicitly formulated.[2]

Definition 2.10. Let $1/2 \le \beta \le 1$. A cut $(S, V \setminus S)$ is a *β-balanced cut* if $|S| \le \beta n$ and $|V \setminus S| \le \beta n$. The *minimum β-balanced cut problem* consists of finding a β-balanced cut with minimum cost. The minimum 1/2-balanced problem is also called *graph bisection problem*, sometimes relaxed this way: $|S| \le \lceil \frac{1}{2} n \rceil$ and $|V \setminus S| \le \lceil \frac{1}{2} n \rceil$.

Definition 2.11. Let $0 < \gamma \le 1$. A *γ-separator* is a set of edges whose removal partitions the graph into connected components of size at most γn. The *minimum γ-separator problem* consists of finding a γ-separator of minimum cost.

Sometimes these problems are formulated in a more general way: each vertex has a weight, and not the cardinality of the components but their weights have to obey the balance criteria. For simplicity, we will only handle the special case defined above, but the presented algorithms can be generalized to the weighted problems.

2.3 LP formulations

In this section, linear programming formulations are presented for the generalized sparsest cut problem (Section 2.3.1), the minimum multicut problem (Section 2.3.2), and the minimum multiway cut problem (Section 2.3.3). In the first two cases, we start with LP formulations for some multicommodity flow problems, and get to the LP for the cut problems by duality. In the third case, the LP of the cut problem is formulated directly. In all cases, the first LPs have exponential size; therefore, we have to formulate equivalent, polynomial-size LPs. As it turns out, this can be done by rewriting the program in terms of graph metrics.

[2]As a matter of fact, the sparsest cut problem has no direct practical relevance, only indirectly, because it has been found helpful in devising approximation algorithms for these, practically more important problems. The other reason why it has been widely studied is its connection to multicommodity flows.

2.3.1 The (generalized) sparsest cut problem

We start with an LP formulation for the maximum concurrent flow problem. Let $\{q_1^i, q_2^i, \ldots\}$ be an enumeration of the paths from s_i to t_i. We will also use q_j^i to denote the characteristic function of this path, i.e. $q_j^i(e)$ is 1 if $e \in q_j^i$ and 0 otherwise. In a solution of the maximum concurrent flow problem, f_j^i denotes the amount of commodity i flowing along the path q_j^i, and f denotes the concurrent flow, i.e. the minimum fraction of the demand that is satisfied.

LP 2.1. maximize f, subject to

$\forall e \in E$:	$\sum_{i,j} q_j^i(e) f_j^i \leq c_e$	(capacity constraint)
$\forall i = 1, \ldots, k$:	$\sum_j f_j^i \geq f \cdot \mathsf{dem}_i$	(at least fraction f of the demand is satisfied)
$\forall i, j$:	$f_j^i \geq 0$	

Of course, this LP might be of exponential size, but this does not cause any problems, as will be shown later. The dual is:

LP 2.2. minimize $c \cdot l$, subject to

$\forall i, j$:	$\lambda_i \leq \sum_e q_j^i(e) l_e$
	$\sum_i \mathsf{dem}_i \lambda_i = 1$
$\forall i = 1, \ldots, k$:	$\lambda_i \geq 0$
$\forall e \in E$:	$l_e \geq 0$

Note that in an optimal solution of LP 2.2, $\lambda_i = \mathsf{dist}_l(s_i, t_i)$. LP 2.2 has a pictorial interpretation. Suppose that the edges of the graph represent a system of *pipes*; c_e denotes the cross-section area of pipe e, which is given in advance. The aim is to determine the lengths of the pipes – denoted by l_e – so that the total volume of the pipe system is minimized. However, the edge lengths cannot all be chosen arbitrarily small because the weighted sum of the $s_i - t_i$ distances must be 1.

We claim that LP 2.2 is a relaxation of the generalized sparsest cut problem. In order to see this, first let us formulate a „fractional" LP (i.e. a program with linear constraints but rational objective function) relaxation of the generalized sparsest cut problem:

LP 2.3. minimize $\frac{c \cdot l}{\sum_i \mathsf{dem}_i \lambda_i}$, subject to

$\forall i, j$:	$\lambda_i \leq \sum_e q_j^i(e) l_e$
$\forall i = 1, \ldots, k$:	$\lambda_i \geq 0$
$\forall e \in E$:	$l_e \geq 0$

Note that – just as in the case of LP 2.2 – in an optimal solution of LP 2.3, $\lambda_i = \mathsf{dist}_l(s_i, t_i)$. It is now easy to see the correspondence between the last two LPs and the generalized sparsest cut problem:

Proposition 2.2. If the constraint $l_e \in \{0, 1\}$ is added to LP 2.3, then the optimum of the resulting integer program equals the ratio of the generalized sparsest cut.

Proposition 2.3. The optimum of LP 2.2 equals the optimum of LP 2.3.

With $\lambda_i = \mathsf{dist}_l(s_i, t_i)$, a connection appears between these linear programs and graph metrics. This becomes even more apparent in the following variant:

LP 2.4. minimize $c \cdot d$, subject to

	$\sum_i \mathsf{dem}_i d_{s_i, t_i} = 1$
$\forall u, v, w \in V$:	$d_{u,w} \leq d_{u,v} + d_{v,w}$
$\forall u, v \in V$:	$d_{u,v} \geq 0$

That is, a metric with the following properties has to be found: (i) the sum of the $s_i - t_i$ distances, weighted by the demands is one; (ii) the sum of all N distances weighted by the edge costs is minimal. Note also that the size of LP 2.4 (unlike the previous LPs) is polynomial.

Proposition 2.4. The optimum of LP 2.4 equals the optimum of LP 2.2.

2.3.2 The minimum multicut problem

Now we start with an LP formulation for the maximum throughput problem:

LP 2.5. maximize $\sum_{i,j} f_j^i$, subject to

$\forall e \in E: \quad \sum_{i,j} q_j^i(e) f_j^i \le c_e \quad$ (capacity constraint)

$\forall i,j: \quad f_j^i \ge 0$

Its dual is:

LP 2.6. minimize $c \cdot l$, subject to

$\forall i,j: \quad \sum_e q_j^i(e) l_e \ge 1$

$\forall e \in E: \quad l_e \ge 0$

Proposition 2.5. If the constraint $l_e \in \{0,1\}$ is added to LP 2.6, then the optimum of the resulting integer program equals the cost of the minimum multicut.

Again, the size of this LP might be exponential, but again, an equivalent, polynomial-size LP can be given. The key observation is that the constraints in LP 2.6 are equivalent to stating that for each i, $\mathsf{dist}_l(s_i, t_i) \ge 1$. This can be formulated in a more compact manner using a potential π^i for each $s_i - t_i$ pair:

LP 2.7. minimize $c \cdot l$, subject to

$\forall (u,v) \in E, i = 1, \ldots, k: \quad \pi_u^i - \pi_v^i \le l_{u,v} \quad$ (potential property)

$\forall i = 1, \ldots, k: \quad \pi_{s_i}^i - \pi_{t_i}^i \ge 1 \quad$ (distance constraint)

$\forall v \in V, i = 1, \ldots, k: \quad \pi_v^i \ge 0$

$\forall e \in E: \quad l_e \ge 0$

Proposition 2.6. The optimum of LP 2.6 equals the optimum of LP 2.7.

2.3.3 The minimum multiway cut problem

Let T denote the set of terminals. Similarly to LP 2.4, we can formulate the following LP:

LP 2.8. minimize $c \cdot d$, subject to

$\forall t, t' \in T \ (t \ne t'): \quad d_{t,t'} = 1$

$\forall u, v, w \in V: \quad d_{u,w} \le d_{u,v} + d_{v,w}$

$\forall u, v \in V: \quad d_{u,v} \ge 0$

Proposition 2.7. If the constraint $d_{u,v} \in \{0,1\}$ is added to LP 2.8, then the optimum of the resulting integer program equals the cost of the minimum multiway cut.

However, the integrality ratio of this relaxation is too high for our purposes [10]. Therefore, we strengthen the relaxation with additional constraints that are valid for the integral solutions:

LP 2.9. minimize $c \cdot d$, subject to

$$\begin{array}{ll} \forall t, t' \in T \ (t \neq t'): & d_{t,t'} = 1 \\ \forall u, v, w \in V: & d_{u,w} \leq d_{u,v} + d_{v,w} \\ \forall u, v \in V: & d_{u,v} \geq 0 \\ \forall v \in V: & \sum_{t \in T} d_{v,t} = k - 1 \\ \forall u, v \in V, \forall S \subseteq T: & d_{u,v} \geq \sum_{t \in S} (d_{u,t} - d_{v,t}) \end{array}$$

Proposition 2.8. If the constraint $d_{u,v} \in \{0, 1\}$ is added to LP 2.9, then the optimum of the resulting integer program equals the cost of the minimum multiway cut.

