Coping with NP-hardness: Approximating minimum bisection and heuristics for maximum clique

Thesis for the Ph.D. Degree

by

Robert Krauthgamer

Supervisor: Prof. Uriel Feige
Department of Computer Science and Applied Mathematics
The Weizmann Institute of Science

Submitted to: Feinberg Graduate School
Weizmann Institute of Science
Rehovot 76100, Israel

April 16, 2001

(Revised December 2001)

Abstract

Many important optimization problems are known to be NP-hard. That is, unless P = NP, there is no polynomial time algorithm that optimally solves these problems on every input instance. We study algorithmic ways for "coping" with NP-hard optimization problems.

One possible approach for coping with the NP-hardness is to relax the requirement for *exact solution*, and devise *approximation algorithms*, i.e. efficient algorithms that produce a solution that is guaranteed to be nearly optimal. In the last decade, our understanding of many NP-hard optimization problems was greatly improved, both from the direction of approximation algorithms and from the direction of hardness of approximation. However, there is still a large gap in our understanding of the approximability of several fundamental problems.

A notable example is the *minimum bisection* problem, that requires to find in a graph a minimum-cost cut that partitions the vertices into two equal-size sets. This problem has applications both in theory and in practice. The seminal work of Leighton and Rao (1988) was largely motivated by this problem, and led to algorithms with approximation ratio $O(\log n)$ for several related problems. However, prior to our work no sublinear (in n) approximation ratio was known for this problem, and its approximability is a famous open problem.

We significantly improve the known approximation ratio for minimum bisection. Our algorithms achieve an approximation ratio $O(\log^2 n)$, which is "in the same ballpark" as the current approximation ratios for many related problems.

Another approach for coping with the NP-hardness is to relax the requirement for *worst-case* analysis, and consider instead *heuristic algorithms* that are successful on *average-case* input instances. One main difficulty in providing rigorous analysis of heuristics lies in realistically modeling average-case instances.

Consider for example the *hidden clique problem.* In a random model for the problem, a random graph on n vertices is chosen (i.e. $G_{n,1/2}$) and then a clique of size k is randomly placed in the graph, and the goal is to find the planted clique in the graph. A semi-random model may extend this random model by allowing an adversary to remove any edge that is not inside the planted clique.

We devise for the hidden clique problem a heuristic that is based on the Lovász theta function, a well-known semidefinite programming relaxation of maximum clique. Our heuristic is successful in the semi-random model when $k \geq \Omega(\sqrt{n})$. In contrast, previous heuristics have similar success in the random model but fail in the semi-random model.

We also study relaxations that are stronger than the Lovász theta function, namely those obtained by the "lift-and-project" method of Lovász and Schrijver (1991). We show that on a random graph $G_{n,1/2}$ the value of these stronger relaxations is comparable to the theta function, and hence they do not extend our heuristic mentioned above to a planted clique of smaller size $k = o(\sqrt{n})$.

This thesis is based on the following papers.

1. U. Feige and R. Krauthgamer. Finding and certifying a large hidden clique in a semirandom graph. *Random Structures Algorithms*, 16(2):195–208, 2000.
2. U. Feige, R. Krauthgamer, and K. Nissim. Approximating the minimum bisection size. In *32nd Annual ACM Symposium on Theory of Computing*, pages 530–536, May 2000.
3. U. Feige and R. Krauthgamer. A polylogarithmic approximation of the minimum bisection. In *41st Annual IEEE Symposium on Foundations of Computer Science*, pages 105–115, November 2000.
4. U. Feige and R. Krauthgamer. The probable value of the Lovász-Schrijver relaxations for maximum independent set. Manuscript, April 2001.

Declaration. I, Robert Krauthgamer, declare that this thesis summarizes my independent work under the supervision of Professor Uriel Feige, with the exception of Section 2.6, whose results were obtained jointly with Kobbi Nissim.

Acknowledgments

It is a great pleasure to acknowledge the people who helped me along the way. My first and foremost gratitude goes to my advisor, Uri Feige, who helped me open the door into the world of scientific research. I am indebted to his remarkable guidance, which led me through many unknown paths during this work. Working with him was a fascinating experience from which I (hope I have) learned a lot.

The Weizmann Institute was a wonderful environment to be a student in. I thank the entire staff for actively creating this environment, and particularly Oded Goldreich, Shafi Goldwasser, David Harel, Moni Naor, David Peleg, Ran Raz, and Adi Shamir for valuable interactions and feedback, and for many courses and seminars.

I am grateful to colleagues with whom I had the pleasure to do research (that did not get into this thesis). I thank Nati Linial, Ori Sasson, Michael Mitzenmacher, Andrei Broder, Guy Kortsarz, T.S. Jayram, Tracy Kimbrel, Baruch Schieber, Maxim Sviridenko, Kobbi Nissim, Shai Halevi and Eyal Kushilevitz for collaborating with me.

Many thanks go to my fellow students at Weizmann for an abundance of informal sessions and discussions, and for their exciting company. A short (but definitely incomplete) list includes Kobbi Nissim, Yehuda Hassin, Michael Langberg, Benny Pinkas, Omer Reingold, Sitvanit Ruah, Elad Shachar, Alon Rosen, Yehuda Lindell, Michael Elkin and Hillel Kugler.

My deepest gratitude goes to my entire family, for their love and for their belief in me. In particular, my parents, Arnon and Rita, deserve all the credit for motivating me since my childhood and for supporting me on this long journey.

Finally, I am most grateful to my wife, who accompanied me in these doctoral studies with great patience and fidelity. Gali, your endless love and encouragement gives me confidence in my way.

Contents

Chapter 1

Introduction

Optimization problems arise naturally in a variety of practical and theoretical contexts. Many such important problems are known to be NP-hard; that is, unless P = NP, there is no polynomial time algorithm that optimally solves these problems on every input instance, see e.g. [GJ79].

We study algorithmic ways for "coping" with NP-hard optimization problems. In many cases, merely classifying a problem as NP-hard does not suffice, and one needs to apply some algorithm for dealing with the problem at hand. It is therefore desirable to devise algorithms that cope with the NP-hardness, by means of relaxing some of the requirements.

One approach for coping with the NP-hardness is to relax the requirement for *exact solution*, and settle for an *approximate solution*, i.e. a solution that is guaranteed to be nearly optimal. These *approximation algorithms* are usually evaluated by their *approximation ratio*, which is the worst-case ratio between the values of the solution provided by the algorithm and that of the optimum solution.

Another approach for coping with the NP-hardness is to relax the requirement for *worst-case* analysis, and consider instead the behavior of an algorithm on *average-case* input instances. We are interested in evaluating these *heuristic algorithms* by providing for them *rigorous performance guarantees* (and not by experimental methods such as benchmarks).

In this work we restrict our attention to the two approaches mentioned above, namely approximation algorithms and analysis of heuristics, but we remark that there are also other approaches. For example, one may relax the polynomial time restriction and seek subexponential time algorithms, trying to improve over straightforward exhaustive search. Another approach examines whether a problem is fixed-parameter tractable, which means that there exists for the problem an algorithm whose running time is polynomial when some parameter of the problem (e.g. the value of the optimum solution) is fixed independently of the input size. It is also possible to combine several approaches, such as heuristics that produce (on average) nearly optimal solutions, or (superpolynomial) branch and bound algorithms that save running time by using approximation algorithms (or heuristics) to skip branches that do not contain (or are unlikely to contain) an optimal solution.

The literature on coping with NP-hard optimization problems is immense, and we will only mention a few references of a broader scope. The famous example of the traveling salesman problem (TSP) is studied thoroughly in [LLKS85], including approximation algorithms, average-case analysis, empirical evaluation, polyhedral methods, and branch and bound methods. The approach of approximation algorithms is studied in [Hoc97] and in [Vaz01] (see also the compendium [CK95]). Average-case analysis (from the viewpoint of random graphs) is surveyed in [FM97]. The approach of fixed-parameter tractability is studied in [DF99]. Various heuristics are evaluated empirically in [Rei94] and in [JT96].

One may wonder whether this diversity of approaches for coping with NP-hardness is really necessary. One possible reason for this is that no one approach seems to be suitable in all cases and effective for all problems. Approximation algorithms, for example, seem promising in several problems (e.g. Euclidean TSP [Aro98]), where an arbitrarily good approximation can be found in polynomial time, but other problems (e.g. maximum clique [Hås99]), cannot be approximated within ratio of $n^{1-\epsilon}$, for any fixed $\epsilon > 0$, unless P = ZPP. The diversity of approaches seems essential also because of the difficulty in comparing different approaches. In particular, an algorithm that performs well with respect to one criterion, might perform poorly with respect to another criterion.

1.1 Basic terminology

An *optimization problem* is a search problem, i.e. each problem instance has a set of feasible solutions, where the size of each solution is polynomial in the size of the instance. Each solution is associated with a *value* (e.g. cost or benefit), which is a positive integer that can be computed from the instance and the solution in polynomial time. The optimization problem can be either a minimization problem or a maximization problem, and it requires to find a feasible solution with an optimal (minimal or maximal, respectively) value.

In general, we specify the computational resources (time, space and randomness) used by an algorithm as a function of the input size, where the input is assumed to be coded in an alphabet of a fixed size (e.g. in binary). However, in graph problems it is more convenient to characterize the input size by the number of vertices in the input graph, which is denoted throughout by n. We say that an algorithm is *efficient* if its running time is polynomial (in the input size).

We next define the optimization problems that we address in this work. These graph problems are known to be NP-hard, see e.g. [GJ79, CK95].

Minimum bisection. A *cut* of a graph is a partition of the graph vertices into two nonempty subsets called the *sides* of the cut, and consists of the edges with one endpoint in each subset. The edges of a cut are also said to *cross* the cut. The *cost* of a cut is the number of edges that cross it. A *bisection* is a cut whose two sides are of equal cardinality. The *minimum bisection problem* requires to find in an input graph a bisection of minimal cost.

Maximum clique. A *clique* in a graph is a subset of its vertices that induce a complete subgraph, i.e. every two vertices in the subset are connected by an edge. The *size* of a clique is the number of vertices in it. The *maximum clique problem* requires to find in an input graph a clique of maximal size.

Maximum independent set (a.k.a. stable set). An *independent set* in a graph is a subset of its vertices that induce an empty subgraph, i.e. every two vertices in the subset are not connected by an edge. The *size* of an independent set is the number of vertices in it. The *maximum independent set problem* requires to find in an input graph an independent set of maximal size.

It is well-known that that a clique in a graph G corresponds to an independent set in the edge complement graph $\bar{G}$, and therefore there is an equivalence between the maximum clique problem and the maximum independent set problem.

1.2 Approximation algorithms

In many applications it is reasonable to compromise on a solution whose value is close to the optimum, if such a solution can be found efficiently. Therefore, the quality of a solution is measured by the proximity of its value to that of the optimal solution. Most commonly, the proximity between two values is measured by their ratio (although it is sometimes plausible to measure it by their difference).

Definitions. A polynomial time algorithm A has *approximation ratio* $r \geq 1$ if for any instance of the problem, algorithm A produces a solution whose value is within a ratio of r from the value of an optimal solution for this instance. We also say that A is an r *approximation algorithm*. Note that the approximation ratio is measured for the worst-case instance of the problem. Typically, r is allowed to be a function of the instance size (e.g. of the number of vertices n in the input graph for graph problems). In the case of a *randomized* approximation algorithm we consider the expected value of the algorithm's solution to the instance (where the expectation is taken over the coin tosses of the algorithm).

A family of algorithms is called a *polynomial time approximation scheme* (PTAS) if the family contains algorithms with approximation ratios that are arbitrarily close to 1, i.e. the family contains a $1+\epsilon$ approximation algorithm for every $\epsilon > 0$. Such a family of algorithms is called a *fully polynomial time approximation scheme* (FPTAS) if for every $\epsilon > 0$, the family contains a $1+\epsilon$ approximation algorithm whose running time is polynomial in $1/\epsilon$.

Background. The approximation ratio of an NP-hard problem is usually studied from two directions. An *approximability result* shows that this problem can be approximated within some ratio r_1, by devising an approximation algorithm for it. An *inapproximability result* shows that the problem cannot be approximated within some ratio r_2, unless P = NP (or a similar assumption). For some problems, the approximability and inapproximability results are essentially tight, namely r_1 and r_2 are equal up to low order terms, yielding an *approximation threshold* that completely classifies this problem in terms of approximation. For other problems, there is currently a large gap between the two types of results, and they are not well-understood in terms of approximation. The compendium [CK95] contains references to most of the results achieved in this field, and many important results are described in [Hoc97] and in [Vaz01].

NP-hard optimization problems might differ quite substantially in terms of their approximation ratios. We mention below a few typical examples, although it should be noted that there are also other (less typical) approximation ratios, see e.g. [GKR$^+$99] and [EP00].

Some problems are known to be NP-hard to solve exactly but have an arbitrarily good fixed approximation ratio. One example for a problem that admit an FPTAS is Knapsack[IK75]. Examples for problems that admit PTAS include Bin-Packing [FdlVL81] and Euclidean-TSP [Aro98].

The "next" level in the quality of approximation is a constant ratio. On the one hand, many problems are known to be approximable within a constant ratio. On the other hand, many of them do not have a PTAS, unless P = NP. That is, for each such problem there is an inapproximability result to within some constant ratio larger than 1. Many of these results are proved using a general technique that follows from the connection between *Probabilistically Checkable Proofs* (PCPs) and inapproximability [FGL$^+$96, AS98, ALM$^+$98]. For example, each of the problems MAX-SAT, MAX-CUT, Vertex cover, Metric-TSP, Multiway cut and Steiner tree is approximable within some constant ratio, but inapproximable (assuming P $\neq$ NP) within another constant ratio, see e.g. [PY91, ALM$^+$98]. A few problems have a constant approximation threshold, e.g. for a version of MAX-SAT known as MAX-E3SAT, a 8/7 approximation algorithm [Joh74] is matched by an inapproximability result [Hås97] within a ratio of $8/7-\epsilon$, for any fixed $\epsilon > 0$ (assuming P $\neq$ NP).

Other problems, such as Set-Cover, Hitting-Set and Dominating-Set, can be approximated [Joh74] within a logarithmic order, but cannot be approximated [Fei98] within $(1-\epsilon)\ln n$, for any $\epsilon > 0$ (unless NP $\subseteq DTIME(n^{O(\log\log n)})$). Several problems have a polylogarithmic approximation ratio and a far from tight (if any) inapproximability result, e.g. cutwidth [LR99], bandwidth [Fei00] and minimum bisection (see Section 1.4).

There are problems which are even harder to approximate. Several problems, such as Label cover, Nearest Lattice Vector (CVP), Nearest Codeword and Longest Path cannot be approximated [ABSS97, KMR97] within $2^{\log^{1-\epsilon} n}$, for any fixed $\epsilon > 0$, unless NP $\subseteq$ $DTIME(n^{\text{polylog} n})$ (for some of these problems, these inapproximability results were slightly improved in [DS99, DKRS99]). However, there is still a large gap between the inapproximability results and the approximability results for these problems, and it is possible that they are even harder to approximate.

Some problems, such as maximum clique and chromatic number, cannot be approximated [Hås99, FK98] within $n^{1-\epsilon}$, for any fixed $\epsilon > 0$, unless P = ZPP. Known algorithms achieve approximation ratios of $O(n/\log^2 n)$ for maximum clique [BH92] and $O(n(\log\log n)^2/\log^3 n)$ for chromatic number [Hal93].

1.3 Analysis of heuristics

Although hard to solve in the worst-case, NP-hard problems may be significantly easier on "average" instances encountered in practice. It is therefore desirable to devise *heuristic algorithms*, that successfully produce an optimal solution on average-case instances. We wish to evaluate heuristics by rigorous analysis methods that explain or predict good behavior of the heuristic in real-life instances.

Most commonly, a rigorous analysis of a heuristic consists of an *input model* and of *performance guarantees*. The input model defines which input instances are considered as average-case instances. The performance guarantees are desired properties that the heuristic should satisfy when it is applied on average-case instances from the input model.

Many results and open problems in this area are surveyed by Frieze and McDiarmid [FM97].

Random input models. A main difficulty in analyzing heuristics is to devise a input model that realistically represents average-case instances that occur in practice. One possible input model is a *random model*, that assumes some probability distribution on the input instances. Usually, the desired performance guarantee is that the heuristic successfully finds an optimal solution on all but a "vanishing" part of the input distribution. Formally, an event in a distribution of graphs on n vertices is said to happen *almost surely* if its probability is $1 - o(1)$ (i.e. approaches 1 as n tends to infinity); then the formal requirement of the performance guarantee stated above is that the heuristic almost surely finds an optimal solution.

A straightforward random model for graph problems is $G_{n,p}$, the *random graph* on n vertices with *edge probability* p, which is formally defined as the distribution generated by placing an edge independently with probability p between each pair of n vertices. For example, the distribution $G_{n,1/2}$ represents a uniform distribution on all graphs on n labeled vertices. In general, however, p may depend on n.

For some problems, the model of a random graph $G_{n,p}$ is an inadequate framework for evaluating heuristics. We illustrated this by the minimum bisection problem. It can be seen that for $p \gg 1/n$, almost surely all the bisections in a random graph $G_{n,p}$ have cost roughly $pn^2/4$. Therefore, good and bad heuristics have nearly the same performance in this random graph model.

Another random model that has been suggested is similar to the random graph model, except that one solution is "planted" in the graph. That is, the graph is created by first choosing a solution at random, and then placing edges in the graph at random in a way that ensures that the chosen solution will almost surely be an optimal solution. For example, a planted bisection model may first choose a random partition of the n graph vertices into two equal-size subsets, and then place edges at random, so that an edge is placed with probability q if it crosses the bisection defined by the chosen partition, and with probability p if it does not. If q is sufficiently smaller than p, then the bisection defined by the chosen partition is, almost surely, a minimum bisection. Several heuristics for this and similar planted bisection models were studied in [BCLS87, Bop87, DF89, JS98, CK99, FK01a].

Semi-random input models. Although the input distributions employed by random models are quite natural, there is usually no claim that these models represent instances that occur in real-life applications. Furthermore, a heuristic that relies excessively on statistical properties of the graphs in these distributions (e.g. all vertices have roughly the same degree), might perform well on these specific distributions, but poorly on more realistic distributions. It is therefore desirable to have input models that represent (effectively) a wider range of instances.

To enrich the input model, Blum and Spencer [BS95] suggested a *semi-random model*, in which the input is generated by a mixture of random and adversarial choices. In the strongest of their semi-random models, a graph is first drawn at random from some distribution, and then an adversary can modify this graph subject to some restrictions. The desired performance guarantee is that regardless of the adversary (i.e. for all adversaries), the heuristic almost surely finds an optimal solution. Here, the probability is taken over the choice of the algorithm's coin tosses and the choice of the graph from the distribution (before modification by the adversary).

Feige and Kilian [FK01a] formalized this semi-random model of [BS95] as a *monotone adversary model*, in which the adversary is allowed to add certain edges and/or remove certain other edges (depending on the problem). For example, they consider the planted bisection model described above, together with an adversary that is allowed to remove edges that cross the planted bisection and to add edges that do not cross it. Clearly, such adversarial moves can only decrease the cost of the planted bisection or increase the cost of other bisections, and so it may appear, intuitively, that the adversary can only make the bisection problem easier. However, as Feige and Kilian note, this monotone adversary can foil many popular techniques for heuristics, e.g. it can alter the degrees of vertices, create bisections that are "locally optimum", and modify the spectrum (eigenvalues of the adjacency matrix) of the graph. A heuristic that is successful in a semi-random model withstands such an adversarial "help", and is therefore more robust.

Interestingly, it is also possible to show hardness results for the semi-random graph model. Blum and Spencer [BS95] show that in a certain semi-random model, there is no successful heuristic for the problem of coloring a graph with 4 colors, unless NP $\subseteq$ BPP. Feige and Kilian [FK01a] show a similar result for the maximum independent set problem.

Evidence for optimality. An average-case algorithm does not have an apriori guarantee on its performance, and it is therefore valuable to certify that the solution it produced on the particular instance at hand is indeed optimal. The algorithm of Boppana [Bop87] for the minimum bisection problem has such a certification property (see also [FK01a]). His algorithm outputs a bisection together with a lower bound (that is obtained by a relaxation) on the minimum cost of a bisection. If the cost of the output bisection is equal to the lower bound then it is clear that the output bisection is indeed an optimal solution. Boppana's analysis shows that this is indeed the case, almost surely.

Average polynomial time. Another possible performance guarantee is that of *average polynomial time*, which means that on any instance the heuristic finds the planted solution (or an optimal solution) and that the expected running time of the heuristic (over the distribution of the input instances) is polynomial. For example, Dyer and Frieze [DF89] show for several graph problems an average polynomial time algorithm in a random model with constant edge probabilities. Some improvements (to smaller edge probabilities and for semi-random models) are given in [SFVM98, Sub99].

Related areas. Note that NP-hard problems are not necessarily easy on the average. Distributions on which a problem is hard on the average case are necessary for cryptography. It is therefore important to identify problems and corresponding distributions, on which the problem is hard on the average.

Levin [Lev86] put a basis for a theory of average NP-completeness. The emphasis in Levin's theory appears to be to identify distributions on which the underlying problem is hard. In contrast, the emphasis in our work is to provide algorithms that perform well on average with respect to distributions that occur in practice, and not necessarily with respect to the most difficult distributions.

1.4 Overview of our results

Our research on coping with NP-hard optimization problems spans the two approaches described above, namely approximation algorithms and analysis of heuristics. In each approach, we concentrate on one graph problem that is fundamental in the sense that a better understanding of it may reflect on our understanding of many other problems and, possibly, of the whole approach.

Approximation algorithms for minimum bisection. In the last decade, our understanding of the approximability of many NP-hard optimization problems was greatly improved, due to both approximation algorithms and hardness of approximation results. For many problems, known algorithm ratios match the hardness of approximation results, up to an order of magnitude or less. However, there is still a large gap in our understanding of the approximability of several fundamental problems.

Notable examples to large gaps between approximability and inapproximability results are graph partitioning problems, and, in particular, minimum bisection. (Recall from Section 1.1 that a bisection is a cut that partitions the vertices into two sets of equal cardinality; the minimum bisection problem requires to find in an input graph a bisection of minimum cost).

In a seminal work, Leighton and Rao [LR88, LR99] obtained a bicriteria approximation (a.k.a. pseudo-approximation) algorithm. That is, given an input graph on n vertices, their algorithm finds a *2/3-balanced cut* (i.e. a cut that partitions the vertices into two sets, each of cardinality at most $2n/3$) whose cost is at most $O(\log n)$ times that of the minimum cost bisection. The techniques and results of Leighton and Rao found many applications and inspired additional work, see e.g. [LR99, Shm97, ENRS99].

