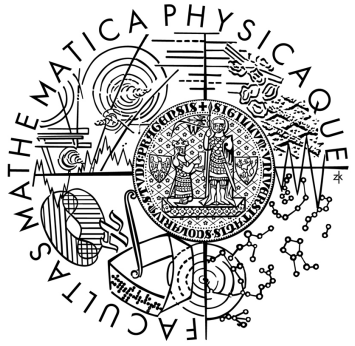
Charles University in Prague
Faculty of Mathematics and Physics

MASTER THESIS



Ján Pecsők

Algorithms for Graph Partitioning

Department of Applied Mathematics

Supervisor of the master thesis: doc. Mgr. Kolman Petr Ph.D
Study programme: Computer science
Specialization: Discrete Models and Algorithms

Prague 2013

I want to thank my supervisor, Petr Kolman, for his patience, advice he gave me and his overall support during writing this thesis. Our communication has always been in a good atmosphere. I have enjoyed our discussions and I liked the feel of cooperation.

I want also to thank my parents for supporting me not only during the time I was writing the thesis, but everytime I really needed it.

I dedicate this thesis to all of my teachers from all phases of my education. I have always feeled lucky to have teachers who not only do their job excellently, but also have great personality I could relate to.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In ..... date ..... signature of the author

Název práce: Algoritmy pro řezy v grafech

Autor: Ján Pecsők

Katedra: Katedra aplikované matematiky

Vedoucí diplomové práce: doc. Mgr. Kolman Petr, Ph.D., KAM

Abstrakt: Problémy hledání řezu v grafu mohou být popsány jako problémy, v kterých jsme žádáni rozdělit graf na 2 nebo více částí. V této práci podáváme přehled metod a konceptů používaných při hledání nejlepších řezů vzhledem k několika kritériím. Dokážeme dualitu mezi problémem hledání multi-komoditního toku a řídkého řezu z práce autorů Leighton a Rao (LR). Dokážeme ji pomocí algoritmu užívajícího lineárního programování a geometrického vnořování. Následně představíme práci autorů Arora, Rao a Vazirani (ARV) a jejich algoritmus založený na semidefinitním programování a také na geometrickém vnořování. Též vysvětlíme koncept expanzních toků poprvé představených v práci ARV. Další rozsáhlá sekce je věnovaná spektrální teorii. Prvky spektrální teorie a koncept expanzních toků se spojí v kapitole o algoritme využívajícího jednokommoditní toky. Nakonec ukážeme výsledky naši implementace varianty algoritmu využívajícího jednokomoditní toky a algoritmu vnořování dle LR.

Klíčová slova: Řezy v grafech, Spektrální teorie, Řídké řezy, Expanze

Title: Algorithms for graph partitioning

Author: Ján Pecsők

Department: Department of Applied Mathematics

Supervisor: doc. Mgr. Kolman Petr, Ph.D., KAM

Abstract: Graph-partitioning problems can be generically defined as a family of problems in which we are asked to partition a graph into two or more components. We present overview of methods and concepts used to find best graph partitions according to several criteria. We prove duality of multi-commodity flow and sparsest cut problem due to work of Leighton and Rao by describing algorithm using a Linear programming relaxation and a geometric embedding. Then we present the work of Arora, Rao and Vazirani (ARV) and their algorithm based on Semidefinite programming relaxation and a geometric embedding. We also explain the concept of expander flows first introduced in the work of ARV. One section of our work is devoted to the spectral graph theory, introducing the concepts of the spectral gap, random walks, conductance and relations between them. We connect the ideas of expander flows and spectral theory in chapter about so called Cut-Matching game framework. Finally we present the performance results of our implementation of the Leighton-Rao and the Cut-Matching game algorithms.

Keywords: Graph partitioning, Spectral theory, Sparse cuts, Expansion, Conductance

Contents

1. Introduction

Graph-partitioning problems can be generically defined as a family of problems in which we are asked to partition a graph into two or more components. As we will see, there are many flavors of these problems, according to what objective we pursue. In this work, we mostly focus on the graph-partitioning problems that partition the vertex set of graph to 2 large parts, such that the number of edges crossing the partition is minimized. Among the other things, the ability to do so is a useful primitive in divide and conquer algorithms for a variety of tasks.

The graph-partitioning problems consist mostly of NP-hard problems and thanks to their numerous applications in science and engineering, they are central topic of research in the study of approximation algorithms. The graph partitioning problems have their place in both theory and practice. Research in this area has great impact on advances in other fields, such as spectral graph theory [41], metric embeddings [10, 56], mixing of Markov chains [43] and so on. As we will see, all the different approaches to graph partitioning problems use knowledge from all of these fields and by exploring and deepening these approaches, they bring new results in theory. The graph-partitioning problems are also connected with clustering problems [47]. Roughly speaking, clustering is the process of organizing objects into groups whose members are similar in some way. Practical areas that use the graph partitioning algorithms include image segmentation [44], social-network analysis [55], PRAM emulation, packet routing in distributed networks [12], VLSI circuit placement [19, 68], web-data graph analysis, sparse matrix factorization [17], parallel computing [52] and much more.

As was mentioned above, our focus will be finding 2-part graph partitions by minimizing the number of edges between the parts. If we denote by $E(S, \bar{S})$ the set of edges between the vertex sets $S \subset V$,$\bar{S} = V \setminus S$ we can straightforwardly say, that we want to find a graph partition $(S, \bar{S})$ such that $|E(S, \bar{S})|$ is minimized. This objective would lead to definition of MinCut problem, which we will describe later. Although this problem has many practical applications as well, often we want also assure that the size of both parts are about the same. One way to provide this quality, is to measure the ratio of cardinality of edge set $E(S, \bar{S})$ to the size of the vertex set $S \subsetneq V$. There are several standard formalizations of this bi-criterion. One commonly used is an *expansion*. Given an undirected possibly weighted, graph $G = (V, E)$[1] *the edge expansion* $\alpha(S)$ of a set of nodes $\emptyset \neq S \subsetneq V$ is defined as:

$$\alpha(S) = \frac{|E(S, \overline{S})|}{\min\{|S|, |\overline{S}|)\}}. \tag{1.1}$$

Where $E(S, \bar{S})$ denotes the set of edges having one end in S and one end in the complement $\overline{S}$ and where $|\cdot|$ denotes the cardinality (or the weight of the set). The *expansion of the graph* G is then defined as:

$$\alpha(G) = \min_{\emptyset \neq S \subsetneq V} \alpha(S). \tag{1.2}$$

A graph with the expansion at least c is called *c-expander*. Usually we call the graph an *expander* if c is at least constant. Sometimes we need a balanced version

[1] In this text we commonly use an undirected graph $G = (V, E), |V| = n, |E| = m$.

of the expansion $\alpha_c(G)$, where we consider only large subsets of vertices:

$$\alpha_c(G) = \min_{S \subsetneq V, c \cdot |V| \leq |S| \leq 1/2 \cdot |V|} \alpha(S). \tag{1.3}$$

Closely related term is *sparsity* of a graph G denoted by $\mathcal{S}_G$ and defined as:

$$\mathcal{S}_G = \min_{\emptyset \neq S \subsetneq V} \frac{|E(S, \overline{S})|}{|S||\overline{S}|}. \tag{1.4}$$

Observe that the sparsity of graph and the expansion differ only by a constant factor:

$$\frac{n}{2}\mathcal{S}_G \leq \alpha(G) \leq (n-1)\mathcal{S}_G. \tag{1.5}$$

For any n-vertex graph the number of vertices in the big half of a cut must lie between $\frac{n}{2}$ and n. Another term used is a conductance of a graph. *The graph conductance* $\Phi(G)$ is defined as:

$$\Phi(G) = \min_{\emptyset \neq S \subsetneq V} \frac{E(S, \overline{S})}{\min\{\mathrm{vol}(S), \mathrm{vol}(\overline{S}\}}, \tag{1.6}$$

where

$$\mathrm{vol}(S) = \sum_{v \in S} \deg_v. \tag{1.7}$$

The conductance of a graph is often called the Cheeger constant h_G. Observe that for a d-regular graph H holds $\frac{\alpha(G)}{d} = \Phi(G)$. The sparsity, expansion and conductance are tightly interconnected. Various results can be cleanly formulated by using one of these. We mostly refer to the sparsity and expansion in this work. Although the results in spectral graph theory, mainly the Cheeger inequality, use the term conductance. Generaly, if we know something about one of these graph qualities, then we also know the others. More on the interreducibilities between these qualities will be in the section 1.1.3.

1.1 Graph partitioning problems and definitions

1.1.1 Single-commodity flows and Minimum s-t cut problem

A definition of a single-commodity flow problem is well known. We use the web resource [34] for this section.

Definition 1. *(*Single commodity maximum flow problem *and* minimum s-t cut problem*) Let $G = (V, E)$ be a directed graph (network) with $s, t \in V$ being the* source *and* sink *of G respectively. then*

- *The* capacity *of an edge is a mapping $c : E \to R^+$, denoted by c_{uv} or $c(u, v)$. It represents the maximum amount of flow that can pass through an edge. Graph is directed, so it can have defined the capacity for both direction of an edge.*

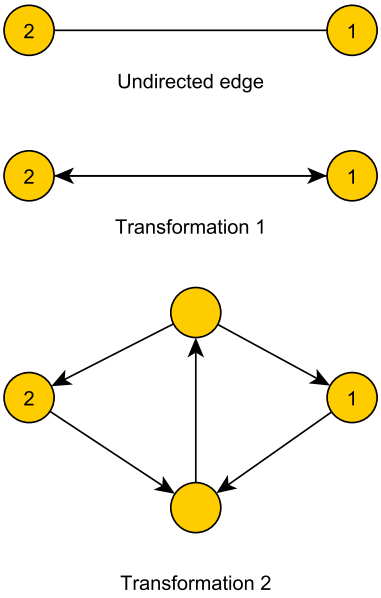


Figure 1.1: For the purpose of solving SINGLE COMMODITY MAXIMUM FLOW PROBLEM, undirected edges can be transformed to the directed edges. We can do it either by simply replacing the edge by 2 directed edges with the same capacity in both directions or by replacing the edge by the graph tranformation 2. The disadvantage of transformation 1 is the fact, that it allows the flow for both directions in the same time, thus creating possibility for a flow circulation. However, many algorithms for SINGLE COMMODITY MAXIMUM FLOW PROBLEM can avoid circulations and this is not a problem.

- *A* flow *is mapping* $f : E \to R^+$*, denoted by* f_{uv} *or* $f(u,v)$*, subject to the following two constraints:*

$$\begin{array}{lcll} f(u,v) & \leq & c(u,v) & \forall (u,v) \in E \quad \textit{capacity constraint} \\ \sum_{u:(u,v)\in E} f(u,v) & = & \sum_{u:(v,u)\in E} f(v,u) & \forall v \in V \setminus \{s,t\} \quad \textit{conservation of flows} \end{array}$$

- *The* value of flow *is defined by* $|f| = \sum_{v\in V} f(s,v)$*, where* s *is the source of* G*. The flow passing from the* source *to the* sink *is represented by this value.*

The SINGLE COMMODITY MAXIMUM FLOW PROBLEM *is to maximize* $|f|$ *, that is, to route as much flow as possible from* s *to* t

- *An* s-t cut $C = (S,T)$ *is a partition of* V *such that* $s \in S$ *and* $t \in T$*. The* cut-set *of* C *is the* $E(S,T) = \{(u,v) \in E | u \in S, v \in T\}$*. Observe, that if all the edges of the set* $E(S,T)$ *are removed, the* value of flow *comes to zero;* $|f| = 0$

- *The* capacity *of an* s-t cut *(*S,T*) is defined by*

$$c(S,T) = \sum_{(u,v)\in E(S,T)} c(u,v)$$

The MINIMUM S-T CUT PROBLEM *is minimizing* $c(S,T)$*, that is, to determine* S *and* T *such that* $s \in S, t \in T$ *and the capacity of the* s-t cut $C = (S,T)$ *is minimal.*

Observe that for the problem definition we used directed graphs. In this work we mostly talk about undirected graphs. So how we can solve SINGLE

COMMODITY MAXIMUM FLOW PROBLEM for undirected graphs? We can do some simple transformations. As we can see in figure 1.1.1.

The following statement proposes the tight duality between MINIMUM S-T CUT PROBLEM and SINGLE COMMODITY MAXIMUM FLOW PROBLEM proved in 1956 independently by Ford and Fulkerson [25] and by Feinstein and Shannon [20]. The theorem can also be proved by an application of the Duality theorem from the theory of linear programming.

Theorem 2. *The maximum value of an* s-t flow *is equal to the minimum capacity over all* s-t cuts

To see that we formulate it as a linear programming problem

$$\begin{aligned}
&\max|f| && && \\
&\text{subject to:} && && \text{(P)}\\
&f(i,j) - c(i,j) \leq 0 && (i,j) \in E && \text{(a)}\\
&\sum_{j:(j,i)\in E} f(i,j) - \sum_{j:(i,j)\in E} f(i,j) \leq 0 && \forall i \in V, i \neq s,t && \text{(b)}\\
&|f| + \sum_{j:(j,s)\in E} f(j,s) - \sum_{j:(s,j)\in E} f(s,j) \leq 0 && && \text{(c)}\\
&-|f| + \sum_{j:(j,t)\in E} f(j,t) - \sum_{j:(t,j)\in E} f(t,j) \leq 0 && && \text{(d)}\\
&f(i,j) \geq 0 && \forall (i,j) \in E &&
\end{aligned}$$

The constraints (b) represent a flow conservation for the internal vertices. The constraints (c) and (d) represent a flow conservation for the *source* and the *sink* vertices respectively. The equalities would be more illustrative but thanks to the fact that LP maximizes $|f|$, we can use inequalities only from one side. Now we formulate dual program to the above linear program. Then we will see that it corresponds to MINIMUM S-T CUT PROBLEM. In dual program, we provide variable for each row in the primal program. Let $d(i,j), (i,j) \in E$ be the variables corresponding to the constraints (a) and the variables $p_i, i \in V \setminus \{s,t\}$ be variables corresponding to the constraints (b) and finally let p_s and p_t be the variables, that correspond to the constraints (c) and (d) respectively. With that in mind, you can see the duality between the following linear program and the one stated above.

$$\begin{aligned}
&\min \sum_{(i,j)\in E} c(i,j)d(i,j) && && \\
&\text{subject to:} && && \text{(D)}\\
&d(i,j) - p_i + p_j \geq 0 && \forall (i,j) \in E && \\
&p_s - p_t \geq 1 && && \\
&p_i \geq 0 && \forall i \in V && \\
&d(i,j) \geq 0 && &&
\end{aligned}$$

Consider a solution of this program, where

$$\begin{aligned} d(i,j) \in \{0,1\},\ (i,j) \in E \\ p_i \in \{0,1\},\ i \in V \setminus \{s,t\} \\ p_s = 1 \\ p_t = 0 \end{aligned} \tag{A}$$

Where an edge (i,j) will be in the cut, if $d(i,j) = 1$. And if the vertex v can be reached from the source, then $p_i = 1$. With these additional constraints the dual LP solves the MINIMUM S-T CUT PROBLEM. To see that (D) has an integral solution, observe that the matrix defining this LP is *totally unimodular*. Which tells us that every extreme point of the polyhedron defining the feasible region is integral and hence the simplex algorithm will return solution satisfying (A).

1.1.2 The formulation of graph partitioning problems

In this section we present the list of NP optimization problems included in book [11]. First we state the definition of the SPARSEST-CUT PROBLEM which together with the EDGE EXPANSION and the GRAPH CONDUCTANCE are in the main focus of this work. In the following sections we describe several algorithms for getting an approximate solution of these problems. In the section 3 we describe multilevel graph partitioning algorithms for $(k, 1+\epsilon)$ - BALANCED PARTITION PROBLEM. The remaining definitions are for readers reference only. Just to see how wide is the area of the graph partitioning problems and what results are known to the date.

Definition 3. *(*SPARSEST-CUT PROBLEM*)Given and undirected graph G=(V,E) and a capacity function $c : E \to R^+$ that assigns a capacity c_E to each edge $e \in E$. Also given a set of demand pairs $(s_1,t_1),(s_2,t_2),\ldots,(s_k,t_k)$ and demand values $d_1,d_2,\ldots,d_k$ such that each demand pair (s_i,t_i) is associated with the demand value d_i for $1 < i < k$.*

Given a set of edges $E' \subset E$ (a cut). Let $c(E') = \sum_{e \in E'} c(e)$. Let $dem(E') = \sum_{i:(s_i,t_i) is\ separated\ by E'} d_i$. Where (s_i,t_i) is separated by E' if they are not connected in $G[E \setminus E']$. Finding the cut $E' \subset E$ that minimizes ratio $\frac{c(E')}{dem(E')}$ is called SPARSEST-CUT PROBLEM.

This is general definition of SPARSEST CUT PROBLEM. If we choose to have demand pair between every 2 nodes in graph and set all demands to 1, we get UNIFORM SPARSEST CUT PROBLEM (UMFP). UMFP can be equally restated as a problem of finding sparsity ratio of a graph as defined in (1.4). Other related problems include:

- EDGE EXPANSION problem is to find *expansion* of a graph as defined in (1.2).
- C-BALANCED EXPANSION problem is to find *balanced expansion* of a graph as defined in (1.3).

- Graph conductance is problem of finding the *graph conductance* of a graph as defined in (1.6).
- Directed Sparsest Cut is directed version of Sparsest cut problem.

Next definition is a very general formulation of graph partitioning problem where we want to find a decomposition of graph vertex set into k possibly balanced partition. We refer more on this problem in section 3.

Definition 4. *(*$(k, 1+\epsilon)$ - balanced partition problem*) Given a graph* $G = (V, E, W_V, W_E)$*, where* V *is set of vertices,* E *is set of edges,* $W_V : V \to R^+$ *is set of weights of vertices.*$W_E : E \to R^+$ *is weight of edges. Objective of* $(k, 1+\epsilon)$ - balanced partition problem *is to choose a partition* $V = V_1 \cup V_2 \cup \cdots \cup V_k$*, such that*

- *The sum of the vertex weights in each* V_i *is "about the same",*

$$\forall i \in \{1, \ldots, k\}, V_i \leq (1+\epsilon)\frac{\sum_{v \in V} W_V(v)}{k}$$

- *The sum of all edge weights of edges connecting all different pairs* $(N_i, N_J), i \neq j$ *is minimized*

$$min|C| = |\{W_E(\{u, v\}) \in E | u \in V_i, v \in V_j, i \neq j\}| = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} w(V_i, V_j)$$

where $w(V_i, V_j) = \sum_{e \in E(V_i, V_j)} W_E(e)$

For example in the parallel computing this way of partitioning of unstructured graphs is very valuable. In this area graph partitioning is mostly used to partition underlying graph model of computation and communication. To give rough description, vertices in the graph represent computation units and edges denote communication. Particularly, if we want to use k processors we want to partition the graph into k blocks of about equal size. Some other variants of this problem are:

- Minimum k-cut is unbalanced version of the problem.
- Minimum C-balanced cut is special case where $k = 2$ and also known as Balanced Edge Separator problem. Directed version is recognized as Directed Balanced Edge Separator problem.

The class of minimization problems follows. These problems are different in that way, that they focus on finding cuts with minimum edge weight or vertex weight, but they do not have the ambition to provide balanced partition.

Definition 5. *(*Minimum vertex k-cut problem*) Let* G *be graph* $G = (V, E)$*, and* T *be a set of terminal vertices* $T = \{s_1, t_1, \ldots, s_k, t_k\}$*. And let* w *be a weight function* $w : V \setminus T \to R^+$*. Then the problem of finding a subset* $C \subseteq V \setminus T$ *of vertices such that by removing from* G *we disconnects each* s_i *from* t_i *for* $1 \leq i \leq k$ *with objective to minimize the sum of the weight of the vertices in the cut, i.e.,* $\sum_{v \in C} w(v)$*, is called* Minimum vertex k-cut problem*.*

Other minimization problems are

- MINIMUM MULTI-CUT PROBLEM is analogous to MINIMUM VERTEX K-CUT PROBLEM, where we have weighted edges, and we are looking for edge cut $E' \subseteq E$ that disconnects all terminal pairs with minimum $w(E') = \sum_{e \in E'} w(e)$.

- MINIMUM MULTI-WAY CUT PROBLEM is special case of MINIMUM MULTI-CUT PROBLEM, where we have instead of set of terminal pairs the terminal set $T \subset V$ and we want to disconnect all vertices of T from each other by minimal weighted edge cut $E' \subseteq E$.

- MINCUT problem is defined analogously to the MAXCUT problem,where we want to find partition $(S, \bar{S})$, with a minimum number of edges $|E(S, \bar{S})|$, in the weighted version we want to minimize the sum of edge weights $\sum_{e \in E(S,\bar{S}} w(e)$. In contrary to the other graph partitioning problems, MINCUT can be computed in polynomial time, thanks to duality of SINGLE COMMODITY FLOW PROBLEM and MINCUT. This duality is a generalization of the duality presented in section 1.1.1.

The last definitions of the section are devoted to maximization problems, where instead of finding minimal cut or cut with minimal sparsity we want to maximize edge or vertex weight of the cut. This is something different from all of the previous problems and also need different approaches.

Definition 6. *(*MAXCUT*) Let $G = (V, E)$, then optimization problem where we want to partition V into disjoint sets S and $\bar{S}$ to maximize $|E(S, \bar{S}|$ (in weighted case $w(S, \bar{S})$) is called* MAXCUT *problem.*

Related problems are

- MAXIMUM BISECTION with additional constraint that the partition must cut the graph into halves of the same size. Analogous problem is MINIMUM BISECTION, where the number of edges is to be minimized.

- MAXIMUM DIRECTED CUT directed version of MAXCUT, in partition $(S, \bar{S})$ we count number of edges from S to $\bar{S}$.

- MAXIMUM K-CUT PROBLEM is version of MAXCUT problem, where set of vertices V is partitioned intp k disjoint sets $V_1, \ldots, V_k$ and the sum of the weight of the edges between the sets is maximized.

- MAXIMUM K-SECTION PROBLEM is balanced version of MAXIMUM K-CUT PROBLEM, where graph must be cut into sets of equal size.

In the end of the section we present the table with the list of graph partitioning problems together with thr known approximations and complexity classes.