In the last program, the number of constraints is exponential. However, it is not difficult to formulate an equivalent program of polynomial size:

LP 2.10. minimize $c \cdot d$, subject to

$$\begin{array}{ll} \forall t, t' \in T \ (t \neq t'): & d_{t,t'} = 1 \\ \forall u, v, w \in V: & d_{u,w} \leq d_{u,v} + d_{v,w} \\ \forall u, v \in V: & d_{u,v} \geq 0 \\ \forall v \in V: & \sum_{t \in T} d_{v,t} = k - 1 \\ \forall u, v \in V: & d_{u,v} \geq \sum_{t \in T} d'(u, v, t) \\ \forall u, v \in V, \forall t \in T: & d'(u, v, t) \geq d_{u,t} - d_{v,t} \\ \forall u, v \in V, \forall t \in T: & d'(u, v, t) \geq 0 \end{array}$$

Proposition 2.9. The optimum of LP 2.9 equals the optimum of LP 2.10.

2.4 Lower bounds on the integrality ratio

All the presented algorithms work by solving the LP-relaxation of the problem, and then rounding it to an integral solution. The approximation factors are proven with respect to the value of the fractional optimum, which is of course a lower bound on the integral optimum. Hence, the best approximation factor achievable this way is the ratio between the integral optimum and the fractional optimum. This ratio is called the *integrality ratio* of the problem.

The presented results can also be regarded as upper bounds on the integrality ratio of the given relaxations. Now we present some lower bounds:

Theorem 2.10 (Leighton & Rao, 1988). There are graphs for which the integrality ratio of the presented relaxation of the sparsest cut problem is $\Omega(\log n)$.

Theorem 2.11 (Garg, Vazirani & Yannakakis, 1994). There are graphs for which the integrality ratio of the presented relaxation of the minimum multicut problem is $\Omega(\log k)$.

It follows that using this type of method, no better approximation factors can be guaranteed than $O(\log n)$ and $O(\log k)$, respectively. Since the algorithms that we present in the next chapters achieve this approximation ratio, it follows that they are optimal in this sense, up to constant factors.

Chapter 3

Shortest-paths metrics

In this chapter, we review approximation algorithms based on shortest-paths metrics. This approach goes back to the work of Leighton and Rao in 1988, which was later improved by Garg, Vazirani, and Yannakakis [19]. The key to this approach is a procedure for growing regions, i.e. balls in the metric space defined by the shortest paths. As it turns out, these balls can be grown in such a way that the resulting parts define sufficiently small cuts.

First, the region-growing procedure is described in detail (Section 3.1). Afterwards, it is shown how this generic procedure can be used for approximating the minimum multicut problem (Section 3.2) and the sparsest cut problem (Section 3.3). Section 3.4 briefly describes how this method can be used to approximate the generalized sparsest cut problem; however, we do not go into detail on this, because the results are inferior to those achievable using other methods presented later. Finally, it is shown in Section 3.5 how the result on sparsest cuts implies an approximation result on β-balanced cuts.

3.1 The region-growing algorithm

3.1.1 Growing a single region

The input for the region-growing algorithm is a graph G with edge costs c, as well as a distance labeling l. (In the applications, l will be the solution of the LP relaxation of the problem.) Moreover, a *root vertex* $v_0 \in V$, from which the region growing starts, is also given, as well as the parameters $\varepsilon \in \mathbb{R}_+$ and $q \in \mathbb{N}$. We will use the notation $B = c \cdot l$. (In the applications, B will be the optimum of the LP, which is exactly the 'volume' corresponding to the solution l.)

The region growing procedure essentially mimics Dijkstra's algorithm to find the shortest paths from v_0. That is, it maintains an *active set*, denoted by A, which is at the beginning $\{v_0\}$, and is extended by one new vertex in each step. The vertices enter A in the order of their distance from v_0.

The aim is to grow a region with small radius and small cut (i.e. $c(\delta(A))$ should be small). In order to formalize these properties, the algorithm is formulated as a *cut packing* procedure. That is, for each set $S \subseteq V$ we define a variable y_S; these variables will always satisfy the *packing property*: $\forall e \in E : \sum_{S: e \in \delta(S)} y_S \le l_e$. At the beginning, each y_S is set to 0. Only the y variable corresponding to the current active set is raised; therefore, the sets with non-zero y values are all subsets of A, and they form a chain.

The *weight* of the set A is defined as

$$\mathsf{weight}(A) = B/q + \sum_{S \subseteq A} c(\delta(S)) y_S \tag{3.1}$$

The set A is said to have a *small cut*, if

$$c(\delta(A)) \le \varepsilon \cdot \mathsf{weight}(A) \tag{3.2}$$

Now the algorithm can be formulated as follows:

Algorithm 3.1.
1. $A = \{v_0\}$.
2. If the cut corresponding to A is small, then output A and stop.
3. Raise y_A as long as the packing property allows it, i.e. until for some edge $(u, v) \in \delta(A)$ (with $u \in A$) $\sum_{S:(u,v)\in\delta(S)} y_S = l_{(u,v)}$ holds.
4. Let $A := A \cup \{v\}$, and goto 2.

That is, the algorithm outputs the first region with small cut.

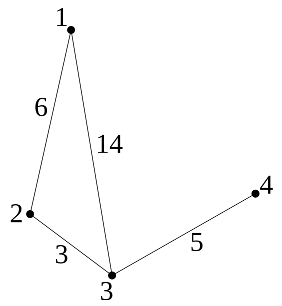

Figure 3.1: Example graph.

Before we get into the analysis of this algorithm, the notion of cut packing and its connection to Dijkstra's algorithm is presented on an example. The example graph is shown in Figure 3.1. The nodes of the graph are numbered from 1 to 4; the numbers near the edges are the edge costs.

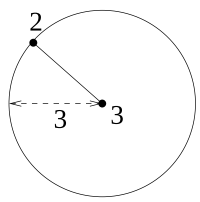

Figure 3.2: The grown region reaches the first node.

The algorithm is started with node 3 as v_0; $A = \{3\}$. All y variables are set to 0. The y variable corresponding to A is slowly increased. This is possible until the packing property is satisfied with equality for one the edges in $\delta(A)$. Clearly, this happens when $y_A = 3$, and the edge between node 3 and node 2 prohibits further increase. This situation is shown in Figure 3.2.

At this point, node 2 is added to A. The variable $y_{\{3\}}$ is not changed anymore; $y_{\{2,3\}}$ is increased as long as the edges in $\delta(A)$ allow. This is possible until $y_{\{2,3\}}$ reaches the value 2, and node 4 is reached. This situation is shown in Figure 3.3.

At this point, node 4 is added to A. The variable $y_{\{2,3\}}$ is not changed anymore; $y_{\{2,3,4\}}$ is increased as long as the edges in $\delta(A)$ allow. This is possible until $y_{\{2,3,4\}}$ reaches the value 4, and node 1 is reached. This final situation is shown in Figure 3.4.

Now let us get back to the analysis of Algorithm 3.1. First some notation is introduced. In the following, the active set of the ith iteration will be denoted by A_i, the output of the algorithm by R. Define the *radius* of A as $\mathsf{rad}(A) = \sum_{S\subseteq A} y_S$. The rad operator has the following properties:

Proposition 3.1.
1. Let $v \in R$ and let A_i be the first set containing v. Then $\mathsf{dist}_l(v_0, v) = \mathsf{rad}(A_{i-1})$.
2. $\forall v \in R : \mathsf{dist}_l(v_0, v) \leq \mathsf{rad}(R)$.

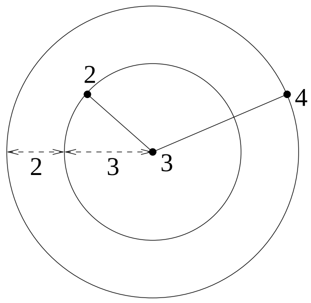

Figure 3.3: The grown region reaches the second node.

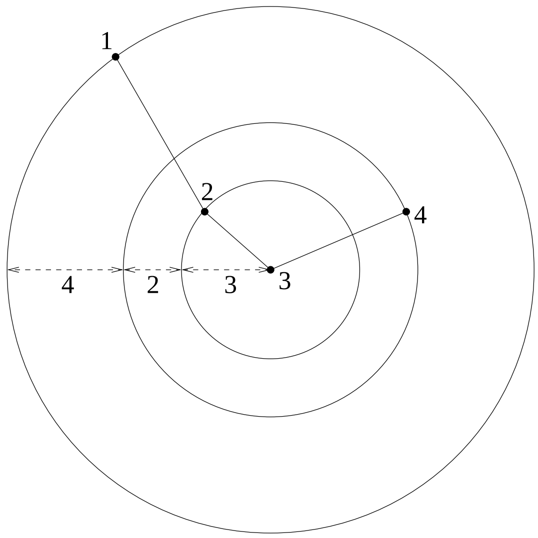

Figure 3.4: The grown region reaches the last node.