However, prior to our work there was no major progress on the approximation ratio of minimum bisection (i.e. when the strict constraint on the cardinalities of two sides of the cut cannot be relaxed). On the one hand, there is no hardness of approximation result that excludes the possibility that minimum bisection admits a PTAS. On the other hand, the known approximation ratio was $n/2$, due to Saran and Vazirani [SV95].

We devise an algorithm that approximates the minimum bisection within a ratio of $O(\log^2 n)$. This approximation ratio improves over the previous (linear in n) approximation ratio significantly, and is, in particular, "in the same ballpark" as the bicriteria algorithms of [LR88, LR99] and [ENRS99].

Our algorithm extends (with the same approximation ratio) to minimum bisection in graphs with arbitrary nonnegative edge costs and polynomially bounded nonnegative integer vertex weights. It also extends to cutting away from the graph k vertices, where k is given as part of the input, and to cutting the input graph into any fixed number of parts of equal cardinality.

Our approximation algorithm follows a divide and conquer approach, where the input graph is recursively divided into smaller parts based on a new cut notion that we define, and the parts' solutions are combined using dynamic programming. Our new cut notion is related to a *min-ratio cut* (i.e. a cut with the minimal ratio between the cost of the cut and the number of vertices in the smaller of its two sides), and we show how it can be computed from an approximate min-ratio cut using flow techniques (i.e. min (s,t)-cut).

The approximation ratio of our algorithm can be described as $O(\tau \log n)$, where τ is the approximation ratio for the problem of finding in a graph a min-ratio cut. Known algorithms for general graphs achieve $\tau = O(\log n)$, see [LR88, LR99] and [AR98, LLR95]. For graphs excluding any fixed graph as a minor (e.g. planar graphs), known algorithms achieve a constant ratio, i.e. $\tau = O(1)$, see [KPR93], and hence in these graphs our algorithm approximates minimum bisection within ratio $O(\log n)$.

We also devise a simpler (randomized) algorithm for minimum bisection, whose approximation ratio is better (than the one above) in the variant that requires to cut away a relatively small number of vertices. Namely, the algorithm finds a cut that separates k vertices (where k is given as part of the input) at a cost that is within a ratio of $1 + \epsilon k / \log n$ from the minimum, for an arbitrarily small constant $\epsilon > 0$. In particular, this algorithm yields a PTAS for the problem of cutting $k = O(\log n)$ vertices from a graph (a problem that is not known to be in P). The algorithm extends to graphs with arbitrary nonnegative edge weights.

These two approximation algorithms for minimum bisection are described in full in Chapter 2. Preliminary versions of these results appeared in [FK00b] and in [FKN00, Section 5].

Analysis of heuristics for maximum clique. The maximum clique problem appears to be difficult on the input model of a random graph $G_{n,1/2}$. It is known that the maximum size of a clique in $G_{n,1/2}$ is roughly $2\log_2 n$, almost surely, see e.g. [AS92]. Several simple and natural algorithms (e.g. the greedy one) find a clique of size roughly $\log_2 n$, almost surely. Karp [Kar76] suggested the problem of finding a clique of size significantly larger than $\log_2 n$, but no efficient algorithm is known to achieve that. Finding cliques of size $\frac{3}{2}\log_2 n$ in a random graph $G_{n,1/2}$ was even suggested in [JP00] as a hard computational problem on which to base cryptographic applications

We focus on the *hidden clique problem*, which is a variant with a planted solution. In the random model of this problem, a random graph $G_{n,1/2}$ is chosen, and then a clique of size k is randomly placed in the graph. The goal is to find in the graph a clique of size k.

For the hidden clique problem in the random model, Kučera [Kuč95] observed that taking the vertices with highest degrees almost surely succeeds in finding the hidden clique, when $k > c\sqrt{n \log n}$ for a sufficiently large constant $c > 0$. Alon, Krivelevich and Sudakov [AKS98] showed an algorithm based on eigenvalue techniques that almost surely finds the hidden clique, when $k \geq \Omega(\sqrt{n})$. Jerrum showed that the Metropolis process does not find the clique, almost surely, when $k = o(\sqrt{n})$.

We devise another heuristic for the hidden clique problem. Our heuristic also finds the hidden clique, almost surely, when $k \geq \Omega(\sqrt{n})$, but it extends to a semi-random model of the problem, in which an adversary is allowed to remove (from the random graph with the planted clique) any edge that is not inside the planted clique. In contrast, the previous algorithms of [Kuč95, AKS98] have similar success in the random model, but fail in the semi-random model, unless $k = \Omega(n)$. An additional useful property of our heuristic is that it almost surely certifies the optimality its solution. Namely, the heuristic produces its solution together with an upper bound on the size of the maximum clique in the input graph, and the value of the solution matches, almost surely, the upper bound.

Our heuristic is based on the Lovász theta function, a well-known semidefinite programming relaxation of the maximum clique problem. For the random model, our main argument is that the relaxation is almost surely tight and corresponds to the planted clique. We then extend the result to the semi-random model by using the monotonicity of the relaxation with respect to removing edges from the graph. Note that in the worst case, the Lovász theta function is far from being tight [Fei97], so in terms of approximation ratio it gives a poor guarantee.

A possible direction for extending our heuristic to a planted clique of smaller size $k = o(\sqrt{n})$, is to use a stronger relaxation than the Lovász theta function. In particular, if the relaxation is monotone with respect to adding edges, it may be plausible to compare the almost sure value of the relaxation on a random graph $G_{n,1/2}$, which we denote by $\hat{k}$, with the size k of the planted clique. If $k < \hat{k}$ then, almost surely, the relaxation value on the hidden clique graph would be at least $\hat{k}$ and the relaxation would not be tight. However, if $k > \hat{k}$ then it may be the case that, almost surely, the relaxation is tight on the hidden clique graph (i.e. has value k), and can be used to find the planted clique. For example, our heuristic mentioned above implements this approach, based on Juhász' proof [Juh82] that the theta function of a random graph $G_{n,1/2}$ is almost surely $\Theta(\sqrt{n})$.

Lovász and Schrijver [LS91] designed a powerful "lift-and-project" procedure that produces semidefinite programming relaxations that are stronger than the Lovász theta function. Their general technique (it can be applied to any 0-1 integer programming problem) produces a sequence of tighter and tighter relaxations, so that the nth relaxation in the sequence is guaranteed to be tight (where n is the number of variables in the integer program). For the maximum clique problem, they show that the first relaxation in the sequence is already at least as tight as the theta function. Furthermore, for any fixed r, the rth relaxation in the sequence can be computed in polynomial time, up to an arbitrarily small error, and is therefore a plausible candidate for a hidden clique heuristic.

We show that on a random graph $G_{n,1/2}$, the value of the rth relaxation in the sequence of Lovász and Schrijver [LS91], for $r = o(\log n)$, is almost surely roughly $\sqrt{n/2^r}$. It follows that the rth relaxation for $r = O(1)$ almost surely has a value of $\Theta(\sqrt{n})$, which is comparable (up to constant factors) to the Lovász theta function. Hence, on the hidden clique graph with planted clique size $k = o(\sqrt{n})$, those relaxations in the sequence that are known to be computable in polynomial time are not tight, almost surely, and offer no improved heuristic under the approach outlined above (since improvement by arbitrarily large constant factors can be achieved by other methods due to [AKS98]).

Our results on heuristics for maximum clique are described in full in Chapter 3. One part of these results appeared in [FK00a] and another part is based on [FK01b].

1.5 Perspectives

An important goal in the area of approximation algorithms is to achieve an approximation threshold for minimum bisection and other graph partitioning problems. Our results in Chapter 2 are a significant improvement in the known approximation ratio for minimum bisection, but there is still a considerable gap in the understanding of this problem in terms of approximation, as no hardness of approximation result is known.

Our new cut notion, called amortized cut, is useful to both approximation and bicriteria approximation algorithms for minimum bisection. We show that an algorithm that finds a ρ-amortized cut, can be used to find a bisection whose cost is at most $O(\rho \log n)$ times that of the minimum cost bisection, and also to find a 2/3-balanced cut whose cost is at most $O(\rho)$ times that of the minimum cost bisection (see Section 2.5.6). Our algorithm achieves an amortized cost $\rho = O(\log n)$ by using an approximation algorithm for min-ratio cuts. However, we show that any graph contains an $O(1)$-amortized cut (which does not follow immediately from the definition, since ρ is not an approximation ratio). Therefore, further investigation of the values of ρ that can be achieved by an efficient algorithm, is interesting in the context of improved algorithms for minimum bisection, and may possibly have other applications.

Devising a realistic model of average-case input instances is a main difficulty in any rigorous analysis of heuristics. Our results for the hidden clique problem in Chapter 3, extend results that were previously known in the random model, to a semi-random model that represents a wider range of input graphs. A semi-random model improves over a random model in many respects, but it also has certain drawbacks. For example, in many semi-random models that use a planted solution, this planted solution is almost surely a unique optimal solution. It is therefore possible that heuristics that perform well in this model, perform poorly in other settings where the optimal solution is not unique.

The technique of Lovász and Schrijver [LS91] can be used for many optimization problems, as it allows to produce efficiently computable relaxations that are tighter than before for (almost) any 0-1 integer programming problem. In particular, the relaxations that it produces can be used in various approaches for coping with the NP-hardness of a problem. So far, this technique has not found applications in the area of approximation algorithms, possibly because many aspects of this powerful technique are not well-understood. We view our work in Chapter 3 as a step in the direction of understanding this technique from various aspects, and our analysis of its performance (for maximum clique) on a random graph, is the first application of the technique to average-case analysis.

Chapter 2

Approximating minimum bisection[*]

2.1 Introduction

Let $G(V, E)$ be an undirected graph with n vertices and m edges, where n is even. For a subset S of the vertices (with $S \neq \emptyset, V$), the *cut* $(S, V \setminus S)$ is the set of all edges in G with one endpoint in S and one endpoints in $V \setminus S$; these edges are said to be cut by $(S, V \setminus S)$. The *cost* of a cut is the number of edges in it.

A cut $(S, V \setminus S)$ is called a *bisection* of G if its two sides, S and $V \setminus S$, are each of size $n/2$. We denote the minimum cost of a bisection of G by b. *Minimum bisection* is the problem of computing b for an input graph G. Garey, Johnson, and Stockmeyer [GJS76] show that this problem is NP-hard, and we address the problem of approximating it.

An algorithm is said to *approximate* a minimization problem within ratio $r \geq 1$ if it runs in polynomial time and outputs a solution whose value (or, if the algorithm is randomized, its expected value over the coin tosses of the algorithm) is at most r times the cost of the optimal solution. A problem is said to have a *polynomial time approximation scheme* (PTAS) if for every fixed $r > 1$ there is an algorithm with approximation ratio r.

A cut $(S, V \setminus S)$ with $|S| = k$ is called a $(k, n-k)$ *cut* of G. Let b_k denote the minimum cost of a $(k, n-k)$ cut in G. In the *minimum* $(k, n-k)$ *cut problem*, we are given a graph G and a number $k \in \{1, \dots, n-1\}$, and we wish to compute b_k. The minimum $(k, n-k)$ cut problem is NP-hard, as it includes minimum bisection as the special case $k = n/2$. Furthermore, the proof of Bui and Jones [BJ92] actually shows that it is NP-hard to compute b_k in graphs of maximum degree 3 and for $k = \alpha n$ (and even $k = n^\alpha$) for any fixed $0 < \alpha < 1$. We address the problem of approximating b_k.

It is not known whether b_k is polynomial time computable when k is a slowly growing function of n, say $k = \log n$. Note that a straightforward exhaustive search on all vertex subsets of size k can find b_k in time $n^{k+\Theta(1)}$, which is polynomial only if $k = O(1)$, i.e. a fixed constant independent of n.

[*]This chapter is based on the full versions of [FK00b] and of [FKN00, Section 5].

2.1.1 Previous work

Leighton and Rao [LR88, LR99] showed how to approximate within ratio $O(\log n)$ *minimum-quotient cuts*, which we shall call *min-ratio cuts*. In these cuts, one wishes to minimize the *cut ratio* (also called *edge expansion* or *flux*) $c/|S|$, where c is the number of edges cut, and $|S|$ is the cardinality of the smaller of the two vertex sets.

A *β-balanced cut* is a cut that partitions the graph into two parts, each of size at most βn. Leighton and Rao [LR88] used the approximate min-ratio cuts to find a 2/3-balanced cut (also called *edge separator*) with at most $O(b \log n)$ edges, see also [LR99, Shm97]. Note that such a 2/3-balanced cut does not provide an $O(\log n)$ approximation for the value of b. For example, when the graph consists of 3 disjoint cliques of equal size, an optimal 2/3-balanced cut has no edges, whereas $b = \Omega(n^2)$.

A straightforward approach for obtaining an exact bisection is to first find an almost balanced cut (e.g. using approximate min-ratio cuts) and then move a few low degree vertices from one side to the other. Using this approach one can approximate bisection within a ratio of $\tilde{O}(\sqrt{m/b})$ (we use $\tilde{O}(f)$ to denote $O(f \cdot \text{polylog } n)$) see e.g. [LR99, Footnote 10] and [FKN00]. This is a dramatic improvement over the naive ratio of $O(m/b)$ (achieved by arbitrarily picking $n/2$ vertices), but might still be larger than n.

In terms of n, the best approximation ratio known prior to our work was $n/2$, due to Saran and Vazirani [SV95]. We presented in [FKN00] an approximation algorithm that achieves approximation ratio $\tilde{O}(\sqrt{n})$. In [FK00b], we improved the approximation ratio to polylogarithmic in n, by using similar techniques (e.g. approximate min-ratio cuts and dynamic programming), but in a more sophisticated way. In this thesis, we describe the improved approximation algorithm from [FK00b], and a related result (that was not improved) from [FKN00, Section 5].

Additional related work include the following. In [AKK99], Arora, Karger and Karpinski show that bisection has a PTAS for everywhere-dense graphs, i.e. graphs with minimum degree $\Omega(n)$. In [GSV99], Garg, Saran and Vazirani give an approximation ratio of 2 for the problem of finding a 2/3-balanced cut of minimum cost in a planar graph. Their result extends to a β-balanced cut, for any $\beta \geq 2/3$, but does not extend to a bisection, which is a 1/2-balanced cut. In [BJ92], Bui and Jones show that for any fixed $\epsilon > 0$, it is NP-hard to approximate the minimum bisection within an *additive* term of $n^{2-\epsilon}$. In terms of approximation ratio, however, there is no known hardness of approximation result which excludes the possibility that bisection has a PTAS. Several heuristics for minimum bisection are studied (in terms of average-case behavior) in [BCLS87, Bop87, DF89, JS98, CK99, FK01a].

2.1.2 Our results

Our main result is an algorithm for approximating the minimum bisection within a polylogarithmic ratio.

Theorem 2.1. *A bisection of cost within ratio of $O(\log^2 n)$ of the minimum can be computed in polynomial time.*

In Section 2.2 we give an overview of the algorithm. On a high level, the algorithm follows a divide-and-conquer approach. The input graph is recursively divided into parts, using a new cut notion which we call an *amortized cut*, and then the parts are combined into a bisection using dynamic programming.

In Section 2.4 we describe our algorithm for approximating bisection, based on a subroutine for finding an amortized cut. If the subroutine is guaranteed to find a ρ-amortized cut in a graph, the algorithm computes a bisection whose cost is within ratio of $1 + O(\rho \log n)$ of the minimum.

In Section 2.3 we devise an algorithm for finding an $O(\log n)$-amortized cut in a general graph. By using this algorithm as a subroutine in the $1 + O(\rho \log n)$ approximation algorithm for bisection, we are guaranteed that $\rho = O(\log n)$, proving Theorem 2.1. The subroutine uses a τ-approximate min-ratio cut in order to find an $O(\tau)$-amortized cut. The best known approximation algorithms for min-ratio cut in general graphs, due to Leighton and Rao [LR88, LR99] and due to [AR98, LLR95], have approximation ratio $\tau = O(\log n)$.

In certain graph families, there is a better approximation ratio τ for the min-ratio cut problem. If these graph families are closed under taking induced subgraphs, then we can approximate bisection within an improved ratio of $O(\tau \log n)$. For example, it is shown in [KPR93] that in graphs excluding any fixed graph as a minor (e.g. bounded-genus graphs) min-ratio cut can be approximated within a constant ratio, i.e. $\tau = O(1)$.

Theorem 2.2. *In graphs excluding any fixed graph as a minor (e.g. planar graphs), a bisection of cost within ratio of $O(\log n)$ of the minimum can be computed in polynomial time.*

In Section 2.5 we show that our results extend to several natural generalizations of the bisection problem. These extensions include, for example, bisection of graphs with arbitrary nonnegative edge costs and graph partitioning into three parts of equal size.

Cutting few vertices. We present a simple randomized algorithm that is aimed towards approximating the minimum $(k, n-k)$ cut problem when k is relatively small. The algorithm and its analysis are described in Section 2.6, where we prove the following theorem. We say that an event happens *with high probability* if its probability approaches 1 as n goes to infinity.

Theorem 2.3. *For every fixed $\epsilon > 0$, there is a polynomial time randomized algorithm that finds, with high probability (over the coin tosses of the algorithm), a $(k, n-k)$ cut whose cost is at most $(1 + \epsilon k/\ln n)b_k$.*

In particular, the above algorithm implies (by the Markov inequality) the following approximation ratios for $k = O(\log n)$ and for $k = \Omega(\log n)$. Note that Corollary 2.2 should be used only when k is slightly larger than $O(\log n)$, while for larger k the approximation ratio of Theorem 2.1 is preferable.

Corollary 2.1. *For any $k = O(\log n)$, there is a PTAS for the minimum $(k, n-k)$ cut problem.*

Corollary 2.2. *For any $k = \Omega(\log n)$, the minimum $(k, n-k)$ cut problem can be approximated within a ratio of $O(k/\log n)$.*

2.1.3 Conventions and notation

We will often denote the two sides of a (not necessarily optimal) bisection as *white* W and *black* B. A graph may have several different bisections of minimum cost. For the analysis, let us fix one of them (arbitrarily) and call it *the fixed optimal bisection* (W^*, B^*).

For V_1, V_2 two disjoint subsets of vertices in a graph, let $e(V_1, V_2)$ denote the number of edges with one endpoint in V_1 and the other endpoints in V_2. Subsets $V_1, V_2 \subset V$ are called a *partition* of V if they are nonempty, disjoint, and their union is equal to V. In our context, V is the vertex set of a graph, and then a partition $V = V_1 \cup V_2$ is equivalent to the cut (V_1, V_2).

A subset of vertices $S \subset V$ with $0 < |S| < |V|$, corresponds to a cut $(S, \overline{S})$ in the graph, where $\overline{S} = V \setminus S$. We denote by $r(S)$ the ratio of this cut, i.e. $r(S) = \frac{e(S,\overline{S})}{\min\{|S|,|\overline{S}|\}}$, and by $r'(S)$ the ratio of this cut towards S, i.e. $r'(S) = \frac{e(S,\overline{S})}{|S|}$. We call S a *part* of the graph, referring either to the set of vertices S or to the subgraph induced on S, depending on the context.

2.2 Overview and techniques

Our approximation algorithm for minimum bisection has three stages, as outlined below.

Stage 1: Decomposition. This stage consists of a sequence of *divide steps.* The input to a divide step is a *part* of the input graph G, i.e. a vertex set and the subgraph induced on it, and the output is a partition of the vertex set into two nonempty subsets, giving two new parts of the graph. These divide steps are applied on the input graph G recursively, until it is decomposed into individual vertices.

The output of the whole decomposition stage is a binary tree T, that we call the *decomposition tree.* Each node i of the tree contains a part V_i obtained in a divide step, as follows. The root of the tree contains the input graph G, the leaves of the tree contain individual vertices of G, and the two direct descendants of a node i are the two subparts obtained in the divide step of its part V_i.

To complete the description of the decomposition stage, we need to explain how a divide step is performed. This is done using a new notion called an *amortized cut*, which we define later in this section. We devise an algorithm for finding amortized cuts in Section 2.3. The decomposition stage is described in more detail in Section 2.4.1.

Stage 2: Labeling. Consider a *labeling* of the decomposition tree T, which labels each (nonleaf) tree node as either white or black. Fixing a parameter $1/2 < \alpha < 1$, we say that a labeling is *α-consistent* with respect to a white-black bisection (W, B) of the input graph if every part V_i (at a tree node i) satisfies that $|W \cap V_i| \leq \alpha|V_i|$ if the label of node i is white, and that $|B \cap V_i| \leq \alpha|V_i|$ if the label of node i is black.

The desired outcome of the labeling stage is a labeling which is α-consistent with the fixed optimal bisection (W^*, B^*), called in short an *opt-consistent labeling.* However, an optimal bisection is not known to the algorithm, so instead of finding an opt-consistent labeling, this stage produces a family of labelings, such that at least one member of the family is opt-consistent. The description of how this is done is deferred to Section 2.4.2. For the purpose of this overview, it will be convenient to think of the labeling stage as if it produces only one labeling, which is opt-consistent.

Stage 3: Combining. Given a decomposition tree T and an arbitrary (not necessarily opt-consistent) labeling of it, the combining stage assigns to each vertex v of the input graph G a *white charge* and a *black charge*. The two charges are simple to compute based on the labels along the path from the root of T to the leaf that contains the vertex v.

The *charge of a bisection* (W, B) of the input graph G (with respect to the labeling) is defined as the sum of the white charges of the vertices of W and the black charges of the vertices of B. The functions white charge and black charge have the property that for every bisection, charge is an upper bound on cost (regardless of the labeling).

If the charge is defined with respect to an opt-consistent labeling of T then our notion of amortized cut used in the decomposition stage guarantees in addition that the charge of the fixed optimal bisection is within a polylogarithmic factor of its cost b. Hence, using the opt-consistent labeling produced by the labeling stage ensures that the input graph G contains a bisection whose charge is within polylogarithmic ratio of b.

Finding a bisection of minimum charge in G is relatively straightforward. Associate with each vertex a *net-charge*, which is its white charge minus its black charge, and pick the $n/2$ vertices with smallest net-charge to form one side W, leaving the remaining $n/2$ vertices in another side B. The bisection (W, B) that we find has minimum charge, and its cost is thus within a polylogarithmic factor of b, the cost of the minimum bisection.

It is interesting to note that finding a minimum cost bisection is an optimization problem with a quadratic objective function (minimizing the number of edges, where edges are pairs of vertices). Finding a minimum charge bisection (given the decomposition tree and an opt-consistent labeling) is an optimization problem with a linear objective function (sum of net-charges over individual vertices). Hence in a sense, our algorithm performs a linearization of a quadratic function, and loses a polylogarithmic factor in the process.