	Complexity	Approximation known
Min Cut	$O(\|V\|.\|E\|.\log(\|V\|^2/\|E\|)$	
Sparsest Cut	NP-complete	$O(\sqrt{\log k}\log\log k)$ [6]
Uniform Sparsest Cut	Apx[51]	$O(\sqrt{\log n})$ [10]
Balanced Edge Separator	NP-hard	$O(\sqrt{\log n})$ [9]
Edge Expansion	NP-hard	$O(\sqrt{\log n})$ [9]
Graph Conductance	NP-hard	$O(\sqrt{\log n})$ [9]
Directed Sparsest Cut	NP-hard	$O(\sqrt{\log n})$ [1]
Directed Balanced Edge Separator	NP-hard	$O(\sqrt{\log n})$ [1]
Minimum k-cut	$O(\|V\|^{k^2})$[30]	$2-\frac{2}{k}$[70]
Minimum Vertex k-cut	NP-hard	$O(\log\|V\|)$[27]
Minimum Multi-way Cut	Apx-complete[18]	$\frac{3}{2}-\frac{1}{k}$ [2][3]
Minimum Multi-cut	Apx-hard	$O(\log\|S\|)$[28][3]
$(k, 1+\epsilon)$-balanced Partitioning Problem	NP-hard	$\epsilon > 0, \log^2 n$ [5]
Max Cut	Apx-Complete [65]	1.1383[29]
Maximum Bisection	Apx-Complete [65]	1.431[75]
Minimum Bisection	NP-hard	$O(\log^{1.5} n)$[22]
Max Directed Cut	Apx-Complete[65]	1.165[21]
Max k-cut	Apx-complete	$1/(1-1/k+2\ln k/k^2)$[26, 58]
Maximum k-section	Apx-complete	$1/(1-1/k+\epsilon.k^3)$[4]

Table 1.1: Graph partitioning problems

1.1.3 Interreducibilities among graph partitioning problems

There are some well-known interreducibilities among graph partitioning problems. The following reductions are described in ARV paper [10]. As was mentioned above for *sparsity* and *expansion* holds inequlity (1.5). The Conductance problem can be reduced to the Sparsest cut problem by replacing each node with a degree d with a clique on d nodes; any sparse cut in the transformed graph does not split cliques; and the sparsity of any cut (that does not split cliques) in the transformed graph is within a constant factor of the conductance of the cut. The sparsest cut problem can be reduced to the conductance problem by replacing each node in graph G with a large (say sized $C = n^2$ size) clique. Cut of conductance Φ in the transformed graph $\bar{G}$ corresponds to a cut in the original graph of sparsity $\frac{\Phi}{C^2}$.

2. Ideas and approaches for solving graph partitioning problems

In this introduction we present a broad overview of section 2. After the introductory first section follows the main part of this work where we present various ideas and approaches to graph partitioning problems. First we start by the introduction of multi-commodity flows. It leads us to work of Leighton and Rao [53], which captures the duality between multi-commodity flows and sparsest-cut problem. As we will see, this duality allows us to construct $O(\log n)$-approximation algorithm for the sparsest cut problem. The formulation of this duality is often referenced as the Leighton-Rao theorem. There are two main directions how to prove this theorem and both can be found in the presentation of Shmoys [72]. One is region-growing algorithm which was presented in the original paper of Leighton and Rao.The other one uses geometric embeddings. In the beginning of the section 2.2 we introduce basic terms and definitions about metric spaces and embeddings. We continue by above mentioned geometric proof of Leighton-Rao theorem. We also choosed to implement and test this algorithm.

In the section 2.2.3 we move our attention to the groundbreaking paper of authors Arora, Rao and Vazirani [10]. This paper as first brought $O(\sqrt{\log n})$-approximation to the sparsest cut problem and other related graph partitioning problems. In the ARV paper are presented 2 algorithms. One based on SDP programming and geometric embeddings and a second one based on expander flows. We present SDP algorithm in section 2.2.3. Although a time complexity of this algorithm is polynomial, a number of constraints in SDP relaxation is too high. The algorithm thus is not practical. But it has great theoretical impact in both graph partitioning theory and theory of geometric embeddings. The second algorithm in ARV paper introduces the expander flows and starts the new branch of research, which we refer to as flow-based approach and present it in section 2.4. Further exploration of the flow-based approach gave a rise to so called cut-matching game, which was introduced in the work of Khandekar, Rao and Vazirani [50]. We further focus on the cut-matching game and present something from work of Orecchia, who together with other authors further develops ideas of this framework in work [63] and his dissertation thesis [62]. A description of the cut-matching game finishes the section.

The last part of section 2 presents a spectral graph theory and its connections to graph partitioning. We present the connection between eigenvectors of graph matrices and good partitions. Main idea of this section is presented in the proof of the Cheeger inequality, which provides basis for spectral algorithm for graph partitioning. Spectral theory has connection to both geometric approach and flow based approach. We also introduce a random walks in this section and how the mixing time of random walks is connected with a spectral gap and the expansion of a graph. As we will see a strategy of the Cut player in the Cut-Matching game uses concepts of the random walks and the spectral theory and thus whole cut-matching framework merges the flow-based approach with the spectral graph

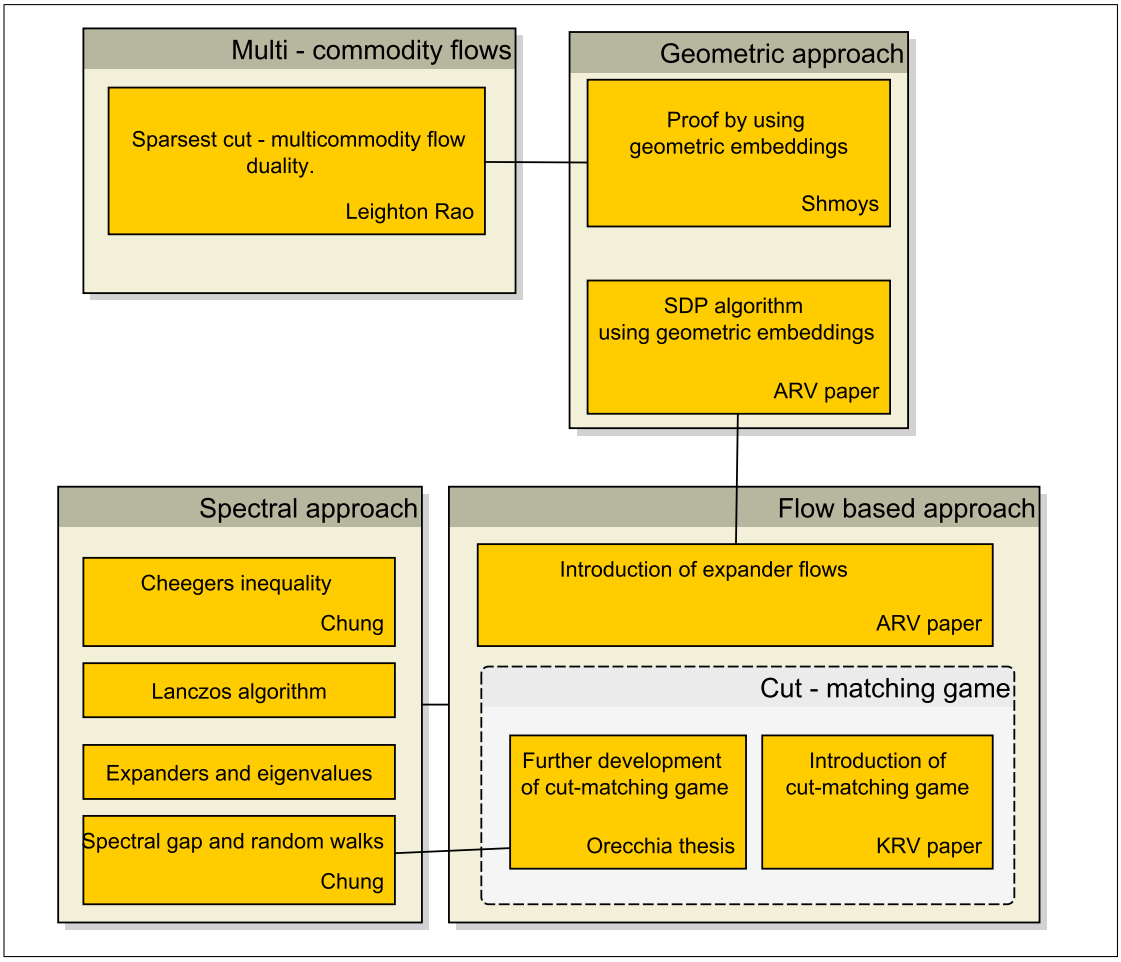


Figure 2.1: Overview of Section 2

partitioning ideas. We can also look at a spectral graph partitioning as an embedding of graph vertices to a one-dimensional space. Thus all these approaches are connected and the best algorithms use ideas from all of them. The figure 2.1 presents diagram with ideas presented in section 2.

2.1 Multi-commodity flows

A *multi-commodity flow problem* (MFP) can be seen as a generalization of the more known single-commodity flow problem we presented in section 1.1.1. Instead of trying to route the flow only between one pair of vertices we want to route the flow between set of pairs of vertices, each with its own commodity.The MFP comes in many flavours. As we will see in this section, there is a similar connection between Multi-commodity flow problem and Sparsest-cut problem as we have seen between Single-commodity maximum flow problem and Minimum s-t cut problem. Unfortunately, duality between the MFP and the Sparsest-cut is not tight. Main result in this area is Leighton-Rao theorem.We will see it later in this section. The definition of MFP we use is based on the one from web resource [35].

Definition 7. *Given a graph of flow network $G = (V, E)$, where edge $(u, v) \in E$ has capacity $c(u, v)$. There are k commodities $K_1, K_2, \ldots, K_k$ defined by $K_i =$*

(s_i, t_i, d_i), *where* s_i *is the* source *and* t_i *is the* sink *of commodity* i*, and* d_i *is the* demand. *The flow of commodity* i *along edge* (u, v) *is* $f_i(u, v)$*. Object is to find an assignment of flow which satisfies the constraints:*

- *Capacity constraints:*

$$\sum_{i=1}^{k} f_i(u, v) \leq c(u, v)$$

- *Flow conservation:*

$$\forall v, u, u, v \neq s_i, t_i \sum_{w \in V} f_i(u, w) = 0, f_i(u, v) = -f_i(v, u)$$

In the maximum multi-commodity flow problem*, there are no demands* d_i *on each commodity, but the total throughput is maximized:*

$$\max \sum_{i=1}^{k} \sum_{w \in V} f_i(s_i, w)$$

In the maximum concurrent flow problem*, we want to maximise the minimal fraction of the flow of each commodity to its demand:*

$$\max \min_{1 \leq i \leq k} \frac{\sum_{w \in V} f_i(s_i, w)}{d_i}$$

In this work we exclusively refer to the maximum concurrent flow problem. To use the MFP to solve Uniform sparsest-cut problem, we can use the Uniform MFP (UMFP). For which holds that each unordered pair of vertices $(u, v) \in V \times V$ there is separate commodity $K_(u, v) = (u, v, 1)$. We denote f the minimal fraction we want to maximize.

$$f = \min_{1 \leq i \leq k} \frac{\sum_{w \in V} f_i(s_i, w)}{d_i}$$

Observe that when we have a cut $(S, \overline{S})$ of the graph G, we need to take $f|S||\overline{S}|$ of the flow across a cut. We assume $\emptyset \neq S \subsetneq V$. We can say that *the capacity of the cut* $C(S)$ is simply the sum of the edge capacities

$$C(S) = \sum_{e \in (S, \overline{S})} C(e).$$

In case all the capacities are equal, the capacity of the cut is proportional to the number of edges of the cut so we can see

$$f|S||\overline{S}| \leq |E(S, \overline{S})| \Rightarrow f \leq \min_{\emptyset \neq S \subsetneq V} \frac{\sum_{e \in E(S, \overline{S})} C(e)}{|S||\overline{S}|} = \min_{\emptyset \neq S \subsetneq V} \frac{|E(S, \overline{S})|}{|S||\overline{S}|} = \mathcal{S}. \quad (2.1)$$

Which is the same as the sparsity defined above (1.4). Although we have inequality $f \leq \mathcal{S}$, there is no max-flow min-cut theorem for UMFP which would always guarantee equality as in the case of single commodity flow problem. We formulate this fact as a lemma with a proof from another work of Leighton and Rao [54].

Lemma 1. *([54])There is a graph G for which $\mathcal{S}_G > f$, where $\mathcal{S}_G$ is the sparsity of G and f is the size of solution of uniform multi-commodity problem.*

Proof. We begin by showing that the max-flow and the min-cut of a UMFP are separated by a $\Theta(\log n)$ gap whenever the underlying graph has certain expansion properties. Let G be a 3-regular n-node c-expander, for some constant $c > 0$. Families of such graphs are well known to exist provided that c is sufficiently small constant [59].

So we have already seen (1.5) that

$$\mathcal{S}_G \geq \frac{c}{n-1}.$$

In the following text we show:

$$f \leq \frac{6}{(n-1)(\log n - 1)},$$

which is a $\Theta(\log n)$ smaller than sparsity $\mathcal{S}$.

Since G is 3-regular, there are at most $n/2$ nodes within the distance $\log n - 3$ of any particular node $v \in V$. Hence for at least half of the $\binom{n}{2}$ commodities the shortest path connecting the source and the sink in G has at least $\log n - 2$ edges. In order to sustain a needed flow for such commodity at least $f(\log n - 2)$ capacity of edges must be used by the commodity. Thus, when we consider a flow for all $\binom{n}{2}$ commodities, the capacity of edges in the graph must be at least

$$\frac{1}{2}\binom{n}{2} f(\log n - 2).$$

Since the graph is 3-regular and has unit capacity edges, the total capacity is at most $\frac{3n}{2}$. Hence,

$$f \leq \frac{3n}{\binom{n}{2}(\log n - 2)} = \frac{6}{(n-1)(\log n - 2)} \leq \frac{6\mathcal{S}_G}{c(\log n - 2)} = O\left(\frac{\mathcal{S}_G}{\log n}\right).$$

□

The preceding example is as bad as things can get, however. This fact is summarized in the following theorem, which is one of main results of Leighton and Rao paper [53].

Theorem 1. *(Leighton Rao Theorem) ([53]) There is a feasible flow with with $\Omega\left(\frac{\mathcal{S}}{\log n}\right)$ for any n-vertex UMFP.*

The preceding result provides an approximate max-flow min-cut relationship for UMFPs that is tight in the worst case. In particular, we know that

$$\Omega\left(\frac{\mathcal{S}}{\log n}\right) \leq f \leq \mathcal{S}$$

for any UMFP. Therefore we can say that maximum flow is always within a $\Theta(\log n)$ factor of the minimum cut and as we showed, the approximation is existentially tight. Now we can introduce a notion of *demand graph.*

Definition 8. *In an instance of MFP let*

$$\{s_1, s_2, \ldots, s_k, t_1, t_2, \ldots, t_k\} = T$$

be set of the source and the sink vertices and let F *to be* $F = \{(s_i, t_i)|i \in \{1, \ldots, n\})\}$. *Then* $H = (T, F)$ *is the* demand graph.

Since the source and the sink vertices can overlap, the demand graph can be any graph, not only 2-factor. In case of UMFP the demand graph is a complete graph on n vertices. In a way we can see the solving UMFP problem as an embedding of a complete graph. Later we will see that we can embed other expander than complete graph to solve graph partitioning problems.

2.2 Geometric approach

The geometric approach to the graph partitioning problems introduces an idea of embedding graph to some geometric space. Main goal is to effectively embed the graph vertices into some abstract space in such way that we keep distances between adjacent vertices as small as possible and let distance between average pair of vertices be some constant, say 1. The point is that the proximity in a geometric space rougly reflects a connectivity in a graph.Well-connected graph components stay close in and clusters of nodes that are connected only with few edges are more separated. There is good chance that even a random cut of such geometric space will also be a good cut of graph.

2.2.1 Basic definitions

Definitions in this section are from the web resource [39]. We start with the definitions of the metric spaces and the normed spaces.

Definition 9. *Let* X *be a set, and let* $d : X \times X \to \mathcal{R}_0^+$. *The pair* (X, D) *is a* metric space *if for all* $x, y, z \in X$,

1. $d(x, y) = d(y, x)$
2. $d(x, y) = 0 \Leftrightarrow x = y$
3. $d(x, y) + d(y, z) \geq d(x, z)$ *(triangle inequality)*

Definition 10. *A* normed space *is* $\mathcal{R}^k$ *for some finite* k *with an associated mapping* $a \to ||a||$ *from* $\mathcal{R}^k$ *to* $\mathcal{R}_0^+$ *such that:*

1. *For all* $\lambda \in \mathcal{R}$, $||\lambda a|| = |\lambda|||a||$
2. $||u + v|| \leq ||u|| + ||v||$
3. $||u|| = 0 \Leftrightarrow a = 0$ *(the zero vector)*

By removing 3rd property we gain so called seminorm.

Observe that (R^k, d) with $d(u, v) = ||u - v||$ is metric space. The examples of norms follows:

1. l_2 is the normal Euclidean norm. For vector $v \in R^k$:

$$||v||_2 = \sqrt{\sum_{i=1}^{k} |v_i|^2}.$$

   This gives the ordinary distance from the origin to the point v, a consequence of the Pythagorean theorem.

2. l_1 norm is often called *Taxicab norm or Manhattan norm*. The name relates to the distance a taxi has to drive in a rectangular street grid to get from the origin to the point v. Norm of a vector v is simply the sum of absolute values of its coordinates.

$$||v||_1 = \sum_{i=1}^{k} |v_i|.$$

   Note that in contrast $\sum_{i=1}^{k} v_i$ is not a norm because it may yield negative results.

3. We can talk about *p-norm*, which is generalization of above 2 norms. For $p = 1$ we get the taxicab norm and for $p = 2$ we get Euclidean norm. The p-norm of vector v is defined as:

$$||v||_p = (\sum_{i=1}^{k} |v_i|^p)^{1/p}.$$

   As p approaches infinity the value is dominated by the largest coordinate. Such norm is called *infinity norm* or *maximum norm* and is referred as l_∞ norm.

$$||v||_\infty = \max(|v_1|, |v_2|, \ldots, |v_n|).$$

4. l_2^2 norm is further used in Section 2.2.3 and is defined as squared Euclidean distance

$$||v||_2^2 = (||v||_2)^2 = \sum_{i=1}^{k} |v_i|^2.$$

As we will see later, we often want to embed points from one normed metric space to another in such way that lengths between points do not change much. How well we can do that is captured by property called *distortion*.

Definition 11. *Let (X_1, d_1) and (X_2, d_2) be metric spaces. An embedding $f : X_1 \to X_2$ has* distortion C *if there is an $r > 0$ such that $\forall x, y \in X_1$,*

$$r \cdot d_1(x, y) \leq d_2(f(x), f(y)) \leq Cr \cdot d_1(x, y).$$

Research of geometric embeddings recognize 3 main types of problems

1. Can we embed metric space of class I into class II with distortion at most C? (Upperbound)

2. Show an example of a metric space in class I that does not embed into any metric space of class II with distortion at most C. (Lowerbound)

3. Determine the minimum distortion required to embed X_1 into X_2 within metric spaces (X_1, d_1) and (X_2, d_2).

Now follows famous result, the theorem of Bourgain from 1985 [14].

Theorem 2. *([14]) For $p > 1$, every n-point metric embeds into l_p with distortion $O(\log n)$.*

We use this theorem in the proof presented in following section.

2.2.2 The Embedding algorithm and proof of Leighton-Rao duality theorem

In this section we give a proof of the Leighton-Rao theorem from the section (2.1). We describe more general result provided by subsequent work in this area. We first formulate MFP as a linear programming (LP) problem. We use information from online materials and lecture notes [36]. Let have MFP as in definition 2.1, with a graph $G = (V_G, E_G)$ and capacities $c(u, v)$. With collection of source and sink pairs with demands $(s_1, t_1, d_1), \ldots, (s_k, t_k, d_k)$ our objective will be that of the MFP The value $y = \min_{1 \leq i \leq k} \frac{\sum_{w \in V} f_i(s_i, w)}{d_i}$ will be maximized. Lastly, $H_G = (V_H, E_H)$ will be the demand graph. We consider both G and H to be undirected.

$$
\begin{array}{rll}
 & \max \quad y & \\
 & \text{subject to} & \text{(P1)} \\
\text{Flow conservation} & \sum_{u \in V_G} f_i(u, v) = \sum_{w \in V_G} f_i(v, w) & \forall i \in \{1, \ldots, k\}, \forall v \in V - \{s_i, t_i\} \\
\text{Capacity constraints} & \sum_{i=1}^{k} f_i(u, v) \leq c(u, v) & \forall (u, v) \in E_G \\
\text{Optimization constraints} & \sum_{v \in V_G} f_i(s_i, v) \geq y.d_i & \forall i \in \{1, \ldots, k\} \\
 & f_i(u, v) \geq 0 & \forall i \in \{1, \ldots, k\}, \forall (u, v) \in E_G
\end{array}
$$

Linear program formulated this way contains linear number of $k|E_G| + 1$ variables. It is possible to give a formulation that involves an exponential number of variables, but for which it is easier to derive the dual.