3. $\forall u, v \in R : \mathsf{dist}_l(u, v) \leq 2 \cdot \mathsf{rad}(R)$.

As already mentioned, the region growing procedure is expected to output a region with small radius and small cut. It is clear from Algorithm 3.1 that R has a small cut. The following lemma bounds its radius:

Lemma 3.2. $\mathsf{rad}(R) < \frac{\ln(1+q)}{\varepsilon}$.

So now we have seen that the region growing procedure outputs a set with small cut (in the sense of Equation 3.2) and small radius (in the sense of Lemma 3.2). This will be used in the next sections.

3.1.2 Growing multiple regions

We now explain how multiple invocations of the above region growing procedure can be used of chop off several parts of the graph in such a way that even the sum of their cuts is small.

Instead of the root vertex v_0, we are now given a set of candidate root vertices $V_0 \subseteq V$. We choose an arbitrary $v_1 \in V_0$, and perform the region growing procedure starting from v_1. The resulting set is denoted by R_1. We continue by removing the vertices in R_1 from the graph, picking a new root from V_0 and so on, until the remaining graph contains no more vertices from

V_0. In each invocation of the region growing procedure, q is fixed to $|V_0|$. The resulting sets are $R_1, \ldots, R_p$ with some $p \leq q$. Let $M = \delta(R_1) \cup \ldots \cup \delta(R_p)$.

The following lemma states that not only is the cost of each cut $\delta(R_i)$ small, but so is even their sum:

Lemma 3.3. 1. $c(M) \leq 2\varepsilon B$

2. $\sum_{i=1}^{p} c(\delta(R_i)) \leq 4\varepsilon B$

So we have seen that this procedure can be used to chop off parts of the graph in such a way that even the sum of their cuts can be bound with respect to B. This will be used in the next two sections for devising approximation algorithms for the minimum multicut and the sparsest cut problem, respectively.

3.2 Approximating the minimum multicut

Given an instance of the minimum multicut problem, we can find an approximately optimal multicut as follows. First, solve the LP relaxation of the problem (LP 2.7). This yields a length vector l. Then apply the algorithm for growing multiple regions with this length assignment and with the following settings: $V_0 = \{s_1, \ldots, s_k\}$, and $\varepsilon = 2\ln(k+1)$. We output M.

Lemma 3.4. *M is a multicut; the cost of M is $O(\log k)$ times the optimum multicut.*

Proof. Our first aim is to show that M is a multicut. Clearly, all sources are contained in $R_1 \cup \ldots \cup R_p$. Because of the choice of ε and q, Lemma 3.2 guarantees that $\mathsf{rad}(R_i) < 1/2$. Together with Proposition 3.1, this shows that the distance (with respect to l) between any two vertices in R_i is less than 1. Since l is a solution of LP 2.7, $\mathsf{dist}_l(s_i, t_i) \geq 1$ for each i, and hence, M splits s_i and t_i for each i, i.e. M is indeed a multicut.

Now we move on to prove the claim on the cost of M. According to Lemma 3.3, the cost of M is at most $4B\ln(k+1)$. Since, by duality, B equals the maximum throughput, and the maximum throughput is at most the minimum multicut, it follows that $c(M)$ is at most a factor of $O(\log k)$ times the minimum multicut. □

Thus we have an $O(\log k)$-approximation algorithm for the minimum multicut problem. Another consequence is the following approximate max-flow-min-cut theorem:

Theorem 3.5 (Garg, Vazirani & Yannakakis, 1995). Maximum throughput $\leq$ minimum multicut $\leq O(\log k)$·maximum throughput. □

Moreover, according to Theorem 2.11, there are graphs for which the ratio between the maximum throughput and the minimum multicut is $\Omega(\log k)$. Thus, this theorem is tight, and no better approximation algorithm can be given using this approach (up to constant factors).

3.3 Approximating the sparsest cut

The aim of this section is to devise an $O(\log n)$ approximation algorithm for the sparsest cut problem, based on the above methods. The results presented here are originally due to Leighton and Rao [32], but here we present the tighter version given by Garg, Vazirani, and Yannakakis [19], and use the presentation of Cheriyan and Ravi [8].

Given an instance of the sparsest cut problem, we proceed as follows. First, we solve the LP-relaxation of the problem (LP 2.4 in the special case with $\mathsf{dem} = 1_N$) to obtain the length values l. As before, the LP-optimum is denoted by B. Our aim is to find a cut with ratio at most $B' = (2 + 8\ln(1+n))B$.

We carry out the procedure for growing multiple regions, with the following parameters: $V_0 = V$, $\varepsilon = B'n^2/(8B)$. The resulting sets are $R_1, \ldots, R_p$. Clearly, if the cut corresponding to one of the R_i-s has ratio at most B', then we are done.

Lemma 3.6. If none of the sets $R_1, \ldots, R_p$ defines a cut with ratio at most B', then $\exists 1 \leq i \leq p$: $|R_i| \geq n/2$.

The set with cardinality at least $n/2$ will be denoted by R^*. We reset all y_S variables, for which $S \not\subseteq R^*$, to zero. Next, we continue to grow R^* further, but now with a slight modification of the single-region-growing procedure: we do not check for the cut-cost condition, but rather continue the growing procedure until all vertices are added to the active set. The active sets in this phase are: $R^* = A_i \subset A_{i+1} \subset \ldots \subset A_j = V$.

Lemma 3.7. $\exists i \leq r \leq j$ for which the ratio of $\delta(A_r)$ is at most B'.

It follows that this algorithm always finds a cut with ratio at most $B' = O(\log n)B$, and thus it is an $O(\log n)$-approximation algorithm. Another consequence is the following approximate maximum-flow-minimum-cut theorem:

Theorem 3.8 (Leighton & Rao, 1988). Maximum uniform concurrent flow $\leq$ sparsest cut $\leq O(\log n)\cdot$maximum uniform concurrent flow. □

Again, it follows from Theorem 2.10, that this theorem is tight and no better approximation algorithms can be constructed for it using this method, up to constant factors.

3.4 Notes on the generalized sparsest cut

The above techniques can be used to devise an $O(\log k \log \sum \mathsf{dem}_i)$ approximation algorithm for the generalized sparsest cut problem [19]. Using the technique of Plotkin and Tardos for scaling the demands [40], this can be improved to $O(\log^2 k)$.

Furthermore, Kahale proved that an α-approximation algorithm for the minimum multicut problem induces an $\alpha \cdot O(\log k)$-approximation algorithm for the generalized sparsest cut problem [26]. (Kahale's approach also makes heavy use of the shortest-paths metric defined by the optimal solution to the LP-relaxation of the problem.) The best approximation algorithm currently known for the minimum multicut problem has an approximation factor of $O(\log k)$ (see above), thus also yielding an $O(\log^2 k)$-approximation algorithm for the generalized sparsest cut problem.

However, we do not go into detail concerning these approaches, because $O(\log k)$-approximation algorithms are also known for this problem. Such an algorithm will be presented in Chapter 5.

3.5 From sparsest cuts to β-balanced cuts

Now that we have an approximation algorithm for the sparsest cut problem, we can also devise a so-called pseudo-approximation algorithm for the minimum β-balanced cut problem.[1] Pseudo-approximation means that the bound on the cost of the produced cut is not with respect to the minimum β-balanced cut, but with respect to the minimum β'-balanced cut for some $\beta' < \beta$.

The algorithm finds an approximately sparsest cut in the graph, removes the smaller part from the graph, finds again the approximately sparsest cut in the remaining graph, again removes the smaller part and so on, until the remaining graph has size at most βn:

Algorithm 3.2.
1. Let $i = 0$ and $G_0 = G$.
2. If $|V(G_i)| \leq \beta n$ then output the cut $(V(G_i), V(G) \setminus V(G_i))$ and stop.
3. Find an α-approximate sparsest cut (A_i, B_i) with $|A_i| \leq |B_i|$ in G_i.
4. Let $G_{i+1} = G_i \setminus A_i$.
5. Let $i = i + 1$ and goto 2.

[1] As already mentioned, this result is the most compelling reason for studying the sparsest cut problem.

The following theorem states that this algorithm is indeed a pseudo-approximation algorithm for the β-balanced cut problem if $\beta \geq \frac{2}{3}$.

Theorem 3.9 (Leighton & Rao, 1988). Suppose that $\beta' < \beta$ and $\beta \geq \frac{2}{3}$. Let $OPT_{\beta'}$ denote the cost of the minimum β'-balanced cut. Then Algorithm 3.2 finds a β-balanced cut of cost $O\left(\frac{\alpha}{\beta-\beta'} \cdot OPT_{\beta'}\right)$.