The above presentation of the combining stage was oversimplified. The output of the labeling stage is not one labeling that is opt-consistent, but rather a large family of labelings, such that at least one of them is opt-consistent. Moreover, this family has exponential cardinality, so we cannot try the above net-charge approach on each labeling separately. Instead, we exploit the structure of this family of labelings and use dynamic programming to compute a labeling from the family and a bisection, such that the charge of this bisection with respect to this labeling is minimum over all labeling-bisection pairs. Details appear in Section 2.4.4.

In the rest of the overview we shall introduce and discuss the notion of *amortized cut*, which is of central importance in bounding the ratio between the charge and the cost of the fixed optimal bisection. To motivate this new notion we present our algorithm as a divide-and-conquer algorithm. We then suggest a kind of cut that is desirable for the algorithm's divide step and call this cut notion an amortized cut.

Divide and conquer approach

A possible divide and conquer approach for a graph problem is to divide the input graph G into two parts (using a cut), solve a subproblem for each part, and then combine the solutions of the two subproblems into a solution for G. This approach can be applied recursively, and then the input graph G is recursively divided into smaller and smaller parts, where each part is associated with a subproblem. Note that the divide step cut is a tool of this approach, and is not intended to be a solution to the subproblem.

In our context, the graph problem is minimum bisection, and we apply this divide and conquer approach for the more general problem of cutting away an arbitrary number of vertices that is given as part of the input (bisection is the special case where the given number is $n/2$). Similarly, the subproblem of each part requires to cut away (from that part) an arbitrary number of vertices that is given in the subproblem. Note that minimum bisection is a cut problem, and therefore in addition to the *divide step cuts* we have here also *solution cuts* (later called *combined cuts*). Note that the solution cut of a part need not be the same as the divide step cut of this part.

Our three stage algorithm outlined above follows this divide and conquer approach. The task of breaking the input graph into smaller and smaller parts is performed by the decomposition stage, whose decomposition tree T represents the recursive structure of the divide steps.

For such a divide and conquer approach to be successful, it is desirable that (i) each of the two subproblems can be solved separately; and (ii) the solutions of the two subproblems can be combined while incurring a relatively small additional cost. Below we provide an overview of how our algorithms handles these issues.

Consider the problem of cutting away k vertices from a part $U \subseteq V$ of the input graph. The corresponding divide step uses a cut (U_1, U_2) of U to break this problem into the two subproblems of cutting away k_1 vertices from U_1 and of cutting away k_2 vertices from U_2, with $k = k_1 + k_2$. (For the sake of exposition assume that k_1, k_2 can be guessed.) Let us assume that the subproblem associated with each subpart U_i is solved separately (by recursion) and the solution obtained for it is a cut (C_i, F_i) with $|C_i| = k_i$ (see also Fig. 2.1). The two solution cuts are then combined into a cut of U that separates $k = k_1 + k_2$ vertices, namely $(C_1 \cup C_2, F_1 \cup F_2)$. Let $Cut(U', k')$ denote the cost of the cut of U' that separates k' vertices and is found by the algorithm. Then the cost of the combined cut is given by

$$Cut(U,k) = Cut(U_1,k_1) + Cut(U_2,k_2) + e(C_1,F_2) + e(C_2,F_1). \tag{2.1}$$

Previous accounting method

The approach of [FKN00] is based on a straightforward upper bound on the cost (2.1) of the combined cut. The additional cost incurred by the divide step, i.e. $e(C_1, F_2) + e(C_2, F_1)$, is at most the cost of all the edges cut by the divide step, i.e. $e(U_1, U_2)$, yielding the upper bound

$$Cut(U,k) \leq Cut(U_1,k_1) + Cut(U_2,k_2) + e(U_1,U_2). \tag{2.2}$$

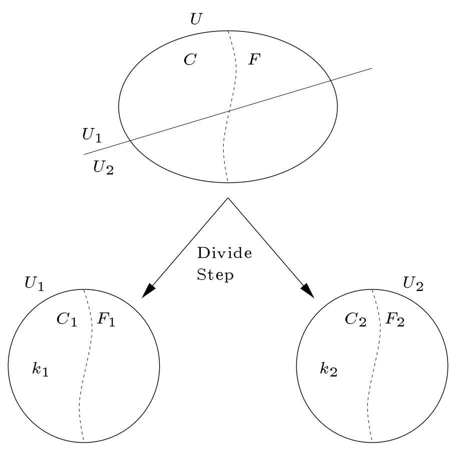

Figure 2.1: The divide and conquer paradigm

We remark that a bound similar to (2.2) is used in divide and conquer algorithms for many other graph problems, such as minimum cut linear arrangement (a.k.a. cutwidth), see e.g. [LR99].

The divide steps of [FKN00] use an approximate min-ratio cut to break each part U. This cut appears to be suitable for the bound (2.2) because it minimizes the cost of the cut (U_1, U_2), and at the same time tries to cut the part U into parts of roughly equal size, so as to minimize the depth of the recursion.

It is particularly instructive to evaluate the quality of our upper bound in the case where the computed cut $(C_1 \cup C_2, F_1 \cup F_2)$ is just the cut induced on U by the optimal bisection (W^*, B^*). Intuitively, we analyze the case where the algorithm happens to find the optimal bisection. In fact, we will later use dynamic programming to find a bisection for which the upper bound is minimized, so such an analysis bounds from above the cost of the output bisection.

There are cases where the upper bound (2.2) is tight (i.e. holds with equality). Indeed, the cuts within each U_i are computed independently of each other, and so it might happen that all the edges between the two parts U_1, U_2 end up in the combined cut. However, this bound is insensitive to cases where only few of the edges that are cut in the divide step end up in the combined cut, leading to a relatively poor approximation ratio.

New accounting method

We introduce a more sophisticated way of bounding the cost of the combined cut. Since $F_1 \subseteq U_1$ and $F_2 \subseteq U_2$ we can bound the cost of the combined cut by

$$Cut(U, k) \leq Cut(U_1, k_1) + Cut(U_2, k_2) + e(C_1, U_2) + e(C_2, U_1). \tag{2.3}$$

Unlike the actual cost (2.1), the upper bound (2.3) can be used in a divide and conquer approach, as follows. Let us call $e(C_1, U_2) + e(C_2, U_1)$ the *charge of the divide step* of U. This charge can be distributed into a charge $e(C_1, U_2)$ of the part U_1, and a charge $e(C_2, U_1)$ of the part U_2. The charge of a part U_i consists of the edges going from C_i to the other part U_{3-i}, and thus depends on the cut (C_i, F_i) chosen in the part U_i, but not on the cut chosen in the other part U_{3-i}. We obtain two separate subproblems (as in each part U_i we want to find a cut (C_i, F_i) for which sum of the cost of this cut and the charge to this part is minimal), enabling a recursive divide and conquer approach. In contrast, the terms $e(C_1, F_2)$ and $e(C_2, F_1)$ of the actual cost of the combined cut depend on the cuts chosen in both parts, and do not allow to break the problem into two separate subproblems.

The new accounting method makes a distinction between the two sides C and F of the combined cut. Unlike e.g. in (2.2), these two sides have different roles in the upper bound (2.3), and we will choose in a certain way which side is referred to as C (and which as F). Since we wish to minimize the charge, it makes sense to choose the smaller of the two sides to be C. In our analysis we have a somewhat relaxed condition, requiring that $|C| \leq \alpha|U|$, for a fixed $1/2 < \alpha < 1$. The task of identifying a side C as required above in each divide step (i.e. each node of the decomposition tree) is performed by the labeling stage, as explained in Section 2.4.2.

The *charge of a bisection* is the upper bound that is obtained by applying the upper bound (2.3) recursively, i.e. it is the sum of the charges of all the divide steps. In Section 2.4.3 we discuss this notion in more detail, and in Section 2.4.4 we show that its current formulation is equivalent to the one from Stage 3 of the algorithm outline (where the identification of a side C at each divide step corresponds to labeling of the decomposition tree T). ¿From the current formulation it is straightforward that the charge of a bisection is always an upper bound on its cost (regardless of the identification of C at each divide step, i.e. the tree labeling).

We call the vertices of $C = C_1 \cup C_2$ *charged* and the vertices of $F = F_1 \cup F_2$ *free*. The edges in the part U can then be classified as charged-charged, charged-free or free-free, according to their two endpoints.

Desired divide step

Rather than find a bisection of minimum cost, our approximation algorithm looks for a bisection of minimum charge. Our desired divide step is therefore one that guarantees that for the fixed optimal bisection, charge can be used to approximate cost. By the labeling stage, it suffices to refer here to charge with respect to an opt-consistent labeling, so from now on we assume that $|C| \leq \alpha|U|$ at each divide step.

Consider the charge of the fixed optimal bisection, and recall that it is the sum of the charges of all the divide steps. The charge of a divide step of a part U is $e(C_1, U_2) + e(C_2, U_1)$ and can be written also as $e(C_1, F_2) + e(C_2, F_1) + 2e(C_1, C_2)$, i.e. the cost of the charged-free edges that the divide step cuts and twice the cost of the charged-charged edges that it cuts. Observe that a charged-free edge is always an edge of the fixed optimal bisection (and vice versa) and that each edge is cut exactly once in the decomposition stage. Hence, all the charged-free edges cut in all the divide steps are exactly all the edges of the fixed optimal bisection. So for the fixed optimal bisection, the difference between charge and cost is twice the cost of all the charged-charged edges cut in all the divide steps.

It is therefore desired that the divide step cuts relatively few charged-charged edges, where relative here is with respect to b, the cost of the fixed optimal bisection. Since b

is the total cost of the charged-free edges that are cut in all the divide steps, we seek an *amortization scheme* that amortizes the total cost of all charged-charged edges cut against the total cost of all charged-free edges cut. The partition of vertices to charged and free is not known to the divide step, and we therefore require that the amortization scheme holds for every possible partition of vertices to charged and free.

A simple amortization scheme can consider each divide step separately and amortize the cost of the charged-charged edges cut in a divide step against the cost of the charged-free edges cut in the same divide step. Suppose that in every divide step the amortized cost in this method is at most ρ, i.e. at every part U we have that $e(C_1, C_2) \leq \rho[e(C_1, F_2) + e(C_2, F_1)]$. Then the total cost of charged-charged edges cut in all divide steps is clearly at most ρb, and the charge of the fixed optimal bisection is at most $(1 + 2\rho)b$.

The problem with this simple amortization scheme is that in order to guarantee that the scheme holds for all possible partitions of vertices to charged and free, ρ might be required to be at least n, a value that is too high for our intended application. For example, consider a graph that consists of two cliques of size $n/2$ connected by an edge e. If the divide step breaks any of the cliques, then letting this clique be C and the other clique be F, the amortization cost will be at least n. Otherwise, the divide step consists of the edge e and then letting C consist of the two endpoints of e, the amortization cost will be infinite.

We employ a more complicated amortization scheme that allows a small amortization cost ρ but introduces an additional logarithmic factor. The reason for the logarithmic factor is that this scheme amortizes against the same edge more than once (but, in a sense, not too many times). Another complication is that this scheme actually has two amortization methods, and it uses at each divide step the one that is better (for that divide step).

Amortized cut

We amortize the cost of the charged-charged edges cut in a divide step against the cost of the charged-free edges *in the part being divided*, i.e. in the divide step of a part U we amortize $e(C_1, C_2)$ against $e(C, F)$. The edges that we amortize against are not cut in this divide step, and hence an edge may receive an amortized cost in many divide steps. However, our amortization scheme described below will guarantee that the total cost amortized against a single edge is at most $O(\rho \cdot \log n)$, for a suitable ρ. Since the edges that we amortize against are charged-free edges and hence edges of the fixed optimal bisection, it would follow that the total cost of the charged-charged edges cut in all the divide steps is at most $O(\rho \log n) \cdot b$, and so the charge of the fixed optimal bisection is $(1 + O(\rho \log n)) \cdot b$.

For motivation, consider the case where the divide steps recursion has depth $O(\log n)$, e.g. when all the divide steps are roughly balanced. In this case, an edge can receive an amortized cost in at most $O(\log n)$ divide steps. Suppose that in every divide step the amortized cost is at most ρ, i.e. in every part U we have that $e(C_1, C_2) \leq \rho \cdot e(C, F)$. Then the total cost amortized against a single edge is at most $O(\rho \log n)$.

We do not require that the divide steps are balanced, but rather scale the amortization cost at a part U according to the imbalance of its divide step. Out of the several possible scaling factors we will use only the following two, where we assume, without loss of generality, that $|U_1| \leq |U_2|$. The first scaling factor is $e(C_1, F_1)/e(C, F)$, and its corresponding amortization method requires that

$$e(C_1, C_2) \le \rho \cdot \frac{e(C_1, F_1)}{e(C, F)} \cdot e(C, F). \tag{2.4}$$

The second scaling factor is $|C_1|/|C|$, and its corresponding amortization method requires that

$$e(C_1, C_2) \le \rho \cdot \frac{|C_1|}{|C|} \cdot e(C, F). \tag{2.5}$$

Alternative formulations. The first amortization method (2.4) can be written also as $e(C_1, C_2) \le \rho \cdot e(C_1, F_1)$. A convenient interpretation of this formulation is that we amortize against the charged-free edges inside U_1, the smaller side of the divide step cut (rather than inside U, the part being divided), and the amortized cost is required to be at most ρ.

The second amortization method (2.5) can be written also as $e(C_1, C_2) \le \rho \cdot r'(C) \cdot |C_1|$ where $r'(C) = e(C, F)/|C|$ (see Section 2.1.3 for the difference between $r'(C)$ and $r(C)$). A convenient interpretation of this formulation is that we amortize against the vertices in C_1, the charged vertices inside the smaller side of the divide step cut, and the amortized cost is required to be at most $\rho \cdot r'(C)$.

Total amortized cost. The total cost amortized in the first method (2.4) is at most $O(\rho \log n) \cdot b$. Indeed, let us use the alternative formulation in which the amortization is only against edges inside U_1, the smaller side of the divide step cut. An edge can be inside U_1 in at most $\log n$ divide steps (since the size of the part it is contained in reduces at each such divide step by a factor of 2). Hence the total cost amortized in this method against a single edge (of the fixed optimal bisection) is at most $O(\rho \log n)$, and the claim follows.

The total cost amortized in the second method (2.5) is also at most $O(\rho \log n) \cdot b$. Indeed, we show in Section 2.4.3 that the total cost amortized in this method against a single edge (of the fixed optimal bisection) is at most $O(\rho \log n)$ (essentially by careful summation of the relevant terms of the form $|C_1|/|C|$), and the claim follows.

Our amortization scheme. Our amortization scheme chooses at each divide step the scaling factor that is better for this divide step, and so it suffices to have that at each part U at least one of (2.4) and (2.5) holds. It follows from the above discussion (see Section 2.4.3 for a full proof) that the total cost amortized in both methods together is at most $O(\rho \log n) \cdot b$.

We can now formally define our desired divide step according to the (alternative formulations of) the two amortization methods described above. We call this cut an amortized cut.

Definition (amortized cut). Let (U_1, U_2) be a cut with $|U_1| \le |U_2|$ in a graph $G'(U, E')$, and let $U = C \cup F$ be a partition of the graph vertices U to charged vertices C and free vertices F. Let us denote $C_i = U_i \cap C$ and $F_i = U_i \cap C$ for $i = 1, 2$, as in Fig. 2.1. Let

$$\rho_e = \frac{e(C_1, C_2)}{e(C_1, F_1)} \qquad \text{and} \qquad \rho_v = \frac{e(C_1, C_2)}{|C_1| \cdot r'(C)} \tag{2.6}$$

where $r'(C) = e(C,F)/|C|$. We call ρ_e the *amortized cost for the edges*, and ρ_v the *amortized cost for the vertices* (note that ρ_e, ρ_v depend on C, F).

The *amortized cost of the cut* (U_1, U_2) is the maximum of $\min\{\rho_e, \rho_v\}$, where the maximum is taken over all partitions $U = C \cup F$ with $0 < |C| \le \alpha|U|$ for a fixed $\frac{1}{2} \le \alpha < 1$. We say that the cut (U_1, U_2) is *ρ-amortized* if its amortized cost is at most ρ.

In order us to correctly handle cases where there is no cost to amortize against, we use the convention that $\frac{0}{0}$ is defined to be 0, and that $\frac{t}{0}$ for $t > 0$ is defined to be ∞. In particular, we may extend (2.6) to the case where $C = \emptyset$ and then ρ_e, ρ_v are defined to be 0.

Convenient characterizations. A convenient characterization of an amortized cut is given in the following proposition, whose proof is straightforward. (We will use this characterization in Section 2.4.)

Proposition 2.3. *A cut (U_1, U_2) with $|U_1| \le |U_2|$ is ρ-amortized if and only if for every $C \subset U$ with $|C| \le \alpha|U|$ and $F = U \setminus C$,*

$$e(C_1, C_2) \le \rho \cdot \max\left\{e(C_1, F_1)\ ,\ \tfrac{|C_1|}{|C|} \cdot e(C, F)\right\}$$

where $C_i = U_i \cap C$ and $F_i = U_i \cap C$ for $i = 1, 2$.

The restriction $|C| \le \alpha|U|$ implies that the two terms $r(C) = \frac{e(C,F)}{\min\{|C|,|F|\}}$ and $r'(C) = \frac{e(C,F)}{|C|}$ differ by no more than a constant factor. Indeed, $\min\{|C|, |F|\} = \Theta(|C|)$ and hence $r(C) = \frac{e(C,F)}{\min\{|C|,|F|\}} = \frac{e(C,F)}{\Theta(|C|)} = \Theta(r'(C))$.

We can therefore characterize the amortized cost of a cut (up to constant factors) in terms of $r(C)$ rather than $r'(C)$. (We will use this characterization in Section 2.3).

Proposition 2.4. *A cut (U_1, U_2) with $|U_1| \le |U_2|$ is $O(\rho)$-amortized if for every partition $U = C \cup F$ with $0 < |C| \le \alpha|U|$,*

$$\min\left\{\frac{e(C_1, C_2)}{e(C_1, F_1)}\ ,\ \frac{e(C_1, C_2)}{|C_1| \cdot r(C)}\right\} \le \rho \tag{2.7}$$

where $C_i = U_i \cap C$ and $F_i = U_i \cap C$ for $i = 1, 2$.

Remarks. Observe that without the restriction $|C| \le \alpha|U|$, the amortized cost ρ might be required to be $\Omega(|U|)$, a value that is too high for our intended application. For example, consider a clique on n vertices and a cut (U_1, U_2) in it with $|U_1| \le |U_2|$. Let one vertex of U_2 be the only free vertex, and the rest of the vertices be charged. The number of charged-charged edges cut is $|U_1| \cdot \Theta(n)$. There are no charged-free edges in U_1, so the amortized cost for the edges is $\rho_e = \infty$. The number of charged vertices in the smaller side is $|U_1|$ and $r'(C) = \frac{n-1}{n-1} = 1$, so the amortized cost for the vertices is $\rho_v = \frac{|U_1|\Theta(n)}{|U_1|\cdot 1} = \Theta(n)$. Therefore, the amortized cost of any cut would be $\rho = \Omega(n)$.

In contrast, we show that the restriction $|C| \le \alpha|U|$ allows to obtain relatively small values of ρ. Namely, there always exists a cut whose amortized cost is $\rho = O(1)$, and a cut whose amortized cost is $O(\log |U|)$ can be computed efficiently. We remark that our constructions are stronger than those required by Proposition 2.4, as they satisfy (2.7) with no restriction on $|C|$. (The point is that we use $r(C)$ rather than $r'(C)$, which makes a significant difference when $|C| \gg |F|$, as in the above clique example.)

Note that the amortized cost ρ is not an approximation ratio. On the one hand, it is not clear from the definition that every graph has an $O(1)$-amortized cut. On the other hand, the amortized cost of a cut may be smaller than 1, as demonstrated by a graph that consists of two cliques of size $n/2$ connected by an edge. The cut that separates the two cliques can be seen to have amortized cost $O(1/n)$.

2.3 Finding an amortized cut

In this section we devise an algorithm for finding $O(\log n)$-amortized cuts in general graphs, and $O(1)$-amortized cuts in graphs excluding any fixed minor (e.g. planar graphs). The input graph for this algorithm is denoted by G (though it may be just a part of the input graph for bisection). We assume that G is connected, as otherwise we can separate a connected component while cutting no edges at all.

Section 2.3.1 shows that every optimal min-ratio cut is an $O(1)$-amortized cut. It follows that in every graph there exists an $O(1)$-amortized cut. An optimal min-ratio cut is NP-hard to find in general graphs, and we thus consider approximate min-ratio cuts.

Section 2.3.2 demonstrates an approximate min-ratio cut which would be a poor divide step for our accounting method. In particular, its amortized cost is high, showing that the arguments of Section 2.3.1 do not immediately extend from optimal min-ratio cuts to approximate ones.

Section 2.3.3 presents an algorithm that uses a τ-approximate min-ratio cut in order to find an $O(\tau)$-amortized cut. Known algorithms for the min-ratio cut problem in general graphs [LR99, AR98, LLR95] have approximation ratio $\tau = O(\log n)$, and we can thus find an $O(\log n)$-amortized cut. For certain graph families a better approximation ratio is possible. For example, in graphs excluding any fixed minor, a ratio of $\tau = O(1)$ is known due to [KPR93], and we can thus find an $O(1)$-amortized cut.

2.3.1 Min-ratio cuts are $O(1)$-amortized

We give an $O(1)$ upper bound on the amortized cost of optimal min-ratio cuts. The proof is based on the characterization given in Proposition 2.4 for an amortized cut. We remark that our proof satisfies (2.7) with no restriction on $|C|$.

Lemma 2.5. *An optimal min-ratio cut in a graph is $O(1)$-amortized.*

Proof. Let (V_1, V_2) be an optimal min-ratio cut in a graph G, and assume, without loss of generality, that $|V_1| \leq |V_2|$. Let $V = C \cup F$ be an arbitrary partition of the graph vertices to charged vertices C and free vertices F, with $0 < |C| < |V|$, and denote $C_i = V_i \cap C$ and $F_i = V_i \cap F$ for $i = 1, 2$ (see also Fig. 2.2). We show below that

$$\min\left\{\frac{e(C_1, C_2)}{e(C_1, F_1)}\ ,\ \frac{e(C_1, C_2)}{|C_1| \cdot r(C)}\right\} \leq 2, \tag{2.8}$$

and then by Proposition 2.4 we will have that (V_1, V_2) is $O(1)$-amortized, which proves the lemma. Note that we can assume that $|C_1| > 0$, as otherwise there is nothing to prove.