In the following formulation P_i is the set of all paths from s_i to t_i in G. We have a variable x_p for each path in P_i, for each $i \in \{1, \ldots, k\}$ corresponding to

how many units of flow from s to t are routed through the path p.

$$\begin{aligned} \max \quad & y \\ \text{subject to:} & \\ \sum_{p \in P_i} x_p &= y.d_i && \forall i \in \{1, \ldots, k\} \\ \sum_{p:(u,v)\in p} x_p &= c(u,v) && \forall (u,v) \in E_G \\ x_p &\geq 0 && \forall p \\ y &\geq 0 \ . \end{aligned} \tag{P2}$$

Now to see the dual of P2, we provide a variable w_i for each $i \in \{1, \ldots, k\}$ and one variable $z(u,v)$ for each $(u,v) \in E_G$. It looks as follows:

$$\begin{aligned} \min \quad & \sum_{(u,v)\in E_G} z(u,v)c(u,v) \\ & \text{subject to:} && && \text{(D2a)} \\ & \sum_{i=1}^{k} w_i.d_i \geq 1 && && \text{(e)} \\ -w_i + & \sum_{(u,v)\in p} z(u,v) \geq 0 && \forall i \in \{1, \ldots, k\}, p \in P_i && \text{(f)} \\ & w_i \geq 0 && \forall i \in \{1, \ldots, k\} \\ & z(u,v) \geq 0 && \forall (u,v) \in E_G \ . \end{aligned}$$

Now consider the graph G with the weights $z(u,v)$ on the edges $(u,v) \in E_G$. From the constraints f we see that w_i is less than shortest path from s_i to t_i and equality holds for the optimum solution. To see that, observe that if some w_i is strictly smaller than the length of the shortest path, we can make it equal to the length of the shortest path without sacrificing feasibility and without changing the cost of the solution (By incrementing w_i we do not break the first constraint (e) and do not change the objective function). The other observation is that, in an optimal solution we have $\sum_{i=1}^{k} w_i.d_i = 1$. To see it, realize that if we had $\sum_{i=1}^{k} w_i.d_i = c > 1$, then we could divide all the w_i and all the $z(u,v)$ by c, and obtain a solution that is still feasible and has a smaller cost. This means that we can formulate the following version of the linear program equivalent to (D2a). We have a variable $l(x,y)$ for every pair of vertices in $E_G \cup E_H$:

$$\min \sum_{(u,v)\in E_G} l(u,v)c(u,v)$$

subject to: (D2b)

$$\sum_{i=1}^{k} l(s_i,t_i).d_i = 1 \quad \text{(g)}$$

$$\sum_{(u,v)\in p} l(u,v) \geq l(s_i,t_i) \qquad \forall i \in \{1,\ldots,k\}, p \in P_i \quad \text{(h)}$$

$$l(u,v) \geq 0 \qquad \forall (u,v) \in E_G \cup E_H$$

Now we use the fact that G and H are undirected. In such case we have a variable $l(u,v)$ for each *unordered* pair u,v. The constraints (h) can be equivalently restated as the *triangle inequalities*

$$l(u_1,u_3) \leq l(u_1,u_2) + l(u_2,u_3)\ \forall (u_1,u_3),(u_1,u_2),(u_2,u_3) \in E_G \cup E_H$$

Now we can see that we require $l(u,v)$ to be non-negative, symmetric and to satisfy the triangle inequality, and so it is a *metric* over V. (Technically, it is a *semi-metric* because we can have distinct vertices at distance zero, and $l(.,.)$ is not defined for all pairs) So one way to look at this formulation is to see it as embedding vertices to some abstract space with $l(.,.)$ metric in such way that average distance between vertices is constant as we mention in the beginning of this section. This is provided by constraint (g). The minimization objective function is to keep the distances between the neighboring vertices short. To see it more clearly, consider the UMFP version of the problem. The demand graph H is the complete graph on n vertices and weights on both the graph G and H are ones on all edges. This would translate to the following LP:

$$\min \sum_{(u,v)\in E_G} l(u,v)$$

subject to: (D2bU)

$$\sum_{i=1}^{k} l(s_i,t_i) = \sum_{(u,v)\in V\times V, u\neq v} l(u,v) = 1 \quad \text{(i)}$$

$$\sum_{(u,v)\in p} l(u,v) \geq l(s_i,t_i) \qquad \forall i \in \{1,\ldots,k\}, p \in P_i \quad \text{(j)}$$

$$l(u,v) \geq 0 \qquad \forall (u,v) \in E_G \cup E_H\ .$$

This way vertices of well connected component in such abstract space will be tightly together and it will be easier to find a sparse cut. In the following section we elaborate on this idea some more.

These observations give us one more alternative formulation:

$$\min_{l(.,.)metric} \frac{\sum_{(u,v)\in E_G} c(u,v).l(u,v)}{\sum_{(s_i,t_i)\in E_H} d_i.l(s_i,t_i)}. \quad \text{(D2c)}$$

In the case of UMFP:

$$\min_{l(.,.)metric} \frac{\sum_{(u,v)\in E_G} l(u,v)}{\sum_{(u,v)\in V\times V, u\neq v} l(u,v)}. \tag{D2cU}$$

The rest of this section is from presentation of Shmoys [72]. Suppose that this length function was derived from an embedding of the input graph G into some low-dimensional space with l_1 norm; that is, there is a function $h : V \to R^d$ such that $l(u,v) = ||h(u) - h(v)||_1$ for each edge $(u,v) \in E_G \cup E_H$. We first shall show that if such embedding is achievable, then we could find a cut of sparsity at most $\sum_{(u,v)\in E_G} l(u,v)c(u,v)$; that is, if the optimal fractional solution (x,y) has the property that it can be derived from such an embedding, then given the embedding, we can find an optimal sparsest cut.

The problem is of course that optimal length function does not have to admit such an embedding. So main problem is to find algorithm which finds graph embedding where distances roughly corresponds to the length function.

Lemma 2. *([72]) Let $l : \binom{V}{2} \to R_0^+$ be a feasible solution to the linear program (D2c) of objective function value α such that there exists a function $h : V \to \mathcal{R}^d$ with the property that*

$$l(u,v) = ||h(u) - h(v)||_1, \forall (u,v) \in E_G \cup E_H. \tag{2.2}$$

Then, given h one can find a cut S of sparsity ratio $\mathcal{S}_G$ at most α in polynomial time.

Proof. First we present some simple consequences of the existence of the embedding h. This embedding defines a metric μ:

$$\forall (u,v) \in V_G, \mu(u,v) \stackrel{\text{def}}{=} ||h(u) - h(v)||_1.$$

These values satisfy the triangle inequality. We can also derive a metric corresponding to each subset $S \subseteq V$: let

$$\mu_S(u,v) = \begin{cases} 1 & \text{if } |\{u=v\} \cup S| = 1 \\ 0 & \text{otherwise} \end{cases}$$

Observe that for any $u, v, w \in V_G$, if (u,w) crosses the cut defined by S, then exactly one of (u,v) and (v,w) must cross this cut too. Such defined metric is often called *cut metric*. The *cut cone* is the convex cone formed by taking all non-negative combinations of cut metrics. The property of μ that we need is that it is in cut cone: that is, there exists $\lambda_S \geq 0$ such that

$$\mu(u,v) = \sum_{S\subseteq V} \lambda_S . \mu_S(u,v), \forall u, v \in V_G \tag{2.3}$$

To prove this we start by considering just the contribution of the first coordinate of our d-dimensional space to μ, $|h_1(u) - h_1(v)|$. Consider such ordering of the vertices V_G, $(v_1, \ldots, v_n)$such that

$$h_1(v_1) \leq f_1(v_2) \leq \cdots \leq h_1(v_n).$$

Then for each pair of vertices $v_j, v_{j'}, j > j'$ we can rewrite the contribution of the first coordinate to $\mu(v_j, v_{j'})$ as

$$h_1(v_j) - h_1(v_{j'}) = \sum_{l=j'}^{j-1} h_1(v_{l+1}) - h_1(v_l).$$

Now let $S(l) = \{v_1, \ldots, v_l\}, l = 1, \ldots, n$. Observe that the term $h_1(v_{l+1}) - h_1(v_l)$ is included in this sum precisely when the distance with respect to the metric $\mu_{S(l)}$, between v_j and $v_{j'}$ is 1. So we can see:

$$h_1(v_j) - h_1(v_{j'}) = \sum_{l=1}^{n} (h_1(v_{l+1}) - h_1(v_l)).\mu_{S(l)}(v_j, v_{j'}).$$

Note that if $j < j'$, then the right side of this equation still gives $|h_1(v_j) - h_1(v_{j'})|$. This way we have shown that the contribution of the first coordinate to μ can be expressed in the claimed form. By iterating this construction for each coordinate and putting them together, we have a decomposition of μ as proposed above (2.3). At the first glance, it looks like we need exponential number of sets for

Algorithm 1 Function returning sparsest cut from graph embedding

```
function EMBED-CUT(G,h)                ▷ Each vertex v ∈ V_G is embedded at
(h_1(v), ..., h_d(v))
    MinSparsity = ∞
    S(i, γ) = {v | h_i(v) ≤ γ}
    SparsestCut
    for all v ∈ V_G do
        for j = 1 → d do
            S = S(j, h_j(v))
            Sparsity ← |E(S, S̄)| / (|S||S̄|)
            if Sparsity < MinSparsity then
                MinSparsity ← Sparsity
                SparsestCut ← S(j, h_j(v))
            end if
        end for
    end for
return SparsestCut
end function
```

decomposition 2.3, but we have just shown decomposition that uses at most nd sets. From shown construction we can derive efficient algorithm to produce the decomposition (Algorithm 1). Finally, we shall see that decomposition (2.3) implies the lemma. Let $\mu = \sum_{S \in \mathcal{Q}} \lambda_S \mu_S$ as in (2.3), where $\mathcal{Q} = \{S | S \subseteq V, \lambda_S > 0\}$. Then by substituting $\mu(u, v)$ for $l(u, v)$ and recalling that $l(\ldots)$ is feasible we see that

$$\alpha = \frac{\sum_{(u,v) \in E_G} c(u, v).l(u, v)}{\sum_{(s_i,t_i) \in E_H} d_i.l(s_i, t_i)} = \frac{\sum_{(u,v) \in E_G} c(u, v) \sum_{S \in \mathcal{Q}} \lambda_S.\mu_S(u, v)}{\sum_{(s_i,t_i) \in E_H} d_i \sum_{S \in \mathcal{Q}} \lambda_S.\mu_S(s_i, t_i)}.$$

By realizing the fact that $\mu_S(u,v) = 1$ holds exactly when (u,v) crosses the cut defined by S we see that

$$\alpha = \frac{\sum_{S\in\mathcal{Q}} \lambda_S \sum_{(u,v)\in(S,\overline{S})} c(u,v)}{\sum_{S\in\mathcal{Q}} \lambda_S \sum_{(s_i,t_i)\in(S,\overline{S})} d_i} \geq \min_{S\in\mathcal{Q}} \frac{\sum_{(u,v)\in(S,\overline{S})} c(u,v)}{\sum_{(s_i,t_i)\in(S,\overline{S})} d_i} = \min_{S\in\mathcal{Q}} \frac{E(S,\overline{S})}{|S||\overline{S}|} \geq \mathcal{S}_G.$$

The first inequality follows from the simple observation that an average cannot be less than the minimum of its components.

Observation 1. *([72]) For any non-negative integers, $a_1,\ldots,a_n$, and positive integers, $b_1,\ldots,b_n$,*

$$\min_{i\in\{1,\ldots,n\}} \frac{a_i}{b_i} \leq \frac{\sum_{i=1}^n a_i}{\sum_{i=1}^n b_i}$$

From duality we have $\alpha = S_G$. Therefore inequalities are equalities and we can find sparsest cut just by trying nd cuts. □

The $O(\log k)$-approximation algorithm for the sparsest cut problem is an immediate corollary of the following result: given any feasible solution $l(\ldots)$ to (D2b) we can construct an embedding h which induces a feasible fractional solution $l'(\ldots)$ such that

$$\sum_{e\in E_G} l'(u,v)c(u,v) = O(\log k) \sum_{e\in E_G} l(u,v)c(u,v).$$

The embedding is constructed by a randomized algorithm. So for any input the solution induced by the embedding is sufficiently good with high probability. The probability depends only on the random choices made by algorithm and not on any assumption about the input.

To describe this algorithm, assume that $|V_H|$ is power of 2, i.e., $|V_H| = 2^\tau$. We use $l(\ldots)$ as a distance function. Furthermore, for any set of vertices $A \subseteq V$ let $l(u,A) = \min_{v\in A} l(u,v)$. Now we let $L = q\log k$ where q is a constant that will be determined later. The dimension of the embedding is $d = \tau L = O(\log^2 k)$. For $l = 1,\ldots,L$ $t = 1,\ldots,\tau$, construct the set A_{tl} by choosing $2^{\tau-t} = \frac{k}{2^t}$ points from V_H uniformly at random with replacement. The embedding function h is then defined to be

$$h_{tl}(v) = l(v, A_{tl}), \forall v \in V_G. \tag{2.4}$$

We first state 2 key lemmas and show how they imply that the embedding is good. Than we will prove both of the lemmas according to presentation of Shmoys [72].

Lemma 3. *([72]) For each edge $(u,v) \in E_G$, $||h(u) - h(v)||_1 \leq d.l(u,v)$.*

Lemma 4. *([72]) With probability at least 1/2*

$$||h(s_i) - h(t_i)||_1 \geq \frac{L.l(s_i,t_i)}{88}, \forall i \in \{1,\ldots,k\} \tag{2.5}$$

In a polynomial time we can check if a particular embedding h satisfies (2.5). Therefore we can arbitrarily increase the probability of success by repeatedly constructing independent embeddings. We can obtain a Las Vegas-style algorithm where the performance is guaranteed and the expected running time is polynomial by iterating the procedure until an embedding satisfying (2.5) is found.

Observe that Lemmas 3 and 4 can be easily combined. In case h satisfies (2.5):

$$\sum_{i=1}^{k} d_i ||h(s_i) - h(t_i)||_1 = \Omega(\log k \sum_{i=1}^{k} d_i l(s_i, t_i)) = \Omega(\log k).$$

On the other hand, Lemma 3 implies that

$$\sum_{(u,v)\in E_G} c(u,v)||h(u) - h(v)||_1 = O(\log^2 k) \sum_{(u,v)\in E_G} c(u,v)l(u,v).$$

Combining these two equations we get the following theorem:

Theorem 3. *([72]) With probability at least* $1/2$ *the embedding* h *induces a feasible solution* $l'(\dots)$ *of objection function value:*

$$\sum_{(u,v)\in E_G} c(u,v)l'(u,v) = O(\log k) \sum_{(u,v)\in E_G} c(u,v)l(u,v).$$

([72]) By applying this theorem starting with the optimal solution $l(u,v)$ to the linear program (D2b) (and thus the optimal solution to (D2c) as well) we obtain the following corollary.

Corollary 4. *There is a randomized polynomial-time algorithm that with probability at least 1/2 computes a cut of sparsity ratio within an* $O(\log k)$ *factor of the optimal solution.*

Now we show the proofs of needed lemmas.

Proof of Lemma 3. For any set $A \subseteq V$ and edge $(u,v) \in E_G$

$$\begin{array}{ll} l(u,A) & \leq \; l(u,v) + l(v,A) \\ l(u,A) - l(v,A) & \leq \; l(u,v). \end{array}$$

Symmetrical argument leads to

$$l(v,A) - l(u,A) \leq l(u,v).$$

By combining these two we get

$$|l(v,A) - l(u,A)| \leq l(u,v).$$

By definition :

$$||h(u) - h(v)||_1 = \sum_{t=1}^{\tau} \sum_{l=1}^{L} |h_{tl}(u) - h_{tl}(v)| = \sum_{t=1}^{\tau} \sum_{l=1}^{L} |l(u,A_{tl}) - l(v,A_{tl}|.$$

Finally when we put everything together, we see that

$$||h(u) - h(v)||_1 \leq \sum_{t=1}^{\tau} \sum_{l=1}^{L} l(u,v) = \tau L l(u,v) = d.l(u,v).$$

□

Now comes the hardest part, the proof of Lemma 4.

Proof of Lemma 4. Firstly, we focus on one particular commodity i and prove that with probability at least $1-\frac{1}{2k}$ holds $||h(s_i)-h(t_i)||_1 = \Omega(L.l(s_i,t_i))$. To see that this claim implies the lemma observe that probability

$$Pr(\forall i \in \{1,\ldots,k\}, ||h(s_i)-h(t_i)||_1 = \Omega(L.l(s_i,t_i))) \geq \sum_{i=1}^{k} 1-(1-\frac{1}{2k}) = \frac{1}{2}.$$

Where inequality holds from unions bound. For $v \in \{s_i,t_i\}$ let

$$\begin{aligned} B_l(v,r) &= \{w \in V_H : l(v,w) \leq r\} \\ B_l^{\circ}(v,r) &= \{w \in V_H : l(v,w) < r\}. \end{aligned}$$

Let $r_0 = 0$ and let r_t be the minimum of the set $\{r|2^t \leq |B_l(s_i,r)| \wedge 2^t \leq |B_l(t_i,r)|\}$. Furthermore, we denote by $\bar{t}$ the minimum of the set $\{t|r_t \geq \frac{l(s_i,t_i)}{4}\}$ and reset $r_{\bar{t}} = \frac{l(s_i,t_i)}{4}$. Observe that thanks to triangle inequality we have ensured that $B_l(s_i,r_{\bar{t}})$ and $B_l(t_i,r_{\bar{t}})$ are disjoint.

We wish to prove that h embeds s_i and t_i in such way that they are reasonably far apart as compared to $l(s_i,t_i)$. Since in l_1 norm we compute distances by summing distances over all coordinates, we can prove this by showing that with sufficiently high probability each coordinate of h contributes a certain distance to the total distance between s_i and t_i. To be more accurate, we will show that each of the coordinates h_{tl} is likely to contribute distance $r_t - r_{t-1}$. So by summing this, we get

$$||h(s_i)-h(t_i)||_1 = \sum_{t=1}^{\tau}\sum_{l=1}^{L}|h_{tl}(s_i)-h_{tl}(t_i)| \geq \sum_{t=2}^{\bar{t}} L(r_t-r_{t-1}) = Lr_{\bar{t}} = \Omega(Ll(s_i,t_i)).$$

Which we wish to prove.

Observe that for a set $A \subseteq V_H$ we have $A \cap B_l^{\circ}(s_i,r_t) = \emptyset$ if and only if $l(s_i,A) \geq r_t$.Thus if we denote by $E_{tl}, t = 1,\ldots,\bar{t}, l = 1,\ldots,L$ the event:

$$A_{tl} \cap B_l^{\circ}(s_i,r_t) = \emptyset \wedge A_{tl} \cap B_l(t_i,r_{t-1}) \neq \emptyset,$$

then E_{tl} implies that

$$|h_{tl}(s_i) - h_{tl}(t_i)| = |l(s_i,A_{tl}) - l(t_i,A_{tl}| \geq r_t - r_{t-1}.$$

And this is exactly what we want. Now we only need to show that E_{tl} is likely to occur. To bound this probability though, we need to calculate some preliminaries.

Consider the following illustrative example. Suppose that we are given a ground set X. We specify the set $G, G \subseteq X$ to be a good set and additionally we specify a bad set $B, B \subseteq X, B \cap G = \emptyset$. Now A is formed by selecting p elements of X independently and uniformly at random from X (with replacement). Then

$$\begin{aligned} Pr[(A \cap G \neq \emptyset) \wedge (A \cap B = \emptyset)] &= Pr[A \cap G \neq \emptyset | A \cap B = \emptyset] Pr[A \cap B = \emptyset] \\ &\geq Pr[A \cap G \neq \emptyset].Pr[A \cap B = \emptyset]. \end{aligned} \tag{2.6}$$

To see the last inequality observe that knowing the elements of A are not selected from B can only increase the probability that they are selected from G. For any set $Y, Y \subseteq X$ we have

$$Pr[A \cap Y = \emptyset] = \left(1 - \frac{|Y|}{|X|}\right)^p.$$

If $p = \frac{|X|}{|Y|}$, then this bound approaches $\frac{1}{e}$ as p goes to infinity and is always within $[1/4, 1/e]$ if $\frac{|X|}{|Y|} \geq 2$. If we take a constant β and $p = \beta(\frac{|X|}{|Y|})$, then holds:

$$\left(\frac{1}{4}\right)^\beta \leq Pr[A \cap Y] \leq \left(\frac{1}{e}\right)^\beta. \tag{2.7}$$

Now we go back to the event $E_{tl}, t = 1, \ldots, \bar{t}, l = 1, \ldots, L$. Without loss of generality the radius r_t was determined by the ball around s_i. To apply the previous calculations set

$$\begin{array}{rcl} A & = & A_{tl} \\ X & = & V_H \\ B & = & B_l^\circ(s_i, r_t) \\ G & = & B_l(t_i, r_t). \end{array}$$

Now we have

$$\begin{array}{rcl} p & = & 2^{\tau - t} \\ |X| & = & 2^\tau \\ |B| & < & 2^t \\ |G| & \geq & 2^{t-1}. \end{array}$$

Which implies $p < \frac{|X|}{|B|}$ and $p \geq (1/2)\frac{|X|}{|G|}$. Moreover, observe that $\frac{|X|}{|B|} > 2$. Now we can use (2.7) in following way:

$$\begin{array}{rcl} Pr[A \cap B & = & \emptyset] \geq \frac{1}{4} \\ Pr[A \cap G & \neq & \emptyset] \geq (1 - (\frac{1}{e})^{\frac{1}{2}}. \end{array}$$

By plugging it to (2.6) we have

$$Pr[E_{tl}] = Pr[(A \cap G \neq \emptyset) \wedge (A \cap B = \emptyset)] \geq \frac{1 - (\frac{1}{e})^{\frac{1}{2}}}{4} \geq \frac{1}{11}, t = 1, \ldots, \bar{t}, l = 1, \ldots, L.$$

Notice that the above calculation also apply in case $t = \bar{t}$.

Now we would like to apply Chernoff bounds, where we would like to have the sum of $0 - 1$ variables. Consider the following, fix the particular $t \in \{1, \ldots, \bar{t}\}$ and define $X_l, l = 1, \ldots, L$ as:

$$X_l = \begin{cases} 1 & \text{if } E_{tl} \text{ occurs} \\ 0 & \text{otherwise} \end{cases}$$

Now we can apply the Chernoff bound to the sum $\sum_{l=1}^L X_l$, to show that it does not deviate too much from its expectation $E[\sum_{l=1}^L X_l] = \sum_{l=1}^L E[X_l] \geq \frac{L}{11}$. We use following statement of Chernoff bound: let $E[\sum_l X_l] = \mu$, then

$$Pr[\sum_l X_l < \frac{\mu}{2}] \leq e^{-\mu/8}.$$

(This follows, for example, from Theorem 4.2 from work of Motwani and Raghavan [61]) Now we can see that, if $\sum_{l=1}^{L} X_l \geq L/22$, then it means that at least for $L/22$ of the coordinates $h_{tl}, l = 1, \ldots, L$ the event E_{tl} occurs. Which leads to

$$\sum_{l=1}^{L} |h_{tl}(s_i) - h_{tl}(t_i)| \geq (r_t - r_{t-1})\frac{L}{22}. \tag{2.8}$$

Since $\mu \geq \frac{L}{11} = \frac{q \log k}{11}$, we can let q to be 200 and resulting probability will be less than $\frac{1}{(2k)\log(2k)}$. Inequality (2.8) fails to hold with probability at most $\frac{1}{2k\log(2k)}$. We know that $\bar{t} \leq \log(2k)$. It follows that (2.8) holds for *every* $t = 1, \ldots, \bar{t}$ with probability at least $1 - (\frac{1}{2k})$. And this finally leads to the fact that with probability at least $1 - (\frac{1}{2k})$,

$$\sum_{t=1}^{\bar{t}} \sum_{l=1}^{L} |h_{tl}(s_i) - h_{tl}(t_i)| \geq \sum_{t=1}^{\bar{t}} (r_t - r_{t-1})\frac{L}{22} = r_{\bar{t}}\frac{L}{22} = l(s_i, t_i)\frac{L}{88}. \tag{2.9}$$

Which completes the proof. This also completes the part where we used the presentation of Shmoys [72]. □

2.2.3 ARV algorithm

In this section we present results and techniques from the work of Arora, Rao, Vaziriani [10]. In this fundamental work they give a $O(\sqrt{\log n})$-approximation algorithms for Sparsest cut, Expansion, Balanced separator and Graph conductance problems. Which is improvement over $O(\log n)$-approximation provided in work of Leighton and Rao [53]. Expander flows as interesting and natural certificates for graph expansion were introduced in this paper. A lot of later research in the area of graph partitioning refers to this work and uses its results. Introduction of the expander flows led to work of Khandekar, Rao and Vaziriani [50], where they used expander flow approach to develop the Cut-Matching game framework. This framework was again further explored by others. We especially recognize the work of Orecchia et al [49, 62]. In this paper Arora et al. give two versions of algorithm. One uses SDP and results in theory of geometric embeddings. The second version uses the expander flows approach.