Since the best known approximation factor for the sparsest cut problem is $\alpha = O(\log n)$ (see Section 3.3), Algorithm 3.2 is also an $O(\log n)$-pseudo-approximation algorithm.

Finally, it should be noted that Algorithm 3.2 invokes the – quite costly – algorithm for computing an approximate sparsest cut $O(n)$ times (since in each iteration of the algorithm, at least one vertex is chopped off the graph, and altogether $(1-\beta)n$ vertices have to be chopped off). Hence, Algorithm 3.2 is rather slow: the time for computing an approximate sparsest cut is dominated by the solution of a linear program, so the running time of Algorithm 3.2 is essentially that of solving $O(n)$ linear programs. A more direct pseudo-approximation algorithm for a generalization of the β-balanced cut problem, involving only the solution of a single linear program, will be presented in the next chapter.

Chapter 4

Spreading metrics

In Section 3.5, we presented an indirect way of approximating the minimum β-balanced cut problem. As already mentioned, balanced cut problems have important practical applications; hence, it is important to construct fast and effective approximation algorithms for them.

In this chapter, we show an approximation result for the minimum γ-separator problem, but with a more direct procedure, yielding a faster algorithm.

First, it should be highlighted that the minimum β-balanced cut problem is a special case of the minimum γ-separator problem, in the following sense:

Proposition 4.1.
1. Every β-balanced cut is also a β-separator.
2. For every $2/3 \leq \gamma \leq 1$, a γ-separator also induces a γ-balanced cut with less or equal cost.

Our aim is to present a pseudo-approximation algorithm for the minimum γ-separator problem. Therefore let us fix $\gamma < \gamma'$, where the goal is to find a γ'-separator, and bound its cost with respect to the cost of the optimal γ-separator.

The presented algorithm makes use of *spreading metrics*. In the following, we will first define the notion of spreading metrics and describe their most important characteristics (Section 4.1). Section 4.2 is devoted to efficiently computing a spreading metric. In Section 4.3 it is demonstrated how, given a spreading metric, the pseudo-approximation for the γ-separator problem works. Finally, in Section 4.4, an effective heuristic, based on spreading metrics, is presented.

4.1 What is a spreading metric?

Strictly speaking, the spreading metric is not necessarily a metric (not even a semi-metric), but only an assignment of length values to the edges, with constraints on the shortest-paths metric it induces. So it would be more appropriate to reserve the name 'spreading metric' to the induced shortest-paths metric; however, we will stick to the original notion.

Spreading metrics were first introduced by Even et al [13] as part of a general optimization framework for a wide class of graph optimization problems. In the case of γ-separators, the methods of this framework can be applied even more directly; this way, we get a highly efficient approximation algorithm for the minimum γ-separator problem [12]. This is going to be presented in this chapter.

Generally, a spreading metric $l : E \to \mathbb{R}_+$ has the following properties:

- Its 'volume' $c \cdot l$ is a lower bound on the optimal solution of the problem;
- Every non-trivial subset of V (i.e. every subset, for which the problem cannot be solved with 0 cost) has a radius with respect to dist_l that is bounded from below by a constant.

The second property means that the spreading metric *spreads* out the non-trivial subsets. This is the origin of the name.

In the context of the minimum γ-separator problem, a spreading metric is defined as follows.

Definition 4.1. $l : E \to \mathbb{R}_+$ is a spreading metric, if

- $c \cdot l$ is a lower bound on the cost of the minimum γ-separator;
- For every subset $S \subseteq V$, for which $|S| > \gamma' n$, and for every $u \in S$, $\exists v \in S : \mathsf{dist}_l(u, v) > \frac{\gamma' - \gamma}{\gamma'}$.

It can be easily seen that the second property in this definition is equivalent to claiming that for every S, for which $|S| > \gamma' n$ (which means that S has to be split by any γ'-separator), the radius of S is higher than $\frac{\gamma' - \gamma}{\gamma'}$.

4.2 Obtaining a spreading metric

A spreading metric can be obtained by solving the following linear program:

LP 4.1. minimize $c \cdot l$, subject to

$\forall q_j^{u,v}$ path from u to v:	$\sum_{e \in E} q_j^{u,v}(e) l_e \geq \lambda_{u,v}$
$\forall S \subseteq V, \forall u \in S$:	$\sum_{v \in S} \lambda_{u,v} \geq \lvert S\rvert - \gamma n$
$\forall e \in E$:	$0 \leq l_e \leq 1$
$\forall u, v \in V$:	$\lambda_{u,v} \geq 0$

Note that it can be assumed that $\lambda_{u,v} = \mathsf{dist}_l(u, v)$. With this observation, it can easily proven that this LP yields indeed a spreading metric:

Lemma 4.2. Let (l, λ) be an optimal solution of LP 4.1. Then l is a spreading metric.

Note that LP 4.1 has exponential size. However:

Lemma 4.3. There is a *polynomial-time separation oracle* for LP 4.1, i.e. a polynomial-time procedure that, given a candidate solution (l, λ), either finds a violated constraint, or proves that it is indeed a solution. Hence, LP 4.1 can be solved optimally in polynomial time using the ellipsoid algorithm [24].

So now, we have a polynomial-time algorithm for computing a spreading metric. However, since our aim is to use the solution of this LP in an approximation algorithm, it might make sense to settle for an approximate solution of the LP as well, if it can be computed significantly faster than the optimal value.

In particular, Even et al. suggest using the framework for fractional packing and covering problems constructed by Plotkin, Tardos, and Shmoys [39] and Young [46]. Using these results, Even et al. show how an $O(1)$-approximate solution of the LP can be found in $\tilde{O}(m^2 n \frac{\gamma'}{\gamma' - \gamma})$ deterministic time, or in $\tilde{O}(m^2 \frac{1}{\gamma' - \gamma})$ expected time using a randomized algorithm [12].

4.3 Approximating the minimum γ-separator

Now everything is ready for explaining the approximation algorithm to the minimum γ-separator problem. We start by computing a spreading metric (or an approximate spreading metric) l. The volume of the spreading metric $c \cdot l$ will be denoted by B.

Our aim is to chop off pieces of the graph such that each piece has size at most $\gamma' n$, and the cut defined by the chopped off pieces is small. The procedure for chopping off the pieces is very similar to the region-growing algorithm of Section 3.1 (which is essentially the algorithm of Dijkstra for determining the distances from a given vertex). The main difference is that now not the radius, but the size of the chopped off pieces has to be small. Therefore, both the notion of the *weight* of the active set and the notion of a *small cut* has to be redefined. Before these new definitions are presented, some notation has to be introduced.

Suppose that the region-growing procedure starts from a vertex v_0. In each step, the active set is a 'ball' of some radius r, i.e. $B(v_0, r) = \{v : \mathsf{dist}_l(v_0, v) < r\}$. Let $E(v_0, r) = E \cap (B(v_0, r) \times B(v_0, r))$ be the edges inside this ball. Now, the weight of $B(v_0, r)$ is defined as follows:

$$\mathsf{weight}(v_0, r) = \frac{B}{n \ln n} + \sum_{e \in E(v_0,r)} c_e l_e + \sum_{(x,y) \in \delta(B(v_0,r)),\ x \in B(v_0,r)} c_{(x,y)}(r - \mathsf{dist}_l(v_0, x))$$

It is useful to compare this definition to the original definition of weight (Equation (3.1) on page 13). A closer look uncovers the similar structure of the two expressions: both contain a term proportional to B, the rest is in both cases the contribution of the edges in the set, with different coefficients.

Now we come to defining what a small cut means. Let $\bar{r} = \frac{\gamma' - \gamma}{\gamma'}$. Suppose that the currently active set is A and w is the next vertex to be added to it, according to the rule of Dijkstra's algorithm. A's cut is said to be small, if the following condition is satisfied:

$$c(\delta(A)) \le \frac{1}{\bar{r}} \ln \left(\frac{\mathsf{weight}(v_0, \bar{r})}{\mathsf{weight}(v_0, 0)} \right) \cdot \mathsf{weight}(v_0, \mathsf{dist}_l(v_0, w))$$

Again, it is useful to compare this with the original definition (3.2) on page 13. In both cases, a small cut is defined as at most a constant fraction of the weight of the currently active set; only the constant is different in the two cases.

Otherwise, the region-growing procedure is the same. In particular, we stop as soon as the currently active set has a small cut (according to the above definition). The output is the current active set, which will be denoted by R.

As already mentioned, the aim is to chop off a piece with small cut and small cardinality. It is clear that R has small cut. The next lemma shows that R has also small cardinality.

Lemma 4.4. $|R| \le \gamma' n$.

This lemma justifies the above redefinition of the weight and the small cut. Using the new definitions, we can chop off pieces with cardinality at most $\gamma' n$, which can then form a γ'-separator. This is why this approach is more direct than that of the previous Chapter: the chopped off pieces satisfy the balance criterion without the need for any further processing.