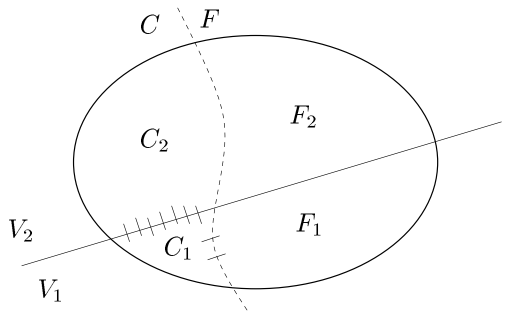

Figure 2.2: The amortized cost of an optimal min-ratio cut (V_1, V_2)

One easy case is when $\frac{e(C_1,C_2)}{e(C_1,F_1)}$ (i.e. the amortized cost for the edges ρ_e) is at most 2, which clearly implies (2.8).

Another easy case is when $\frac{e(C_1,C_2)}{|C_1|} \leq 2r(V_1)$. Since (V_1, V_2) is an optimal min-ratio cut, we also have that $r(V_1) \leq r(C)$. We obtain that $\frac{e(C_1,C_2)}{|C_1| \cdot r(C)} \leq 2\frac{r(V_1)}{r(C)} \leq 2$, and therefore (2.8) holds.

We next prove that one of the two easy cases above must hold, as otherwise we must have that $r(F_1) < r(V_1)$, in contradiction with (V_1, V_2) being an optimal min-ratio cut. Indeed, assume that $e(C_1, C_2)/e(C_1, F_1) > 2$ and $\frac{e(C_1,C_2)}{|C_1|} > 2r(V_1)$. Since $r(V_1) = \frac{e(V_1,V_2)}{|V_1|}$ is the average degree from V_1 to V_2, it can be represented as the following convex combination of the average degree from C_1 to V_2 and the average degree from F_1 to V_2, namely

$$r(V_1) = \frac{|F_1|}{|V_1|} \cdot \frac{e(F_1, V_2)}{|F_1|} + \frac{|C_1|}{|V_1|} \cdot \frac{e(C_1, V_2)}{|C_1|}.$$

Since $r(F_1) = \frac{e(F_1,V_2)+e(F_1,C_1)}{|F_1|}$ (note that $|F_1| \leq |V_1| \leq \frac{1}{2}|V|$), we can represent $r(V_1)$ also as

$$r(V_1) = \frac{|F_1|}{|V_1|} \cdot r(F_1) + \frac{|C_1|}{|V_1|} \cdot \left[\frac{e(C_1, V_2) - e(F_1, C_1)}{|C_1|}\right].$$

By the above two assumptions (that exclude the easy cases) we have that

$$\frac{e(C_1, V_2) - e(F_1, C_1)}{|C_1|} \geq \frac{e(C_1, C_2) - e(F_1, C_1)}{|C_1|} \geq \frac{\frac{1}{2}e(C_1, C_2)}{|C_1|} > r(V_1).$$

The last two inequalities imply that

$$r(V_1) > \frac{|F_1|}{|V_1|} \cdot r(F_1) + \frac{|C_1|}{|V_1|} \cdot r(V_1).$$

We obtained that some convex combination of $r(F_1)$ and $r(V_1)$ is smaller than $r(V_1)$, and we can therefore conclude that $r(F_1) < r(V_1)$. This contradicts the fact that (V_1, V_2) is an optimal min-ratio cut, and completes the proof of Lemma 2.5. ☐

The converse of Lemma 2.5 is not true, and an $O(1)$-amortized cut can be an $\Omega(n)$-approximate min-ratio cut, as follows from the next proposition with $t = O(1)$.

Proposition 2.6. *Fix a constant $1/2 < \alpha < 1$ for the definition of an amortized cut. Then for every $t = o(n)$, there is an $O(1/t)$-amortized cut which is an $\Omega(n/t)$-approximate min-ratio cut.*

Proof. Consider the a graph on n vertices, for a sufficiently large n, that consists of three cliques as follows. V_1 is a clique on t vertices, V_2 is a clique on αn vertices, and V_3 is a clique on the remaining $\Omega(n)$ vertices. In addition, the graph contains one edge connecting V_1 to V_2, and one edge connecting V_2 to V_3.

The cut $(V_1, V_2 \cup V_3)$ has amortized cost $O(1/t)$. Indeed, let $C \cup F$ be a partition of the vertices with $|C| \leq \alpha n$. We may assume that C contains both endpoints of the edge between V_1 and V_2, as otherwise the cut contains no charged-charged edges and its amortized cost is 0. So we have that the cost of the charged-charged edges cut is 1, and that both V_1 and V_2 contain at least one charged vertex. If V_1 contains also at least one free vertex, then the number of charged-free edges in V_1 is at least $t-1$ and hence $\rho_e = \frac{e(C_1,C_2)}{e(C_1,F_1)} \leq 1/(t-1)$. Otherwise, we have $C_1 = V_1$; since there are at most αn charged vertices, and at least one of them is in V_1, we have that V_2 contains also free vertices and thus $e(C,F) \geq \Omega(n)$; it follows that $\rho_v = \frac{e(C_1,C_2)}{e(C,F)} \cdot \frac{|C|}{|C_1|} \leq O(1/t)$.

The cut $(V_1, V_2 \cup V_3)$ is an $\Omega(n/t)$-approximate min-ratio cut. Indeed, the ratio of this cut is $r(V_1) = 1/t$, while the cut $(V_3, V_1 \cup V_2)$ is an optimal min-ratio cut and has ratio $r(V_3) = O(1/n)$. □

The next corollary follows from Lemma 2.5.

Corollary 2.7. *In every graph there exists an $O(1)$-amortized cut.*

Corollary 2.7 is optimal up to constant factors, and there are graphs for which any cut has amortized cost $\Omega(1)$. For example, consider a clique on n vertices. Given a cut (V_1, V_2) with $|V_1| \leq |V_2|$, let α be the constant in the amortized cut definition, and take $(\alpha - 1/2)n$ vertices of V_2 and all of V_1 to be the charged vertices. It can be seen that $\rho_e = \infty$ and $\rho_v = \Theta(1)$, and so the amortized cost of the cut (V_1, V_2) is $\Omega(1)$, as claimed.

2.3.2 Approximate min-ratio cuts might be poor amortized cuts

We demonstrate that an approximate min-ratio cut of a graph might be a poor divide step, and in particular a poor amortized cut. Consider, for example, the following graph G on $2n + 2\sqrt{\epsilon n}$ vertices for a fixed $0 < \epsilon < 1$ (see also Fig. 2.3). The vertex set of the graph is $F_1 \cup F_2 \cup C_1 \cup C_2$ where each of F_1, F_2 are of size n, each of C_1, C_2 are of size $\sqrt{\epsilon n}$, and each of the four subsets forms a clique. These four cliques are connected as follows. Between F_1 and F_2 there are n edges that form a matching (i.e. have no common endpoint). Between C_1 and C_2 there are all possible ϵn edges, thus $C_1 \cup C_2$ forms a clique. There are also $2\sqrt{\epsilon n}$ edges between F_i and C_i (for $i = 1, 2$) so that their endpoints at F_i are distinct and each vertex of C_i is an endpoint of exactly two of these edges.

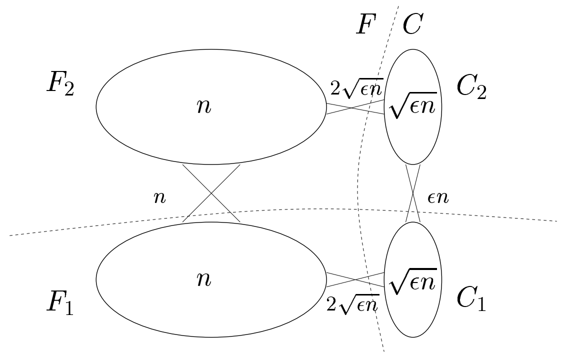

Figure 2.3: A poor divide step by an approximate min-ratio cut

Let $C = C_1 \cup C_2$ be the charged vertices, and $F = F_1 \cup F_2$ the free vertices. Such a partition to charged and free may reflect the "right" cut of $2\sqrt{\epsilon n}$ vertices from the graph G (if, e.g., the input graph for bisection consists of this graph G and a clique on $2n - 2\sqrt{\epsilon n}$ vertices).

Consider a divide step based on the cut $(F_1 \cup C_1, F_2 \cup C_2)$, whose ratio is nearly optimal. Indeed, an optimal min-ratio cut in this graph is $(F_1, C_1 \cup F_2 \cup C_2)$ and its ratio is $1+2\sqrt{\epsilon}/\sqrt{n}$. The cut $(F_1 \cup C_1, F_2 \cup C_2)$ has a slightly higher ratio of $(1+\epsilon)(1-o(1))$, and so it is a $1+\epsilon$ approximate min-ratio cut.

Observe that the cut $(F_1 \cup C_1, F_2 \cup C_2)$ is a poor divide step. It cuts ϵn charged-charged edges while the total number of charged-free edges in G (and the bisection cost in the input graph) is only $4\sqrt{\epsilon n}$. According to the new accounting method, such a divide step does not give an approximation ratio better than $\Omega(\sqrt{\epsilon n})$.

The observation that the cut $(F_1 \cup C_1, F_2 \cup C_2)$ is a poor divide step is supported by its high amortized cost. The amortized cost for the edges is $\rho_e = \epsilon n / 2\sqrt{\epsilon n} = \sqrt{\epsilon n}/2$. The ratio of the cut (C, F) is $r(C) = r'(C) = 2$, so the amortized cost for the vertices is $\rho_v = \epsilon n / (\sqrt{\epsilon n} r'(C)) = \sqrt{\epsilon n}/2$. We conclude that a $1 + o(1)$ approximate min-ratio cut might have amortized cost $\rho \geq \min\{\rho_e, \rho_v\} = \sqrt{\epsilon n}/2$.

2.3.3 Finding $O(\tau)$-amortized cut

We present an algorithm that finds an $O(\tau)$-amortized cut, given a subroutine for computing a τ-approximate min-ratio cut. The algorithm is motivated by the $O(1)$ upper bound on the amortized cost of a min-ratio cut shown in Section 2.3.1. In particular, we examine what additional properties are required in order to extend the analysis of Lemma 2.5 from optimal min-ratio cuts to approximate ones.

The proof of Lemma 2.5 uses *twice* the fact that (V_1, V_2) is an optimal min-ratio cut. In the first usage we had that $\frac{e(C_1,C_2)}{|C_1|\cdot r(C)} \leq 2\frac{r(V_1)}{r(C)} \leq 2$, which extends to the case where (V_1, V_2) is an approximate min-ratio cut with the approximation ratio carried over to the amortized cost, i.e. if (V_1, V_2) is a τ-approximate min-ratio cut then we have $\frac{e(C_1,C_2)}{|C_1|\cdot r(C)} \leq 2\frac{r(V_1)}{r(C)} \leq 2\tau$.

The second time we used the fact that (V_1, V_2) is an optimal min-ratio cut was to say that $r(F_1) < r(V_1)$ cannot hold and gives a contradiction. In general, this usage does not extend to an approximate min-ratio cut, as demonstrated by the example in Section 2.3.2. However, the proof does extend to an approximate min-ratio cut if we have the additional property that the ratio of V_1 is minimal over all its subsets F_1, i.e. $r(V_1) \leq r(F_1)$ for all $F_1 \subset V_1$. We therefore obtain that the proof of Lemma 2.5 extends to approximate min-ratio cuts as follows.

Lemma 2.8. *Let (V_1, V_2) be a τ-approximate min-ratio cut in a graph, with $|V_1| \leq |V_2|$. If $r(V_1) \leq r(F_1)$ for every $F_1 \subset V_1$ then (V_1, V_2) is an $O(\tau)$-amortized cut.*

Note that the proof of Lemma 2.8 is not symmetric with respect to the two amortization methods. It guarantees that either $e(C_1, C_2)/e(C_1, F_1) \leq 2$ (i.e. the amortized cost for the edges ρ_e is at most 2), or $\frac{e(C_1,C_2)}{|C_1| \cdot r(C)} \leq 2\tau$ (i.e. the amortized cost for the vertices ρ_v is $O(\tau)$). In contrast, in the proof of Lemma 2.5 for optimal min-ratio both amortization costs are $O(1)$.

The amortized cut algorithm. We use Lemma 2.8 to devise an algorithm that finds an $O(\tau)$-amortized cut based on a τ-approximate min-ratio cut. The algorithm, described in Fig. 2.4, starts with a τ-approximate min-ratio cut (V_1, V_2) and then "fixes" it so that it would also be "minimal" with respect to containment, as required by Lemma 2.8. It then follows that the output cut is $O(\tau)$-amortized.

In order to "fix" the cut (V_1, V_2), the algorithm uses minimum (s,t)-cuts in a related graph G', which is defined in step 2. The related graph G' contains edges of the input graph G, as well as new edges. The edges from G have unit capacity, while the capacity of the new edges is some parameter $p > 0$. Step 3 then finds the optimal value of p with respect to the minimum (s,t)-cut. Before discussing implementation issues of step 3, let us analyze the algorithm correctness.

Lemma 2.9. *The cut $(S, V \setminus S)$ output by algorithm* FindAmortized *is a τ-approximate min-ratio cut. In addition, every nonempty subset of V_1 has ratio at least as large as S, i.e. $r(S) = \min\{r(S') : \emptyset \neq S' \subseteq V_1\}$.*

Proof. Consider an arbitrary value p and an arbitrary (s,t)-cut in the related graph G' with the corresponding set $S \subset V_1$ (see Fig. 2.5). The cut consists of (i) edges between s and $V_1 \setminus S$ (each of capacity p) (ii) edges between S and $V_1 \setminus S$ (these are edges from the input graph G) and (iii) edges between S and t (these are the edges between S and V_2 in the input graph G). The capacity of this (s,t)-cut is thus

$$\text{cap}(S) = p \cdot |V_1 \setminus S| + e(S, V \setminus S)$$

where, as usual, $e(\cdot,\cdot)$ denotes the number of corresponding edges in the input graph G. In the special case of the empty set $S = \emptyset$, the capacity of the (s,t)-cut is

$$\text{cap}(\emptyset) = p \cdot |V_1|$$

Fixing the value of p, let us compare the capacity of the cut defined by the empty set $\emptyset$ with that of an arbitrary set $S \neq \emptyset$, i.e. $\text{cap}(\emptyset)$ vs. $\text{cap}(S)$. The empty set $\emptyset$ yields a smaller

```
Algorithm FindAmortized.

  1. Find in the input graph $G = (V, E)$ a $\tau$ approximate min-
     ratio cut $(V_1, V_2)$ with $|V_1| \leq |V_2|$.

  2. Create a related graph $G'$:

     – Merge all vertices of $V_2$ into a single vertex $t$, remov-
     ing self loops at $t$, and keeping all edges to $V_1$, including
     parallel edges.

     – Add a new vertex $s$ which is connected to each vertex
     of $V_1$ by an edge whose capacity (weight) is a parameter
     $p > 0$.

  3. Let $S$ denote the vertices of $V_1$ which are on the same side
     with $s$ in a minimum $(s, t)$-cut of $G'$.

     – Find (e.g. by binary search) the minimum $p > 0$ for
     which $S \neq \emptyset$. (Possibly, $S = V_1$).

  4. Output the cut $(S, V \setminus S)$ of the input graph.
```

Figure 2.4: Algorithm for amortized cuts

capacity whenever

$$\begin{aligned} p \cdot |V_1| &< p \cdot |V_1 \setminus S| + e(S, V \setminus S) \\ &\Updownarrow \\ p &< \frac{e(S, V \setminus S)}{|S|} = r(S) \end{aligned}$$

where $r(S)$ is the ratio of the cut $(S, V \setminus S)$ in the input graph G (note that $|S| \leq |V_1| \leq \frac{1}{2}|V|$ and that $r(S) > 0$ if G is connected).

We claim that the value of p found at step 3 is essentially $p^* = \min\{r(S) : \emptyset \neq S \subseteq V_1\}$. Indeed, when $p < p^*$, a minimum (s, t)-cut in G' corresponds to $S = \emptyset$, and when $p > p^*$, a minimum (s, t)-cut yields a set $S \neq \emptyset$. When $p = p^*$, a minimum (s, t)-cut can be obtained either by $S = \emptyset$, or by (one or more) $S \neq \emptyset$ with $r(S) = p^*$.

When $p = p^* + \epsilon$ for a very small $\epsilon > 0$, only the sets $S \neq \emptyset$ with $r(S) = p^*$ give smaller capacity than the empty set, and thus a minimum (s, t)-cut is obtained by one of these sets S. By the definition of p^*, this set $\emptyset \neq S \subset V_1$ has minimal ratio $r(S)$ over all nonempty subsets of V_1, i.e. $r(S) = \min\{r(S') : \emptyset \neq S' \subseteq V_1\}$, as claimed. Furthermore, since $S = V_1$ is included in this range, we get that $r(S) \leq r(V_1)$ and hence $(S, V \setminus S)$ is a τ-approximate min-ratio cut, finishing the proof. We remark that a slightly modified algorithm can guarantee in addition that $r(S) < r(S')$ for every $S' \subset S$ with $S' \neq \emptyset, S$. Details omitted. $\square$

Theorem 2.4. *Given a subroutine for computing a τ-approximate min-ratio cut, algorithm* FindAmortized *finds an $O(\tau)$-amortized cut.*

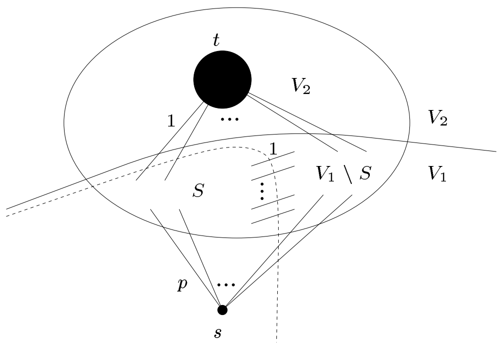

Figure 2.5: An (s, t)-cut in the related graph G'

Proof. Lemma 2.9 guarantees that the cut found by the algorithm satisfies the requirements of Lemma 2.8, from which it follows that the cut is $O(\tau)$-amortized. □

We now address the issue of implementing step 3. Observe that p^* is the maximum value p for which the empty set $\emptyset$ gives a minimum (s, t)-cut. Since, by definition, p^* is the ratio $r(S)$ of a set S, it has only n^3 possible values, which can be exhaustively searched. Alternatively, p^* can be found in $O(\log n)$ iterations of binary search, since as an exact multiple of $1/|S|$ it is bounded between 0 and n, and the difference between any two of its possible values is more than $1/n^2$.

Once we find p^*, we need to find a set $S \neq \emptyset$ that gives a minimum (s, t)-cut for p^*. We can either guess a vertex of V_1 and merge it with s before computing the minimum (s, t)-cut for p^*, or alternatively compute a minimum (s, t)-cut for $p = p^* + \epsilon$ with e.g. $\epsilon = 1/n^2$.

2.4 The bisection algorithm

In this section we describe our approximation algorithm for bisection and prove the following theorem. (See Section 2.2 for the definition of an amortized cut.)

Theorem 2.5. *Given a subroutine that finds a ρ-amortized cut, a bisection within ratio of $1 + O(\rho \log n)$ of the minimum can be found in polynomial time.*

2.4.1 Decomposition stage

The decomposition stage recursively divides the input graph $G = (V, E)$ into smaller and smaller parts using a ρ-amortized cut subroutine (e.g. the one devised in Section 2.3). Each part is further divided unless it consists of a single vertex.

The decomposition stage builds a rooted binary tree T, called the *decomposition tree*, which corresponds to the recursive decomposition of the input graph G in a natural way, as follows. (Throughout, we call the vertices of T *nodes*, to avoid confusion with the vertices of the input graph G.) Each tree node i contains a part $V_i \subseteq V$ that was found during the recursive decomposition. The root node of T contains V, i.e. the whole input graph G. Let us denote the two children of a nonleaf node i by $L(i)$ and $R(i)$. Then their two parts

$V_{L(i)}, V_{R(i)}$ are the result of dividing V_i, i.e. the ρ-amortized cut found in V_i is $(V_{L(i)}, V_{R(i)})$. A leaf of the tree T contains a part that consists of a single vertex of G. Therefore T contains exactly n leaves and $n-1$ nonleaf nodes.

2.4.2 Labeling stage

Recall the following definitions from Section 2.2. A *labeling* of the decomposition tree T labels each nonleaf node of the tree as either white or black. Fixing a parameter $1/2 < \alpha < 1$, we say that a labeling is *α-consistent* with respect to a white-black bisection (W, B) of G if every tree node i satisfies that: If the label of node i is white then $|W \cap V_i| \leq \alpha |V_i|$, and if the label of node i is black then $|B \cap V_i| \leq \alpha |V_i|$ (where V_i is the part contained in node i). A labeling is called *opt-consistent* if it is α-consistent with the fixed optimal bisection (W^*, B^*).

The labeling stage produces a family $\mathcal{F}$ of labelings. The cardinality of $\mathcal{F}$ is exponential in n, so rather than listing its members explicitly, the labeling stage produces an implicit representation of $\mathcal{F}$. The actual work of the labeling stage is to *mark* certain nodes of T, and these nodes implicitly define the family $\mathcal{F}$, as described below.

The labeling stage marks some of the nodes of T in a process that goes from the root of T towards its leaves, as follows. The root of T is always marked, and any other node i in the tree is marked in this process if its closest marked ancestor j satisfies $|V_i| \leq \frac{1}{2\alpha}|V_j|$ (as before, V_i and V_j are the parts contained in the nodes i and j, respectively). Note that the constant α is chosen so that $\frac{1}{2} < \alpha < 1$, implying $\frac{1}{2} < \frac{1}{2\alpha} < 1$.

A labeling of T is said to be *derived* from the marked nodes, if the label of every unmarked node is the same as the label of its closest marked ancestor (there is no restriction on the labels of the marked nodes). Note that in this case the labels of the marked nodes uniquely define the labels of all the internal tree nodes.

The family $\mathcal{F}$ produced by the labeling stage consists of all the labelings that can be derived from the marked nodes. Since each of the $\Omega(n)$ marked nodes can be labeled arbitrarily by one of two colors, the resulting family of labelings has exponentially large cardinality, and we cannot explicitly list all the family members. Instead, the algorithm implicitly represents this family $\mathcal{F}$ by identifying which are the marked nodes.

Lemma 2.10. *The family of labelings $\mathcal{F}$ contains at least one opt-consistent labeling.*

Proof. Let the white-black cut (W, B) be the fixed optimal bisection. Consider the labeling that is derived from the marked nodes, with the label of each marked node i being the color in minority among the vertices of V_i.