As was mentioned in the beginning of section 2.2 from geometric approach the key idea of underlying algorithms for graph partitioning is to spread out the vertices in some abstract space while not stretching the edges too much. Finding a good graph partition is then accomplished by partitioning this abstract space. In this view, spectral algorithms are mapping the vertices to the points on the real line in such way. Intuitively speaking, snipping this line at a random point should cut few edges leading to a good cut. In section 2.2.2 we described embedding which uses linear programming approach.

Arora et al. in their paper [10] present approach, which maps the vertices to points in an n dimensional space such that the average squared distance between vertices is a fixed constant, but the average squared distance between the endpoints of edges is minimized. Furthermore, they show that these squared distances form a metric referred as l_2^2 metric.

$$l_2^2(u, v) = (||u - v||_2)^2$$

.

We achieve embedding of vertices with this metric by mapping the vertices to the points on the unit sphere in $\mathcal{R}^n$ such that the squared distances form a metric. Mainly we need to choose these point in such way that triangle inequality holds within l_2^2 metric. We refer to this embedding as an l_2^2-*representation* of the graph.

Definition 12. *(l_2^2 Representation) An l_2^2-representation of a graph $G = (V, E)$ is an assignment of a point (vector) to each node, say $v_i, i \in V$, such that for all i, j, k hold:*

$$|v_i - v_j|^2 + |v_j - v_k|^2 \geq |v_i - v_l|^2 \text{ (triangle inequality).}$$

An l_2^2-representation is called a unit-l_2^2 representation if all points lie on the unit sphere. Equivalently can be said that all vectors have unit length.

Alternative descriptions and remarks:

- Geometrically speaking, the above triangle inequality says that every v_i and v_k subtend a nonobtuse angle at v_j.
- Every positive semidefinite $n \times n$ matrix has a *Cholesky factorization*, namely, a set of n vectors $v_1, v_2, \ldots, v_n$ such that $M_{ij} =< v_i, v_j >$. Thus a unit-l_2^2 representation for an n node graph can be alternatively viewed as a positive semidefinite $n \times n$ matrix M whose diagonal entries are 1 and $\forall i, j, k, M_{ij} + M_{jk} - M_{ik} \leq 1$.

When the average l_2^2 distance among all vertex pairs is some fixed constant ϱ, we say that it is *well-spread*. The sum of the l_2^2 distances between the endpoints of edges will be the *value* of such representation. Now observe that every c-balanced cut in the graph corresponds to a l_2^2-representation in a natural way: map each side of the cut to the one of two antipodal points on the unit sphere. It is easy to see that l_2^2 distance between these points is 4. Then it is clear that value of such representation is 4 times the cut capacity, since only edges contributing to the value are those that cross the cut and each of them contributes 4. Next consider case when on the one side of such cut there are at least cn vertices and on the other side there are $(1 - c)n$ vertices. Then the average l_2^2 distance between the vertices is at least:

$$\frac{4cn(1-c)n}{\binom{n}{2}} \geq \frac{4c(1-c)n^2}{n^2} = 4c(1-c).$$

Now it should be clearly seen that following SDP is relaxation for $\alpha_c(G)$ (scaled by cn).

$$\min \frac{1}{4} \sum_{(i,j)\in E} ||v_i - v_j||^2 \tag{k}$$

subject to:

$$||v_i||^2 = 1 \qquad \forall i \in V \tag{l}$$

$$||v_i - v_j||^2 + ||v_j - v_k||^2 \geq ||v_i - v_l||^2 \qquad \forall i, j, k \in V \tag{m}$$

$$\sum_{i<j} ||v_i - v_j||^2 \geq 4c(1-c)n^2 \qquad . \tag{n}$$

Objective function (k) corresponds to the number of edges crossing the cut, dividing this value by cn we get $\alpha_c(G)$. Constraints (l, m) provide unit-l_2^2 representation. Thanks to the last constraint (n) the average l_2^2 distance between vertices is high. Constraint (n) in above SDP also motivates following definition:

Definition 13. *An l_2^2-representation is* c-spread *if equation (n) holds.*

Similarly the following is SDP for sparsest cut, scaled by n.

$$\min \sum_{(i,j)\in E} ||v_i - v_j||^2 \tag{o}$$

subject to:

$$||v_i - v_j||^2 + ||v_j - v_k||^2 \geq ||v_i - v_l||^2 \qquad \forall i, j, k \in V \tag{p}$$

$$\sum_{i<j} ||v_i - v_j||^2 = 1 \tag{q}$$

For Sparsest Cut Problem we give different relaxation. For any cut $(S, \bar{S})$ consider a vector representation that places all nodes in S at one point of the sphere of radius $1/(|S||\bar{S}|)$ in l_2^2 metric and all nodes in $\bar{S}$ at the diametrically opposite point. Distance between these 2 points is then $1/|S||\bar{S}|$. To see how this representation of a cut is feasible solution to the above SDP consider following. Objective function (o), corresponds to sparsity ratio and is exactly $\frac{E(S,\bar{S})}{|S||\bar{S}|}$ when using the above representation. Constraint (p) provide triangle inequality. We do not have requirement to assign unit-length vectors to vertices as in (l). The last constraint (q) again keep average squared distance between vertices to be constant and is $\frac{|S|\cdot|\bar{S}|}{|S||\bar{S}|} = 1$ considering above representation. And thus, this SDP is a relaxation of Sparsest Cut Problem.

Of course, optimal solution will not in general correspond to a cut. To provide $\sqrt{\log n}$-approximation algorithm for Sparsest cut and Expansion we need to prove $\sqrt{\log n}$ integrality gap of these relaxations is $O(\sqrt{\log n})$. The crux of ARV paper is to extract a low-capacity cut from this embedding.

The key to this is following result about the geometric structure of well-spread l_2^2-representations.

Definition 14. *(Δ-Separated) If $v_1, v_2, \ldots, v_n \in \mathcal{R}^D$, and $\Delta \geq 0$, two disjoint sets of vectors S, T are $\delta -$ separated if for every $v_i \in S, v_j \in T, ||v_i - v_j||^2 \geq \Delta$.*

Theorem 5. *([10]) For every $c > 0$, there are $c', b > 0$ such that every c-spread unit-l_2^2-representation with n points contains Δ-separated subsets S, T of size $c'n$, where $\Delta = \frac{b}{\sqrt{\log n}}$. Furthermore, there is a randomized polynomial-time algorithm for finding these subsets S, T.*

These sets S and T can be used to find a good cut as follows: consider all points within some distance $\delta \in [0.\Delta]$ from S, where δ is chosen uniformly at random. The value of a representation determines the quality of this cut. Starting with a representation of minimum value, the expected number of edges crossing such a cut must be small, since the length of a typical edge is short relative to Δ.

More formally this can be described as corollary to Theorem 5.

Corollary 6. *([10]) There is a randomized polynomial-time algorithm that with high probability finds a cut that is c'-balanced and has size $O(W\sqrt{\log n})$. Where $W = \frac{1}{4}\sum_{(i,j)\in E} ||v_i - v_j||^2$ is the optimum value of SDP defined by equations (k)-(n).*

Proof. Thanks to Theorem 5 we have algorithm that produces Δ-separated subsets S,T, for $\Delta = \frac{b}{\sqrt{\log n}}$. To each edge $e = \{i, j\}$ associate its scaled l_2^2 length $w_e = \frac{1}{4}||v_i - v_j||^2$. Now we have $W = \sum_{e\in E} w_e$. Sets S and T are at least Δ apart with respect to this distance. Let V_s denote the vertices whose vectors are contained within distance s of S and V_0 be the vertices in S. To produce cut we pick random number r between 0 and Δ. Then we output the cut $(V_r, V \setminus V_r)$. We know that $S \subseteq V_r$, $T \subseteq V \setminus V_r$ and $|S|, |T| \geq c'n$, thus this is c'-balanced cut.

Let E_s be the set of edges leaving V_s. Each edge $e = \{i, j\}$ only contributes to E_s for s in the open interval (s_1, s_2), where $s_1 = d(i, V_0)$ and $s_2 = d(j, V_0)$ where $d(i, V) = \min_{j\in V} \frac{1}{4} l_2^2(i, j)$. From the triangle inequality follows that $|s_2 - s_1| \leq w_e$. Hence

$$W = \sum_{e\in E} w_e \geq \int_{s=0}^{\Delta} |E_s|.$$

And thus the expected value of $X = |E_s|$ over the interval $\{0, \Delta\}$ is

$$E[X] = \int_{s=0}^{\Delta} |E_s| \frac{1}{\Delta} \leq \frac{W}{\Delta}.$$

Thanks to Markov inequality:

$$P(X \geq 2E[X]) \leq 1/2.$$

The algorithm thus produces a cut of size at most $2W/\Delta = O(W\sqrt{\log n})$ with the probability at least 1/2. □

The value of Δ can be improved only by constant factor [10].

The following algorithm is the realization of Corollary 6. For a given a c-spread l_2^2 representation (which we can obtain by SDP program described above), it finds Δ-separated sets of size $\Omega(n)$

The proof that SET-FIND works for $\Delta = \Omega(1/\sqrt{\log n})$ can be find in ARV paper [10] and we will not cover it here. To sum up, ARV algorithm for Balanced Separator first computes SDP relaxation for $\alpha_c(G)$ which results in c-spread unit-l_2^2-representation $v_1, v_2, \ldots, v_n \in \mathcal{R}^d$. This representation is the input for

SET-FIND:
Input: A c-spread unit-l_2^2-representation $v_1, v_2, \ldots, v_n \in \mathcal{R}^d$.
Parameters: Separation parameter Δ, balance to be achieved c', and projection gap, δ.

1. Project the points on a uniformly random line u passing through the origin, and compute the largest value m where half the points v, have $\langle v, u\rangle \geq m$. Then, we specify that

$$S_u = \{v_i : \langle v_i, u\rangle \geq m + \frac{\delta}{\sqrt{d}}\},$$

$$T_u = \{v_i : \langle v_i, u\rangle \leq m\}.$$

   If $|S_u| < 2c'n$, HALT.

2. Pick any $v_i \in S_u, v_j \in T_u$ such that $|v_i - v_j|^2 \leq \Delta$, and delete v_i from S_u and v_j from T_u. Repeat until no such v_i, v_j can be found and output the remaining sets S, T.

Figure 2.2: SET-FIND

the SET-FIND algorithm, which produces $\Omega(1/\sqrt{\log n})$-separated sets S and T. Finally Corollary 5 gives us the desired cut.

Algorithm for achieving approximation $O(\sqrt{\log n})$ for SPARSEST CUT is similar. Let β be the optimal solution of SDP (o)- (q). Then following holds.

Theorem 7. *([10]) There is a polynomial-time algorithm that, given a feasible SDP solution with value β, produces a cut $(S, \bar{S})$ satisfying*

$$\frac{|E(S, \bar{S}|}{|S|} = O(\beta \cdot n \cdot \sqrt{\log n}).$$

The proof branches into two cases. One uses SET-FIND procedure from balanced version of a problem. The other case must satisfy prerequisites of following lemma

Lemma 5. *For every constant c, τ where $c < 1, \tau < 1/8$ there is a polynomial-time algorithm for the following task. Given any feasible solution $\beta = \sum_{\{i,j\}\in E} ||v_i - v_j||^2$ to SDP (o)- (q), and node k such that the geometric ball of squared-radius τ/n^2 around v_k contains at least cn vectors, the algorithm finds a cut $(S, \bar{S})$ with expansion at most $O(\frac{\beta n}{c})$.*

We just give the algorithm that provides the above result. Full proof as well as this lemma can be find in the work ARV [10]. Consider a geometric ball of squared radius $\frac{\tau}{n^2}$ around vertex v_k as stated in lemma. Let X be the subset of nodes that corresponds to the vectors in this ball. Now let V_s be the set of nodes whose distance from X is at most s, for $s \geq 0$. And let E_s be the set of edges leaving V_s. The algorithm consists of doing a breadth-first search on the weighted graph starting from X. We find s for which the cut $(V_s, \bar{V}_s)$ has the lowest sparsity and output this cut.

In case the conditions of Lemma (5) do not hold, we run SET-FIND.

This concludes section about ARV algorithm using SDP. In section 2.4 wel introduce the notion of expander flows as approximate certificates of expansion.

2.3 Spectral graph theory

In this section we first introduce some facts from the spectral graph theory, especially different kinds of matrices assiociated with graphs and their eigenvalues and eigenvectors. Very important is the introduction of spectral gap. We want to present the connection between conductance and spectral gap by showing the proof of the Cheeger inequality in the subsection 2.3.5. Thanks to the tight relation of conductance, expansion and sparsity (see section 1.1.3) the inequality really shows the relation of spectral gap with the other two criteria as well. This leads to the straightforward spectral algorithm for graph partitioning presented in the section 2.3.7.

The other concepts we want to point out are the introduction of random walks, relation of spectral gap with random walks and mixing of random walks in the section 2.3.6. Together with the Cheeger inequality we also get the connection of random walks with expansion graph properties. This will be important in the section 2.4.2 about the Cut-Matching game, where we use these connections and use some ideas from the spectral graph theory.

2.3.1 Background from linear algebra

Int this subsection we present well-known facts from linear algebra, mainly with focus on eigenvalues. We used the publication of Jukna [45] for this section.

A scalar λ is an *eigenvalue* of a square real matrix A if the equation $Ax = \lambda x$ has solution $x \in \mathcal{R}^n, x \neq \vec{0}$. This can be also expressed by stating that λ is a root of *characteristic polynomial* $p_A(z) = det(A - zI)$, with I being a unit matrix with ones on the diagonal, and zeros elsewhere. A non-zero vector x with $Ax = \lambda x$ is called an *eigenvector* corresponding to the eigenvalue λ. Observe that p_A has degree n, so we can have at most n (complex) eigenvalues. If the matrix A is symmetric that is $A^\intercal = A$, then all its eigenvalues are real numbers. We state some standard facts about eigenvalues of a real symmetric $n \times n$ matrix $A = (a_{ij})$:

1. A has exactly n (not necessarily distinct) real eigenvalues $\lambda_1 \geq \cdots \geq \lambda_n$.

2. There exists a set of n eigenvectors $x_1, \ldots, x_n$, one for each eigenvalue that are normalized and mutually orthogonal that is, $||x_i||_2 = 1$ and $\langle x_i, x_j \rangle = 0$ over the real numbers. Hence, vectors $x_1, \ldots, x_n$ form an *orthonormal* basis of $\mathcal{R}^n$.

3. The rank of A is equal to the number of its nonzero eigenvalues, including multiplicities: $rank(A) = |\{i : \lambda_i \neq 0\}|$.

4. The sum of all eigenvalues $\sum_{i=1}^n \lambda_i$ is equal to the trace $tr(A) = \sum_{i=1}^n = \sum_{i=1}^n a_{ii}$.

5. The product $\prod_{i=1}^n \lambda_i$ of all eigenvalues is equal to det(A).

6. Perron-Frobenius theorem: If $A = (a_{ij})$ is a real $n \times n$ matrix with nonnegative entries $a_{ij} \geq 0$ and irreducible, then there is a real eigenvalue r of A such that

$$\min_{i \in \{1,2,\ldots,n\}} \sum_{j=1}^{n} a_{ij} \leq r \leq \max_{i \in \{1,2,\ldots,n\}} \sum_{j=1}^{n} a_{i,j}$$

and any other eigenvalue λ satisfies $|\lambda| \leq r$. A matrix is *reducible* if there is a subset $I \subseteq \{1, 2, \ldots, n\}$ such that $a_{ij} = 0$ for all $i \in I$ and $j \notin I$. In particular, an adjacency matrix of a graph is irreducible iff the graph is connected. The Perron–Frobenius eigenvalue r is simple. Both right and left eigenspaces associated with r are one-dimensional.

Important characteristic of matrices, which is connected to eigenvalues is whether given matrix is *semidefinite*. A symmetric $n \times n$ matrix A is called *positive semidefinite*, if all of its eigenvalues are nonnegative. This property is often denoted by $A \succeq 0$. The matrix is *positive definite*, if all of its eigenvalues are positive. There are many equivalent ways of defining positive semidefinite matrices some of which we state in the list below, which we have from the publication of Lovasz [57].

Lemma 6. *([45]) For a real symmetric $n \times n$ matrix A, the following are equivalent:*

1. *A is positive semidefinite.*
2. *The quadratic form $x^\intercal Ax$ is nonnegative for every $x \in \mathcal{R}^n$.*
3. *$A = U^\intercal U$ for some matrix U.*
4. *A is nonnegative linear combination of matrices of the type $xx^\intercal$.*
5. *The determinant of every symmetric minor of A is nonnegative.*

Observe that from 2 follows that the diagonal entries of any positive semidefinite matrix are nonnegative. The sum of two positive semidefinite matrices is again positive semidefinite, this follows from 2 again.

Definition 15. *Let $A \in \mathcal{R}^{n \times n}, x \in \mathcal{R}^n$. The quantity*

$$r_A(x) = \frac{x^\intercal Ax}{x^\intercal x} \in \mathcal{R}$$

is called a Rayleigh quotient.

Observe that if x is an eigenvector of A, then $Ax = \lambda x$ and

$$r_A(x) = \frac{\lambda x^\intercal x}{x^\intercal x} = \lambda.$$

In generality one can show that $\forall x \in \mathcal{R}^n, \lambda_{\min} \leq r(x) \leq \lambda_{\max}$.

Theorem 8. *For the k-th largest eigenvalue λ_k of a symmetric $n \times n$ matrix A holds*

1.

$$\lambda_k \leq r_A(x), \text{ for } x \in \langle u_1, \ldots, u_k \rangle \tag{2.10}$$

and

$$\lambda_k \geq r_A(x) \text{ for } x \in \langle u_1, \ldots, u_{k-1} \rangle^{\perp} \tag{2.11}$$

where $u_1, \ldots, u_k$ are eigenvectors associetad with eigenvalues, $\lambda_1, \ldots, \lambda_k$.

2.

$$\lambda_k = \max_{dimU=k} \min_{x \in U} \frac{x^\intercal Ax}{x^\intercal x} = \min_{dimU=k-1} \max_{x \perp U} \frac{x^\intercal Ax}{x^\intercal x}. \tag{2.12}$$

In terms of Rayleigh quotient it is

$$\lambda_k = \max_{dimU=k} \min_{x \in U} r_A(x) = \min_{dimU=k-1} \max_{x \perp U} r_A(x), \tag{2.13}$$

where the maximum/minimum is over all subspaces U of a given dimension and over all nonzero vectors x in the respective subspace. In particular, this leads to:

$$\lambda_1 = \max_{x \neq 0} \frac{x^\intercal Ax}{x^\intercal x} = \max_{||x||_2=1} x^\intercal Ax$$

and

$$\lambda_2 = \max_{x \perp \vec{1}} \frac{x^\intercal Ax}{x^\intercal x} = \max_{x \perp \vec{1}, ||x||_2=1} x^\intercal Ax,$$

where $\vec{1}$ is the all-1 vector and the second equality follows since we can replace x by $x/||x||$, since the first maximum is over all nonzero vectors x.

Proof. We only prove the first part of an equation (2.12). The proof of the second one is analogous. First note that the Rayleigh quotient is the invariant under replacing x by any nonzero multiple of cx. Thus we can without loss of generality assume that x is a unit vector that is $||x||_2 = 1$ and hence $x^\intercal x = 1$ (by replacing $x \to cx$ with $c = 1/||x||_2$, if necessary).

Now we can take an orthonormal basis of eigenvectors $u_1, \ldots, u_n$. As this is the basis of $\mathcal{R}^n$ x can be written as $x = \sum_{i=1}^n a_i u_i$ and the expression $x^\intercal Ax$ reduces to

$$x^\intercal Ax = (\sum_{i=1}^n a_i u_i)^\intercal A(\sum_{i=1}^n a_i u_i) = \sum_{i,j=1}^n \langle u_i, \lambda_j u_j \rangle = \sum_{i=1}^n \lambda_i,$$

where the last equality follows from the fact that the scalar product of vectors u^i and u^j is 1 if $i = j$, and is 0 otherwise. Similarly we have that $x^\intercal x = \sum_{i=1}^n a_i^2$, and for unit vector x we get $\sum_{i=1}^n a_i^2 = 1$. Consequently, the Rayleigh quotient $r_A(x)$ of unit vector x can be interpreted as the weighted average of the eigenvalues.

Now let U be the subspace of $\mathcal{R}^n$ generated by the first k eigenvectors $u_1, \ldots, u_k$. We get $x^\intercal Ax = \sum_{i=1}^k a_i^2 \lambda_i$ and $\sum_{i=1}^k a_i^2 = 1$ for any unit vector $x \in U$. Observe that weighted average $r_A(x)$ of the eigenvalues $\lambda \geq \cdots \geq \lambda_k$ is at least the smallest of the first k eigenvalues, so $\min_{x \in U} r_A(x) \geq \lambda_k$ holds for this special k-dimensional subspace U. This proves 1. Because $U = \langle u_1, \ldots, u_k \rangle$ and

$$\forall x \in U \ \lambda_k \leq \frac{x^\intercal Ax}{x^\intercal x} = r_A(x).$$

This also proves that exists $U \subseteq \mathcal{R}^n$ with $\dim U = k$, for that holds $\lambda_k \leq \min_{x \in U} r_A(x)$ and thus it also holds that

$$\lambda_k \leq \max_{\dim U = k} \min_{x \in U} \frac{x^\intercal A x}{x^\intercal x}.$$

On the other hand, consider any subspace U of dimension k and subspace V of dimension $n-k+1$ generated by the last $n-k+1$ eigenvectors $u^k, u^{k+1}, \ldots, u^n$. Since $r_A(z)$ is a weighted average of the last $n-k+1$ eigenvectors $u_k, u_{k+1}, \ldots, u_n$ and the largest of these eigenvalues being λ_k, this proves

$$\forall x \in V = \langle u_1, \ldots, u_n \rangle^\perp \; \lambda_k \geq r_A(x).$$

We know that these two subspaces must have a nontrivial intersection. There must exist a nonzero vector $z \in U \cap V$. Without a loss of generality we can assume that $z = \sum_{j=k}^n b_j u^j$ is a unit vector, thus $z^\intercal z = \sum_{j=k}^n b_j^2 = 1$ and we obtain $r_A(z) = \sum_{j=k}^n b_j^2 \lambda_j \leq \lambda_k$, since $r_A(z)$ is a weighted average of the last $n-k+1$ eigenvalues and the largest of these eigenvalues being λ_k. So we have that $\min_{x \in U} r_A(x) \leq r_A(z) \leq \lambda_k$ holds for any k-dimensional subspace. This proves that

$$\lambda_k \geq \max_{\dim U = k} \min_{x \in U} \frac{x^\intercal A x}{x^\intercal x}.$$

□

2.3.2 Introduction to spectral graph theory

In this section we introduce some concepts and methods of spectral graph theory. The spectral graph theory has a connection to virtually all methods of the graph partitioning. The *spectrum* of matrix is the collection of its eigenvalues. By analysing spectrum of matrices associated with a graph, we can figure out lot of information about the graph itself. Now we introduce some of these matrices, see how they relate to each other. We will follow the publication of Laszlo [57]. We introduce the adjacency matrix, the Laplacian and the transition matrix of the random walk and their eigenvalues.