Now the algorithm for finding a γ'-separator is very simple: the above method is called repetitively to chop off pieces of size at most $\gamma' n$, until the remaining part also has size at most $\gamma' n$.

It remains to show that the cost of the resulting γ'-separator can be bounded using the cost of the optimal γ-separator:

Theorem 4.5 (Even, Naor, Rao & Schieber, 1999). The above algorithm finds a γ'-separator of cost at most $OPT_\gamma \cdot \left(\frac{\gamma'}{\gamma' - \gamma} + o(1) \right) \cdot \ln n$.

Therefore, we now have an $O(\log n)$-pseudo-approximation algorithm for the γ-separator problem. Concerning the running time of the algorithm, one can state that it is polynomial; furthermore, the most costly part is obtaining the spreading metric. Taking into account the results mentioned in Section 4.2 for obtaining an approximate spreading metric, this part can significantly be accelerated, yielding a highly efficient algorithm for the γ-separator problem. Moreover, it is easy to see that if an $O(1)$-approximate spreading metric is used instead of a 'real' spreading metric, the above algorithm remains an $O(\log n)$-pseudo-approximation algorithm, only the constant in the big-O is increased accordingly. (However, as a more thorough analysis uncovers, this constant is still better than those in other approaches for balanced cuts [12].)

With some minor changes, the above algorithm can also be modified to yield an $O(\log(B/c_{min}))$-pseudo-approximation algorithm (which is better in some cases), where c_{min} is the minimum edge cost [12].

4.4 A heuristic based on spreading metrics

In this section, we briefly review a heuristic for the γ-separator problem, that is based on spreading metrics. This approach was suggested by Könemann, Even, and Naor [30]. It is important to note that this is only a heuristic, i.e. there is no guarantee for the cost of the separator it finds; however, there is empirical evidence that it finds separators of small cut in practical problem instances. The reason why we present it is that it reveals an interesting property – and possible use – of spreading metrics.

The key idea is the following. Suppose we have a spreading metric (or approximate spreading metric) l. The above algorithm, that yields an approximately minimal γ-separator, essentially *rounds* the spreading metric to an integral solution of the linear program, corresponding to a γ-separator. The rounding procedure is quite complex, but informally, it has the following property: edges for which l_e is small, are not likely to participate in the resulting separator, whereas edges with a high l_e value are more likely to participate.

Based on this insight, Könemann et al. propose the following procedure. After calculating l, all edges with $l_e \leq t$, for some threshold value t, are contracted. The resulting smaller graph is denoted by G_t. In G_t, each vertex has a weight $w(v)$ corresponding to the number of vertices of G from which it has been created. Then, any available graph partitioning heuristic can be used to find a low-cost γ-separator in the weighted graph G_t. In the last step, the found separator is transformed back to a γ-separator of the original graph.

Könemann et al. used the publicly available partitioning algorithm k-Metis [28] as internal cut heuristic. They report excellent results on practical problem instances. In particular, this hybrid algorithm consistently outperforms k-Metis. This shows that the edges with low l_e values really do not participate in the best γ-separators, and hence they can be really contracted, thus decreasing the size of the problem, without sacrificing quality. Moreover, it is interesting to note that k-Metis itself uses a similar coarsening and uncoarsening strategy, but uses less sophisticated (actually, local) measures to determine which edges to contract. From the fact that this hybrid algorithm outperforms k-Metis, one can deduce that the spreading metric offers high-quality, global information on which edges should be contracted. However, it should also be noted that computing the spreading metric takes much more time than running k-Metis.

Chapter 5

ℓ_p-Embeddings

In the previous chapters, we grew 'balls' with respect to some graph metric, and it turned out that, with the appropriate ball growing algorithm, these balls defined approximately optimal multicuts, sparsest cuts, and γ-separators. This geometric picture suggests that it might be beneficial to embed the graph – as a discrete metric space – into the real space $\mathbb{R}^h$ for some h, and use real geometric means (such as real balls or hyperplanes) for finding low-cost cuts in the graph. A similar approach has been found very useful for approximating the maximum cut [22].

In this chapter, this approach is investigated in connection with minimum cut problems. In particular, we present two results that are based on this paradigm: an approximation algorithm for the generalized sparsest cut (Section 5.2), and another one for the minimum multiway cut (Section 5.3). However, first in Section 5.1, the basic definitions and facts are given.

5.1 Embedding discrete metric spaces in real spaces

In the following, (V, d) is a discrete metric space, i.e. a finite set V with a metric d defined on it (in the applications, V will be the vertex set of the input graph), $n = |V|$.

As real spaces, we consider $\mathbb{R}^h$ for some h, equipped with a metric. As it turns out, two metrics play important roles in the applications: the ℓ_1 metric (also called Manhattan metric): $\|x - y\|_1 = \sum_{i=1}^{h} |x_i - y_i|$; and the ℓ_∞ metric: $\|x - y\|_\infty = \max_{i=1}^{h} |x_i - y_i|$. In the following, ℓ_1^h denotes $\mathbb{R}^h$ equipped with the ℓ_1 metric and ℓ_∞^h denotes $\mathbb{R}^h$ equipped with the ℓ_∞ metric.

Our aim is to embed the discrete metric space (V, d) into a real space, so that standard geometric methods can then be used to find a low-cost cut of the graph. Ideally, we would like to think of the graph as a subspace of the real metric space, with its edge costs defined by the (ℓ_1 or ℓ_∞) metric of the real space. This is possible if an embedding can be found that does not alter the distances. This is captured by the following definition.

Definition 5.1. Let (V, d) and (V', d') be two metric spaces. The mapping $\varphi : V \to V'$ is *isometric* if $\forall u, v \in V : d'(\varphi(u), \varphi(v)) = d(u, v)$.

It is not obvious that every graph can be embedded into an appropriate real space. Moreover, it is not obvious how such an embedding can be constructed (if it is possible at all).

The following proposition shows that in the case of ℓ_∞-embeddability the answer is positive:

Proposition 5.1. Every discrete metric space on n points has an isometric embedding into ℓ_∞^n. Such an embedding can be constructed in polynomial time.

However, as will be shown later, our ultimate goal is to embed the discrete metric space into ℓ_1^h for some appropriate h. Unfortunately, this is generally not possible in an isometric way. (We will show this, or actually, a much stronger version of this claim, later.)

Nevertheless, since we are aiming at approximation algorithms only, it might make sense to allow small changes in the distances. As it turns out, we can guarantee ℓ_1-embeddability for all graphs by allowing small changes in the metric. This is formalized in the following.

Definition 5.2. Let (V, d) and (V', d') be two metric spaces. The mapping $\varphi : V \to V'$ is a *contraction* if $\forall u, v \in V : d'(\varphi(u), \varphi(v)) \leq d(u, v)$ The *distortion* of the contraction φ is $\max\left\{\frac{d(u,v)}{d'(\varphi(u),\varphi(v))} : u, v \in V\right\}$.

As can be easily seen, the distortion of a contraction is always at least 1. Our aim is to construct a contraction with small distortion. The following is a well-known theorem from the field of functional analysis.

Theorem 5.2 (Bourgain, 1985). For each discrete metric space (V, d) there exists a contraction φ into ℓ_1^h with distortion $O(\log n)$, where $h = O(\log^2 n)$.

The original proof of Bourgain is non-constructive [6]. The first constructive proof, making use of a randomized algorithm, is due to Linial, London, and Rabinovich [33]. Later, the same authors presented a deterministic polynomial-time algorithm for the same problem; however, with a worst bound of $O(n^2)$ on h [34]. Fortunately, the value of h is not very important for our applications; it just should be polynomial in n.

In our application, we will actually use a slight generalization of this result:

Theorem 5.3 (Linial, London & Rabinovich, 1995). For each discrete metric space (V, d) and $X \subseteq V$ there exists a contraction φ into ℓ_1^h, so that $\forall u, v \in X : \frac{d(u,v)}{\|\varphi(u)-\varphi(v)\|_1} = O(\log |X|)$. Such a contraction can be found in random polynomial time, in which case $h = O(\log^2 |X|)$, or in deterministic polynomial time, in which case $h = O(n^2)$.

As can be seen, this is really a generalization. Substituting $X = V$ returns the original result. However, it is easy to prove this generalization based on the specialized result.

Another interesting question in connection with Bourgain's theorem is whether the $O(\log n)$ distortion guarantee is tight. In his original paper, Bourgain proved that the best achievable distortion is $\Omega(\log n / \log \log n)$ for certain graphs. However, as will be shown later, the application to graph cuts proves that the best achievable distortion is actually $\Omega(\log n)$, and hence the theorem is tight.