This labeling is clearly in the family $\mathcal{F}$, and we claim that it is also opt-consistent. Indeed, the label of a marked node i is by definition the minority color in V_i. The label of an unmarked node i is the same as the label of its closest marked ancestor j. Suppose, without loss of generality, that this label (of i and j) is white. Then at most half the vertices of V_j are white, i.e. $|W \cap V_j| \leq \frac{1}{2}|V_j|$. Observe that $V_i \subset V_j$ and $|V_i| > \frac{1}{2\alpha}|V_j|$ and hence $|W \cap V_i| \leq |W \cap V_j| \leq \frac{1}{2}|V_j| < \alpha |V_i|$. Hence, this labeling of $\mathcal{F}$ is opt-consistent. $\square$

2.4.3 The charge of a bisection

We now formally define the *charge of a bisection* (W, B) with respect to the decomposition tree T and a labeling of it. The reference to T will later be omitted, as we always refer to the tree computed in the decomposition stage.

Definition (charge). Let (W, B) be a bisection of the input graph, and assume we are given a decomposition tree T and a labeling of it. For each (nonleaf) node i of T, if i is labeled white then we let (see Fig. 2.6) $C_i = W \cap V_i$ and $F_i = B \cap V_i$, and if i is labeled black then we let $C_i = B \cap V_i$ and $F_i = W \cap V_i$. We obtain a cut (C_i, F_i) of the part V_i, and say that C_i is *charged* and F_i is *free*. The *charge of the divide step* of a (nonleaf) node i is defined as

$$e(C_i \cap V_{L(i)}, V_{R(i)}) + e(C_i \cap V_{R(i)}, V_{L(i)}).$$

The *charge of the bisection* (W, B) is defined as the sum of all the divide steps charges, i.e.

$$\sum_{i \in T} e(C_i \cap V_{L(i)}, V_{R(i)}) + e(C_i \cap V_{R(i)}, V_{L(i)}).$$

(These charges are defined with respect to T and a labeling of it.)

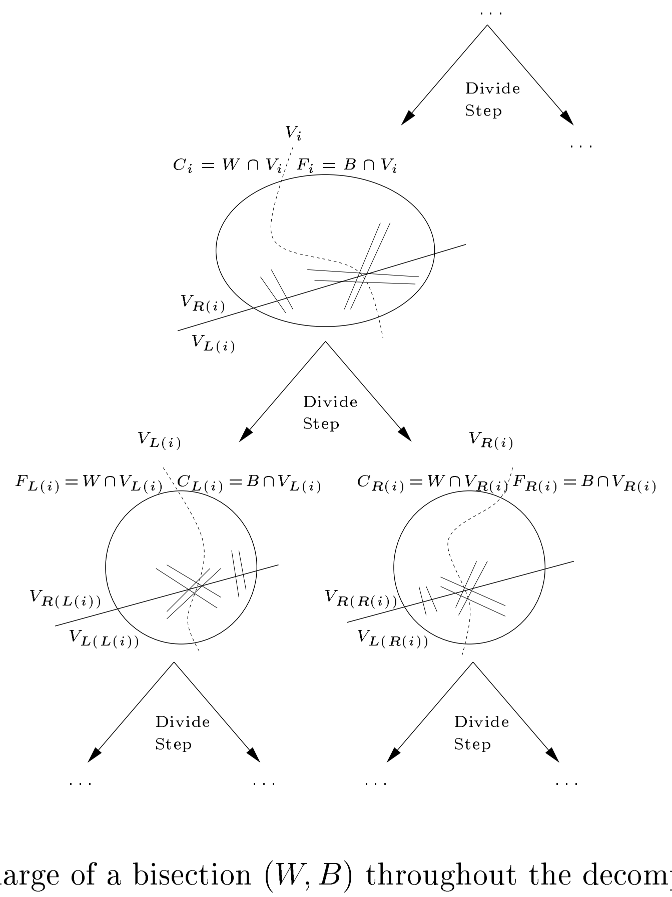

Figure 2.6: The charge of a bisection (W, B) throughout the decomposition tree

Bisection charge vs. cost

In certain conditions, a bisection charge can approximate its cost. As shown below, the charge of a bisection upper bounds its cost, and the gap between them is not too large if the charge is taken with respect to an α-consistent labeling (as in the case of the fixed optimal bisection and an opt-consistent labeling).

Lemma 2.11. *The charge of a bisection (W, B) with respect to any labeling is at least as large as its cost.*

Proof. As we have seen in section 2.2, the true cost of the (W, B) edges cut in a divide step i is $e(C_i \cap V_{L(i)}, F_i \cap V_{R(i)}) + e(C_i \cap V_{R(i)}, F_i \cap V_{L(i)})$, and is therefore not larger than the charge of this step. The proof follows by summing over all divide steps, since the decomposition stage eventually divides the graph into individual vertices, and so every edge of the bisection (W, B) is cut at some divide step. $\square$

Lemma 2.12. *The charge of a bisection (W, B) with respect to a labeling that is α-consistent with it is at most $e(W, B) \cdot (1 + O(\rho \log n))$.*

Proof. Consider a bisection (W, B) and a labeling of T that is α-consistent with it. As we have seen in Section 2.2 and in Lemma 2.11 the charge of a divide step is larger than the true cost of the (W, B) edges cut in that step by the cost of the charged-charged edges cut in that divide step. Summing over the divide steps we get that the charge of (W, B) the fixed optimal bisection is larger than its cost by $2\sum_i e(C_i \cap V_{L(i)}, C_i \cap V_{R(i)})$, where i ranges over all (nonleaf) nodes i in T. We use the shorter notation $C_L = C_i \cap V_{L(i)}$ and $C_R = C_i \cap V_{R(i)}$, where i is clear from the context.

To upper bound $2\sum_i e(C_L, C_R)$, observe that each part V_i is divided using a ρ-amortized cut, and that the α-consistent labeling guarantees that $|C_i| \le \alpha|V_i|$ for all nodes i, so we can use the amortization scheme of Section 2.2. Namely, let us assume, without loss of generality, that the decomposition stage places in the left child of a node i the smaller of the two subparts of V_i, i.e. $|V_{L(i)}| \le |V_{R(i)}|$ for every nonleaf node i. Then by Proposition 2.3 we can upper bound

$$e(C_L, C_R) \le \rho \cdot \max\left\{e(C_L, F_L)\ ,\ \tfrac{|C_L|}{|C_i|} \cdot e(C_i, F_i)\right\},$$

and obtain

$$2\sum_i e(C_L, C_R) \le 2\rho \cdot \left\{\sum_i e(C_L, F_L) + \sum_i \tfrac{|C_L|}{|C_i|} \cdot e(C_i, F_i)\right\}\ . \tag{2.9}$$

Therefore, to complete the proof of Lemma 2.12 it suffices to upper bound the sums in the curly brackets (i.e. the total cost amortized in each of the two methods) by $e(W, B) \cdot O(\log n)$.

Consider first $\sum_i e(C_L, F_L)$. The edges that contribute to this sum are charged-free edges and hence edges of the bisection (W, B). An edge in the cut (C_L, F_L) must be inside $V_{L(i)}$, the smaller side of the cut of V_i, and any single edge can be inside $V_{L(i)}$ in at most $\log n$ divide steps i throughout the tree T. Hence, $\sum_i e(C_L, F_L)$ consists of at most $\log n$ times the cost of every edge of the bisection (W, B), and therefore this sum is at most $e(W, B) \cdot \log n$.

Consider next $\sum_i \frac{|C_L|}{|C_i|} \cdot e(C_i, F_i)$, and recall our convention that $\frac{0}{0}$ is defined to be 0. The edges of $e(C_i, F_i)$ contribute to the sum their cost scaled by a factor of $\frac{|C_L|}{|C_i|}$. Each edge of

$e(C_i, F_i)$ is a charged-free edge and hence an edge of the bisection (W, B). However, an edge of the bisection (W, B) belongs to $e(C_i, F_i)$ if and only if this edge is inside V_i. The nodes i for which this edge is inside V_i are all on a path from the root to a leaf of the decomposition tree T, and therefore the total contribution of this edge is at most its cost scaled by the sum of $\frac{|C_L|}{|C_i|}$ over that path in T.

We claim that the sum of $\frac{|C_L|}{|C_i|}$ over any path from the root to a leaf is bounded by $O(\log n)$. It follows from this claim that $\sum_i \frac{|C_L|}{|C_i|} \cdot e(C_i, F_i)$ can be described as the cost of every edge of the bisection (W, B) scaled by at most $O(\log n)$, and therefore this sum is at most $e(W, B) \cdot O(\log n)$.

To prove the claim, consider an arbitrary path from the root to a leaf, and denote the path nodes by $1, 2, \ldots, p+1$. At each node i the charged side (i.e. C_i) may be either W or B, depending on the label of the node, so denoting $w_j = |W \cap V_j|$ and $b_j = |B \cap V_j|$, we have that $\frac{|C_L|}{|C_i|}$ is either $\frac{w_{L(i)}}{w_i}$ or $\frac{b_{L(i)}}{b_i}$, and clearly at most their sum. Hence,

$$\sum_{i=1}^{p} \frac{|C_L|}{|C_i|} \leq \sum_{i=1}^{p} \frac{w_{L(i)}}{w_i} + \sum_{i=1}^{p} \frac{b_{L(i)}}{b_i}$$

Consider first $\sum_1^p \frac{w_{L(i)}}{w_i}$, and observe that w_i is a nonincreasing sequence, since in the tree, node i is a parent of node $i+1$. If node $i+1$ is a left child (of its parent node i), then $w_{L(i)} = w_{i+1}$ and hence $\frac{w_{L(i)}}{w_i} = \frac{w_{i+1}}{w_i} \leq 1$. The number of such nodes i is at most $\log n$, since the path from the root to a leaf can contain at most $\log n$ left children i (recall that $|V_{L(i)}| \leq |V_{R(i)}|$). The contribution of all such nodes i to $\sum_1^p \frac{w_{L(i)}}{w_i}$ is therefore at most $\log n$.

If node $i+1$ is a right child (of its parent i), then $w_{L(i)} = w_i - w_{i+1}$, and the contribution of all such nodes i is at most $\sum_1^p \frac{w_i - w_{i+1}}{w_i}$. Clearly, $\frac{w_i - w_{i+1}}{w_i} \leq \frac{1}{w_i} + \ldots + \frac{1}{w_{i+1}+1}$ and hence the contribution of all such nodes i to $\sum_1^p \frac{w_{L(i)}}{w_i}$ is at most $\sum_1^p \frac{w_i - w_{i+1}}{w_i} \leq \frac{1}{w_1} + \ldots + \frac{1}{2} + 1 = H(w_1) \leq H(n)$ where $H(k) = \sum_1^k \frac{1}{j}$ is the k-th harmonic number.

We conclude that $\sum_1^p \frac{w_{L(i)}}{w_i} \leq \log n + H(n) \leq O(\log n)$. Similarly, $\sum_1^p \frac{b_{L(i)}}{b_i} = O(\log n)$, and together we get that $\sum_1^p \frac{|C_L|}{|C_i|} \leq O(\log n)$, proving the claim and the lemma. □

Corollary 2.13. *The charge of the fixed optimal bisection (W^*, B^*) with respect to an opt-consistent labeling is at most $b(1 + O(\rho \log n))$.*

Distributing charge to vertices

It will be convenient (algorithmically) to distribute the charge of a bisection (W, B) (with respect to T and a labeling) to the vertices of the input graph, as follows. For each vertex $v \in V_i$ let the *cross-degree* of v at node i, denoted $cross_i(v)$, be the cost of the edges that are incident at v and are cut in divide step i. We define the *charge of a vertex $v \in V$* as the sum of the cross-degree of v at all nodes i for which v belongs to the charged side, i.e. $\sum_{i:v \in C_i} cross_i(v)$. The next lemma proves that distributing the charge of a bisection to the graph vertices is indeed correct.

Lemma 2.14. *The charge of a bisection (W, B) is the sum of the charges of all vertices in G.*

Proof. The charge of a divide step of node i is equal to the sum of the cross-degrees at node i of all vertices $v \in V_i$, i.e.

$$e(C_i \cap V_{L(i)}, V_{R(i)}) + e(C_i \cap V_{R(i)}, V_{L(i)}) = \sum_{v \in C_i} cross_i(v) \ .$$

Summing over all nodes i in the tree T, the lefthandside is, by definition, the bisection charge, and the righthand side is the sum of the charges of all vertices in G. The proof follows. $\square$

Distributing the charge to the vertices of G is important algorithmically. The charge of a vertex depends on (and can be easily computed from) the side of this vertex in the bisection (W, B), the decomposition tree T, and the labeling of T, but it does not depend on the side of the cut (W, B) that other vertices of the graph belong to. It follows that the charge of a bisection (W, B) with respect to a given decomposition tree T and a labeling of it, depends *linearly* on the placement of vertices into W and B. This formulation of charge will be exploited by (the dynamic programming in) the combining stage.

2.4.4 Combining stage

The combining stage computes a bisection of the input graph G and a labeling of the decomposition tree T, such that the bisection charge with respect to the labeling is at most $b \cdot (1 + O(\rho \log n))$. It then follows from Lemma 2.11 that the cost of the computed bisection is at most $b \cdot (1 + O(\rho \log n))$, as desired.

Consider first the case where an opt-consistent labeling is known. Then it suffices to compute a bisection of G whose charge with respect to this opt-consistent labeling is minimal, because Corollary 2.13 guarantees that the charge of the computed bisection is at most $b \cdot (1 + O(\rho \log n))$. Below we describe a simple procedure for finding a bisection of G with minimal charge with respect to a given labeling.

However, we do not know how to efficiently find an opt-consistent labeling, and therefore we go over all the labelings in the family $\mathcal{F}$. Specifically, using a more complicated procedure described below the combining stage finds a bisection of G and a labeling from $\mathcal{F}$, such that the charge of the bisection with respect to the labeling is minimal over all such bisection-labeling pairs. Lemma 2.10 guarantees that at least one of these labelings is opt-consistent, in which case Corollary 2.13 applies. Hence, the bisection-labeling pair computed by this procedure satisfies that the charge of the bisection with respect to the labeling is indeed at most $b \cdot (1 + O(\rho \log n))$.

Minimizing charge over a given labeling

Finding a bisection of minimum charge with respect to a given labeling is relatively straightforward. By Lemma 2.14, the charge of a bisection (W, B) is the sum of the vertex charges. Since the decomposition tree T and the labeling are fixed, the charge of a vertex depends only on its side in the bisection (W, B). We can therefore compute for each vertex v what is its charge when it belongs to W, called the *white charge* of v, and what is its charge when it belongs to B, called the *black charge* of v. (Note that summing the white charge and the black charge of a vertex gives the degree of that vertex in G.)

The charge of a bisection (W, B) is then the sum of the white charges of W and the black charges of B. To find a bisection (W, B) with minimum charge with respect to the given labeling, we can thus compute for each vertex its net-charge (white charge minus black charge), and take W to be the $n/2$ vertices with smallest net-charge. (This algorithm for the case where a labeling is given was used in the algorithm outline in Section 2.2, where we assumed that the labeling stage produces an opt-consistent labeling.)

Minimizing charge over the family $\mathcal{F}$

The combining stage uses dynamic programming to find a bisection and a labeling from the family $\mathcal{F}$, so that the charge of the bisection with respect to this labeling is minimum over all such bisection-labeling pairs.

The dynamic programming table Q has entries of the form $Q(i, k, g)$, where i is a node of the decomposition tree T, k is an integer between 0 and $|V_i|$, and g is a *guess list* that contains the labels of the marked ancestors of node i. Throughout, i is considered an ancestor of itself.

An entry $Q(i, k, g)$ in the table contains the optimal solution to the following problem: Choose k vertices of V_i and a labeling from $\mathcal{F}$ that agrees with g, so that when these k vertices are placed in the side W and the remaining vertices of V_i are placed in the side B, the sum of the charges of all the vertices of V_i with respect to the chosen labeling, is minimal over all such choices. Note that when we only consider labelings from the family $\mathcal{F}$ that agree with g, the labels of all the ancestors of i are uniquely defined from g, while the marked descendants of i can have arbitrary labels.

For a leaf node i, the table entry $Q(i, k, g)$ can be computed directly, as follows. Since i is a leaf node, the part V_i consists of a single vertex, say v, and k can be either 0 or 1. If $k = 0$ then v is necessarily in B, and if $k = 1$ then v is necessarily in W. The guess list g gives the labels of all the nodes on the path from the leaf i to the root, and hence all the labels that can possibly affect the charge of v. Since k and g uniquely define all the data that the charge of v depends on, $Q(i, k, g)$ is just the charge of v, and can be computed directly as $\sum_j cross_j(v)$ where j ranges over all ancestors of i whose label (according to g) agrees with the side of v (as follows from k).

For a nonleaf node i, the table entry $Q(i, k, g)$ can be efficiently computed from table entries of its children nodes $L(i), R(i)$. Indeed, choosing k vertices from V_i is equivalent to choosing j vertices from one child part $V_{L(i)}$ and $k-j$ vertices from the other child part $V_{R(i)}$, so we need to add up two entries, each corresponding to one child node. The optimal value of j is not known, but it can be exhaustively searched. The guess list g can be extended into lists g_L, g_R for the children nodes, in possibly more than one way. Therefore,

$$Q(i, k, g) = \min_{0 \le j \le k} \min_{g_L, g_R} \{Q(L(i), j, g_L) + Q(R(i), k - j, g_R)\}$$

where g_L, g_R range over all possible extensions of g, as described below. If a child node $L(i)$ is a marked node, then there are two possible ways to extend the list g into a list g_L (by adding a label for $V_{L(i)}$), and the optimum $Q(i, k, g)$ is achieved by taking the one which is better. If a child node $L(i)$ is not a marked node, then the only extension is $g_L = g$, because i and $L(i)$ have the same marked ancestors. The possible extensions of the child node $R(i)$

are similar. It follows that each table entry of a nonleaf node i can be computed from table entries of its children $L(i), R(i)$ in time $O(|V_i|) = O(n)$.

To fill all the table entries, start from the entries that correspond to leaf nodes i and go upwards the decomposition tree T. In particular, the entries $Q(i_{root}, n/2, g)$ will be computed for the root node i_{root}. At the root node, the guess list g contains the label of the root, and thus has only two possible values. (In fact, the two entries must be the same due to symmetry.) The combining stage outputs $\min_g Q(i_{root}, n/2, g)$, which by definition, is the minimum charge of all bisections of the input graph with respect to any labelings from $\mathcal{F}$, as desired. A bisection that achieves this minimum charge can also be computed. Simply go over the table entries in the reversed order of computation, and recover at each entry the values of j, g_L, g_R that gave the optimum. Alternatively, associate with each entry $Q(i,k,g)$ a set of k vertices of V_i which is optimal for it, and its corresponding labels.

Lemma 2.15. *The combining stage finds in polynomial time a bisection of the input graph G and a labeling from the family $\mathcal{F}$, so that the charge of the bisection with respect to the labeling is minimal over all such bisection-labeling pairs.*

Proof. The above discussion shows that the algorithm correctly computes every entry $Q(i,k,g)$, and a bisection-labeling pair as desired.

The size of the table Q is polynomial in n. Indeed, there are only $O(n)$ tree nodes i. For each tree node i, the range of k contains $O(|V_i|) = O(n)$ possible values. In addition, at each tree node i the guess list g contains labels of at most $O(\log n)$ ancestor nodes, and thus g assumes polynomially many values. The polynomial bound on the size of the table Q follows.

An entry for a leaf nodes i is computed efficiently. An entry for a nonleaf node is efficiently computed from previously computed entries. By the upper bound on the table size we conclude that all the table entries are computed in polynomial time, and in particular $Q(i_{root}, n/2, g)$. ☐

Corollary 2.16. *The combining stage finds bisection of the input graph (and a labeling of T) such that bisection charge (with respect to the labeling) is at most $b(1 + O(\rho \log n))$.*

Proof. By Lemma 2.15 and Corollary 2.13 there exists a bisection of G and a labeling of $\mathcal{F}$ such that the bisection charge with respect to the labeling is at most $b(1 + O(\rho \log n))$. The proof then follows by applying Lemma 2.10. ☐

This corollary completes the proof of Theorem 2.5, since by Lemma 2.11 the charge of a bisection is an upper bound on its actual cost.

2.5 Extensions

Our results extend to several variants (and generalizations) of the minimum bisection problem, including the case of edges with arbitrary nonnegative costs (Section 2.5.1), the case of vertices with polynomially bounded nonnegative integer weights (Section 2.5.2), the variant that requires, in addition, to separate a given pair of vertices s and t (Section 2.5.3), the case of cutting away from the graph an arbitrary number of vertices (instead of $n/2$) that is given as part of the input (Section 2.5.4), the case of cutting the input graph into a fixed number of equal-size parts (Section 2.5.5), and the case of finding a 2/3-balanced cut whose cost is small relative to the minimum bisection cost b (Section 2.5.6).

In what follows, the *basic bisection problem* refers to the minimum bisection problem that was defined in Section 2.1. In contrast, the *extended bisection problems* refer to the variants of the problem specified above. We discuss each extended problem separately, but it is straightforward to combine together several extensions (e.g. to allow both edge costs and vertex weights as described above, and require that the total weight of the vertices cut away is a number k that is given in the input).

We consider two approaches for extending our approximation algorithm from the basic bisection problem to an extended problem. One approach is to *reduce* the extended problem to the basic one. Another approach is to *modify the algorithm* that we devised for the basic bisection problem so that it handles also the extended variant. As we discuss below, each approach has its own advantages and so it is valuable to show both approaches for each extended problem. We indeed show that for almost all the extended problems specified above both approaches can be applied, although for a few problems we provide only a modified algorithm.

A major advantage of the reduction approach is that it is self contained and not restricted to the particular algorithm that we devise, so future improvement in the approximation ratio for the basic problem may lead to an immediate improvement also for the extended problem. Most of our reductions transform an approximation ratio $f(n)$ for the basic problem into an approximation ratio $f(n^{O(1)})$ for the extended problem (because they increase the number of vertices n by a polynomial), and so for the current approximation ratio $f(n)$, which is polylogarithmic, these reductions increase the approximation ratio by at most a constant factor. The techniques used in our reductions are similar to those devised in [BJ92, BCLS87] for the (different) purpose of proving NP-hardness results.

The advantages of the algorithm modification approach are that it preserves aspects that are specific to our algorithm, such as an improved $O(\log n)$ approximation ratio for planar graphs, and that it is usually more efficient (and therefore practical) than the reduction approach. A drawback of the algorithm modification approach is that it requires to go again through the algorithm's analysis. In particular, we might be required to verify that the approximate min-ratio cut algorithm (that we use as a black-box) can be extended accordingly. However, the necessary changes in the algorithm and its proof are usually straightforward.