Let G be a finite, undirected, simple graph $G = (V, E)$ with a node set $V = \{1, \ldots, n\}$. The *adjacency matrix* of G is defined as the $n \times n$ matrix $A_G = (A_{ij})$ in which

$$A_{ij} = \begin{cases} 1, & \text{if } i \text{ and } j \text{ are adjacent,} \\ 0, & \text{otherwise.} \end{cases}$$

This definition can be extended to the weighted case and to the graphs with multiple edges. We just let A_{ij} be the weight of the edge (i, j) or the number of edges connecting i and j in case of multigraph. Information about loops can be on the diagonal.

We define *Laplacian* of the graph as the $n \times n$ matrix L_G in which:

$$L_{ij} = \begin{cases} d_i & \text{if } i = j \\ -A_{ij} & \text{if i and j are adjacent} \\ 0 & \text{otherwise} \end{cases} \tag{2.14}$$

Here d_i denotes the degree of the node i. In the case of weighted graphs we define $d_i = \sum_{j=1}^{n} A_{ij}$. So it holds

$$L_G = D_G - A_G,$$

where D_G is the diagonal matrix of the degrees of G.

One commonly used matrix which is derived from the Laplacian is *the normalized Laplacian of Graph* $\mathcal{L}(G)$ defined as:

$$\mathcal{L}_{u,v} = \begin{cases} 1 & \text{if } u = v \text{ and } d_v \neq 0 \\ -\frac{1}{\sqrt{d_v d_u}} & \text{if u and v are adjacent} \\ 0 & \text{otherwise} \end{cases} \tag{2.15}$$

Let D_G denote the diagonal matrix with the (v, v)-th entry having value d_v. Then we can write:

$$\mathcal{L} = D_G^{-1/2} L_G D_G^{-1/2}.$$

Because the order of $\mathcal{L}(G)$ equals to the number of graph vertices and $\mathcal{L}(G)$ is symmetric, we have n nonegative eigenvalues and their corresponding eigenvectors. Let $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ be the eigenvalues of $\mathcal{L}$ and $w_1, \ldots, w_n$ associated eigenvectors. We use them in the following sections. Observe that the vector

$$w_1 = \frac{1}{\sqrt{2m}} \cdot \vec{1} D^{1/2} \tag{2.16}$$

is associated with the eigenvalue λ_1. As we will see, the second eigenvalue λ_2 is often referred as *the spectral gap* λ_G. We further introduce the spectral gap in the section 2.3.3.

The *transition matrix of the random walk on* G is defined as the $n \times n$ matrix $P_G = (P_{ij})$ in which

$$P_{ij} = \frac{1}{d_i} A_{ij}. \tag{2.17}$$

So $P_G = D_G^{-1} A$. We use the transition matrix in the section 2.3.6 about random walks.

The matrices A_G and L_G are symmetric thus their eigenvalues are real. The matrix P_G is not symmetric, but it is conjugate to a symmetric matrix. Let

$$N_G = D_G^{-1/2} A_G D_G^{-1/2},$$

then N_G is symmetric and

$$P_G = D_G^{-1/2} N_G D_G^{1/2}.$$

The matrix N_G is symmetric, so its eigenvalues are real. The matrices P_G and N_G have the same eigenvalues and so all eigenvalues of P_G are real as well. We denote the eigenvalues of our matrices

$$\begin{aligned} A_G: &\quad \beta_1 \geq \beta_2 \geq \cdots \geq \beta_n, \\ L_G: &\quad \mu_1 \leq \mu_2 \leq \cdots \leq \mu_n \\ P_G: &\quad \nu_1 \geq \nu_2 \geq \cdots \geq \nu_n, \\ \mathcal{L}_G: &\quad \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n. \end{aligned}$$

To complete the list of common matrices associated with graph, we also use the *incidence matrix* of G. It comes in two flavors. Let $G = (V, E)$ and $V = \{1, \ldots, n\}$ and $E = \{e_1, \ldots, e_m\}$, and let B_G be the $n \times m$ matrix for which

$$(B_G)_{ij} = \begin{cases} 1 & \text{if } i \text{ is an endpoint of } e_j \\ 0 & \text{otherwise} \end{cases}$$

The following matrix similar to this and many times is more useful. In graph G fix an orientation of each edge and let vec(G) be the resulting graph. Then let $B_{\vec{G}}$ be the $n \times m$ matrix for which

$$(B_{\vec{G}})_{ij} = \begin{cases} 1 & \text{if } i \text{ is the head of } e_j \text{ in } \vec{G} \\ -1 & \text{if } i \text{ is the tail of } e_j \text{ in } \vec{G} \\ 0 & \text{otherwise} \end{cases}$$

Now to relate the incidence matrix to the Laplacian L_G, observe that independently of chosen orientation of edges holds:

$$L_G = B_{\vec{G}} B_{\vec{G}}^{\intercal}.$$

It is worth while to express this equation in terms of quadratic forms:

$$x^{\intercal} L_G x = \sum_{(i,j) \in E)}^{n} (x_i - x_j)^2. \tag{2.18}$$

2.3.3 The largest eigenvalue and the spectral gap

In this section we continue to follow the publication of Laszlo [57] and explore the properties of the largest eigenvalue $\beta_{\max}$ associated with Adjacency matrix and introduce the notion of the spectral gap. Thanks to the Perron-Frobenius Theorem we immediately see that if G is connected, then the largest eigenvalue $\beta_{\max}$ of A_G has multiplicity 1. This eigenvalue corresponds roughly to the average degree of the graph. More precisely, let $d_{\min}$ denote the minimum degree of G, let $\bar{d}$ be the average degree, and let $d_{\max}$ be the maximum degree. Then the following lemma holds [57].

Lemma 7. *([57]) For every graph G,*

$$\max\{\bar{d}, \sqrt{d_{max}}\} \leq \beta_{max} \leq d_{max}.$$

On the other hand, for the Laplacian L_G this corresponds to the smallest eigenvalue, which is really uninteresting, since it is 0.

Much more interesting value is the gap between the second and the first eigenvalue. If the graph is connected, then the largest eigenvalue of the adjacency matrix as well as the smallest eigenvalue of the Laplacian have multiplicity 1. The gap between this and the nearest eigenvalue is called *the spectral gap* or *the eigenvalue gap* λ_G. As the spectral gap is also referred the second smallest eigenvalue λ_2 of the normalized Laplacian $\mathcal{L}$ of G and we use it in this form. In this form we prove the Cheeger inequality in the section 2.3.5 and the relation of the spectral gap to the speed of random walk convergence in the section 2.3.7.

2.3.4 Expanders and eigenvalues

We have already mentioned the expanders in the previous sections. Now we explore the connection between the expanders and the eigenvalues. We still follow the publication of Laszlo [57] in this section. Expanders play an important role in many applications of the graph theory, in particular in the computer science. The most important expanders are d-regular expanders, where $d \geq 3$ is a small constant. It is relatively easy to prove that such graphs exist by the ways of probabilistic method. But an explicit construction of such graphs is not easy. Besides the random construction methods the first deterministic construction was found by Margulis [59]. Most of these constructions are based on deep algebraic facts. An example of random construction is to pick d random perfect matching on $2n$ nodes (independently, uniformly over all perfect matchings) and let G be the union of them. Then after some moderately difficult analysis we deduce that with positive probability for sufficiently small c is G d-regular c-expander. Now we want to state and prove spectral characterization of d-regular expanders. Since we consider only regular graphs, the adjacency matrix, the Laplacian and the transition matrix are easily expressed and so we shall only consider the adjacency matrix.

Theorem 9. *([57]) Let G be a d-regular graph.*

1. *If $d - \beta_2 \geq 2cd$, then G is an c-expander.*
2. *If G is an c-expander, then $d - \beta_2 \geq \frac{c^2}{5}$.*

Proof of this statement can be found in Lovasz paper [57]. We present the connection between the graph expansion properties and the spectral gap in the section 2.3.5 by proving the Cheeger inequality. The Cheeger inequality expresses this connection in terms of conductance, but expansion and conductance are closely related and for d-regular graphs holds $\frac{\alpha(G)}{d} = \Phi(G)$.

2.3.5 The proof of the Cheeger inequality using eigenvectors

In this section we follow the publication of Chung [42] and present the proof of the Cheeger inequality in the form

$$2\Phi(G) \geq \lambda_G \geq \frac{\alpha_G^2}{2} \geq \frac{\Phi_G^2}{2},$$

where α_G is the minimum conductance of all sets associated with the largest i coordinates of the eigenvector w_2 which is associated with λ_G. This way we can focus on a linear number of choices for the cut, using the order determined by the eigenvector instead of trying the exponential number of possibilities. The above Cheeger inequality guarantess that the cut resulting from this efficient algorithm has the conductance within quadratic factor of the optimum. This inequality provides connection between the eigenvectors and the good cuts. As we have seen in the section 2.3.1, the eigenvalue can be expressed in terms of the Rayleigh quotient. First observe that to the first eigenvalue $\lambda_1 = 0$ of $\mathcal{L}$ is associated

eigenvector $v_1 = D_G\vec{1}$. This is n-th largest eigenvalue of $\mathcal{L}$ and $\lambda_G = \lambda_2$ is the $n-1$-largest eigenvalue, so thanks to the equation (2.11) it holds that

$$\lambda_G \leq r_{\mathcal{L}}(x) \text{ for } x \in U = \{x \in \mathcal{R}^n, x \perp w_1\} = \langle w_n, \ldots, w_2 \rangle.$$

From characterization of Rayleigh quotient we know that, $r_{\mathcal{L}}(v_k) = \lambda_k$. So it holds

$$\lambda_G = \min_{x \perp w_1} r_{\mathcal{L}}(x) = \min_{x \perp 1/\sqrt{2m} \cdot D_G^{1/2}\vec{1}} \frac{x^\intercal \mathcal{L} x}{x^\intercal x}.$$

Recall the equation (2.16) from the section 2.3.2 for that w_1 is the first eigenvector of the normalized Laplacian. Now we show the inequality

$$\lambda_G \leq 2\Phi(G).$$

Consider

$$\frac{x^\intercal \mathcal{L} x}{x^\intercal x} = \frac{x^\intercal D^{-1/2} L D^{-1/2} x}{x^\intercal x} = \frac{(D^{-1/2}x)^\intercal L (D^{-1/2}x)}{x^\intercal x}.$$

Now substitute $y = D^{-1/2}x$ and use quadratic form of Laplacian (2.18). Observe that effectively the substitution can be written as $y_i = \frac{x_i}{\sqrt{d_i}}$ for i-th coordinate. This leads to

$$\frac{(D^{-1/2}x)^\intercal L (D^{-1/2}x)}{x^\intercal x} = \frac{\sum_{i \sim j}(y_i - y_j)^2}{\sum_{i=1}^n y^2 d_i}, \tag{2.19}$$

where we denote by $i \sim j$ the adjacency between vertex i and j in G. Thus the sum goes over all edges of G. Let define another form of rayleigh quotient $R_{\mathcal{L}}$ as:

$$R_{\mathcal{L}}(y) = \frac{\sum_{i \sim j}(y_i - y_j)^2}{\sum_{i=1}^n y^2 d_i}. \tag{2.20}$$

Now let S be the set of vertices, such that $\Phi(G) = \frac{|E(S,\bar{S})|}{\min\{vol(S), vol(\bar{S})\}}$, then let $\tilde{x}$ be the vector

$$\tilde{x}_i = \begin{cases} \sqrt{d_i}(1 - \frac{vol(S)}{vol(G)}) & i \in S \\ \sqrt{d_i}(-\frac{vol(S)}{vol(G)}) & i \in \bar{S} \end{cases}$$

You can see that $\tilde{x} \perp w_1$:

$$\begin{aligned}
\tilde{x}^\intercal w_1 &= \tilde{x}^\intercal \cdot \tfrac{1}{\sqrt{2m}} D_G^{1/2}\vec{1} && \text{see (2.16)} \\
&= \tfrac{1}{\sqrt{2m}} \cdot (\textstyle\sum_{i \in S} \sqrt{d_i}(1 - \frac{vol(S)}{vol(G)})\sqrt{d_i} + \sum_{i \in \bar{S}} \sqrt{d_i}(-\frac{vol(S)}{vol(G)})\sqrt{d_i}) \\
&= \tfrac{1}{\sqrt{2m}} \cdot \left(vol(S)(1 - \tfrac{vol(S)}{vol(G)}) + vol(\bar{S})(-\tfrac{vol(S)}{vol(G)}) \right) \\
&= \tfrac{1}{\sqrt{2m}} \cdot \left(\tfrac{vol(S)vol(G) - vol(S)(vol(S) + vol(\bar{S})}{vol(G)} \right) \\
&= \tfrac{1}{\sqrt{2m}} \cdot \left(\tfrac{vol(S)vol(G) - vol(S)(vol(G))}{vol(G)} \right) \\
&= 0.
\end{aligned}$$

Now we substitute to equation (2.19) (Let $\tilde{y}_i = \frac{\tilde{x}_i}{\sqrt{d_i}}$). For numerator we get

$$\sum_{i \sim j}(\tilde{y}_i - \tilde{y}_j)^2 = \sum_{i \sim j, i \in S, j \in \bar{S}} 1 = |E(S, \bar{S})|.$$

For denominator we get

$$\begin{aligned}
\sum_{i=1}^{n} y_i^2 d_i &= \sum_{i\in S}(1-\tfrac{vol(S)}{vol(G)})^2 + \sum_{i\in \bar{S}}(-\tfrac{vol(S)}{vol(G)})^2 \\
&= vol(S)(1-\tfrac{2vol(S)}{vol(G)} + \tfrac{vol^2(S)}{vol^2(G)}) + vol(\bar{S})\tfrac{vol^2(S)}{vol^2(G)} \\
&= \tfrac{vol^2(S)(vol(S)+vol(\bar{S}))+vol(S)vol^2(G)-2vol(S)vol(G)}{vol^2(G)} \\
&= \tfrac{vol^2(S)vol(G)+vol(S)vol^2(G)-2vol(S)vol(G)}{vol^2(G)} \\
&= \tfrac{vol(S)(vol(S)+vol(G)-2vol(S))}{vol(G)} \\
&= \tfrac{vol(S)(vol(\bar{S}))}{vol(G)} \\
&\geq 1/2\min\{vol(S), vol(\bar{S})\}.
\end{aligned}$$

Putting everything together, we have

$$\lambda_G \leq r_{\mathcal{L}}(\tilde{x}) = \frac{(D^{-1/2}\tilde{x})^\intercal L(D^{-1/2}\tilde{x})}{\tilde{x}^\intercal\tilde{x}} = \frac{\sum_{i\sim j}(\tilde{y}_i-\tilde{y}_j)^2}{\sum_{i=1}^n \tilde{y}^2 d_i} \leq \frac{2|E(S,\bar{S}|}{\min\{vol(S), vol(\bar{S})\}} \leq 2\Phi(G).$$

Already in this part of the proof you can see how the eigenvectors relate to the good cuts. Now we focus on the second part of the inequality, namely

$$\lambda_G \geq \frac{\alpha_G^2}{2} \geq \frac{\Phi^2(G)}{2},$$

where α_G is the minimum conductance of subsets S_i consisting of vertices with the i largest values in the eigenvector associated with λ_G, over all i. Now let f denote an eigenvector achieving λ_G. Then let $g = D^{-1/2}\cdot f$, speaking of coordinates it is $g_i = \frac{f_i}{\sqrt{d_i}}$. Recall equation (2.19) to see that

$$\lambda_G = \frac{(D^{-1/2}f)^\intercal L(D^{-1/2}f)}{f^\intercal f} = \frac{\sum_{i\sim j}(g_i-g_j)^2}{\sum_{i=1}^n g^2 d_i} = R_{\mathcal{L}}(g). \tag{2.21}$$

As we have seen, $\lambda_G = \min_{x\perp w_1} r_{\mathcal{L}_G}(g)$ and $\frac{1}{\sqrt{2m}}\cdot\sum_{i\in V}\sqrt{d_i}\cdot f_i = 0$. From this and from the definition of g follows:

$$\sum_{i\in V} g_i d_i = 0\ . \tag{2.22}$$

We order the vertices so that

$$g_{i_1} \geq g_{i_2} \geq \cdots \geq g_{i_n}.$$

Let $S_j = \{i_1, \ldots, i_j\}$ and define

$$\alpha_G = \min_j \Phi(S_i).$$

Now let r denote the largest integer such that $vol(S_r) \leq vol(G)/2$. This together with (2.22) this leads to

$$\sum_{i\in V} g_i^2 d_i = \min_c \sum_{i\in V}(g_i-c)^2 d_i \leq \sum_{i\in V}(g_i-g_{i_r})^2 d_i\ .$$

Now we let g^+ and g^- be the positive and negative part of $g - g_{i_r}$ as follows:

$$g_i^+ = \begin{cases} g_i - g_{i_r} & \text{if } g_i \geq g_{i_r} \\ 0 & \text{otherwise} \end{cases}$$

$$g_i^- = \begin{cases} |g_i - g_{i_r}| & \text{if } g_i \leq g_{i_r} \\ 0 & \text{otherwise} \end{cases}$$

We consider

$$\begin{aligned} \lambda_G &= \frac{\sum_{i\sim j}(g_i - g_j)^2}{\sum_{i\in V} g_i^2 d_i} \\ &\geq \frac{\sum_{i\sim j}(g_i - g_j)^2}{\sum_{i\in V}(g_i - g_{i_r})^2 d_i} \\ &\geq \frac{\sum_{i\sim j}((g_i^+ - g_j^+)^2 + (g_i^- - g_j^-)^2)}{\sum_{i\in V}((g_i^+)^2 + (g_i^-)^2) d_i} \end{aligned}$$

We can assume, without loss of generality that $R_{\mathcal{L}}(g^+) \leq R_{\mathcal{L}}(g^-)$ and therefore we have $\lambda_G \geq R_{\mathcal{L}}(g^+)$ since

$$\frac{a+b}{c+d} \geq \min\{\frac{a}{b}, \frac{b}{d}\}.$$

We here use the notation

$$\tilde{vol} = \min\{vol(S), vol(G) - vol(S)\},$$

so that

$$|E(S_i, V \setminus S_i)| \geq \alpha_G \cdot \tilde{vol}(S_i).$$

Than we have

$$\begin{aligned} \lambda_G &\geq R_{\mathcal{L}}(g^+) \\ &= \frac{\sum_{i\sim j}(g_i^+ - g_j^+)^2}{\sum_{i=1}^n (g_i^+)^2 d_i} \\ &= \frac{(\sum_{i\sim j}(g_i^+ - g_j^+)^2)(\sum_{i\sim j}(g_i^+ + g_j^+)^2)}{(\sum_{i=1}^n (g_i^+)^2 d_i)(\sum_{i\sim j}(g_i^+ - g_j^+)^2)} \\ &\geq \frac{\sum_{i\sim j}((g_i^+)^2 - (g_j^+)^2)^2}{2(\sum_{i=1}^n (g_i^+)^2 d_i)^2} && \text{by the Cauchy-Schwarz inequality,} \\ &= \frac{(\sum_{j=1}^{n-1} |(g_{i_j}^+)^2 - (g_{i_{j+1}}^+)^2| \cdot |E(S_i, V \setminus S_i)|)^2}{2(\sum_{i=1}^n (g_i^+)^2 d_i)^2} && \text{by counting,} \\ &\geq \frac{(\sum_{j=1}^{n-1} |(g_{i_j}^+)^2 - (g_{i_{j+1}}^+)^2| \cdot |\alpha_G| \tilde{vol}(S_i))^2}{2(\sum_{i=1}^n (g_i^+)^2 d_i)^2} && \text{by the def. of } \alpha_G \\ &= \frac{\alpha_G}{2} \cdot \frac{(\sum_{j=1}^{n-1} (g_{i_j}^+)^2 (|\tilde{vol}(S_j) - \tilde{vol}(S_{j+1})|))^2}{(\sum_{i=1}^n (g_i^+)^2 d_i)^2} \\ &= \frac{\alpha_G}{2} \cdot \frac{(\sum_{j=1}^{n} (g_{i_j}^+)^2 d_{i_j})^2}{(\sum_{i=1}^n (g_i^+)^2 d_i)^2} \\ &= \frac{\alpha_G}{2} \ . \end{aligned}$$

Which finally proves

$$2\Phi(G) \geq \lambda_G \geq \frac{\alpha_G}{2} \geq \frac{\Phi^2(G)}{2} \ .$$

2.3.6 Spectral gap and random walks

In this section we introduce the notion of random walks and how they are connected to the spectral gap and expansion properties of graphs. In this section we use monology of Chung [41] and work of Lovasz [57].

A *random walk* on a graph $G = (V, E)$ is a random sequence $(v^0, v^1, \dots)$ of nodes constructed as follows: We specify some initial distribution σ on vertex set V. We choose a vertex $v^0 \in V$ according to the distribution σ as a starting point. Now we select a vertex v^1 from the neighbors of v^0. Probability of selecting each neighbor is $1/d(v^0)$. Then we select v^2 from the neighbors of v^1 uniformly at random and so on. The probabilities $P[v^k = v], v \in V$ form the distribution denoted by σ^k. Important feature of the random walk is the fact that it is independent of its history. It does not matter how many times you get to some vertex $v \in V$. The probability $P[v^{i+1} = v | v^i = u]$ of getting from the vertex u to the vertex v is always the same. It is independent of the number of steps or the number of visits. Observe the simple fact that

$$\forall u \in V, \sum_{v \in V} P(u, v) = 1,$$

where P is the transition probability matrix introduced in section 2.3.2 and $P_{uv} = P(u, v)$. We can see the distribution $\sigma : V \to \mathcal{R}$ as a non-negative real vector with $\sum_{v \in V} \sigma(v) = 1$. It also holds that $\sigma^k = \sigma \cdot P^k$. The random walk is said to be *ergodic* if there is a unique *stationary distribution* π satisfying

$$\lim_{s \to \infty} \sigma P^s = \lim_{s \to \infty} \sigma^s = \pi.$$

Observe that the necessary conditions for the ergodicty of P are

1. *irreducibility*, i.e., for any $u, v \in V$, there exists some s such that $P^s(u, v) > 0$. This means that the graph is connected. Every vertex $v \in V$ is reachable from every vertex $u \in V$.