Finally, we will also use the following lemma about ℓ_1-embeddings:

Lemma 5.4. Let (V, d) be a discrete subspace of ℓ_1^h, i.e. d is induced by the ℓ_1-metric. Then d is the conical combination (i.e. linear combination with non-negative coefficients) of at most $h \cdot (n-1)$ cut metrics. The participating cut metrics and their coefficients can be determined in polynomial time.

Proof. First we show that d is the sum of h 1-dimensional ℓ_1-metrics. For this purpose, we define the metric d_i as follows: $d_i(u, v) = |u_i - v_i|$ $(1 \leq i \leq h)$. It is clear that d_i is induced by a 1-dimensional ℓ_1 metric. Furthermore, d is the sum of the d_i metrics, since for any u and v

$$d(u, v) = \|u - v\|_1 = \sum_{i=1}^{h} |u_i - v_i| = \sum_{i=1}^{h} d_i(u, v)$$

Hence, d is indeed the sum of h 1-dimensional ℓ_1-metrics. Therefore, it suffices to address the special case when d is 1-dimensional. The aim is to prove that d is the conical combination of at most $n-1$ cut metrics.

Since d is induced by the ℓ_1^1 metric, we can visualize the vertices of the graph as points on the real line; let their order be $v_1, \ldots, l_n$. Let $\mu_i = d(v_i, v_{i+1})$ and $S_i = \{v_1, \ldots, v_i\}$ $(1 \leq i \leq n-1)$. We claim that $d = \sum_{i=1}^{n-1} \mu_i d_{S_i}$, where d_{S_i} is the cut metric defined by S_i. Indeed, $d(v_i, v_{i+j}) = \sum_{p=i}^{i+j-1} \mu_p = \sum_{p=1}^{n-1} \mu_p d_{S_p}(v_i, v_{i+j})$, since $d_{S_p}(v_i, v_{i+j})$ is 1 if $i \leq p \leq i+j-1$ and 0 otherwise. $\square$

5.2 Approximating the generalized sparsest cut

In Section 3.3, we presented an $O(\log n)$-approximation algorithm for the sparsest cut problem, i.e. the special case in which there is a unit demand between all pairs of vertices. It was also mentioned in Section 3.4, that those techniques lead to an $O(\log^2 k)$-approximation algorithm for the generalized sparsest cut problem. In this section, we present an $O(\log k)$-approximation algorithm for the generalized sparsest cut problem, which is the currently known best result. It was published independently by Aumann and Rabani [5], and Linial, London, and Rabinovich [34].

The algorithm starts by solving the LP-relaxation of the generalized sparsest cut problem (LP 2.4). The optimal solution of this LP is a metric d on the graph G with the following properties:

- the sum of the $s_i - t_i$ distances, weighted by the demands, is one
- the sum of all the N distances, weighted by the edge costs, is minimal

Based on Proposition 5.1, the optimal solution can also be interpreted as follows: it defines an embedding φ into ℓ_∞^n, such that the ℓ_∞-metric on the embedded graph has the above two properties. This embedding will be referred to as the optimal ℓ_∞-embedding.

The following lemma uncovers a remarkable connection between the ratio of the generalized sparsest cut and an optimal embedding into ℓ_1^h, and thus motivates why we are interested in an optimal ℓ_1-embedding in place of an optimal ℓ_∞-embedding.

Lemma 5.5. The ratio of the generalized sparsest cut equals

$$z = \min \left\{ \mathsf{cost}(\varphi) : \varphi \text{ is an embedding of } V \text{ into some real space } \mathbb{R}^h \right\}$$

where, for an embedding $\varphi : V \to \mathbb{R}^h$,

$$\mathsf{cost}(\varphi) = \frac{\sum_{(u,v)\in E} c_{(u,v)} \|\varphi(u) - \varphi(v)\|_1}{\sum_{i=1}^k \mathsf{dem}_i \|\varphi(s_i) - \varphi(t_i)\|_1}$$

Therefore, the generalized sparsest cut problem is equivalent to finding the optimal ℓ_1-embedding, which will be denoted by φ_0. It follows that finding φ_0 is NP-hard. However, using Bourgain's theorem, we can construct a contraction into ℓ_1^h with small distortion. Specifically, we use Theorem 5.3, with $X = \{s_1, \ldots, s_k, t_1, \ldots, t_k\}$ to yield an embedding φ_1, so that φ_1 is a contraction, and

$$\mu = \max_{1\le i\le k} \frac{d(s_i,t_i)}{\|\varphi_1(s_i) - \varphi(t_i)\|_1} = O(\log k)$$

It follows that $\forall 1 \le i \le k : \|\varphi_1(s_i) - \varphi(t_i)\|_1 \ge d_{s_i,t_i}/\mu$, and therefore,

$$\mathsf{cost}(\varphi_1) = \frac{\sum_{(u,v)\in E} c_{(u,v)} \|\varphi_1(u) - \varphi_1(v)\|_1}{\sum_{i=1}^k \mathsf{dem}_i \|\varphi_1(s_i) - \varphi_1(t_i)\|_1} \le \frac{\sum_{(u,v)\in E} c_{(u,v)} d_{u,v}}{\sum_{i=1}^k \mathsf{dem}_i \frac{d_{s_i,t_i}}{\mu}} = \mu\frac{z}{1} = \mu z \tag{5.1}$$

Using Lemma 5.4, we can write

$$\mathsf{cost}(\varphi_1) = \frac{\sum_{(u,v)\in E} \sum_{j=1}^p c_{(u,v)} d_{S_j}(u,v)}{\sum_{i=1}^k \sum_{j=1}^p \mathsf{dem}_i d_{S_j}(s_i,t_i)} = \frac{\sum_{j=1}^p \sum_{(u,v)\in E} c_{(u,v)} d_{S_j}(u,v)}{\sum_{j=1}^p \sum_{i=1}^k \mathsf{dem}_i d_{S_j}(s_i,t_i)}$$

where $d_{S_1}, \ldots, d_{S_p}$ are the cut metrics given by Lemma 5.4 ($p \le hn$).

We will now use the following simple fact:

Proposition 5.6. Let $x_1, \ldots, x_p$ and $y_1, \ldots, y_p$ non-negative numbers. Then

$$\min_{1\le i\le p} \frac{x_i}{y_i} \le \frac{\sum_{i=1}^p x_i}{\sum_{i=1}^p y_i} \le \max_{1\le i\le p} \frac{x_i}{y_i}$$

Using Proposition 5.6, we can write

$$\mathsf{cost}(\varphi_1) \geq \min_{1\leq j\leq p} \frac{\sum_{(u,v)\in E} c_{(u,v)} d_{S_j}(u,v)}{\sum_{i=1}^{k} \mathsf{dem}_i d_{S_j}(s_i,t_i)} \tag{5.2}$$

Let j^* be the index at which the minimum is reached. Notice that the right-hand side of (5.2) is exactly the ratio of the cut defined by S_{j^*}. Therefore, according to (5.2), the ratio of the cut defined by S_{j^*} is at most $\mathsf{cost}(\varphi_1)$. Combining this with (5.1), we have that the ratio of this cut is at most μz, where z is the cost of the generalized sparsest cut. Hence, the cost of this cut is at most a factor of μ off the optimum. Putting all the pieces together:

Theorem 5.7 (Aumann & Rabani, 1994; Linial, London & Rabinovich, 1994). Maximum concurrent flow $\leq$ generalized sparsest cut $\leq O(\log k)\cdot$maximum concurrent flow. A cut with cost at most $O(\log k)$ times the maximum concurrent flow can be found in polynomial time. □

To sum up the algorithm: first the LP-relaxation is solved, and the resulting metric space is embedded into ℓ_1^h as in Bourgain's theorem. Then, the embedded metric is decomposed according to Lemma 5.4 into a conical combination of cut metrics. Finally, the best of these cut metrics in terms of cut ratio is returned.

Aumann and Rabani also proved the following interesting property: after embedding the graph in ℓ_1^h, a cut with ratio at most a factor of $O(\log k)$ off the optimum can be found by *cutting the graph with a hyperplane*.

Note that the obtained theorem is indeed a generalization of the Leighton–Rao theorem (Theorem 3.8). Also note that the factor of $O(\log k)$ originates from the distortion guaranteed by Bourgain's theorem. In other words, if for a class of graphs smaller distortion can be guaranteed, then we get an improved approximation factor.

However, we know from Theorem 2.10, that the gap between the maximum concurrent flow and the generalized sparsest cut can be $\Omega(\log k)$ in the worst case. As a consequence, not only Theorem 5.7, but also Bourgain's theorem is tight.

Finally, it should be noted that the same technique can be applied to the minimum multicut problem as well to yield an $O(\log^2 k)$-approximation algorithm. However, this is not interesting because we have already presented an $O(\log k)$-approximation algorithm for it using the region-growing algorithm. Nevertheless, it is interesting to note that the strength of these two methods are complementary, as shown in Table 5.1.