2.5.1 Edge costs

Suppose that the edges of the input graph G have arbitrary nonnegative costs, and that the cost of a bisection is the total cost (i.e. sum of the costs) of its edges, and we wish to find a bisection of G of (approximately) minimum cost.

Reduction. We reduce the extended problem of bisection with edge costs (described above) to the basic bisection problem, as follows. Given a graph G with edge costs as an input, we first guess the most costly edge in a minimum cost bisection of G, by exhaustively trying all $O(n^2)$ edges in the input graph. By scaling all edge costs, we can assume, without loss of generality, that the cost of the guessed edge is n^2. It follows that the cost b of the optimum bisection is at least n^2 but smaller than n^4. We then round down all edge costs to their closest integer, which can decrease the cost of any bisection by at most $\binom{n}{2} \leq b/2$ and therefore by a factor of at most 2. We next change to n^5 every edge cost that is larger than n^5, which does not affect the cost of nearly optimal bisections (i.e. whose original cost was within ratio of roughly n from the minimum). Finally, we replace each vertex of the graph by a clique of size n^5, and each edge (u, v) of cost t by t unit cost edges placed arbitrarily between the clique of u and the clique of v (since $t < n^{10}$ we can do that with no parallel edges).

The bisection of minimum cost b in G corresponds to a bisection of cost $\Theta(b)$ in the resulting graph. Hence, applying our algorithm for the basic problem on the resulting graph (which has n^6 vertices) yields a bisection whose cost is $O(b(\log n^6)^2) = O(b \log^2 n)$. This bisection cannot split any of the cliques that we created, as otherwise its cost will be at least $n^5 - 1 \gg b \log^2 n$, and it therefore must correspond to a bisection of G, whose cost is roughly the same, namely $O(b \log^2 n)$, as required.

Modified algorithm. We modify our algorithm for the basic bisection problem so that it handles the extended problem with edge costs, as follows. Rather than considering the number of edges we always consider their cost, e.g. $e(V_1, V_2)$ denotes the sum of the costs of the edges with one endpoint in V_1 and one endpoint in V_2. The corresponding changes in our algorithm and analysis are straightforward. Note that the amortized cut algorithm (see Fig. 2.4) requires (in step 1) a subroutine that computes an approximate min-ratio cut with respect to the edge costs, but known algorithms (e.g. due to [LR99]) provide this subroutine. Note also this algorithm's binary search (step 3) takes $O(M \log n)$ iterations, where M is the number of bits used to represent an edge cost, and so the running time is polynomial in the input size. The resulting approximation ratio is the same as for the basic problem, i.e. $O(\log^2 n)$.

2.5.2 Polynomial vertex weights

Suppose that the vertices of the input graph G have nonnegative integer weights that are bounded by a polynomial n^c (where n is the number of vertices in G), and let a bisection be a cut that separates half of the total weight (i.e. sum of the weights) of the vertices of V. We wish to find a bisection of G of (approximately) minimum cost. Note that if the weights are allowed to be exponential in n, finding any bisection of the graph is equivalent to the partition (or subset-sum) problem, and therefore NP-hard.

Reduction. We reduce the extended problem of bisection with vertex weights (described above) to the basic bisection problem, as follows. Given a graph G with vertex weights as an input, we replace each vertex of cost w in G by a clique of $\max\{1, w \cdot n^3\}$ unit weight vertices, and replace each edge (u, v) in G by one edge placed arbitrarily between the clique of u and the clique of v. In addition, for each vertex of weight 0 in G we place in the graph a new isolated vertex of unit weight.

A bisection of minimum cost b in G corresponds to a bisection of the same cost b in the resulting graph. Hence, applying our algorithm for the basic problem on the resulting graph (which has at most n^{c+4} vertices) yields a bisection whose cost is $O(b(c+4)^2 \log^2 n)$. This bisection cannot split any of the cliques that we created, as otherwise its cost will be at least $n^3 - 1 \gg b \cdot (c+4)^2 \log^2 n$. Furthermore, the vertices of the created cliques of size at least n^3 must be partitioned evenly by this bisection, as otherwise their partition deviates from an even one by at least n^3 (these clique sizes are multiples of n^3) which is much more than the total number of remaining vertices, $2n^2$ (recall that we added isolated vertices for vertices of weight 0 in G). The computed bisection of the resulting graph therefore corresponds to a bisection of G, whose cost is the same, namely $O(b(c+3)^2 \log^2 n)$, as required.

Modified algorithm. We modify our algorithm for the basic bisection problem so that it handles the extended problem with vertex weights, as follows. Rather than considering the number of vertices in a part we always consider their total weight, e.g. $r(S)$ denotes the cost of the cut $(S, V \setminus S)$ divided by the minimum between the weight of S and the weight of $V \setminus S$. The corresponding changes in our algorithm and analysis are straightforward. Note that the amortized cut algorithm (see Fig. 2.4) requires (in step 1) a subroutine that computes an approximate min-ratio cut with respect to the vertex weights, but known algorithms (e.g. due to [LR99]) provide this subroutine. Note also that in this algorithm's related graph G' (step 2) the capacity of an edge between a vertex $v \in V_1$ and the new vertex s is p times the weight of v_1. The resulting approximation ratio is the same as for the basic problem, i.e. $O(\log^2 n)$.

2.5.3 Separating two vertices from each other ($s - t$ cut)

Suppose that the input graph G contains two special vertices s and t, and we wish to find a bisection that separates s from t and has minimum cost. (Note that the converse restriction, namely that s, t will not be separated, is equivalent to merging them into one vertex of weight 2, and therefore follows from Section 2.5.2).

Reduction. We reduce the extended problem of a bisection that separates s from t to the extended problem of bisection with vertex weights (described in Section 2.5.2), as follows. Given an input graph G with special vertices s, t as above, we let the vertices s, t have weights n and let all other vertices of G have weight 1. The total weight of s and t together is $2n$, while the total weight of all other vertices is $n - 2$ (and thus smaller), so every bisection of the resulting graph must separate s from t. It follows that every bisection of the resulting graph corresponds to a bisection of G that separates s from t and has the same cost, and vice versa. We can therefore find a bisection of G that separates s from t and its cost is within $O(\log^2 n)$ from the minimum.

Modified algorithm. We modify our algorithm for the basic bisection problem so that it handles the extended problem of a bisection that separates s from t, as follows. We change the dynamic programming table Q of the combining stage, so that every entry $Q(i, k, g)$ contains two solutions (if they exist); one solution with the k chosen vertices containing s but not t, and the other solution with the k chosen vertices not containing any of s and t. Computing the table entries is straightforward, and the output of the algorithm is $\min_g Q(i_{root}, n/2, g)$, where the minimum is taken only over solutions that contain s and not t. The necessary changes in our analysis are straightforward. The resulting approximation ratio is the same as for the basic problem, i.e. $O(\log^2 n)$.

2.5.4 Cutting an arbitrary given number of vertices

Suppose that the input consists of a graph G and a number k, and we wish to find a minimum cost cut that separates exactly k vertices.

Reduction. We reduce the problem of cutting away a given number k of vertices to the problem of bisection with vertex weights (described in Section 2.5.2), as follows. Given an input graph G and a number k (assume, without loss of generality, that $k \leq n/2$), we let the vertices of G have weight 1, and add to the graph an isolated vertex of weight $n - 2k$. It is clear that every bisection of the resulting graph corresponds to a cut of G that separates k vertices and has the same cost, and vice versa. We can therefore find a cut of G that separates k vertices and its cost is within $O(\log^2 n)$ from the minimum.

Modified algorithm. We modify our algorithm for the basic bisection problem so that it handles the extended problem of cutting a given number of vertices, as follows. The only change in the algorithm is in the combining stage, that now outputs $\min_g Q(i_{root}, k, g)$, where Q is the dynamic programming table (see Section 2.4.4). The necessary changes in our analysis are straightforward. The resulting approximation ratio is the same as for the basic problem, i.e. $O(\log^2 n)$.

2.5.5 Cutting into a fixed number of parts

Suppose that we wish to find a cut that separates the input graph G into a fixed number p of parts of equal size.

We do not know of a reduction from this extended problem to the basic bisection problem. A recursive bisection approach has a poor performance in general, although it may be useful in some special cases and if some requirements are relaxed, see [ST97] and the references therein.

Modified algorithm. We modify our algorithm for the basic bisection problem so that it handles the problem of cutting the graph into p parts of equal size, as follows. The cost of a cut that partitions V into p parts $V^1, \dots, V^p$ is

$$\sum_{j<l} e(V^j, V^l) = \frac{1}{2} \sum_j e(V^j, V \setminus V^j).$$

Therefore, by scaling the value of every possible solution by a factor of 2 (which clearly does not affect any approximation ratio issues), we obtain that the objective function of the extended problem has the convenient form $\sum_j e(V^j, V \setminus V^j)$. Observe that each cut $(V^j, V \setminus V^j)$ corresponds to separating V^j from the other parts, which are grouped into one part $V \setminus V^j$. Thus, each summand $e(V^j, V \setminus V^j)$ in the objective function is similar to the basic bisection problem (with the minor exception that the two sides are not of the equal sizes). Below we describe the modifications to the three stages of the algorithm, which works simultaneously on all p cuts $(V^j, V \setminus V^j)$. Its analysis is based on applying the new accounting method of Section 2.2 separately to each of these p cuts.

The decomposition stage computes a decomposition tree T exactly as in the algorithm for the basic problem (see Section 2.4.1). Observe that the amortized cut notion does not depend on the cut that we seek, and so the obtained decomposition (and its tree T) can be used for all cuts $(V^j, V \setminus V^j)$.

We extend the notion of a labeling of the decomposition tree, as follows. An extended labeling of T assigns to every tree node a vector of p "basic" labels, one label for each cut $(V^j, V \setminus V^j)$. An extended labeling corresponds to deciding at each tree node i and for each j, which of V^j and $V \setminus V^j$ is considered charged (and which is considered free) in the part V_i. Note that an extended labeling can be viewed as a vector, whose coordinate j forms a basic labelings for $(V^j, V \setminus V^j)$.

The labeling stage marks some nodes of the tree T exactly as in the algorithm for the basic problem (see Section 2.4.2). This stage implicitly defines a family $\overline{\mathcal{F}}$ that consists of all extended labelings in which every unmarked node has the same label as its closest marked ancestor (there is no restriction on the labels of the marked nodes). It is straightforward that $\overline{\mathcal{F}}$ contains at least one extended labeling, for which every coordinate j (forms a basic labeling that) is α-consistent with the cut $(V^j, V \setminus V^j)$. We can restrict the number of possible labels at the marked (and hence also unmarked) nodes from 2^p to $p+1$ values, as follows. Similar to the proof of Lemma 2.10 it is sufficient for our purposes that $\overline{\mathcal{F}}$ contains the labeling where V^j is considered free at a marked node i if more than half the vertices of the part V_i are from V^j. At any part V_i, the latter can happen for at most one value of j, and so it suffices to consider only labelings where at most one V^j is free.

We extend the notion of a charge of a vertex, as follows. The extended charge of a vertex v with respect to an extended labeling is the sum of the basic charges of v with respect to each of the p coordinates of this extended labeling.

The combining stage uses dynamic programming on a table $\overline{Q}$, whose entries are of the form $\overline{Q}(i, \overline{k}, \overline{g})$, as follows. i is a tree node. $\overline{k} = (k_1, \dots, k_p)$, where k_j is the desired size of the jth part and $\sum_j k_j = |V_i|$. $\overline{g} = (g_1, \dots, g_p)$ where g_j is a guess list that contains the jth label of every marked ancestor of i. An entry $\overline{Q}(i, \overline{k}, \overline{g})$ contains the optimal solution to the following problem: Choose a partition of V_i into subsets with sizes according to $\overline{k}$, and choose a labeling from $\overline{\mathcal{F}}$ that agrees with g, so that the sum of the extended charges of all the vertices of V_i with respect to the chosen labeling, is minimal over all such choices. Note that this problem requires some correlation between p cuts, and therefore $\overline{Q}(i, \overline{k}, \overline{g})$ is generally not equal to $\sum_j Q(i, k_j, g_j)$ (where Q is the basic table).

The rules for computing the entries of the table $\overline{Q}$ are a straightforward extension of those for the table Q (see Section 2.4.4). The algorithm computes all the table entries and then outputs $\min_{\overline{g}} \overline{Q}(i_{root}, \overline{k}, \overline{g})$ where $\overline{k} = (n/p, \dots, n/p)$.

The running time of this modified algorithm is polynomial in n (for fixed p). Indeed, the decomposition stage and the labeling stage are exactly as in the algorithm for the basic bisection problem, so let us consider the dynamic programming table $\overline{Q}$ of the combining stage. The number of tree nodes i is $O(n)$, and the range of $\overline{k}$ contains at most n^p possible values. The vector $\overline{g}$ contains one of $p+1$ possible values for each of the $O(\log n)$ marked ancestors (of the relevant tree node i), so $\overline{g}$ assumes one of $n^{O(\log p)}$ values. It follows that the size of the table $\overline{Q}$ is $n^{p+O(\log p)}$. Each table entry is computed efficiently from previously computed entries, and hence the combining stage takes polynomial time.

To analyze the approximation ratio, let $V^1, \ldots, V^p$ be the optimal partition of the input graph into p parts of equal size. Recall that the extended charge of a vertex is the sum of its basic charges with respect to each cut $(V^j, V \setminus V^j)$, and we can therefore apply the analysis of the basic algorithm for each cut $(V^j, V \setminus V^j)$ separately. It follows that the output value is guaranteed to be at most $O(\log^2 n) \cdot \sum_j e(V^j, V \setminus V^j)$. Furthermore, one can obtain from the table $\overline{Q}$ a cut (into p parts of equal size) whose cost is at most (half) this value, i.e. within a ratio of $O(\log^2 n)$ from the minimum.

2.5.6 Bicriteria approximation and balanced cuts

Suppose that we wish to find a 2/3-balanced cut (recall that a cut is called β-balanced if it partitions the graph into two parts, each of size at most βn) whose cost is guaranteed to be small relative to the minimum cost b of a bisection (i.e. a 1/2-balanced cut). Here, the minimum bisection problem is relaxed in two respects, as the solution cut is allowed to have cost larger than b and also to deviate from the cardinality constraints (for its two sides). Algorithms for such problems are sometimes referred to as bicriteria approximation and sometimes as pseudo-approximation.

Known bicriteria approximation algorithms find a 2/3-balanced cut whose cost is at most $O(b \log n)$. Leighton and Rao [LR88, LR99] show how an algorithm that finds a τ approximate min-ratio cut can be used to find a 2/3-balanced cut of cost $O(b\tau)$; the approximation ratio $\tau = O(\log n)$ that they achieve is the best currently known, see also [Shm97]. Even, Naor, Rao and Schieber [ENRS97] devise a different algorithm that also finds a 2/3-balanced cut of cost $O(b \log n)$.

We show below that amortized cuts can be used to obtain also bicriteria approximation algorithms (in addition to approximation algorithms) for minimum bisection. In fact, our algorithm is similar to the one of [LR88, LR99], except that we use amortized cuts instead of approximate min-ratio cuts.

Lemma 2.17. *An algorithm that finds a ρ-amortized cut can be used to find a 2/3-balanced cut of cost $b(1 + O(\rho))$.*

Proof. Given an input graph $G(V, E)$ on n vertices, use the algorithm that finds a ρ-amortized cut, as follows. Repeatedly find (in the graph) a ρ-amortized cut and remove (from the graph) the smaller of its two sides, until the graph contains no more than $2n/3$ vertices. Denoting by S the set of vertices that remain in the graph after the last iteration, output the cut $(S, V \setminus S)$.

It is straightforward to see that $n/3 < |S| \leq 2n/3$, and hence the output cut $(S, V \setminus S)$ is a 2/3-balanced cut. We prove below that the total cost of all edges cut by the amortized cuts (throughout the iterations) is at most $b(1+O(\rho))$. It would then follow immediately that $e(S, V \setminus S) \leq b(1+O(\rho))$, as required.

We now upper bound the total cost of all edges cut in the amortized cuts. Let (W, B) be a fixed optimal bisection of cost b, and call the vertices of W white, and the vertices of B black. The total cost of white-black edges cut is clearly at most b. We show below that the total cost of all white-white edges cut is $O(b\rho)$. By the symmetry between W and B, we will then have a similar upper bound on the total cost of the black-black edges cut, and obtain the desired upper bound of $b(1+O(\rho))$ on the total cost of all edges cut.

To show that the total cost of white-white edges cut in the amortized cuts is $O(b\rho)$, we consider the white vertices W as charged in all the amortized cuts, and then white-white edges are charged-charged edges. The algorithm applies a ρ-amortized cut in parts of G that contain at least $2n/3$ vertices. At least $n/2 - n/3 = n/6$ of the vertices in such a part are black, while at most $n/2$ of them are white, and hence at most $3/4$ of the vertices in this part are considered charged. Taking a constant $\alpha \geq 3/4$ in the definition of an amortized cut, we have that the cost of the charged-charged edges cut can be amortized in one of two amortization methods (see Section 2.2).

In one amortization method the cost of the charged-charged edges cut is amortized against charged-free edges in the smaller side of the cut, with amortized cost at most ρ. Observe that an edge can be in the smaller side of the amortized cut (the side that is removed) in at most one iteration, so the total cost amortized in this method (in all the iterations) against one charged-free edge is at most ρ. Hence, the total cost amortized in this method (in all the iterations) is at most $b\rho$.

In the other amortization method the cost of the charged-charged edges cut is amortized against charged-free edges in the part being divided, with amortized cost at most $\rho|C_1|/|C|$, where C denotes the charged vertices in the part being divided and C_1 denotes the charged vertices in the smaller side of the cut. The total cost amortized in this method (in all the iterations) against one charged-free edge is then upper bounded by ρ times the sum of $|C_1|/|C|$ over all iterations. Recall that the charged vertices are the white vertices, and so $|C| \geq n/6$ in all amortized cuts (i.e. iterations). Furthermore, each vertex is in the smaller side of the cut (the side that is removed) in at most one iteration, and so the sum of $|C_1|$ over all iterations is at most $n/2$. It follows that the total cost amortized in this method (in all the iterations) against one charged-free edge is at most 3ρ, and hence the total cost amortized in this method is at most $b \cdot 3\rho$.

We conclude that the total cost of all charged-charged (i.e. white-white) edges cut in all the iterations is at most $b \cdot 4\rho$. As described above, this proves that the total cost of all edges cut in all the iterations is at most $b(1+8\rho) = b(1+O(\rho))$, and the lemma follows. □

We remark that a $2/3$-balanced cut of cost $b(1+O(\rho))$ can be found also by modifying the algorithm we devised for the basic bisection problem so that its combining stage outputs $\min_{g, n/3 \leq k \leq n/2} Q(i_{root}, k, g)$ (and its corresponding cut). Indeed, the proof of Lemma 2.17 shows a $2/3$-balanced cut whose charge (with respect to a certain labeling in $\mathcal{F}$) is at most $b(1+O(\rho))$. Details omitted.

2.6 Cutting a few vertices from a graph

In this section we present a randomized algorithm for approximating the minimum $(k, n-k)$ cut problem when k is relatively small. In particular, we prove Theorem 2.3 by showing that for an arbitrary fixed $\epsilon > 0$, the algorithm finds, with high probability, a $(k, n-k)$ cut whose cost is at most $(1 + \epsilon k/\ln n)b_k$. The algorithm appears in Section 2.6.1. Some extensions of this algorithm are described in Section 2.6.2.

Techniques. Our algorithm utilizes random edge contraction and dynamic programming. Random edge contraction was introduced by Karger and Stein [KS96] to devise efficient algorithms for the minimum cut problem. Each iteration of their algorithm selects an edge at random and merges its endpoints, so as to form clusters of vertices. If no edge of a fixed minimum cut $(S, V \setminus S)$ is ever contracted, then every cluster is contained entirely either in S or in $V \setminus S$. When only two clusters remain, they correspond to the fixed minimum cut. It can be shown that there is a noticeable probability that no edge of the fixed minimum cut is ever contracted, and then the algorithm succeeds.

Our algorithm also applies random edge contractions iteratively, but instead of requiring that only two clusters remain, we stop at an earlier point, in which we are guaranteed that dynamic programming will find a nearly minimum $(k, n-k)$ cut. The algorithm actually does not know the "right" stopping point, and therefore tries all possible stopping points (taking the best solution).

2.6.1 A randomized algorithm

Our algorithm for finding a $(k, n-k)$ cut (of nearly minimum cost) uses the random edge contraction technique of Karger and Stein [KS96]. It consists of repeating the following algorithm Contract sufficiently many times in order to amplify its success probability.

Algorithm Contract works in iterations, where each iteration consists of (i) a random edge contraction stage followed by (ii) a combining stage that computes a cut of the graph that corresponds to a $(k, n-k)$ cut of the input graph. (Both stages are described below). The algorithm proceeds with the iterations until there are no edges in the graph (to contract) and then it outputs a cut of minimum cost among all $(k, n-k)$ cuts found throughout the iterations (if any).

Let us now describe in more detail the two stages that form an iteration of algorithm Contract. A schematic description of the algorithm appears in Figure 2.7.

In the *contraction stage* we choose an edge uniformly at random and contract it by merging its two endpoints. If as a result there are several edges between some pairs of (newly formed) vertices (i.e. parallel edges), we retain them all. Edges between vertices that were merged are removed, so that there are never any self-loops.

We refer to the vertices of the formed graph as *clusters*. Each cluster is a set of vertices (of the input graph) merged together. Note that the edges inside a cluster are removed from the graph. The *size* of a cluster is the number of vertices in it, and its *degree* is the number of edges leaving the cluster.

In the *combining stage* we find in the graph (of the current iteration) a set of clusters whose total size is exactly k, and for which the sum of cluster degrees is minimal. Note that any set of clusters, and in particular the one that we find, corresponds to a $(k, n-k)$ cut (of the input graph) whose cost is no more than the sum of degrees of these cluster.

It is straightforward to see that the combining stage can be implemented in polynomial time using dynamic programming, see e.g. [CLR90, Chapter 16].

```
Algorithm CONTRACT.
Input: Graph on $n$ vertices and a number $k$.
Output: $(k, n-k)$ cut in the graph.

  1. While the graph contains edges, do
     1.1. edge contraction stage;
     1.2. combining stage to find a $(k, n-k)$ cut.
  2. Output the cut of minimum cost among the
     cuts found in step 1.2 (if any).
```

Figure 2.7: Algorithm for finding a $(k, n-k)$ cut

Lemma 2.18. *The running time of algorithm* CONTRACT *is polynomial in n.*

Proof. Each edge contraction decreases the number of vertices by 1, and thus the number of iterations is bounded by n. Each iteration takes a polynomial time and the proof follows. □

We analyze the success probability of the algorithm based on the following desired scenario. Suppose that the edges chosen to be contracted do not belong to a fixed optimum cut $(S, V \setminus S)$, i.e. these edges are either inside S or inside $V \setminus S$, until at some point the edges inside S (that remain in the graph) have a small cost relative to the cost of the optimum cut. At this point, it can be seen that the combining stage must find a $(k, n-k)$ cut (of the input graph) whose cost is nearly optimal.