2. *aperiodicity*, i.e., $\gcd\{s : P^s(u, u) > 0\} = 1$. This means that random walk does not repeatedly alternate between 2 or more parts of graph. This holds for any non-bipartite graph.

As it turns out, these are also sufficient conditions (see e.g. [16]). Our major concern is to determine number of steps for P^s to be *close* to its stationary distribution, given an arbitrary initial distribution. It is easy to check that the distribution $\pi_i = \frac{d_i}{2m}$ is stationary. Algebraically, this means that π is a left eigenvector of P with eigenvalue 1:

$$\pi^\intercal P = \pi^\intercal.$$

We recall the observation that

$$P = D^{-1}A = D^{-1/2}(I - \mathcal{L})D^{1/2} \ . \tag{2.23}$$

It is important to realize the connection of eigenvalues of the transition matrix P and the normalized Laplacian $\mathcal{L}$. Consider the following: let $w_i, 1 \le i \le n$ be orthonormal eigenvectors of $\mathcal{L}$, then define vectors $\forall i, 1 \le i \le n, \nu_i = w_i \cdot D^{1/2}$. Now by multiplying the P by ν_i from the left we get:

$$\begin{aligned} \nu_i \cdot P = w_i \cdot D^{1/2} D^{-1/2}(I - \mathcal{L})D^{1/2} &= w_i - w_i \mathcal{L} D^{1/2} \\ &= w_i - \lambda_i \cdot w_i D^{1/2} = (1 - \lambda_i) w_i D^{1/2} \ . \end{aligned} \tag{2.24}$$

Now we want to show how quickly $f \cdot P^k$ converges to the stationary distribution for any initial distribution $f : V \to \mathcal{R}$. Suppose that

$$f \cdot D^{-1/2} = \sum_{i=1}^{n} a_i w_i,$$

thus also holds

$$f = \sum_{i=1}^{n} a_i w_i D^{1/2}.$$

Recall the equation (2.16) from the section 2.3.2 for vector w_1. This leads to:

$$a_1 = \frac{\langle \sum_{i=1}^{n} a_i w_i, w_1 \rangle \cdot \sqrt{2m}}{\sqrt{2m}} = \frac{\langle f \cdot D^{-1/2}, \vec{1} \cdot D^{1/2} \rangle}{||\vec{1} \cdot D^{1/2}||_2} = \frac{\langle f, \vec{1} \rangle}{\sqrt{2m}} = \frac{1}{\sqrt{2m}}.$$

The first equality can be checked by realizing that vectors $w_i, 1 \leq i \leq n$ are orthonormal. The second equality should be clear from discussion above. The last equality follows from the fact that:

$$\sum_{v \in V} f(v) = \langle f, \vec{1} \rangle = 1. \tag{2.25}$$

The stationary distribution can be expressed as:

$$\pi = \frac{1}{2m} \vec{1} D.$$

Then putting all together we get

$$\begin{aligned}
||f \cdot P^s - \pi||_2 &= ||f \cdot P^s - \tfrac{1}{2m}\vec{1}D||_2 && \text{see (2.25)} \\
&= ||f \cdot P^s - a_1 w_1 D^{1/2}||_2 && \text{see (2.16)} \\
&= ||f D^{-1/2}(I - \mathcal{L})^s D^{1/2} - a_1 w_i D^{1/2}||_2 && \text{see (2.23)} \\
&= ||\textstyle\sum_{i=2}^{n} (1-\lambda_i)^s a_i w_i D^{1/2}||_2 && \text{see (2.24)} \\
&= \sqrt{\textstyle\sum_{j=1}^{n} \left(\sum_{i=2}^{n} (1-\lambda_i)^s a_i w_i^j \sqrt{d_j}\right)^2} \\
&\leq (1-\lambda')^s \max_{x \in V} \sqrt{d(x)} \sqrt{\textstyle\sum_{j=1}^{n} \left(\sum_{i=2}^{n} a_i w_i^j\right)^2} \\
&\leq (1-\lambda')^s \max_{x \in V} \sqrt{d(x)} ||f D^{-1/2}||_2 \\
&= (1-\lambda')^s \max_{x \in V} \sqrt{d(x)} \sqrt{\textstyle\sum_{i=1}^{n} (f_i \frac{1}{\sqrt{d_i}})^2} \\
&\leq (1-\lambda')^s \frac{\max_{x \in V} \sqrt{d(x)}}{\min_{y \in V} \sqrt{d(y)}} ||f||_2 \\
&\leq (1-\lambda')^s \frac{\max_{x \in V} \sqrt{d(x)}}{\min_{y \in V} \sqrt{d(y)}} \\
&\leq e^{-s\lambda'} \frac{\max_{x \in V} \sqrt{d(x)}}{\min_{y \in V} \sqrt{d(y)}},
\end{aligned}$$

where

$$\lambda' = \begin{cases} \lambda_2 & \text{if } 1 - \lambda_2 \geq \lambda_n - 1 \\ 2 - \lambda_n & \text{otherwise} \end{cases}$$

Or in another words:

$$|1 - \lambda'| = \max_i |1 - \lambda_i|.$$

After substituting ϵ for $||f \cdot P^s - \pi||_2$ and expressing s we get

$$s \geq 1/(\lambda' \log \left(\frac{\max_{x \in V} \sqrt{d(x)}}{\epsilon \min_{y \in V} \sqrt{d(y)}} \right)),$$

where the l_2 distance between fP^s and stationary distribution is at most ϵ. So we see that (up to logarithmic factors), s is the reciprocal of the spectral gap that governs the mixing time.

The appearance of the smallest eigenvalue λ_n is usually not important in most applications. In fact, only λ_2 is crucial in the following sense. Note that λ' is either λ_2 or $2 - \lambda_n$. Suppose the latter holds i.e., $\lambda_n - 1 \geq 1 - \lambda_2$. Then we can modify our random walk as follows. At each step we flip a coin and move with a probability $1/2$ and stay where we are with a probability $1/2$. The stationary distribution of this modified random walk (called lazy random walk) is the same and the transition matrix P is replaced by $1/2(P + I)$. This corresponds to the random walk on graph G' formed by adding d_v loop edges to each vertex v. The eigenvalues of the normalized Laplacian $\mathcal{L}(G')$ are half of the original, $\bar{\lambda_i} = \lambda_i/2 \leq 1$. Therefore,

$$1 - \bar{\lambda_2} \geq 1 - \bar{\lambda_n} \geq 0,$$

and the convergence bound in l_2 distance for the modified random walk becomes

$$2/\lambda_2(\log \left(\frac{\max_{x \in V} \sqrt{d(x)}}{\epsilon \min_{y \in V} \sqrt{d(y)}} \right)).$$

Together with the Cheeger inequality we have the relation between the spectral gap (recall that $\lambda_2 = \lambda_G$), the conductance and the mixing time of the random walks:

$$||f \cdot P^s - \pi||_2 \leq (1 - \lambda_G)^s \frac{\max_{x \in V} \sqrt{d(x)}}{\min_{y \in V} \sqrt{d(y)}} \leq (1 - \Phi(G)^2/2)^s \frac{\max_{x \in V} \sqrt{d(x)}}{\min_{y \in V} \sqrt{d(y)}} .$$

2.3.6.1 Mixing of random walks

In this section we want to intuitively describe the connection between the random walks and the expansion properties of graphs which we proved in section 2.3.6. If the random walk on a graph converges to the stationary distribution rapidly, then the graph has high expansion. On the other hand, slow convergence implies the low graph connectivity and the expansion. Why is it so? The initial distribution can be seen as the spread of particles on the vertices, each particle doing its own random walk. There is a good chance that in well connected components the number of particles at each vertex will be about the same after small number of steps. But when there is bridge or small cut between two graph parts, then it is hard for particles to get from one side to another. Another way to illustrate the concept is to spread the negative and positive charge on vertices. Negative charge will have value $-1/n$ and positive charge $1/n$. It will be given by function $f : \mathcal{R} \to V$, which can also be seen as a vector. We suppose that $\langle f, \vec{1} \rangle = 0$, where $\vec{1}$ is the vector of all ones. That is, the sum of all charges is zero. We mix the charge by application the random walk by multiplying the vector of charges

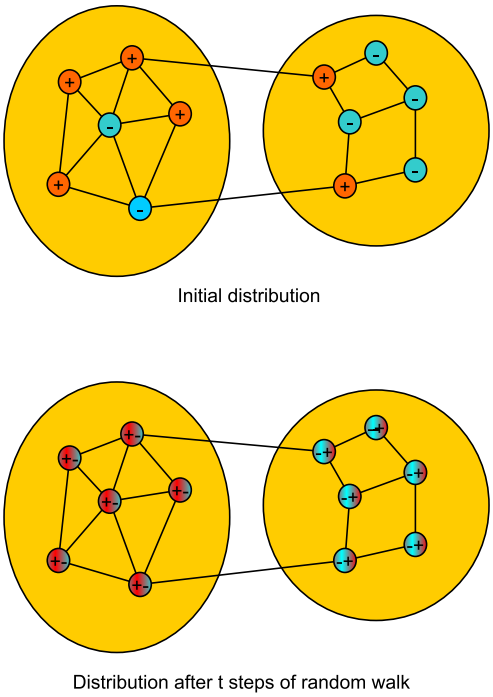


Figure 2.3: Charge spreading

f by transition matrix P. Let f_t be the assignment of charge values after t steps of the random walk. Then $f_0 = f, f_t = Pf_{t-1}$. After few steps, we can expect that the distribution of charge in well connected components will be about the same. Look at figure 2.3 for the graphic illustration.

Now thanks to the vector f_t we can identify these components and find a sparse cut. This principle is used to find good partitions by random walks. In spectral theory the spectral gap is connected to the mixing time of random walk and its eigenvector roughly resembles the vector f_t.

2.3.7 Spectral graph partitioning

In the previous sections we have seen the connection of the spectral gap λ_G and its associated eigenvector to the conductance $\Phi(G)$ of graph G. We can use this to formalize the spectral algorithm for graph partitioning. The key is computing the eigenvector associated with λ_G sometimes called *Fiedler eigenvector* named after the Czech mathematician Miroslav Fiedler who initiated the study of the algebraic graph theory [23]. The algorithm 2 is pseudocode of the simple version of spectral algorithm.

Algorithm 2 Spectral partitioning

function SPECTRALPARTITIONING($G = (V, E)$)
w:=second eigenvector of $\mathcal{L}$
Sort the entries of $w = (w_1, \ldots, w_n)$ as $w_{i_1} \leq \cdots \leq w_{i_{n/2}} \leq w_{i_n}$
$S_j = \{i_1, \ldots, i_j\}$
$\overline{S_j} = V \setminus S_j$
$k = \arg\min_{i=1}^{n} \Phi(S_i)$ **return** $(S_k, \overline{S_k})$
end function

2.3.7.1 The analogy of spectral partitioning with a vibrating string

This section and its subsection are inspired by work of Demmel [38]. Another way to find the intuition behind it, is to consider the knowledge from either physics or music. We know that taut string when plucked has certain *modes of vibration* or *harmonics*. Snapshot of such modes can look like this: Surprisingly, the eigenvectors of graph posses similar qualities. Especially, the second eigenvector of $L(G)$ corresponds to partition of graph to two parts effectively. Half of the string is labeled +, and half is labeled -, bisecting the string into two equal sized connected components. First eigenvector cant say anything about the structure of the graph, but the 3rd, 4th and following eigenvectors can be used for partitioning on more parts. Although, best results are achieved with 2nd eigenvector. Physical model that lies behind this analogy consist from a finite identical masses (nodes) connected by identical springs (edges). For G being the chain of nodes, by writing down Newton's Laws of motion for the masses and solving for the frequencies and shapes of the vibrational modes we will get precisely the eigenvalues and eigenvectors of the Laplacian $L(G)$ of the graph G. Therefore, the second eigenvector of $L(G)$ is the shape of second mode of vibration, and divides the graph in half.

Same intuition applies also in the of more complicated graphs. For example on planar graphs, we can think of them as a kind of trampoline, again, the second mode of vibration divides the graph (trampoline) into two halves. More about physical model behind it and about this intuition, can be again found as web resource [38].

2.3.7.2 Lanczos algorithm

The question remains, how to calculate this Fiedler vector. We do not need the exact numbers, so we can use some approximation. By treating $L(G)$ as a dense matrix, common tools as routine in Matlab, or dsyevx in LAPACK can compute the Fiedler eigenvector in time $O((4/3) \cdot n^3)$ [38]. For sparse graphs this clearly is not cost effective.

For this problem we can choose the *Lanczos algorithm*. Given any $n \times n$ sparse symmetric matrix A, Lanczos algorithm computes a $k \times k$ symmetric tridiagonal matrix T, whose eigenvalues are good approximations of the eigenvalues of A, and whose eigenvectors can be used to get the approximate eigenvectors of A. Typically the most expensive part of the algorithm are k matrix-vector multiplications with A. We hope to achieve good enough approximation with k much smaller than n. This way we approximate only small subset of A's n eigenvalues.

Fortunately, the smallest and the largest eigenvalues converge first, including λ_2. There are many flavors of Lanczos algorithm, we present the one, from work of Kabelikova [46].

Algorithm 3 Lanczos algorithm

function LANCZOS(Matrix $L \in \mathcal{R}^{n \times n}$, ϵ as tolerance)
  $v :=$ random vector $v \in \mathcal{R}^n$
  $u = L \cdot v$
  $j = 0$
  while $|\beta_j| > \epsilon$ **do**
    $\alpha_j = v^\intercal u$;
    $u = u - \alpha_j \cdot v$;
    $\beta_j = ||u||_2$;
    for i=0 To n **do**
      $tmp = v_i$;
      $v_i = \frac{u_i}{\beta_j}$;
      $u_i = -tmp \cdot \beta_j$;
    end for
    $u = u + L \cdot v$;
    $j = j + 1$
  end whilereturn $T_j \in \mathcal{R}^{j \times j}$
end function

At each step j, the algorithm produces tridiagonal matrix

$$T_j = \begin{pmatrix} \alpha_0 & \beta_0 & 0 & \dots & 0 \\ \beta_0 & \alpha_1 & \beta_1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \beta_{j-3} & \alpha_{j-2} & \beta_{j-2} \\ 0 & \dots & 0 & \beta_{j-2} & \alpha_{j-1} \end{pmatrix}$$

Whose eigenvalues approximate the eigenvalues of L. Let z be the second eigenvector of T_j. Then the corresponding approximate eigenvector of L is

$$w_2 = \sum_{i=0}^{j} z_i \cdot v_i$$

Where v_i are stored vectors produced at each step of the algorithm. For more implementation details see the work of Kabelikova [46] and the work of Parlett, Simon, Stringer [66].

2.4 Flow based approach

2.4.1 Expander flows

As we stated before, the time complexity of most graph partitioning problems is such that we do not hope for any polynomial time algorithm to solve them.

Problem of deciding whether a given graph G has expansion at least α is coNP-complete [13]. This means it has no short certificate unless the polynomial hierarchy collapses. Although, we can work with "approximate" certificates. In the section 2.1 we have seen the introduction of MFP and the notion of the demand graphs. Note that every multicommodity flow in graph G can be viewed as an *embedding* of a weighted demand graph H. As we have seen in the section 2.1 Leighton and Rao [53] used the approximate duality of Uniform Multicommodity Flow Problem and Sparsest cut problem to provide approximation algorithm. UMFP can be viewed as a embedding of complete graph into the underlying graph which provide approximate certificate for expansion. Not only we can tell if we can embedd such demand graph into the underlying graph, but we can measure how successfully we can do it. The measurement of success is called *congestion*. We say that H can be embedded (or routed) in G with congestion γ if flow of c_e units can be routed in G between the end-points of e *simultaneously* for all $e \in H$ without violating the edge-capacities of G by a factor more than γ. To provide the certificate of expansion we need to construct graph H which is well-connected, i.e. it has large expansion and can be routed with small congestion γ in the input graph G. If we succeed in building such graph, we can use it as a proof that $\alpha(G) > \alpha(H) \cdot \frac{1}{\gamma}$. To see this, consider arbitrary cut $(S, \bar{S}), 0 < |S| \leq n/2$ on vertices of G (and H as well, since vertices of G and H coincide). We know that to route H in G, we need to take at least $\alpha(H).|S|$ amount of traffic across the cut. The fact that we can do it with the congestion γ means that we can route at least $\frac{1}{\gamma} \cdot \alpha(H).|S|$, so we know

$$E(S, \bar{S}) \geq \frac{1}{\gamma} \cdot \alpha(H).|S|, \tag{2.26}$$

thus

$$\alpha(G) = \min_{\emptyset \neq S \subsetneq V} \alpha(S) \geq \frac{E(S, \bar{S})}{|S|} = \frac{\alpha(H)}{\gamma}.$$

This leads to generalization and improvement over Leighton-Rao approach, where we basically routed complete graph into G. Complete graph has indeed great expansion, but is routed with high congestion as well. As we have seen, It could approximate the expansion within an $\Theta(\log n)$. In ARV paper [10] there are results representing a continuation of that work but with a better certificate; for any graph $\alpha(G) = \alpha$ they can exhibit a certificate to the effect that the expansion is at least $\Omega(\alpha/\sqrt{\log n})$. But how effectively find such expander and how to route it? This is the main focus of research of flow-based approach to graph partitioning. One interesting way how to do it gave autors Arora, Hazan and Kale [7]. They creatively formulated problem as zero-sum game and used Freund-Shapire framework to effectively solve it. They achieved $O(n^2)$ time and retained $O(\sqrt{\log n})$ approximation of Sparsest cut problem. Next section is devoted to another creative way of computing expander flows.

2.4.2 The Cut-Matching game

Further exploration of the flow based approach gave a rise to so called Cut-Matching game framework, introduced in the work of Khander, Rao and Vazirani [50]. The goal of the game is to build a graph with a high expansion which

can be embedded into the underlying graph with the low congestion. The result of the game is either such expander proving lower bound on the expansion or the algorithm finds a sparse cut that proves upper bound on the expansion. Thus in its original form the algorithm approximately solves the decision problem of EXPANSION. The Cut-Matching game also uses the spectral theory ideas and concepts of random walks. Before reading this section we recommend reader to review the section 2.3.6. The key is the connection of spectral gap, mixing time of random walks and the graph expansion.

Following description of the Cut-Matching game is from the work of Orecchia [62] and we also use information from the paper on the Cut-Matching game [49] and the KRV paper [50]. Orrechia in his dissertation explored and further developed this framework. An instance of the Cut-Matching game can be defined as a triple $(\mathcal{H}(n), f(n), g(n))$ with an input graph $G = (V, E_G)$, where $\mathcal{H}(n)$ is a multi-round game between players $\mathcal{C}$, the Cut player and $\mathcal{M}$, the Matching player. $f(n)$ and $g(n)$ are positive functions. The strategy $\mathcal{C}$ identifies the Cut player and the strategy $\mathcal{M}$ identifies the Matching player. In the beginning of the game we have an empty weighted graph H_1 on the V, the vertex set of G. We assume $|V|$ to be even. Let $H_t = (V, E_t, \omega_t)$ be the weighted graph as a result after $t-1$ rounds of the game. Every round $t \geq 1$ starts by the Cut player $\mathcal{C}$ choosing a bisection $(S_t, \overline{S_t})$ of V. How the Cut player $\mathcal{C}$ chooses this bisection may depend on H_t and also on the actions of the players in the previous rounds. The Matching player then picks a perfect matching M_t across the bisection $(S_t, \overline{S_t})$ of V. The Matching player can choose any matching between S_t and $\overline{S_t}$. The graph M_t of perfect matching is then added to the graph H_t resulting in the graph H_{t+1}. Thus $H_{t+1} \stackrel{\text{def}}{=} G_t + M_t$, where the sum denotes edgewise addition of the weights. As M_t is unweighted, the weights of M_t are assumed to be one on each matching edge. The game terminates after $T \stackrel{\text{def}}{=} g(n)$ rounds. There are 2 possible winning criteria. First one, the *Expansion criterion* was introduced by KRV and it says that the Cut player $\mathcal{C}$ wins if $\alpha(G_{T+1})$ is at least $f(n).g(n)$. Otherwise the winner is the Matching player $\mathcal{M}$. The *Gap criterion* (or Spectral gap criterion) introduced in work of Orrechia, says that the Cut player $\mathcal{C}$ wins if $\lambda_{H_{T+1}} \geq f(n)$. Otherwise the Matching player $\mathcal{M}$ wins.

The following figure further summarizes framework of the Cut-matching Game, figure is based on the one from work of Orecchia [62].

Cut-Matching Game $(\mathcal{H}(n), f(n), g(n))$ with input graph $G = (V, E_G)$:

- $H_1 := (V, \emptyset, 0)$ be an empty weighted graph, where $|V| = n$, n is even.
- Fix the Cut player $\mathcal{C}$ and the Matching player $\mathcal{M}$.
- For $t = 1, \ldots, T = g(n)$,
  1. $\mathcal{C}$ chooses a bisection $(S_t, \bar{S}_t)$ of V.
  2. $\mathcal{M}$ chooses a perfect matching $M_t = (V, E(M_t))$ across $(S_t, \bar{S}_t)$.
  3. $G_{t+1} = G_t + M_t$
- Winning criteria:
  - Gap criterion: $\mathcal{C}$ wins if $\lambda_{H_{T+1}} \geq f(n)$. Otherwise $\mathcal{M}$ wins.
  - Expansion criterion: $\alpha(H_{T+1}) \geq f(n) \cdot g(n)$

Figure 2.4: Cut-matching game, with Gap criterion and Expansion criterion

The goal of the Cut player is to eventually build the expander graph H and prove the lower bound on expansion of G. The goal of the Matching player is to prove that such graph could not be built and to find a sparse cut that provides upper bound on the expansion. The Cut player gives a bisection $(S, \bar{S})$ as a suggestion to the Matching player. The Matching player either finds a cut with a low expansion that is somewhere "near" the the bisection $(S, \bar{S})$ or it fails to do so and outputs matching M that can be routed in G with low congestion. And thanks to the choice of bisection, the addition of matching M to the graph H increases expansion of H by a constant factor. We explore more about the strategies of the Matching player and the Cut player in next 2 sections. Consider the following lemma of KRV [50].