Table 5.1: Approximation factors achievable with the region-growing algorithm vs. with ℓ_1-embedding

Problem	Region-growing algorithm	ℓ_1-embedding
Sparsest cut	$O(\log n)$	$O(\log n)$
Generalized sparsest cut	$O(\log^2 k)$	$O(\log k)$
Minimum multicut	$O(\log k)$	$O(\log^2 k)$

5.3 Approximating the minimum multiway cut

In this section, we address the minimum multiway cut problem. Unlike for the problems handled so far, we can now formulate an $O(1)$-approximation algorithm. Note that even the first, quite simple combinatorial algorithm for this problem, suggested by Dahlhaus, Johnson, Papadimitriou, Seymour and Yannakakis [11] was an $O(1)$-approximation. More specifically, its approximation factor was $2(1-1/k)$. The next breakthrough was the algorithm of Călinescu, Karloff and Rabani [10], yielding an approximation factor of $3/2-1/k$. This algorithm makes use of a special ℓ_1-embedding that we are going to present here (Section 5.3.1). Beside a better approximation factor, it also provided further improvement possibilities, that will be surveyed in Section 5.3.2.

5.3.1 The basic result

Recall that in Section 2.3, an LP formulation for the minimum multiway cut problem has been given (LP 2.9). Now its connection to ℓ_1-embeddings will be shown.

Let Δ_k denote the following $k-1$-dimensional simplex in $\mathbb{R}^k$: $\Delta_k = \{x \in \mathbb{R}^k : x \geq 0 \wedge x \cdot 1_k = 1\}$. The jth unit vector of $\mathbb{R}^k$ is denoted by e^j ($j = 1, \ldots, k$). The face of Δ_k that is opposite its vertex e^j will be called the jth face of the simplex.

Definition 5.3. $\varphi : V \to \Delta_k$ is a *simplex-embedding* if $\forall t_j \in T : \varphi(t_j) = e^j$. The cost of a simplex-embedding φ is defined as: $\mathsf{cost}(\varphi) = \frac{1}{2}\sum_{(u,v)\in E} c_{u,v}\|\varphi(u) - \varphi(v)\|_1$. Let $z = \min\{\mathsf{cost}(\varphi) : \varphi \text{ is a simplex-embedding}\}$.

Proposition 5.8. z equals the optimum of LP 2.9.

Clearly, z is a lower bound on the value of the minimum multiway cut. In the following, an algorithm is presented for rounding an optimal solution of the above relaxation to a multiway cut with cost at most $z(\frac{3}{2} - \frac{1}{k})$. Let φ be an optimal simplex-embedding. The output of the rounding algorithm is a simplex-embedding φ' that maps every vertex of the graph to one of the vertices of the simplex, thus defining a multiway cut of cost $\mathsf{cost}(\varphi')$.

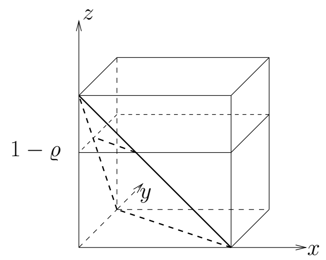

Figure 5.1: The two-dimensional simplex Δ_3. Also marked is the line $\{p \in \Delta_3 : p_j = 1-\varrho\}$ for a given j and ϱ.

The essence of the rounding algorithm is that it maps those vertices of the graph that are 'near' to a given vertex of the simplex, to that vertex of the simplex. More specifically, we fix a number $\varrho \in (0,1)$, and we say that a point $p \in \Delta_k$ is near to vertex e^j of the simplex if $p_j > 1-\varrho$ holds. Note that $\{p \in \mathbb{R}^k : p_j = 1-\varrho\}$ is a hyperplane in $\mathbb{R}^k$ which intersects Δ_k in a hyperplane segment that is parallel to the jth face of the simplex. This hyperplane segment divides the simplex into two parts; the set $\{p \in \Delta_k : p_j > 1-\varrho\}$ is the one containing e^j, i.e. its points are indeed near to e^j. This is illustrated in Figure 5.1.

The algorithm works by considering the vertices of the simplex in a specific order, and assigning to each vertex of the simplex those vertices of the graph that are near to it and have not been assigned to another vertex of the simplex. This is done for the first $k-1$ vertices of the simplex; then all remaining vertices of the graph are mapped to the kth vertex of the simplex. This is illustrated in Figure 5.2. This scheme was called SPARC (Side-PARallel Cuts) by Karger et al. [27].

Now the algorithm can be summarized as follows:

Algorithm 5.1.
1. Let the permutation σ be either $(1, \ldots, k)$ or $(k-1, k-2, \ldots, 1, k)$, each with probability $1/2$.

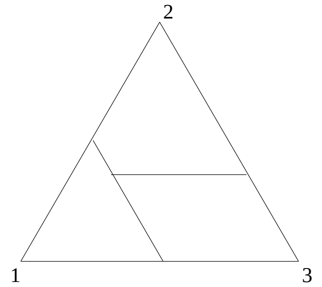

Figure 5.2: Operation of the rounding algorithm

2. Choose a uniform random $\varrho \in (0, 1)$.
3. Let $S = V \setminus T$ (S is the set of vertices of the graph that have to be assigned to a vertex of the simplex)
4. **for** $i = 1$ **to** $k - 1$ **do**: assign the vertices in $S \cap \{v \in V : \varphi(v)_{\sigma_i} > 1 - \varrho\}$ to the ith vertex of the simplex and remove them from S
5. Assign the remaining vertices (i.e. those still in S) to the kth vertex of the simplex.

Denoting the output of the rounding algorithm by φ', the main result can be formulated as follows:

Theorem 5.9 (Călinescu, Karloff & Rabani, 1998). $E(\mathsf{cost}(\varphi')) \leq (\frac{3}{2} - \frac{1}{k}) \cdot \mathsf{cost}(\varphi)$

Here, the expected value is taken with respect to the choice of σ and ϱ. However:

Proposition 5.10. The above rounding algorithm can be derandomized to yield a polynomial-time algorithm.

Thus we have a polynomial-time $(\frac{3}{2} - \frac{1}{k})$-approximation algorithm for the minimum multiway cut problem.

5.3.2 Improvements

This result can be further improved. A natural way of extending the above algorithm is to choose for each $i = 1, \ldots, k$ a new ϱ_i (randomly and independently) instead of using a single ϱ value for all of them. We will refer to this version of the algorithm as the *extended rounding algorithm*.

Theorem 5.11 (Karger, Klein, Stein, Thorup & Young, 1999). With the extended rounding algorithm, $\mathsf{cost}(\varphi') \leq 1.3438 \cdot \mathsf{cost}(\varphi)$. Moreover, the extended rounding algorithm can also be derandomized to yield a polynomial-time approximation algorithm with this approximation factor.

For $k \geq 14$, the approximation factor guaranteed by Theorem 5.11 is better than that guaranteed by Theorem 5.9.

It is currently an open problem whether better rounding algorithms can be given for the presented geometric relaxation of the minimum multiway cut problem. In other words, the integrality ratio of this relaxation is not known. The following two results might be important steps into this direction:

Theorem 5.12 (Freund & Karloff, 2000). The integrality ratio of the presented relaxation is at least $8/\left(7 + \frac{1}{k-1}\right)$.

Theorem 5.13 (Karger, Klein, Stein, Thorup & Young, 1999; Cunningham & Tang, 1999). For $k = 3$, the integrality ratio of the presented relaxation is $12/11$. Moreover, the extended rounding algorithm offers this approximation factor.

These upper and lower bounds on the integrality ratio are shown together in Figure 5.3.

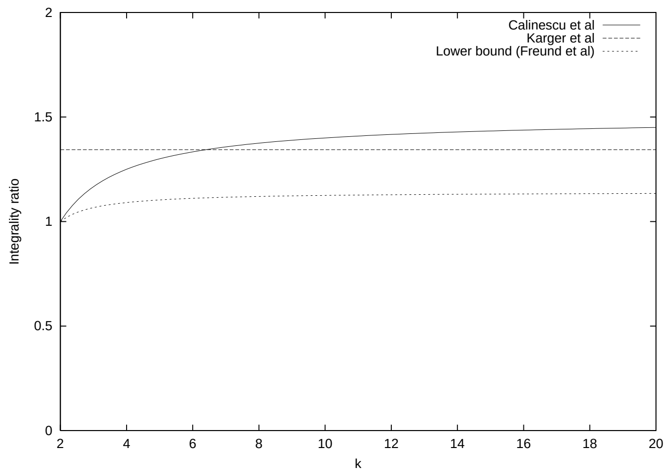

Figure 5.3: Known bounds on the integrality ratio of the presented relaxation of multiway cut

Chapter 6

Conclusions and open problems

In this work, the most important metric-based approximation algorithms for graph cut problems have been reviewed. The reader may wish to refer back to Table 1.1 for an overview of the presented results.