Lemma 2.19. *For every (not necessarily fixed) $\mu > 0$, algorithm* CONTRACT *outputs a $(k, n-k)$ cut whose cost is at most $(1+\mu k)b_k$ with probability at least $e^{-2/\mu}$.*

Proof. For the analysis, fix one cut $(S, V \setminus S)$ with $|S| = k$ whose cost b_k is minimum. Note that algorithm CONTRACT is not aware of this cut.

Consider a run of the algorithm, and let A_t (for $0 < t < n$) be the event that the graph G_t resulting from the first t contractions satisfies the following two conditions:

(a) The total cost of edges of G_t with both endpoints in S is at most $\mu k b_k/2$.

(b) No cluster of G_t contains vertices both from S and from $V \setminus S$.

Equivalently,

(b') None of the first t contracted edges belongs to the optimum cut $(S, V \setminus S)$.

We claim that if the event $A = \cup_t A_t$ happens then the algorithm succeeds, i.e. finds a $(k, n-k)$ cut of cost at most $(1+\mu k)b_k$. Indeed, assume that the event A_t happens and consider the combining stage of iteration t, which is performed on G_t. ¿From (b) we have that every cluster in G_t is either a subset of S or a subset of $V \setminus S$. Therefore, the clusters contained in S have together all the vertices of S, and thus their total weight is k. ¿From (a) it follows that the sum of degrees of these clusters (in G_t) is at most $b_k + 2(\mu k b_k/2) = b_k(1+\mu k)$. The combining stage of iteration t will therefore find a set of clusters of total weight k and whose sum of cluster degrees is no larger, which gives a $(k, n-k)$ cut (of the input graph), with cost at most $b_k(1+\mu k)$.

We next lower bound the probability of the event A. Let us say that an iteration is *successful* if the edge chosen to be contracted is inside S, a *ruin* if it is from the optimum cut $(S, V \setminus S)$, and *void* if it is inside $V \setminus S$. By (a) and (b'), the event A is equivalent to saying that the cost of edges inside S reduces to $\mu k b_k/2$ or less before any ruin iteration occurs. In this sense, the event A is affected by the successful and ruin iterations, but not by the void iterations. In other words, we need to compute the probability that an iterations is successful conditioned on the iteration not being void. As long as the cost of edges inside S, denoted $|E_S|$, is more than $\mu k b_k/2$, the conditioned probability for a successful iteration is

$$\frac{|E_S|}{|E_S| + b_k} = \left(1 + \frac{b_k}{|E_S|}\right)^{-1} \geq \left(1 + \frac{1}{\mu k/2}\right)^{-1}.$$

For the event A to happen we need that the first $k-1$ or less iterations that are not void will all be successful, and thus

$$\Pr[A] \geq \left(1 + \frac{1}{\mu k/2}\right)^{-(k-1)} > e^{-\frac{2(k-1)}{\mu k}} > e^{-2/\mu}.$$

The probability that the algorithm outputs a $(k, n-k)$ cut of cost at most $b_k(1+\mu k)$ is at least $\Pr[A] > e^{-2/\mu}$, as claimed. $\square$

For example, taking $\mu = \epsilon/\ln n$ for a fixed $\epsilon > 0$ we obtain the following.

Corollary 2.20. *For every fixed $\epsilon > 0$, with probability at least $n^{-2/\epsilon}$, algorithm* Contract *outputs a $(k, n-k)$ cut whose cost is at most $(1 + \epsilon k/\ln n)b_k$.*

We can amplify the above success probability by repeating algorithm Contract polynomially many (roughly $n^{2/\epsilon}$) times and taking from all the repetitions the cut of minimum cost. We then obtain Theorem 2.3.

2.6.2 Extensions

Edge costs. Suppose that the edges of the input graph have arbitrary nonnegative costs, and let the cost of a cut be the total cost (i.e. sum of the costs) of its edges.

Our results for approximating the minimum $(k, n-k)$ cut extend to this case of edge costs. The algorithm should be modified so that that the probability of choosing an edge (for contraction) is proportional to its cost, and that the degree of a cluster is the cost of the edges leaving the cluster. The proof follows.

***s−t* cuts.** Suppose that the graph contains two special vertices s, t that must be separated, i.e. we wish to find a minimum cost cut $(S, V \setminus S)$ with $|S| = k$, $s \in S$ and $t \in V \setminus S$.

Unlike the minimum cut algorithm of Karger and Stein [KS96] that does not extend to $s - t$ cuts (see e.g. [MR95, Problem 1.8]), our approximation ratio does extend to this $s - t$ cut variant of the problem. The proof follows by modifying the combining stage to consider only clusters that do not contain t and such that at least one of them contains s.

Vertex weights. Suppose that the vertices of G have nonnegative integer weights. A *w-cut* cuts away vertices of total weight w, i.e. it is a cut $(S, V \setminus S)$ for which the sum of weights of S is w. Let b_w be the minimum cost of a w-cut.

We consider the problem of finding a nearly optimal w-cut, i.e. whose cost approximates b_w. We assume that the vertex weights are bounded by a polynomial in n, since for exponential vertex weights it is NP-hard to decide whether G contains a w-cut (as this is simply the subset-sum problem).

Let $b_{w,k}$ be the minimum cost of a cut that cuts away $k \in \{1, \ldots, n-1\}$ vertices of total weight w (and ∞ if no such cut exists). Modifying the combining stage to find a w-cut (using dynamic programming), it is straightforward to extend the proof of Lemma 2.19 and show that if $b_{w,k}$ is finite then with probability at least $e^{-2/\mu}$ algorithm Contract finds a cut of cost at most $(1 + \mu k) b_{w,k}$. By taking sufficiently many repetitions with $\mu = \epsilon / \log n$ for a fixed $\epsilon > 0$, we conclude that one can find in polynomial time a w-cut whose cost is at most $\min_k \{(1 + \epsilon k / \log n) b_{w,k}\}$ with high probability. Note that the minimum in the latter bound is not necessarily obtained at a value of k for which $b_{w,k} = b_w$.

2.7 Concluding remarks.

Designing an algorithm that finds a cut of amortized cost better than $O(\log n)$ remains an important open question. An efficient algorithm that accomplishes that will not only improve the approximation ratio for minimum bisection (by Theorem 2.5), but also the bicriteria approximation ratio for minimum bisection (by Lemma 2.17), which will lead, in turn, to improved approximation ratios for many other problems, see [LR99, Section 3].

Finding a cut whose amortized cost is better than $O(\log n)$ is, in a sense, no harder (and possibly easier) than approximating min-ratio cuts within a ratio better than $O(\log n)$, as the former problem is reducible (by Theorem 2.4) to the latter. Furthermore, an $O(1)$-amortized cut always exists (by Corollary 2.7), and we know of no hardness result for the problem of finding such a cut.

Chapter 3

Heuristics for maximum clique*

3.1 Introduction

Let $G(V, E)$ be a graph on n vertices. A *clique* in G is a subset of the vertices every two of which are connected by an edge. The *maximum clique* problem requires to find a clique of maximum size in an input graph G. The *clique number* of G denoted $\omega(G)$, is the maximum size of a clique in G.

An *independent set* (a.k.a. *stable set*) in G is a subset of the vertices no two of which are connected by an edge. The *maximum independent set* problem requires to find an independent set of maximum size in an input graph G. The *independence number* (a.k.a. *stability number*) of G denoted $\alpha(G)$, is the maximum size of an independent set in G. It is straightforward that a clique in G forms an independent set in the edge complement graph $\overline{G}$, so $\omega(G) = \alpha(\overline{G})$. It follows that the maximum clique problem and the maximum independent set problem are equivalent in many respects, including in our context. For consistency with the related literature, we refer to one problem in some parts and to the other problem in others.

The maximum clique problem is fundamental in the area of combinatorial optimization, and is closely related, in addition to the maximum independent set problem, also to the *vertex cover* problem (the vertex complement of an independent set) and the *chromatic number* problem (minimum cover by independent sets). The maximum clique problem (or even finding $\omega(G)$) is one of the first problems shown to be NP-hard [Kar72].

A common way to cope with NP-hardness of a problem is to devise algorithms that give approximate solutions. An *efficient* (i.e. polynomial time) algorithm is said to have an *approximation ratio* $r > 1$ for the maximum clique problem if for every input graph, the ratio between $\omega(G)$ and the size of the clique returned by the algorithm is at most $r = r(n)$. It is known through work culminating in [Hås99] that for any fixed $\epsilon > 0$ it is impossible to approximate the clique number $\omega(G)$ within a ratio of $n^{1-\epsilon}$, unless NP has randomized polynomial time algorithms (NP=ZPP). The best approximation algorithm that is known for $\omega(G)$, due to [BH92], has approximation ratio $O(n/\log^2 n)$.

The intractability of the maximum independent set problem in the worst case suggests

*This chapter is based on [FK00a] and on [FK01b].

studying the performance of algorithms on average instances. A possible rigorous description of average instances is by probabilistic models, see e.g. the survey [FM97] on average-case analysis of several graph algorithms on random graphs.

The problem of finding a maximum clique on a random graph appears to be difficult. Let $G_{n,1/2}$ denote the random graph of n labeled vertices obtained by choosing, randomly and independently, each pair of vertices to be an edge with probability $1/2$. It is known that the clique number of $G_{n,1/2}$ is roughly $2\log_2 n$, *almost surely*, i.e. with probability that approaches 1 as n tends to infinity. Several simple and natural algorithms (e.g. the greedy one) find a clique of size roughly $\log_2 n$, almost surely. However, no algorithm is known to find efficiently an independent set of size significantly larger than $\log_2 n$, see [Kar76]. Finding cliques of size $\frac{3}{2}\log_2 n$ in random graphs was even suggested as a hard computational problem on which to base cryptographic applications, see [JP00].

The hidden clique problem. Jerrum [Jer92] and Kučera [Kuč95] suggested independently the following *hidden clique problem.* A random graph $G_{n,1/2}$ is chosen and then a clique of size k is randomly placed in the graph and we wish to find in this graph, denoted $G_{n,1/2,k}$, a maximum clique. Jerrum showed that the Metropolis process will not find the clique when $k = o(\sqrt{n})$. Kučera observed that when $k > c\sqrt{n \log n}$ for an appropriate constant c, the vertices of the planted clique would almost surely be the ones with the largest degrees in G, and hence it is easy to recognize them efficiently. Alon, Krivelevich and Sudakov [AKS98] showed an algorithm that almost surely finds the planted clique whenever $k \geq \Omega(\sqrt{n})$. Their algorithm is based on spectral properties of the graph, namely, it uses the eigenvector that corresponds to the second largest eigenvalue of the adjacency matrix of the graph.

Performance guarantees for heuristics. A major motivation for studying various probabilistic input models in general is to evaluate algorithms performance in real-life applications. It would be encouraging if we could rigorously show that really difficult instances are very rare, and we are more likely to encounter in practice a “solvable” instance. However, it is difficult to establish a connection between probabilistic input models and instances that occur in practice. For example, random graph models are usually highly regular (all vertices are of roughly the same degree). While most graphs indeed have this property, real-life instances not necessarily do. It is therefore desirable to have an algorithm which is effective on a wider range of instances.

One approach to enrich the class of solvable instances is to consider a *semi-random model*, in which the input is generated by a mixture of random and adversarial choices. Blum and Spencer [BS95] introduced two variants of the semi-random model. In one variant, an adversary makes its choices for the graph edges, but each of these choices is flipped with some small probability (“noise”). In the other variant, a random graph is chosen first, and then an adversary can modify this graph subject to some restrictions. This last variant of the semi-random model was formalized by Feige and Kilian [FK01a] as a *sandwich model.* First, two instances $G_{\min}$ and $G_{\max}$, both containing the same planted clique of size k, are generated using random decisions. Then, an adversary is allowed to choose any graph G^* which is sandwiched in between, i.e. $G_{\min} \subseteq G^* \subseteq G_{\max}$, where inclusion is with respect to edges.

The algorithm of Feige and Kilian [FK01a] finds a clique of linear size ($k = \Omega(n)$), in the following sandwich setting. $G_{\min}$ is the empty graph except a clique of size k. $G_{\max}$ is the complete graph except for roughly $n \log n$ random missing edges chosen from those edges connecting the (same) clique and the rest of the graph. An adversary then chooses $G_{\min} \subseteq G^* \subseteq G_{\max}$, and thus has complete control over the edges which are not adjacent to the clique vertices, and large control over the edges connecting the clique to the rest of the graph. The algorithm of [FK01a] uses semidefinite programming and matchings techniques and finds, almost surely, a clique of k vertices in the graph.

Since average-case algorithms do not have an a priori guarantee on their performance, it is important to certify that the algorithm is indeed successful on the particular instance at hand. Boppana [Bop87] shows an algorithm with such a certification property for the minimum bisection problem (see also [FK01a]). The algorithm outputs a bisection together with a lower bound on the size of the optimal bisection. The analysis shows that the output bisection and lower bound are equal, almost surely, in which case the algorithm proves the optimality of its output bisection.

We present an algorithm for finding a clique of size $k \geq \Omega(\sqrt{n})$ planted in a random graph $G_{n,1/2}$. Our algorithm improves over the algorithm of [AKS98] in two respects:

1. Extends to a semi-random model, where an adversary may remove edges from $G_{n,1/2,k}$ (the graph of [AKS98]), except for the edges forming the planted clique of size k. In the sandwich model terminology, let $G_{\max} = G_{n,1/2,k}$ be the graph of [AKS98], and let $G_{\min}$ be the empty graph except the same clique of size k. Then our algorithm finds a clique of size k in an arbitrary graph G^* sandwiched in between $G_{\min}$ and $G_{\max}$, almost surely over the distribution of $G_{\min}$ and $G_{\max}$. Observe that in this sandwich model, the vertices of the clique will not necessarily have higher degree, even if $k \gg \sqrt{n \log n}$, so also the algorithm of Kučera would not work in this model.

2. Certifies optimality of its solution. Using a semidefinite programming relaxation of the clique problem, the algorithm provides an upper bound on the size of the maximum clique in the graph. The upper bound matches, almost surely, the solution of the algorithm, proving that the clique output by the algorithm is the optimal one.

3.1.1 Semi-random model for the hidden clique problem

We study heuristics for the following sandwich model of the hidden clique problem. The semi-random graph G^* on n labeled vertices is constructed by a combination of random and adversarial decisions, as follows.

1. Random graph: for any pair of vertices i, j the edge (i, j) is placed in the graph with probability $1/2$. This gives the random graph $G_{n,1/2}$.

2. Planted clique: a subset Q of k vertices is chosen at random. Let $G_{\min}$ be the empty graph except a clique on the vertices of Q. Let $G_{\max}$ be the random graph $G_{n,1/2}$ with a planted clique on the vertices of Q (i.e. an edge is forced between every pair of vertices in Q). $G_{\max}$ is the $G_{n,1/2,k}$ graph of [AKS98].

3. Adversarial component: having complete knowledge of $G_{\max}$, an adversary may remove from the graph arbitrary edges except those which form the clique on Q (i.e. it is not allowed to remove edges both of whose endpoints are in Q). This gives the input graph G^* which is sandwiched in between $G_{\min}$ and $G_{\max}$.

Both graphs $G_{\min}, G_{\max}$ contain the same clique Q, and thus any sandwiched graph G^* must also have a clique on Q. An adversary can remove any of the edges of $G_{\max}$ outside the clique Q, so it has control over roughly half of all possible edges in the graph.

Observe that the algorithm is essentially required to output $G_{\min}$. The adversary is given $G_{\max}$ and may only remove edges, as long as G^* still contains $G_{\min}$. Thus, it may appear as if the adversarial moves only make the problem easier. Nevertheless, many algorithms that would recover a large clique in $G_{\max}$ would fail on G^*. A major motivation for the sandwich graph model is to identify those algorithms which are robust enough to withstand such an adversarial "help".

3.1.2 Relaxations of the problem

Lovász theta function. A well known relaxation of the independent set problem is the *theta* function of a graph $\vartheta(G)$, introduced by Lovász [Lov79] (see also [GLS93, Chapter 9] or Knuth's survey [Knu94]). The formulation of the theta function as a semidefinite program implies that, up to arbitrary precision, it can be computed in polynomial time, see e.g. [GLS93]. Note that the equivalence between the independent set problem and the clique problem through the edge complement graph $\overline{G}$, implies that $\vartheta(\overline{G})$ is an efficiently computable relaxation of the clique number $\omega(G)$.

In terms of approximation ratio, the theta function appears to have little to offer. The ratio between $\vartheta(\overline{G})$ and the clique $\omega(G)$ can be as large as $n^{1-o(1)}$, as shown in [Fei97]. Indeed, Hastad [Hås99] shows that no polynomial time computable function approximates $\omega(G)$ within a ratio of $n^{1-\epsilon}$, unless NP has random polynomial time algorithms.

Also on the average there is a gap between the Lovász theta function $\vartheta(\overline{G})$ and the clique $\omega(G)$. While the clique number of a random graph $G_{n,1/2}$ is almost surely roughly $2\log_2 n$, see e.g. [AS92], it is shown by Juhász [Juh82] that the value of the theta function is almost surely $\Theta(\sqrt{n})$.

Our approach is motivated by Juhász' result [Juh82] that the theta function of a random graph $G_{n,1/2}$ is $\Theta(\sqrt{n})$, almost surely. When a clique of size $k \geq c\sqrt{n}$, for a sufficiently large constant $c > 0$, is planted in a random graph, the theta function (being a relaxation) must increase to at least k. Furthermore, it is plausible that such a noticeable increase in the theta function will allow to find the planted clique. Indeed, we show that on this graph, which is the hidden clique graph $G_{n,1/2,k}$, the value of the theta function is almost surely exactly k, and then we can find (with some extra work) the planted clique.

In contrast, when a clique of size $k = o(\sqrt{n})$ is planted in a random graph, the monotonicity properties of the theta function, see e.g. [Knu94, Sections 18-19]), guarantee that its value can only increase, but not by more than k. It follows that on the hidden clique graph $G_{n,1/2,k}$, the value of the theta function is also almost surely $\Theta(\sqrt{n})$, and it is therefore possible that the planted clique has no noticeable effect on the theta function.

A possible direction for extending the above approach to a planted clique of smaller size $k = o(\sqrt{n})$, is to use a relaxation that is stronger than the Lovász theta function. In particular, it is desirable to obtain a relaxation whose value on a random graph $G_{n,1/2}$ is almost surely $o(\sqrt{n})$.

The general Lovász-Schrijver technique. Lovász and Schrijver [LS91] propose a general technique for obtaining stronger and stronger relaxations of 0-1 integer programming problems. They devise several procedures, called *matrix-cut operators*, that produce from a convex (e.g. linear programming) relaxation $P \subseteq [0,1]^n$ of the problem, a convex set that is an improved relaxation for the 0-1 vectors in P. That is, the resulting convex set is contained in P and contains all the 0-1 vectors in P. The matrix-cut operators follow a *lift-and-project* approach; they lift the convex relaxation P into a higher (quadratic) dimension by introducing new variables and new constraints, and project it back into the original space.

The two main matrix-cut operators of Lovász and Schrijver [LS91] are denoted by N and N_+. The difference between the two operators is that the lifting of the latter involves, in addition, a positive semidefinite constraint. That is, if P is a linear programming relaxation, then $N(P)$ is also a linear programming relaxation, while $N_+(P)$ is a semidefinite programming relaxation.

The matrix-cut operators can be applied iteratively, say $r \geq 0$ times, and the iterated operators are denoted N^r and N_+^r. The *N-rank of a convex relaxation P* is defined as the number of iterations of the N operator, that are needed to obtain the convex hull of the 0-1 vectors of P (i.e. a perfectly tight relaxation). The *N_+-rank* is defined similarly. Lovász and Schrijver [LS91] show that the N-rank of a relaxation is always at most the dimension (i.e. number of variables) d. The N_+ operator is a strengthening of the N operator, and hence also the N_+-rank is always at most d. Goemans and Tunçel [GT00] and Cook and Dash [CD00] show that there exist relaxations whose N_+-rank meets the upper bound d.

Furthermore, Lovász and Schrijver [LS91] show that the N and N_+ operators have the following important algorithmic property. If one can efficiently optimize (linear objective functions) over a relaxation P, then it is possible to efficiently optimize over the relaxation produced from P by the operator. It follows that for every *fixed* $r \geq 0$, the iterated operators N^r and N_+^r also satisfy this property.

Strong relaxations for maximum independent set. To obtain relaxations of the maximum independent set problem, Lovász and Schrijver [LS91] apply their general technique of matrix-cut operators on a classical linear programming relaxation FRAC of the problem. The relaxation FRAC is a linear program of polynomial size, and hence for every fixed $r \geq 0$, one can efficiently optimize over N_+^r(FRAC). In contrast, the dimension (i.e. number of variables) d of FRAC is the number of vertices n in the graph, and hence optimizing over N^n(FRAC) is NP-hard.

Lovász and Schrijver [LS91] show that the semidefinite programming relaxation N_+(FRAC) is at least as strong as the Lovász theta function. It follows, for example, that for any graph on which the theta function is not tight, the relaxation N_+^r(FRAC) for $r \geq 2$ is stronger than the theta function.

The *N-rank of a graph* is defined as the N-rank of the relaxation FRAC. The *N_+-rank* is defined similarly. It follows that for graphs with bounded N_+-rank, the stable set problem can be solved in polynomial time. This includes, in particular, perfect graphs, since their N_+-rank is at most 1 by the above connection to the theta function.

Stephen and Tunçel [ST99] study the case where the graph G on n vertices is the line graph of an h-vertex graph H. They show that the N_+-rank of G is at most $\lfloor h/2 \rfloor$, and that this bound is met if H is a complete graph on an odd number of vertices, in which case $n = \binom{h}{2}$ and the N_+-rank of G is $\Omega(\sqrt{n})$. Note that stable sets in G correspond to matchings in H, and that a maximum weight matching can be found efficiently; it follows that there are graphs with unbounded (and rather large) N_+-rank in which the (weighted) stable set problem can be solved in polynomial time.