Lemma 8. *([50]). Consider an instance graph $G = (V, E_G)$ with $|V| = n$, $|E_G| = m$. Assume there exists a winning the Cut player strategy $\mathcal{C}$ for $(H(n), f(n), g(n))$ under the Expansion criterion and that this strategy runs in time $T(n)$ per round. Let $T_{flow} = \tilde{O}(m + n^{3/2})$. Then, there is an $O(\frac{1}{f(n)})$- approximation algorithm for the* Expansion *problem on G that runs in time $\tilde{O}(g(n) \cdot (T(n) + T_{flow}))$.*

The idea behind the proof is the one from the beginning of the chapter. Suppose we are trying to decide whether the graph G has the expansion larger then β. We gradually build graph H. It takes $g(n)$ rounds of the algorithm. In each round the Matching player performs a flow computation that attempts to find a cut of expansion less than β that is well-correlated to the bisection given by $\mathcal{C}$. The Matching player either finds such cut and game stops and outputs a cut of expansion less than β or the Matching player finds a perfect matching M between $(S_t, \bar{S}_t)$ that is routed with congestion $\frac{1}{\beta}$ in graph G. Matching M then serves as a certificate that no low-expansion cut exists "near" $(S_t, \bar{S}_t)$. If the Cut player wins, then we end up with expander H with expansion $\alpha(H) \geq f(n)g(n)$ by the Expansion criterion or by spectral gap $\lambda_G \geq f(n)$ in case of the Spectral gap criterion. The expander H can be routed in G with congestion $\frac{g(n)}{\beta}$. From 2.26

we see that

$$\alpha(G) \geq \frac{f(n)g(n)\beta}{g(n)} \geq f(n) \cdot \beta.$$

Hence, we can distinguish between $\alpha(G) < \beta$ and $\alpha(G) \geq \beta \cdot f(n)$ which implies $\frac{1}{f(n)}$-approximation algorithm for Expansion. Moreover, to solve this decision problem we only require $g(n)$ maxflow computations. Using binary search to guess the optimal value for the expansion increases the running time by a logarithmic factor, so the total running time is $\tilde{O}(g(n) \cdot (T(n) + T_{flow}))$.

From the above discussion can be clearly deduced that we are ideally looking for the Cut player strategies that achieve large $f(n)$ and small $g(n)$, as they yield better approximations in lower running time. KRV strategy and all other strategies presented in work of Orecchia [62] are randomized and provably successful with high probability. These strategies achieve $g(n) = O(Polylog(n))$ and $f(n) = \Omega(1/polylog(n))$. For us will be the most important the main result of KRV [50], which is the existence of the cut strategy yielding $f(n) = \Omega(1/\log^2 n)$ and $g(n) = O(\log^2 n)$ under the Expansion criterion. This leads to $O(\log^2 n)$-approximation for Expansion. We denote the strategy for the Cut player of this game C_{KRV}. Orecchia [62] in his work presents other 2 Cut player strategies denoted by C_{NAT} and C_{EXP}. Following table sumarizes results for different Cut player strategies:

Strategy	f(n)	g(n)	Running time	Approximation achieved
C_{KRV}	$\Omega(1/(\log^2 n))$	$O(\log^2 n)$	$\tilde{O}(n)$	$O(\log^2 n)$
C_{EXP}	$\Omega(1/(\log n))$	$O(\log^2 n)$	$\tilde{O}(n)$	$O(\log n)$
C_{NAT}	$\Omega(1/(\log n))$	$O(\log^2 n)$	$\tilde{O}(n)$	$O(\log n)$

Table 2.1: the Cut player strategies results according to Expansion criterion

The Matching player strategy is the same for all variants of the Cut-Matching game. One of the main results of Orrechia [62] regarding the Cut-Matching game is the first lower bound on the performance of any Cut player under the Expansion criterion. It implies that no approximation algorithm for Expansion designed following the Cut-Matching framework can achieve an approximation ration better than $\Omega(\sqrt{\log n})$. Curiously this is also the best approximation known for any polynomial-time algorithm for Expansion [10]. This result we formulate in the following theorem according to Orrechia [62]:

Theorem 10. *([62]) There is a Matching player $\mathcal{M}^*$ that is successful against any Cut player on the game $(\mathcal{G}(n), O(1/\sqrt{\log n}), g(n))$ for all $g(n)$ under the Expansion criterion.*

This lower bound is tight as there is a Cut player strategy, albeit inefficient one that is successful under the Expansion criterion. This strategy requires the use of multicommodity flows and is mentioned in various papers [71, 7, 8]. Thus this lower bound settles the question of the power of the Cut-Matching game.

2.4.2.1 The Matching player strategy

The Matching player in the Cut-Matching game framework takes the bisection $(S, \bar{S})$ of the vertex set V, which is the same for both graph G and graph H_t.

The Matching player uses single-commodity flow procedure to either find perfect matching between S and $\bar{S}$ or to find a sparse cut. In figure 2.5 we can see the procedure.

Input: A bisection $(S, \bar{S})$ of the vertex set of graph $G = (V, E_G)$ and expansion estimate α

1. Construct flow network as follows: Assign each edge in G a capacity of $1/\alpha$ (which can be assumed to be integral), add a source node with an outgoing unit-capacity arc to each vertex in S, and add a sink node with an incoming unit-capacity arc from each vertex in S.
2. Compute maximum flow between the source and the sink.
   (a) If the flow-value is $\frac{n}{2}$, then compute matching M on $(S, \bar{S})$. Matching is obtained by decomposing the flow into flow-paths in a standard manner (see [2]) and output a M.
   (b) If flow value is less then $\frac{n}{2}$, find a minimum cut $(C, \bar{C})$ separating the source and the sink in the flow network and output $(C \setminus \{source\}, \bar{C} \setminus \{sink\})$ and End.

Figure 2.5: The Matching player strategy

If a maximum flow of value $n/2$ is found, then the procedure finds a matching between S and $\bar{S}$ that can be embedded in G with the congestion $1/\beta$. If not, then the following lemma with proof from KRV [50] paper ensures that the procedure outputs a cut of expansion at most β.

Lemma 9. *([50]) (The Matching player lemma) If the maximum flow between the source and the sink has value less than $n/2$, then the procedure outputs a cut in G of expansion less than β.*

Proof. Recall the max-flow min-cut theorem for single commodity flows presented in the section 1.1.1. If the flow has value less than $n/2$, then the minimum cut separating the source and the sink has a capacity less than $n/2$. Let the number of edges in the cut incident to the source (resp. sink) be n_s (resp. n_t). The remaining capacity of the cut is less than $n/2 - n_s - n_t$ and thus uses at most $\beta \cdot (n/2 - n_s - n_t)$ edges in the original graph G. Furthermore, the cut consisting of the edges in the graph separates at least $n/2 - n_s$ vertices in S from $n/2 - n_t$ vertices in $\bar{S}$. The expansion of this cut is at most $\frac{\beta \cdot (n/2 - n_s - n_t)}{\min(n/2 - n_s, n/2 - n_t)}$ which is at most β. □

2.4.2.2 The Cut player strategies

In this section we give an explicit algorithmic formulation of the above mentioned cut strategies and describe C_{KRV} strategy in more detail. At rounds t, given graph H_t which is the union of perfect matchings $M_1, \ldots, M_{t-1}$ and a transition probability matrix P_t on H_t, the Cut player runs the procedure described in the figure 2.6 to output the bisection. The matrix P_t depends on used Cut player strategy. We focus mainly on random walk of C_{KRV} player. It is a sequential

Input:H_t as union of matchings $\{M_1, \ldots, M_t\}$

1. Choose r, a random unit n-dimensional vector orthogonal to $\vec{1}$.
2. Perform random walk on H_t, random walk is dependent on strategy chosen by the Cut player. Let P_t be the probability transition matrix on H_t
3. Let $y_t = P_t \cdot r_t$
4. Sort the entries of $y_t = (y_1, \ldots, y_n)$ as $y_{i_1} \leq \cdots \leq y_{i_{n/2}} \leq y_{i_{n/2+1}} \leq \cdots \leq y_{i_n}$
5. Let $S_t = \{i_1, \ldots, i_{n/2}\}$ and return $(S_t, \bar{S}_t)$

Figure 2.6: The Cut player algorithmic template

composition of the lazy random walks across each matching $M_1, \ldots, M_{t-1}$. This random walk proceeds as follows: in step i the particle stays put with probability $1/2$ and traverses the incident matched edge in M_i with probability $1/2$. More formally, for $t \in \{1, \ldots, T\}$ given H_t we define:

$$P_{t+1}^{KRV} = \left(\frac{I + A(M_t)}{2}\right) \cdot \cdots \cdot \left(\frac{I + A(M_1)}{2}\right) = \prod_{i=t}^{1} \left(\frac{I + A(M_i)}{2}\right).$$

The goal of the random walk is to find a cut $(S, \bar{S})$ in H_t with a low number of edges and thus with a low expansion. Then there si a guarantee that any matching that could the Matching player add to H_t improve its expansion. To find such a cut we use the concept described in the section 2.3.6.1. We generate distribution of "charge" by generating vector r. It is orthogonal to $\vec{1}$, thus the sum of its coordinates is 0, generating equal amount of "positive" and "negative charge". As we described in the section 2.3.6.1, applicating the random walk on the graph with charges, charges rapidly achieve their average within each "well-connected" component. These average weights are typically non-zero (either positive or negative) because of the random initial assignments. Now selecting $n/2$ vertices with the lowest "charges" should help separate some of these "well-connected" components from the rest (say the negative from the positive).

Our goal in C_{KRV} strategy is to either build 1/2- expander or find a sparse cut. How do we know that H_t has already expansion 1/2? We measure the progress of the algorithm by so called *potential function*. It provides information, how well the random walk on H_t *mixes*. It is a measure of how far from uniform the resulting distribution of the associated random walk is when starting from a random vertex. We call H_t *mixing* if for any starting position of the particle the probability that the particle reaches an vertex v is at least $1/2n$. This definition implies that the H_t forms a graph with edge expansion at least 1/2. Formally

$$\phi(t) = \sum_{i,j \in \{1,\ldots,|V|\}} (P_{i,j}^t - 1/n)^2 = \sum_{i \in V} ||P^t e_i - \frac{1}{n}\vec{1}||_2^2 = ||P^t - \pi||_F^2,$$

where e_i is vector of all-zeros except 1 at i-th coordinate. Last equality leads t

the form of Frobenius norm. Frobenius norm is defined on matrices as:

$$A \in \mathcal{R}^{m \times n}, ||A||_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}.$$

Observe that for the empty graph $\phi(0) = n - 1$. If $\phi(t) < \frac{1}{4n^2}$, then the graph H_t is mixing. The KRV algorithm starts with the empty graph H_0 and while the graph is not mixing, it tries to find a new matching to add to the graph, which is embeddable in G with congestion $1/\beta$ and which reduces potential by a factor of $(1 - \Omega(1/\log n))$. If it succeeds in doing this, $t = O(\log^2 n)$ iterations suffice to produce a mixing graph H_t with expansion $\alpha(H_t) = 1/2$ and embedded in G with a congestion $O(\log^2 n/\beta)$. The proof that the potential function is sufficiently reduced in every iteration of algorithm can be found in KRV paper [50]. Detailed description of other Cut-player strategies with detailed analysis can be found in work of Orecchia [62].

3. Multilevel graph partitioning

In this section we briefly introduce methods of solving graph partitioning problems, that includes partitioning of graph to more than 2 parts. In this section we follow the presentation of Sanders and Schulz [69] and continue with section about Kernighan-Lin algorithm from Demmel [38]. Formal definitions we gave in the section 1.1.2, namely in definition 4. A successful heuristic for partitioning large graphs is the *multilevel graph partitioning* (MGP) approach, where the graph is recursively *contracted* to achieve smaller graphs which should reflect the same basic structure as the input graph. To the smallest, and the most coarsened graph an *initial partitioning* algorithm is applied. The contraction is undone and at each level a *local refinement* method is used to improve the partitioning by the coarser level.

First we introduce some basic preliminaries. In a graph $G = (V, E)$ a *matching* $M \subseteq E$ is set of edges that do not share any common nodes, i.e., the graph (V, M) has maximum degree one. By *contracting* an edge $\{u, v\}$ we mean replacing the nodes u and v by a new node z connected to the former neighbors of u and v. For vertex weighted case we can set $c(z) = c(u) + c(v)$ so the weight of a node at each level is the sum of weights it is representing in the original graph. For unweighted case it can still be useful to provide mere number of such vertices. If replacing edges of the form $\{u, v\}, \{v, w\}$ would generate two parallel edges $\{z, w\}$, we insert a single edge with the weight being the sum of the two, $w(\{z, w\}) = w(\{u, w\}) + w(\{v, w\})$. To undo the contraction of an edge, we *uncontract* it.

As we have already mentioned the multilevel approach to graph partitioning consists of three main phases.

1. The contraction (coarsening) phase.

2. The initial partitioning.

3. The uncoarsening with local refinement phase.

In the contraction phase algorithm iteratively identifies matchings $M \subseteq E$ and contract edges in M. This way the size of the original graph should be quickly reduced while maintaining the global structure of the input. Different types of heuristics can be used to choose good matching M. In the initial partitioning phase conventional algorithm for partitioning is chosen.

In the uncoarsening phase the matchings are iteratively uncontracted. To improve the quality of a cut, after uncontracting matching, the refinement algorithm moves nodes between blocks. One of this refinement methods is Kernighan-Lin algorithm.

3.0.3 Kernighan-Lin

In this section we briefly describe the Kernighan-Lin algorithm following the web resource [33]. It is used in MGP approach implemented in software packages METIS, Chaco and others. It is used to the uncoarsening with local refinement phase. The main idea is to find 2 nodes between initial partitions and swap them.

The main problem is to identify which pair of nodes improves the partition. First we define a few terms. Consider partition (P_1, P_2) of a graph $G(V = P_1 \cup P_2, E)$. For each vertex $u \in V$ lets define the *external cost* E_u and the *internal cost* I_u of u as follows. If $u \in P_1$ then $I_u = \sum_{v \in P_1, v \sim u} w(u, v)$, that is the sum of the weights of edges between u and other nodes in P_1. The external cost of u is the sum of the costs of edges between u and the nodes in P_2. Furthermore, let

$$D_u = E_u - I_u$$

be the difference between the external and internal costs of u. Let T be the weights between P_1 and P_2, we want to minimize. If the vertices $u \in P_1$ and $w \in P_2$ are swapped, then the reduction in cost is

$$T_{old} - T_{new} = D_u + D_w - 2w(u, w),$$

where $w(u, w)$ is the weight of the edge between u and w. If there is no edge, then $w(u, w) = 0$. The algorithm is trying to find and execute optimal series of swapping, which maximizes $T_{old} - T_{new}$. The output is the partition of the graph to $\tilde{P_1}$ and $\tilde{P_2}$. The general scheme for K-L follows:

Algorithm 4 Kernighan-Lin

function KERNIGHAN-LIN((G = (V, E), Initial partition V=$P_1 \cup P_2$)
  $A_1 := P_1, B_1 = P_2$
  For all values $a \in A_1$ and $b \in B_1$ determine D_a and D_b respectively.
  repeat
    for $i := 1$ to $\frac{|V|}{2}$ **do**
      $g_i = \max_{a_i \in A_1, b_i \in B_1} D_{a_i} + D_{b_i} - 2 \cdot w(a_i, b_i)$
      swap(a_i, b_i) between A_1 and B_1
      remove a_i and b_i from further consideration in this pass
      update D values for the elements of $A_1 = A_1 \setminus \{a_1\}$ and $B_1 = B_1 \setminus \{b_i\}$
    end for
    Find k which maximizes $g_{\max} = \sum_{i=1}^{k}$
    if $g_{\max} > 0$ **then**
      Exchange $a_1, \ldots, a_k$ in P_1 with $b_1, \ldots, b_k$ in P_2
      $\tilde{P_1} = P_1, \tilde{P_2} = P_2$
    end if
  until $g_{\max} \leq 0$
  return $(\tilde{P_1}, \tilde{P_2})$
end function

3.1 Additional materials and software

The area of graph partitioning is of great interest and huge amount of research has been done on the subject. The reader can explore the works [24, 74] for more material on MGP. In year 2011, started prestigious DIMACS Implementation Challange on the subject graph partitioning and graph clustering [37]. It had ambition to test state of the art implementations of graph partitioning algorithm on real world graph data. The main publication result is a book Graph

partitioning and Graph clustering [31]. The most general purpose methods, that are able to obtain good partitions for large real world graphs are based on the multilevel principle outlined in this section. Well known software packages based on this approach include, Jostle [74], METIS [48, 73] and Scotch [67]. Another graph partitioner, based on the central idea to (un)contract only a single edge between two levels is KaSPar [64]. We can also mention packages KaPPa [32] and DiBaP [60] with different flavors of multilevel partitioning algorithm.

4. Implementations of algorithms

In this section we presentl implementations of Leighton-Rao algorithm using the geometric embeddings described in the section 2.2.2 and the Cut-Matching game with the KRV Cutting-Player strategy described in the section 2.4.2. We used external libraries for finding single-commodity flows [15] in the Cut-Matching game and for solving linear program (The GNU Linear Programming Kit (GLPK)) in Leighton-Rao algorithm.

Although we knew it has high time complexity in theory, we also tried to implement SDP algorithm from ARV paper [10]. However, the number of constraints in SDP was too much and we stopped the implementation after realizing, that on the current hardware we can solve SDP relaxation for graphs with maximum of 20-30 vertices. This algorithm is important for theory, but is not usable in practice, at least not in this form. More practical SDP based approach is given in the work of Arora, Hazan and Kale [7] and Orecchia [62].

Solving linear program relaxation in Leighton-Rao algorithm is the most time consuming part of the algorithm. Solving linear program with $O(n^3)$ variables and constraints is too much for graphs with more than 100 vertices. On the other hand, our implementation of Cut-Matching game is usable on graphs up to thousends vertices. One limitation that our implemantation of Cut-Matching game has, is the fact that algorithm can not find cuts in graphs with expansion greater than 1. It follows from the fact, that in the decision problem version of Cut-Matching game we expect the inversion $1/\beta$ of input expansion β to be integral. Typically this is not a problem, because graphs usually have much lower expansion, but it is constraint for denser graphs. Scaling the algorithm for the higher expansions and solving the demand of integrality can be subject of the future work. Pseudoalgorithms for Cut-Matching game are in Appendix 1.

Solving single-commodity flow problem proved to be much less time-consuming. Of course, we must note that although we tried our best to implement these algorithms efficiently, there is plenty of room for optimization. We closely followed the theory, while there are known techniques that could improve the running times. We consider these implementations as a proof-of-concept prototypes. Further research and optimization could lead to more practical implementations. We tested the algorithms on manually generated graphs. We generated them in such way, that for most of them we know what is the optimum expansion. First set of graphs consist of grid networks with different number of nodes.

	Cut-Match 30	Cut-Match 200	Leighton-Rao	Opt
Grid 5×10	0.25	0.2	0.66667	0.2
Grid 10×10	0.302326	0.2	1.06	0.2
Grid 20×10	0.142857	0.196721	NA	0.1
Grid 20×25	0.083333	0.083333	NA	0.083333
Grid 40×25	0.0725	0.0725	NA	0.05
Grid 40×50	0.04	NA	NA	0.04

Table 4.1: The expansions found in grid graphs.

	Cut-Match 30	Cut-Match 200	Leighton-Rao
Grid 5×10	130	130	792350
Grid 10×10	600	6140	3921578
Grid 20×10	1550	23320	NA
Grid 20×25	65180	208500	NA
Grid 40×25	313040	747100	NA
Grid 40×50	2619430	NA	NA

Table 4.2: The time results for grid graphs in ms.

We can see, that algorithm Cut-Matching game was pretty successful in finding sparse cuts, while Leighton-Rao algorithm was not that efficient. We tested 2 versions of Cut-Matching algorithm with parameter *Iterations* being 30 first time and 200 second time. As was described in the section 2.4.2, the Cut player with KRV strategy wins, when the graph H_T has expansion at least 1/2. In our implementation we give the Matching player more chances to find a sparse cut. The parameter expresses how many iterations Cut-Matching game tries to do after it build 1/2-expander graph H_T.

Leighton-Rao algorithm has 2 parameters. One specifies time which is used to solve LP relaxation. In our test we set it always to 1200 seconds. If algorithm does not find optimal solution in this time, it provides best feasible solution it found. The second parameter is the number of trials to do geometric embeddings. The solution of LP relaxation is used to generate random embedding as was described in section 2.2.2 and which eventually produces desired cut. Leighton-Rao algorithm outputs best cut from all trials. We set this parameter to 100 in all tests.

The next class of graphs consist of so called planted bisections. Graph with planted bisection consists of two random, relatively dense graphs connected by a small number of randomly picked edges . We describe the planted bisection graph by 3 numbers. For example the graph PB $8 \times 8 \times 3$ has 2 components each with 8 nodes and it has exactly 3 edges connecting these two parts.

	Cut-Match 30	Cut-Match 200	Leighton-Rao	Opt
PB $8 \times 8 \times 3$	0.375	0.375	0.375	0.375
PB $25 \times 25 \times 10$	0.4	0.4	2.70833	0.4
PB $50 \times 50 \times 5$	0.1	0.1	0.1	0.1
PB $100 \times 100 \times 10$	0.1	0.1	NA	0.1
PB $500 \times 500 \times 40$	0.08	0.08	NA	0.08

Table 4.3: Expansion found in plant bisection graphs.

We can see that Cut-Match KRV algorithm found bisection in all cases in relatively short time, whlie LR algorithm found only the first and third bisection and it took some time. Now we present the performance of both algorithms for paths and cycles.

	Cut-Match 30	Cut-Match 200	Leighton-Rao
PB $8 \times 8 \times 3$	< 1	460	15440
PB $25 \times 25 \times 10$	100	2210	381680
PB $50 \times 50 \times 5$	930	9830	3759441
PB $100 \times 100 \times 10$	420	35290	NA
PB $500 \times 500 \times 40$	70110	805330	NA

Table 4.4: Time results for plant bisection graphs in ms.

	Cut-Match 30	Cut-Match 200	Leighton-Rao	Opt
Cycle 10	0.4	0.4	0.4	0.4
Cycle 50	0.08	0.08	0.285714	0.08
Cycle 100	0.0434783	0.04	0.181818	0.04
Cycle 200	0.037037	0.02	NA	0.02
Cycle 500	0.008	0.008	NA	0.008
Cycle 1000	0.004	0.004	NA	0.004
Cycle 2000	0.002	0.002	NA	0.002

Table 4.5: Expansion found in the cycles.