The presented approximation factors are the best known ones. Hence, the best known approximation factors can be achieved using metric-based approaches in the case of most NP-hard graph cut problems. This shows that metric-based approaches are very powerful. On the other hand, the presented algorithms also demonstrate the wide spectrum of metric-based approaches. Thus, probably the most important question for future research is how one can generalize these methods and transfer them to other problem formulations.

Furthermore, for each of the addressed problems, important open questions remain:

(Generalized) Sparsest cut. We presented an algorithm that is provably best possible (up to constant factors) among those that bound the ratio of the resulting cut with respect to the optimum of the presented LP. However, other LP formulations can exist with smaller integrality ratios. Moreover, it is possible that better results can be achieved in a completely different way, i.e. without an LP-relaxation. In particular, finding an $O(1)$-approximation would be a breakthrough.

Minimum multicut. The same applies as in the case of the (generalized) sparsest cut.

Minimum multiway cut. As already discussed, the integrality ratio of the presented geometric relaxation of the problem is not known. It might be possible to construct a better approximation algorithm based on the same relaxation, but with a more sophisticated rounding method. Moreover, it is possible that better approximation factors can be reached by using completely different methods. On the other hand, it is known that, unless P=NP, the approximation factor cannot be arbitrarily close to one. It would be very interesting to know how far the currently known best approximation factors lie from the best possible (unless P=NP).

Minimum β-balanced cut. Just as with the (generalized) sparsest cut: it is an open question whether better approximation results can be achieved using better relaxations, or using completely different methods. However, in contrast to the above problems, here only pseudo-approximations are known. Hence, a real approximation algorithm would be certainly a breakthrough. Moreover, since this problem has many important practical applications, it would also be useful to try to decrease the constant in the $O(\log n)$-approximation, so that it is really applicable in practice. Moreover, the typical running time also plays an important role.

Minimum γ-separator. The same applies as in the case of the minimum β-balanced cut problem.

Moreover, there are hardly any results on the hardness of approximation to these problems. This is certainly also an important direction for future research.

Finally, we would like to mention a related result concerning the minimum bisection. The minimum bisection problem is of course a special case of the minimum β-balanced cut problem; hence the presented pseudo-approximations apply. However, better results are known for this special case. Currently, the best is a real approximation algorithm with an approximation ratio of $O(\log^2 n)$, due to Feige and Krauthgamer [15]. This is a rather complex algorithm, which does not make use of graph metrics, at least not directly. However, it uses an algorithm for finding approximate sparsest cuts as a subroutine.

Bibliography

[1] T. F. Abdelzaher and K. G. Shin. Period-based load partitioning and assignment for large real-time applications. *IEEE Transactions on Computers*, 49(1), 2000.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, 1993.

[3] C. J. Alpert and A. B. Kahng. Recent developments in netlist partitioning: A survey. *VLSI Journal*, 19(1-2):1–81, 1995.

[4] Sanjeev Arora, David Karger, and Marek Karpinski. Polynomial time approximation schemes for dense instances of NP-hard problems. *Journal of Computer and System Sciences (JCSS)*, 58(1):193–210, 1999.

[5] Y. Aumann and Y. Rabani. An O(log k) approximate min-cut max-flow theorem and approximation algorithm. *SIAM Journal on Computing*, 27(1):291–301, 1998.

[6] J. Bourgain. On Lipschitz embedding of finite metric spaces in Hilbert space. *Israel Journal of Mathematics*, 52:46–52, 1985.

[7] T. N. Bui and C. Jones. Finding good approximate vertex and edge partitions is NP-hard. *Information Processing Letters*, 42:153–159, May 1992.

[8] J. Cheriyan and R. Ravi. Approximation algorithms for network problems. `http://www.math.uwaterloo.ca/~jcheriya/PS_files/ln-master.ps`, 1998.

[9] F. R. K. Chung. Separator theorems and their applications. In B. Korte, L. Lovász, H. J. Prömel, and A. Schrijver, editors, *Paths, flows, and VLSI-layout*, pages 17–34. Springer-Verlag, 1990.

[10] G. Călinescu, H. Karloff, and Y. Rabani. An improved approximation algorithm for multiway cut. In *ACM Symposium on Theory of Computing*, pages 48–52, 1998.

[11] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994.

[12] G. Even, J. Naor, S. Rao, and B. Schieber. Fast approximate graph partitioning algorithms. *SIAM Journal on Computing*, 28(6):2187–2214, 1999.

[13] G. Even, J. Naor, S. Rao, and B. Schieber. Divide-and-conquer approximation algorithms via spreading metrics. *Journal of the ACM*, 47(4):585–616, 2000.

[14] Guy Even, Joseph Naor, Baruch Schieber, and Madhu Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998.

[15] U. Feige and R. Krauthgamer. A polylogarithmic approximation of the minimum bisection. *SIAM Journal of Computing*, 31(4):1090–1118, 2002.

[16] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[17] A. Frank. Packing paths, circuits and cuts – a survey. In B. Korte, L. Lovász, H. J. Prömel, and A. Schrijver, editors, *Paths, flows, and VLSI-layout*, pages 47–100. Springer-Verlag, 1990.

[18] N. Garg, V. Vazirani, and M. Yannakakis. Multiway cuts in directed and node weighted graphs. In *Proceedings of the 21st International Colloquium on Automata, Languages and Programming*, pages 487–498, 1994.

[19] N. Garg, V. Vazirani, and M. Yannakakis. Approximate max-flow min-(multi)cut theorems and their applications. *SIAM Journal on Computing*, 25(2):235–251, 1996.

[20] N. Garg, V. V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.

[21] Naveen Garg, Huzur Saran, and Vijay V. Vazirani. Finding separator cuts in planar graphs within twice the optimal. In *FOCS*, pages 14–23, 1994.

[22] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995.

[23] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. *Journal of the ACM*, 35:921–940, 1988.

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

[25] T. C. Hu. Multicommodity network flows. *Operations Research*, 11:344–360, 1963.

[26] N. Kahale. On reducing the cut ratio to the multicut problem. Technical Report TR 93-78, DIMACS, 1993.

[27] D. R. Karger, P. Klein, C. Stein, M. Thorup, and N. E. Young. Rounding algorithms for a geometric embedding of minimum multiway cut. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 668–678, 1999.

[28] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1999.

[29] Philip N. Klein, Serge A. Plotkin, and Satish Rao. Excluded minors, network decomposition, and multicommodity flow. In *Symposium on the Theory of Computing (STOC)*, pages 682–690, 1993.

[30] J. Könemann, G. Even, and J. Naor. Graph partitioning using spreading metrics: An empirical evaluation. Technical Report GSIA Working Paper #2002-E21, GSIA, Carnegie Mellon University, 2002.

[31] T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *IEEE Symposium on Foundations of Computer Science*, pages 422–431, 1988.

[32] T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832, 1999.

[33] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 577–597, 1994.

[34] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995.

[35] Z. Á. Mann and A. Orbán. Optimization problems in system-level synthesis. In *Proceedings of the 3rd Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications*, 2003.

[36] S. T. McCormick, M. R. Rao, and G. Rinaldi. Easy and difficult objective functions for max cut. *Math. Program., Ser. B*, 94(2-3):459–466, 2003.

[37] H. Nagamochi and T. Ibaraki. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM Journal of Discrete Mathematics*, 5:54–66, 1992.

[38] J. Naor and L. Zosin. A 2-approximation algorithm for the directed multiway cut problem. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, 1997.

[39] S. A. Plotkin, D. B. Shmoys, and É. Tardos. Fast approximation algorithms for fractional packing and covering problems. In *FOCS*, pages 495–504, 1991.

[40] S. A. Plotkin and É. Tardos. Improved bounds on the max-flow min-cut ratio for multicommodity flows. *Combinatorica*, 15(3):425–434, 1995.

[41] F. Shahrokhi and D. W. Matula. The maximum concurrent flow problem. *Journal of the ACM*, 37(2):318–334, April 1990.

[42] D. B. Shmoys. Cut problems and their application to divide-and-conquer. In D. S. Hochbaum, editor, *Approximation algorithms*, pages 192–235. PWS Publishers, 1996.

[43] H. D. Simon and S.-H. Teng. How good is recursive bisection? *SIAM Journal on Scientific Computing*, 18(5):1436–1445, 1995.

[44] H. Stone. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Transactions on Software Engineering*, 3(1):85–93, Jan 1977.

[45] F. Vahid and D. Gajski. Clustering for improved system-level functional partitioning. In *Proceedings of the 8th International Symposium on System Synthesis*, 1995.

[46] N. Young. Randomized rounding without solving the linear program. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 170–178, 1995.