	Cut-Match 30	Cut-Match 200	Leighton-Rao
Cycle 10	10	230	3040
Cycle 50	270	2900	1474853
Cycle 100	1040	11050	3251559
Cycle 200	5230	39300	NA
Cycle 500	129350	296030	NA
Cycle 1000	630100	1358849	NA
Cycle 2000	4275735	8998491	NA

Table 4.6: Time results for cycles in ms.

	Cut-Match 30	Cut-Match 200	Leighton-Rao	Opt
Path 10	0.2	0.2	0.2	0.2
Path 50	0.05	0.04	0.25	0.04
Path 100	0.0212766	0.02	0.142857	0.02
Path 200	0.0106383	0.01	NA	0.01
Path 500	0.004	0.004	NA	0.004
Path 1000	0.002	0.002	NA	0.002

Table 4.7: Expansion found in the paths.

	Cut-Match 30	Cut-Match 200	Leighton-Rao
Path 10	10	220	3030
Path 50	390	3480	1476136
Path 100	1040	10600	3258634
Path 200	10940	436900	NA
Path 500	136370	320680	NA
Path 1000	776940	1783203	NA

Table 4.8: Time results for paths in ms.

Results for cycles and paths confirm the trend. The Cut-Match KRV algorithm is relatively faster and finds lower expansion cuts than the Leighton-Rao algorithm. We must consider also the fact, that the Leighton-Rao algorithm depends on the solution of LP relaxation for which we can obtain only approximate solution. In theory Leighton-Rao embedding algorithm also has high constant in the approximation factor. On the other hand the Cut-Match KRV algorithm performs suprisingly well. We find most amusing, that it found planted bisection in PB graphs.

Our conclusion from the above performance results is that while Cut-Matching game performs well in prototype implementation the LR embedding algorithm does not seem scalable. The solving of LP relaxation is rather cumbersome and can not be radically optimized or avoided. The Cut-Matching game seems to be elegant and exciting framework which can be further developed into algorithms for the practical use.

Conclusion

We have seen how rich is the area of graph partitioning problems. We introduced various flavours of the problems and presented the known results in the area. We focused our attention on three closely related qualities of the graph partitioning: the sparsity, the expansion and the conductance. We have seen the various ideas and ways how to approach the graph partitioning problems. First we started by the duality of multi-commodity flow problem and sparsest cut problem due to work of Leighton and Rao [53]. We gave the proof of this theorem by presenting algorithm of geometric embeddings from the presentation of Shmoys [72]. We also implemented this algorithm and presented its performance. As we have explained in the previous section, solving the LP relaxation used in the algorithm seems to be a problem to the algorithm being practical. Moreover, the constant in the approximation factor appeared to be high in our implementation.

After presenting Leighton-Rao embedding algorithm in the section 2.2.2 we moved to the work of Arora, Rao Vazirani (ARV [10]). They were first who introduced $O(\sqrt{\log n})$-approximation to the most of graph partitioning problems. We presented their algorithm based on SDP relaxation and geometric embedding. Although we knew it has high time complexity in theory, we tried to implement it. But after implementing the SDP relaxation part, we realized it can not successfully compute sparse cuts for graphs with more than 20-30 nodes. The SDP algorithm presented in ARV paper has its place in theory, but it was not meant to run on the current real hardware. Further exploration of SDP approach, which could be more practical was given in the work of Arora, Hazan and Kale [7]. They used techniques from the game theory and solving of zero-sum games. The further exploration of SDP approach was also given in the work of Orecchia [62].

In the section 2.3 we introduced ideas and concepts of the spectral graph theory. We have seen various matrices related to graphs and how their spectrum relates to the expansion qualities of graph. We explained the relations between the conductance, the spectral gap and the random walks. We formulated basic spectral algorithm for graph partitioning and gave some intuition behind it.

The spectral approach concepts and expander flows are connected in the framework of the Cut-Matching game we presented in the section 2.4.2. We also implemented one version of this algorithm and tested on our data. For some sets of graphs it performed suprisingly well. On all data sets it performed better than Leighton-Rao embedding algorithm in both running time and expansion approximation. As expected, solving of single-commodity flow problem proved to be faster than solving LP relaxation and the algorithm is promising candidate for more practical implementations.

We also took a notion of Multilevel graph partitioning in section 3, which has its use in the most algorithms partitioning graph to more than 2 parts. The concept is used in widely used software packages like METIS [48], KaPPa [32], Jostle [74] and others.

We hope that we have provided a compact overview of graph partitioning problems and have comprehensibly presented the main concepts and ideas behind various approaches.

Bibliography

[1] Afarwal, A., Charikar, M., Makarychev, K., Makarychev, Y. $O(\sqrt{\log n})$ approximation algorithms for Min UnCut, Min 2CNF Deletion, and directed cut problems. Proceedings of the 37th Annual ACM Symposium on Theory of Cumputing, 573-581, 2005.

[2] Ahuja, R., Magnati, T., Orlin, J. Network flows: Theory, algorithms, and applications. Prentice Hall Eaglewood Cliffs, NJ, 1993.

[3] An improved approximation algorithm for multiway cut. Proceedings of the 30th Annual ACM Symposium on Theory of Computing. ACM-SIAM, 48-52, 1998

[4] Andersson, G. An approximation algorithm for MAX p-SECTION. Proceedings of the 16th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science 1563. Springer-Verlag. 1999

[5] Andreev, K., Räcke, H. Balanced graph partitioning. Proceedings of the 16th annual ACM symposium on Parallelism in algorithms and architectures. 120-124, 2004

[6] Arora S., Lee J. R., Naor, A. Euclidean distortion and the sparsest cut., Proceedings of the 37th Annual ACM Symposium on Theory of Computing. 553-562, 2005.

[7] Arora, S., Hazan, E., Kale, S. $O(\sqrt{\log n})$-approximation to sparsest cut in $\tilde{O}(n^2)$ time. Proceedings of the IEEE Symposium on Foundations of Computer Science. 238-247, 1985.

[8] Arora, S., Kale S., A combinatorial, primal-dual approach to semidefinite programs. Proceedings of the 39th Annual ACM Symposium on Theory of Computing. 227-236, 2007.

[9] Arora, S., Rao, S., Vazirani, U. Expander flows, geometric embeddings and graph partitioning. Journal of the ACM. 5:1-5:37, 2009.

[10] Arora, S., Rao, S., Vazirani, U. Expander flows, geometric embeddings, and graph partitionings. Proceedings of the 36th Annual ACM Symposium on Theory of Computing.

222-231, 2004.

[11] Ausiello G., Crescenzi P., Gambosi G.,Kann V., Marchetti-Spaccamela A.,Protasi M. Complexity and Approximation. Springer Verlag. 1999.

[12] Awerbuch, B., Peleg, D. Sparse partitions (extended abstract). 503-513, 1990.

[13] Blum, M., Karp, R., Vornberger, O., Papadimitriou, C., Yannakakis, M. The complexity of testing whether a graph is a superconcentrator. Information Processing Letters. 13:164-167, 1981.

[14] Bourgain, J. On lipschitz embeddings of finite metric spaces in hilbert space. 52(1-2) 46-52, 1985.

[15] Boykov, Y., Kolmogorov, V. , An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI). 2004.

[16] Breiman, L. Probability. Published by Addison-Wesley. 1968.

[17] Bui, T. N., Jones, C. A heuristic for reducing fill-in in sparse matrix factorization. The 6th SIAM Conference on Parallel Processing for Scientific Computing. 445-452, 1992.

[18] Dahlhaus, E., Johnson, D. S., Papadimitriou, C. H., Seymour, P. D., Yannakakis, M. The complexity of multiterminal cuts. SIAM Journal on computing. 864-894, 1994.

[19] Dunlop, A., Kernighan, B. A procedure for placement of standard-cell VLSI circuits. IEEE Transactions on Computer-Aided Design. CAD-4(1):92-98, 1985.

[20] Elias, P.,Feinstein, A., Shannon, C. E. A note on the maximum flow through a network. IRE. Transactions on Information Theory. 2, 4, 117-119, 1956.

[21] Feige, U., Goemans, M. X. Approximating the value of two prover proof systems, with applications to MAX 2SAT and MAX DICUT. Proceedings of the 3rd Israel Symposium on Theory of Computing and Systems. IEEE Computer Society, 182-189, 1995.

[22] Feige, U., Krauthgamer, R. A Polylogarithmic Approximation of the Minimum Bisection. Society for Industrial and Applied Mathematics. 99-130, 2006.

[23] Fiedler, M. Algebraic connectivity of graphs. Czechoslovak Mathematical Journal 23(98):298-305, 1973.

[24] Fjallstrom, P.O. Algorithms for graph partitioning: A survey. Linkoping Electrinic Articles in Computer and Information Science 3(10), 1998.

[25] Ford, L. R., Fulkerson, D. R. Maximal flow through a network. Canadian Journal of Mathematics. 8:399-400, 1956.

[26] Frieze, A., Jerrum, M. Improved approximation algorithms for MAX k-CUT and MAX BISECTION. Algorithmica 18. 67-81, 1997.

[27] Garg, N., Vazirani, V., Yannakakis, M. Multiway cuts in directed and node weighted graphs. Proceedings of the 21st International Colloquium on Automata, Languages and Programming. Lecture Notes in Computer Science 820. Springer-Verlag. 487-498, 1994.

[28] Garg, N., Vazirani, V., Yannakakis, M. Primal-dual approximation algorithms for integral flow and multicut in trees. Algorithmica 18. 3-20, 1997.

[29] Goemans, M. X., Williamson, D. P. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. Journal of ACM. 42, 1115-1145, 1995.

[30] Goldschmidt, O., Hochbaum, D. S. Polynomial algorithm for the k-cut problem. Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science. IEEE Computer Society, 444-451, 1988.

[31] Graph Partitioning and Graph Clustering. ISBNs: 978-0-8218-9038-7 (print); 978-0-8218-9869-7 (online) Volume 588 2013, DOI: http://dx.doi.org/10.1090/conm/588

[32] Holtgrewe, M., Sanders, P., Schulz, C. Engineering a Scalable High Quality Graph Partitioner. 24th IEEE International Parallel and Distributed Processing Symposium. 2010.

[33] http://en.wikipedia.org/wiki/Kernighan%E2%80%93Lin_algorithm

[34] http://en.wikipedia.org/wiki/Max-flow_min-cut_theorem

[35] http://en.wikipedia.org/wiki/Multi-commodity_flow_problem

[36] http://lucatrevisan.wordpress.com/2011/03/02/cs261-multicommodity-flow/

[37] http://www.cc.gatech.edu/dimacs10/

[38] http://www.cs.berkeley.edu/ demmel/cs267/lecture20/lecture20.html

[39] http://www.cs.princeton.edu/courses/archive/spr05/cos598B/lecture1.pdf

[40] http://www.cs.sandia.gov/ bahendr/partitioning.html

[41] Chung, F. Spectral Graph Theory. AMS Publications. 1997.

[42] Chung, F. Four proofs for the Cheeger inequality and graph partition algorithms. CCM 2007.

Vol. I I, 1-4, 2007.

[43] Jerrum, M., Sinclair A., Approximating the permanent. SIAM Journal on computing, 18(6):1149-1178, 1989.

[44] Jianbo, S., Jitendra, M. Normalized cuts and image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence. 22:888-905, 2000.

[45] Jukna S. Extremal Combinatorics: With Applications in Computer Science.

[46] Kabelíková P. Graph Partitioning Using Spectral Methods, 2006, VŠB - Technical University of Ostrava Faculty of Electrical Engineering and Computer Science Department of Computer Science.

[47] Kannan, R., Vempala, S., Vetta, A. On clusterings - good, bad and spectral. Proceedings of the 41th Annual IEEE Symposium on Foundations of Computer Science. 367, 2000.

[48] Karypis, G., Kumar, V. MeTis:Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0. http://www.cs.umn.edu/ metis, 2009, University of Minnesota, Minneapolis, MN

[49] Khandekar, R., Khot, S. A., Orecchia, L., Vishnoi, N. K. On a Cut-Matching Game for the Sparsest Cut Problem. EECS Department. University of California. Berkeley 2007.

[50] Khandekar, R., Rao, S., Vazirani, U. Graph partitioning using single commodity flows. STOC 2006:Proceedings of the 38th Annual ACM symposium on Theory of Computing. 385-390, 2006.

[51] Klein, P., Plotkin, S. A., Rao, S. Excluded minors, network decomposition, and multicommodity flow. Proceedings of the 25th Annual ACM Symposium on Theory of Computing. 682-690, 1993.

[52] Kumar, V., Grama, A., Gupta, A., Karypis, G. Introduction to Parallel Computing. The Benjamin/Cumming Publishing Company, Inc. 1994.

[53] Leighton, T., Rao, S. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. Proceedings of the 29th Annual Symposium on Foundations of Computer Science. 422-431. 1988.

[54] Leighton, T., Rao, S. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. Journal of the ACM. Volume 46 Issue 6, 787-832, 1999.

[55] Leskovec, J., Lang, K. J., Dasgupta, A., Mahoney, M. W. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. CoRR, abs/0810.1355, 2008.

[56] Linial, N., London, E., Rabinovich, Y., The geometry of graphs and some of its algorithmic applications. Combinatorica. 15:577-591., 1995.

[57] Lovász L. Eigenvalues of graphs. 2007. Available from:http://www.cs.elte.hu/ lovasz/eigenvals-x.pdf

[58] Mahajan, S., Ramesh, J. Derandomizing semidefinite programming based approximation algorithms. Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science. IEEE Computer Society. 162-169, 1995.

[59] Margulis, G.A. Explicit constructions of concentrators. Problems of Information Transmission 9. 325-332, 1973.

[60] Meyerhenke, H., Monien, B., Sauerwald, T. A new diffusion-based multilevel algorithm for computing graph partitions of very high quality. IEEE International Symposium on Parallel and Distributed Processing. 1-13, 2008.

[61] Motwani, R., Raghavan, P. Randomized algorithms. 1995, Cambridge University, Cambridge.

[62] Orecchia, L. Fast Approximation Algorithms for Graph Partitioning using Spectral and Semidefinite-Programming Techniques. 2011. Dissertation thesis. EECS Department, University of California, Berkeley.

[63] Orecchia, L., Schulman, L., Vaziriani, U., Vishnoi, N. On partitioning graphs via single commodity flows. Proceedings of the 40th Annual ACM Symposium on Theory of Computing. 461-470, 2008.

[64] Osipov, V., Sanders, P. n-Level Graph Partitioning. 18th European Symposium on Algorithms. 2010.

[65] Papadimitriou, C. H., Yannakakis M. Optimization, approximation, and complexity classes. Journal of computer and system sciences. 425-440, 1991.

[66] Parlett, B.N., Simon, H., Stringer, L.M. On Estimating the Largest Eigenvalue With the Lanczos Algorithm. Mathematics of computation,. 38(157):13-165, 1982.

[67] Pellegrini, F.:Scotch home page, http://www.labri.fr/pelegrin/scotch

[68] Sandeep, N. Bhatt, Leighton, F. T. A framework for solving VLSI graph layout problems. Journal of Computer and System Sciences, 28(2):300-343, 1984.

[69] Sanders, P.,Schulz, C. Engineering Multilevel Graph Partitioning Algorithms. Karlsruhe Institute of Technology (KIT)

[70] Saran, H., Vazirani, V. Finding k-cuts within twice the optimal. Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science. IEEE Computer Society, 743-751, 1991.

[71] Sherman, J. Breaking the multicommodity flow barrier for $O(\sqrt{\log n})$-approximations to sparsest cut. Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science, 2009.

[72] Shmoys, D. B. Cut problems and their application to divide-and-conquer. In D. S. Hochbaum, Approximation Algorithms for NP-hard Problems. 192-235, 1997

[73] Schloegel, K., Karypis, G., Kumar, V. Graph Partitioning Algorithms. Technical report, Karlsruhe Institute of Technology. 2010.

[74] Walshaw, C., Cross, M. Jostle: Parallel Multilevel Graph-Partitioning Software - An Overview. Magoules, F. (ed.) Mesh Partitioning Techniques and Domain Decomposition Techniques. 27-58, 2007.

[75] Ye, Y. A .699 approximation algorithm for Max-Bisection, MATHEMATICAL PROGRAMMING, to appear. http://dollar.biz.uiowa.edu/col/ye/

List of Tables

Appendix 1 - Pseudo-codes for the Cut-Matching Game

Algorithm 5 Cut strategy template

function CUTSTRATEGY(G_t) ▷ G_t is union of perfect matchings $M_1, \ldots, M_{t-1}$
  $r_t = Random(\{x | x \in \mathcal{R}^n, ||x||_2 = 1, < x, \vec{\mathbf{1}} >= 0\})$
  $P_t = RandomWalk_C(G_t)$
    ▷ P_t is probability transition matrix on G_t dependent on used cut-strategy C
  $y_t = P_t.r_t$
  Sort the entries of $y_t = (y_1, \ldots, y_n)$ as $y_{i_1} \leq \cdots \leq y_{i_{n/2}} \leq y_{i_{n/2+1}} \leq \cdots \leq y_{i_n}$
  $S_t = \{i_1, \ldots, i_{n/2}\}$
  $\overline{S_t} = V \setminus S_t$
  return $(S_t, \overline{S_t})$
end function

Algorithm 6 Matching Player strategy

function CUTORFLOW($G, (S_t, \overline{S_t}), \alpha$)
  (G_α, s, t) =ConstructFlowNetwork($G, (S_t, \overline{S_t}), \alpha$)
  $(G_{flow}, FlowValue)$ =SingleCommodityMaxFlow(G_α, s, t)
  if $FlowValue = n/2$ **then**
    Matching M_t =FlowToMatching(G_{flow}) **return** M_t
  else
    MinCut=FlowToMinCut(G_{flow}) **return** MinCut
  end if
end function

Algorithm 7 Cut-Matching game for EXPANSION decision problem, $\alpha(G) < \alpha$?

function CutMatchGame($G = |V, E|, \alpha$)
  $n = |V|$
  f=f(n)
  g=g(n) ▷ functions g and f depends on Cutting player strategy
  $V = \{1, \ldots, n\}$
  $G_1 = (V, \emptyset, 0)$ ▷ G_1 is empty weighted graph on n vertices, n is even
  for $t = 1 \to g$ **do**
    $(S_t, \overline{S_t})$ =CutStrategy(G_t)
    $Result = CutOrFlow((S_t, \overline{S_t}), \alpha)$
    if TypeOf($Result$) is Cut **then**
      $SparseCut = Result$
   return (YES, SparseCut) ▷ Matching player wins, $\alpha(G) < \alpha$,
    else
      $M_t = Result$
      $G_{t+1} = G_t + M_t$
    end if
  end for
   return (NO,G_{T+1}) ▷ Cutting player wins $\alpha(G) \geq \alpha.f$
end function

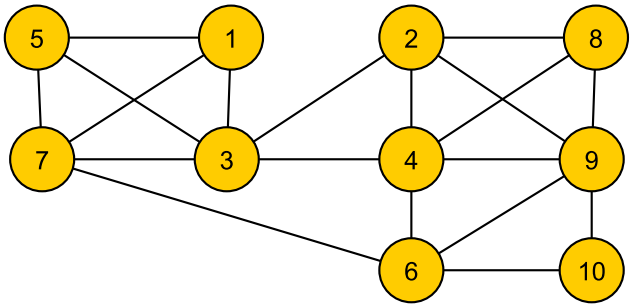
Algorithm 8 Cut-Matching EXPANSION optimalization algorithm

function ExpansionCutMatch(G)(G, a, b)
  $\alpha = \frac{a+b}{2}$
  (IsExpLower, certificate)=CutMatchGame(G, α)
  if IsExpLower = true **then**
    $b = \alpha$
  else
    $a = \alpha$
  end if
  if $b - a \leq f(n)$ **then**
    if certificate is not Sparse cut **then**
      (IsExpLower, certificate)=CutMatchGame(G, b)
    end if
   SparseCut=certificate
   return (SparseCut)
  else
   return ExpansionCutMatch(G,a,b)
  end if
end function

Appendix 2 - User Manual for program Partition

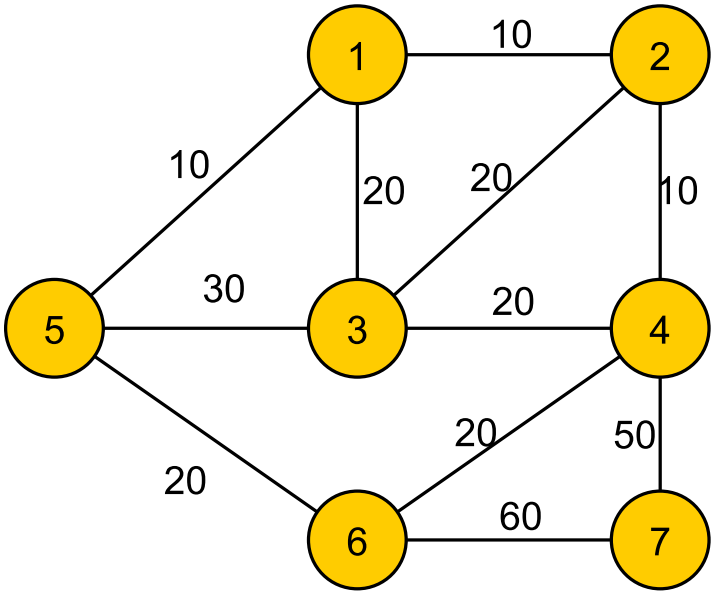
Program partition was compiled for Linux and Unix-like systems. It is used from command line interface. Its only paramater is graph input file. The format of the graph file is simple. In unweighted case on the first line there is the number of nodes followed by number of edges. Then each $i + 1$-th line represents list of neighbors of vertex i. The example of representation of unweighted graph is in figure 4.1. Format of input is simplification of the one that use METIS [48, 73], Jostle [74] and other partitioning programs. The format is well-known and its variant was also used on the DIMACS 10th implementation challange [37].

In weighted case on the first line there is again the number of nodes followed by number of edges and then followed by number 1. List of neighbors is the same, but each neighbor is followed by the weight of the edge that leads to him. The example of weighted graph is in figure 4.2.




```
10  19
3  5  7
3  4  8  9
1  2  4  5  7
2  3  6  8  9
1  3  7
4  7  9  10
1  3  5  6
2  4  9
2  4  6  8  10
6  9
```

Figure 4.1: Unweighted graph example.




```
7  11  1
5  10  3  20  2  10
1  10  3  20  4  10
5  30  4  20  2  20  1  20
2  10  3  20  6  20  7  50
1  10  3  30  6  20
5  20  4  20  7  60
6  60  4  50
```

Figure 4.2: Weighted graph example.