3.1.3 Our results

We present an algorithm for the semi-random model of the hidden clique problem of Section 3.1.1. Our algorithm is based on the Lovász theta function, and its performance is summarized in the next theorem. Throughout, we say that an event occurs *almost surely* if its probability, over the distribution of $G_{\max}$, tends to 1 when $n \to \infty$. The adversarial component is, of course, always assumed to have the worst possible effect.

Theorem 3.1. *For any $k = \Omega(\sqrt{n})$, there is a polynomial time algorithm that, given a semi-random graph G^*, outputs, almost surely, a clique of size k together with a tight upper bound of k on the size of the largest clique in G^*.*

The key to the proof of Theorem 3.1 is the following lemma, that characterizes the value of the theta function on a random graph with a planted clique of a sufficiently large size k. Note that this graph is exactly the graph $G_{\max}$ of our semi-random model. Throughout, the term *with extremely high probability* will be used to denote a probability $1 - e^{-n^r}$ for some constant $r > 0$.

Lemma 3.1. *Let $G = G_{n,1/2,k} = G_{\max}$, where $k > c'\sqrt{n}$ for a large enough constant c'. Then with extremely high probability $\vartheta(\overline{G}) = k$.*

We also examine the asymptotic behavior on a random graph $G_{n,1/2}$ of relaxations of Lovász and Schrijver [LS91]] that are stronger than the theta function. In particular, we show that the typical value of the semidefinite programming relaxation $N_+^r(\text{FRAC})$ on a random graph is, loosely speaking, "roughly" $\sqrt{n/2^r}$ for $r = o(\log n)$. We note that this characterization answers (up to a constant factor) a question of Knuth [Knu94, Section 37,Problem P6].

Theorem 3.2. *For every fixed $\delta > 0$ and $r = o(\log n)$, the value of the relaxation $N_+^r(\text{FRAC})$ on a random graph $G_{n,1/2}$ is at least $\sqrt{n/(2+\delta)^{r+1}}$ and at most $4\sqrt{n/(2-\delta)^{r+1}}$, almost surely.*

Recall that the strongest relaxations of Lovász and Schrijver [LS91] whose value is known to be efficiently computable are $N_+^r(\text{FRAC})$ for $r = O(1)$. Theorem 3.2 shows that on a random graph, the typical value of these relaxations is smaller than that of the theta function by no more than a constant factor. In the hidden clique problem, the planted clique size k that a heuristic can handle can be improved by an arbitrarily large constant factor using a method of [AKS98], and therefore it appears that the improvement offered by these stronger relaxations can be achieved by other methods.

We use Theorem 3.2 to characterize, up to a constant factor, the typical N_+-rank of a random graph $G_{n,1/2}$.

Theorem 3.3. *The N_+-rank of a random graph $G_{n,1/2}$ is almost surely $\Theta(\log n)$.*

Our results for the N_+ operator extend to a somewhat stronger variant of the matrix-cut operators of Lovász and Schrijver [LS91]. This operator, that we denote by $N_{\mathrm{FR}+}$, is specialized for the maximum independent set problem and retains the important algorithmic property of N_+ that an efficient optimization over P implies an efficient optimization over $N_{\mathrm{FR}+}(P)$.

Organization. Section 3.2 proves Lemma 3.1, by using a known formulation of the theta function as an eigenvalue minimization problem. This formulation can be interpreted as duality of semidefinite programming, see e.g. [Ali95].

Section 3.3 gives a proof of Theorem 3.1, based on Lemma 3.1. In Section 3.4.2 we address the case where the size of the planted clique is $k > c'\sqrt{n}$ (where c' is as in Lemma 3.1), and show that a direct application of Lemma 3.1 gives an algorithm that recognizes, almost surely, the vertices of the planted clique in $G_{\max}$. In Section 3.3.2 we use an idea from [AKS98] to extend this algorithm to the case $k \geq c\sqrt{n}$ for $c < c'$. It will follow quite easily in Section 3.3.3 that our algorithm has two additional performance guarantees, which are the robustness against the sandwich model adversary and a certificate (almost surely) for the optimality of its solution. In Section 3.3.4 we discuss the extension of the algorithm to a random graph with edge probability different than 1/2.

Section 3.4 gives a technical description of the matrix-cut operators of Lovász and Schrijver [LS91] (including our variant $N_{\mathrm{FR}+}$), and is intended mainly to readers who are unfamiliar with these operators. We present the formal definitions in Section 3.4.1, collect some basic properties in Section 3.4.2, and review known bounds on the N-rank and N_+-rank in Section 3.4.3.

Section 3.5 describes our results on matrix-cuts in a random graph. The lower bound on the value of the relaxation $N_+^r(\mathrm{FRAC})$ is shown in Section 3.5.1, and the upper bound is shown in Section 3.5.2.

Preliminaries. Throughout, we omit the graph $G(V, E)$ if it is clear from the context. We let n denote the number of vertices in the graph G, and assume, without loss of generality, that $V = \{1, \ldots, n\}$. For a vertex i in the graph, let $\Gamma(i)$ denote the set of the vertices that are adjacent to i in the graph, i.e. $\Gamma(i) : \{j : ij \in E\}$, and let $\Gamma(S)$ denote the set of vertices that are adjacent to at least one vertex of S, i.e. $\Gamma(S) := \cup_{i \in S}\Gamma(i)$.

An $n \times n$ (real) matrix Y is *positive semidefinite* if Y is symmetric and $x^T Y x \geq 0$ for all $x \in \mathbb{R}^n$. It is well-known that a symmetric matrix Y is positive semidefinite if and only if all the eigenvalues of Y are nonnegative.

A *Gram matrix representation* of an $n \times n$ matrix Y is a set of real-valued vectors $\{v_1 \ldots, v_n\}$ such that $Y_{ij} = v_i^T v_j$ for all i, j. It is well-known that a matrix Y is positive semidefinite if and only if it has a Gram matrix representation.

3.2 The theta function in a hidden clique graph

In this section we prove Lemma 3.1. Throughout this section, let us denote by G the graph $\overline{G_{\max}}$, which can be equivalently described as a random graph $G_{n,1/2}$ with a planted independent set on k randomly chosen vertices. Since G has a (planted) independent set of size k and the theta function is a relaxation of the independent set problem, $\vartheta(G) \geq \alpha(G) \geq k$.

The main part is to show the other direction, i.e. that with extremely high probability $\vartheta(G) \leq k$. The theta function has several equivalent formulations (cf. [Lov79, GLS93, Knu94]. We will use the formulation as an eigenvalue minimization problem:

$$\vartheta_2(G) = \min_M \{\lambda_1(M)\}$$

where M is an $n \times n$ real symmetric matrix with $M_{ij} = 1$ whenever vertices i, j are non-adjacent in G, and $\lambda_i(M)$ denotes the i-th largest eigenvalue of the matrix M. We derive an upper bound on $\vartheta(G)$ by "guessing" a particular matrix M, and clearly

$$\vartheta_2(G) \leq \lambda_1(M) \tag{3.1}$$

So it suffices to show that with extremely high probability, our choice of M is such that $\lambda_1(M) \leq k$.

Assume, without loss of generality, that the vertices of the planted independent set are the first k coordinates. Then our chosen M looks like

$$M = \left[\begin{array}{c|c} J_k & B^t \\ \hline B & C \end{array}\right] = \left[\begin{array}{c|cccc} J_k & & 1/-1+x_j & & \\ \hline & 1 & & & 1/-1 \\ & & 1 & & \\ 1/-1+x_i & & & \ddots & \\ & 1/-1 & & & 1 \end{array}\right]$$

where J_k is the all ones matrix of order k, B^t is the transpose of B, and B, C are as follows. Roughly half of the entries of B and C have to be $+1$ because they correspond to pairs of non-adjacent vertices in the graph G. We set all other entries of C to be -1. The remaining entries of B are chosen so that the sum of each row of B is 0. At each row, we choose all entries to be equal. Formally, we define the entries of C to be $c_{ij} = 1$ if $(i, j) \notin E$ and $c_{ij} = -1$ otherwise (for all $k < i, j \leq n$). B is defined by $b_{ij} = 1$ if $(i, j) \notin E$ and $b_{ij} = -1 + x_i$ otherwise (for all $i > k$ and $j \leq k$), where $x_i = (2deg(x_i, Q) - k)/deg(x_i, Q)$, and $deg(x_i, Q)$ denotes the number of vertices in Q which are adjacent to x_i in the graph.

It is easy to see that k is an eigenvalue of M. By our choice of B the vector with 1 in its first k entries and 0 otherwise is an eigenvector whose eigenvalue is k. In order to prove that this vector corresponds to the largest eigenvalue, i.e. $\lambda_1(M) = k$, it suffices to show that $\lambda_2(M) < k$.

Let us describe M as the sum of three matrices $M = U + V + W$. the three matrices are random but correlated, as follows.

U is a random 1/-1 symmetric matrix, with 1 on the diagonal.

V is a random 0/2 symmetric matrix in the upper left $k \times k$ block (with 0 in the diagonal), and 0 otherwise. The matrix V is correlated with U by $v_{ij} = 1 - uij$ for all $1 \leq i, j \leq k$.

W is the correction matrix for having the row sums of B equal to 0.

$$U = \begin{bmatrix} 1 & & 1/-1 \\ & 1 & \\ & & \ddots & \\ 1/-1 & & & 1 \end{bmatrix} \qquad V = \left[\begin{array}{c|c} 0/2 & 0 \\ \hline 0 & 0 \end{array}\right] \qquad W = \left[\begin{array}{c|c} 0 & 0/x_j \\ \hline 0/x_i & 0 \end{array}\right]$$

From [FK81] we know that with probability $1 - e^{-n^r}$ for some constant $r > 0$:

$$\forall i \geq 1, |\lambda_i(U)| \leq c_1\sqrt{n} \qquad \forall i \geq 2, |\lambda_i(V)| \leq c_2\sqrt{k}$$

By the Weyl theorem (cf. [HJ85, page 181]),

$$\lambda_2(M) \leq \lambda_1(U) + \lambda_2(V) + \lambda_1(W) \leq c_1\sqrt{n} + c_2\sqrt{k} + \lambda_1(W) \tag{3.2}$$

To bound $\lambda_1(W)$, it suffices to bound $Tr(W^2)$ because

$$Tr(W^2) = \sum_i \lambda_i(W^2) = \sum_i (\lambda_i(W))^2 \geq (\lambda_1(W))^2 \tag{3.3}$$

Since W is symmetric $Tr(W^2) = \sum_i (W_i W_i^t) = \sum_{i,j} W_{ij}^2 = 2\sum_{i<j} W_{ij}^2$. Look at the i-th row of B and the corresponding row of W. Let us denote by S_i (for $k < i \leq n$) the number of non-zero (i.e. x_i) entries in this row of W, i.e. $S_i = deg(i, Q)$. Then $Tr(W^2) = 2\sum_{i=k+1}^n S_i x_i^2$. Recall x_i was chosen so that the corresponding row sum in B would be zero, so

$$x_i = (2S_i - k)/S_i$$

Since $S_i = deg(i, Q)$ is binomially distributed $S_i \sim B(k, 1/2)$, then $k = 2ES_i$ and we get

$$Tr(W^2) = 2\sum_{i=k+1}^{n} S_i((2S_i - k)/S_i)^2 = 8\sum_{i=k+1}^{n} (S_i - ES_i)^2/S_i \tag{3.4}$$

The following Lemma allows us to bound these quantities.

Lemma 3.2. *With extremely high probability (at least $1 - e^{-n^r}$ for some constant $r > 0$)*

1. $\sum_{i=k+1}^n (S_i - ES_i)^2 \leq k^3/96$.
2. $S_i \geq k/3$.

Proof. 1. We shall bound $Y = \sum_{i=k+1}^{n}(S_i - ES_i)^2$ by Azuma's inequality, cf. [AS92]. Let us first compute its expectation:

$$E(S_i - ES_i)^2 = Var(S_i) = k/4$$

$$EY = (n-k)k/4 \leq nk/4$$

Let $Y_0, Y_1, Y_2, \ldots, Y_{k(n-k)} = Y$ be a Doob martingale of Y defined by exposing one by one each of the $k(n-k)$ Bernoulli trials (recall $S_i \sim B(k, 1/2)$). Let us now bound the Martingale difference. To see how an exposure of a single Bernoulli trial can affect the final result Y, assume that all other Bernoulli trials have been fixed. The sum $Y = \sum_i (S_i - ES_i)^2$ is then fixed except for the contribution of a single S_i (which includes the yet unexposed Bernoulli trial). When that trial is exposed, the contribution of the corresponding S_i will be either $(v - ES_i)^2$ or $(v + 1 - ES_i)^2$, for some value $0 \leq v \leq k/2 - 1$. The difference between the two values is maximized when $v = 0$ or $v = k/2 - 1$. The Martingale difference is thus bounded by $\Delta = (\frac{k}{2})^2 - (\frac{k}{2} - 1)^2 = k - 1$. By Azuma's inequality, for any $\lambda > 0$

$$Pr\left[|Y_{k(n-k)} - EY_{k(n-k)}| \geq \lambda\Delta\sqrt{k(n-k)}\right] \leq 2e^{-\lambda^2/2}$$

Take $\lambda = n^{1/4}$ to get

$$Pr\left[|Y - EY| \geq n^{1/4}k\sqrt{k(n-k)}\right] \leq e^{-\sqrt{n}/4}$$

Since $EY \leq nk/4$ and $k > c'\sqrt{n}$ for a sufficiently large constant $c' > 0$ we conclude that with extremely high probability, $Y \leq EY + n^{3/4}k^{3/2} \leq k^3/4c'^2 + k^3/c'^{3/2} \leq k^3/96$, as claimed.

2. Follows immediately from Chernoff bound:

$$Pr[S_i \leq k/3] = Pr[S_i \leq 2/3 \cdot ES_i] \leq e^{-k/36} \leq e^{-c'\sqrt{n}/36}$$

□

Using (3.3),(3.4) and Lemma 3.2 we get that with extremely high probability

$$(\lambda_1(W))^2 \leq Tr(W^2) < 8 \cdot k^3/96 \cdot (3/k) = k^2/4$$

and using (3.1),(3.2) we arrive at

$$\lambda_2(M) \leq \lambda_1(U) + \lambda_2(V) + \lambda_1(W) < c_1\sqrt{n} + c_2\sqrt{k} + k/2 < k$$

as claimed.

3.3 Algorithm for the hidden clique problem

In this section we prove Theorem 3.1 and describe its algorithm, extensively relying on the result of Lemma 3.1. We shall start with the strictly random model, in which the input graph is $G_{\max}$ and there is no adversary. First (Section 3.4.2) we address the basic case of $k > c'\sqrt{n}$ for a large enough constant c', where we can easily use Lemma 3.1. Then (Section 3.3.2) we improve the result to any $k = \Omega(\sqrt{n})$. We use the approach of [AKS98] of guessing a fixed number of vertices from the planted clique, in order to reduce the problem to the basic case. Finally (Section 3.3.3), we show how our analysis easily extends to the semi-random model with the exact same algorithm.

The monotonicity properties of the theta function, cf. [Knu94, Sections 18-19], are used throughout this section. Specifically, addition (or removal) of an edge (or a vertex) from a graph has a monotone effect on the theta function, similarly to the independence number $\alpha(G)$. For example, adding (resp. removing) an edge may only decrease (resp. increase) the theta function.

Observe that in all cases Q is almost surely the unique maximum clique. Indeed, any clique containing a vertex from $V \setminus Q$, almost surely contains at most $(1+o(1))k/2$ vertices from Q, and at most $2\log n$ vertices from $V \setminus Q$, which is altogether much smaller that $|Q|$.

3.3.1 The basic case

In the basic case we assume that the input graph is $G_{\max} = G_{n,1/2,k}$ (so there is no adversary), and assume also that $k > c'\sqrt{n}$ for a large enough constant c'.

Finding the clique vertices can then be performed by testing separately each vertex v to see whether it belongs to the planted clique or not. To test if v belongs to the planted clique, remove v from the graph $G_{\max}$ to get the graph $G_{\max} \setminus v$, and check how this removal affects the theta function. We can analyze $G_{\max} \setminus v$ by using the principle of deferred decisions. If the vertex v belongs to the planted clique, then $G_{\max} \setminus v$ has a clique of size $k-1$ in an otherwise random graph. If v does not belong to the planted clique, then $G_{\max} \setminus v$ has a clique of size k in an otherwise random graph.

Applying Lemma 3.1 on $G_{\max}$ and on $G_{\max} \setminus v$ for all vertices v, and using the union bound we get the following observation.

Observation. With extremely high probability, $\vartheta(\overline{G_{\max}}) = k$ and

$$\vartheta(\overline{G_{\max} \setminus v}) = \begin{cases} k-1 & \text{if } v \text{ belongs to the planted clique;} \\ k & \text{otherwise;} \end{cases}$$

This suggests the following simple algorithm for the basic case:

```
Algorithm BasicFind.
Input: Graph $G = G_{\max}$ where $k > c'\sqrt{n}$ for a large enough constant $c'$.

  1. $P \leftarrow \{v : \vartheta(\overline{G \setminus v}) < \vartheta(\overline{G}) - 1/2\}$

  2. Output $P$ and $\vartheta(\overline{G})$.
```

The discussion above shows that with extremely high probability, the output P of algorithm **BasicFind** is the planted clique Q of size k, and $\vartheta(\overline{G}) = |P|$, proving its optimality.

Improved Algorithm

The same observation leads to a more efficient algorithm, which uses only one computation of the theta function. The theta function has several equivalent formulations, cf. [Lov79, GLS93, Knu94], but we will use a particular geometric maximization form, described as ϑ_4 in [GLS93, Chapter 9.3], as follows. An *orthonormal representation* of G is a sequence of unit length vectors $\{u_i \in \mathbb{R}^n : i \in V\}$, such that $u_i \cdot u_j = 0$ whenever i, j are non-adjacent vertices. Then

$$\vartheta_4(G) = \max_{d,\{u_i\}} \sum_{i \in V} (d \cdot u_i)^2$$

where $d \in \mathbb{R}^n$ ranges over all vectors of unit length, and $\{u_i \in \mathbb{R}^n : i \in V\}$ is an orthonormal representation of $\overline{G}$.

Using semidefinite programming, it is possible to solve $\vartheta_4(G)$ within arbitrary small additive error, and arrive at a corresponding vector d and an orthonormal representation $\{u_i\}$. (More precisely, formulation $\vartheta_3(G)$ describes a semidefinite program, whose solution can be efficiently transformed to an equivalent orthonormal representation for $\vartheta_4(G)$. See the above references for details).

Suppose now that we solve $\vartheta_4(G)$ for our input graph $G = \overline{G_{\max}}$, and we get a solution $d, \{u_i\}$. We claim that the k vertices whose contribution $(d \cdot u_i)^2$ is the largest, are the vertices of the clique, with extremely high probability. This, of course, will enable us to recognize the vertices of the planted clique with only one such computation of the theta function.

To prove the claim, assume that the result of the above observation regarding $\vartheta(\overline{G_{\max}})$ and $\vartheta(\overline{G_{\max} \setminus v})$ indeed holds (which happens with extremely high probability). Then $\vartheta(\overline{G_{\max}}) = k$, and thus the solution $d, \{u_i\}$ we get from the semidefinite programming has a value of at least $k - \epsilon$ (for arbitrary small fixed $\epsilon > 0$). In this solution, the contribution of every vertex of the planted clique, $q \in Q$, is at least $1 - \epsilon$, or otherwise, the same orthonormal representation would give that $\vartheta_4(\overline{G_{\max} \setminus q}) \geq \sum_{i \neq q} (d \cdot u_i)^2 > k - 1$, contradicting the above observation. However, the contribution of any vertex $j \in V \setminus Q$ is bounded, because j is (with extremely high probability) a neighbor of some $q \in Q$, so $u_j \cdot u_q = 0$ and therefore $(d \cdot u_j)^2 + (d \cdot u_q)^2 \leq 1$, and $(d \cdot u_j)^2 \leq \epsilon$. This shows that the contribution of every vertex $q \in Q$ is larger than that of vertex $j \in V \setminus Q$, as claimed.

As an estimate for $k = |Q|$ we can take the value we get from the semidefinite programming for $\vartheta_4(\overline{G_{\max}})$, rounded to the nearest integer. This gives the following algorithm, which outputs $P = Q$ and $\vartheta = k$ with extremely high probability.

```
Algorithm ImprovedBasicFind.
Input: Graph $G = G_{\max}$ where $k > c'\sqrt{n}$ for a large enough constant $c'$.

  1. Compute $\vartheta \leftarrow \vartheta_4(\overline{G})$ within small additive error $\epsilon = 1/3$, together with
     a corresponding orthonormal representation $\{u_i\}$ and its handle $d$.

  2. $P \leftarrow \{i : (d \cdot u_i)^2 > 1/2\}$.

  3. Output $P$ and $\vartheta$.
```

3.3.2 Smaller values of k

Following [AKS98], the main idea in improving the algorithm for $G_{\max}$ to any $k = \Omega(\sqrt{n})$ is to guess a constant number of vertices from the planted clique. We can then work on the subgraph induced on those vertices which are common neighbors to all of our guess set. The induced subgraph is much smaller the $G_{\max}$, and is random except for the planted clique of size k. Thus we improve the ratio between the size of the clique and the size of the graph by a constant factor, and can use the algorithm of the basic case (either **BasicFind** or **ImprovedBasicFind** of Section 3.3.1).

Guessing a constant number of vertices from the graph can be replaced by an exhaustive search on a polynomial number of possibilities. Let $\hat{N}(S)$ denote the set of vertices neighboring to all of S. The algorithm for $k \geq c\sqrt{n}$ with arbitrary fixed $c > 0$ is as follows.

```
Algorithm **FindClique.**
Input: Graph $G = G_{\max}$ where $k \geq c\sqrt{n}$.

  1. $s \leftarrow 2\lceil \log_2(c'/c) \rceil + 3$

  2. For all subsets $S$ of $s$ vertices, do

      2.1. $V_1 \leftarrow \hat{N}(S)$.
      2.2. $(P_S, \vartheta_S) \leftarrow$ **BasicFind**(the subgraph induced on $V_1$)
      2.3. End-for.

  3. Output the set $S \cup P_S$ which is a clique in $G$ and has maximum size
     over all choices of $S$.

  4. Output $\vartheta = s + \max_S \vartheta_S$.
```

We claim that for any fixed c, algorithm **FindClique** almost surely outputs the planted clique Q and a tight upper bound $\vartheta = k$. First observe that for any fixed subset S of size s, the cardinality of $\hat{N}(S)$ in the random graph $G_{n,1/2}$ is a binomially distributed random variable with parameters $n - s$ and $1/2^s$. Thus, almost surely, $|\hat{N}(S)| = (1 + o(1))n/2^s$ for all subsets of vertices of size s in $G_{n,1/2}$. Planting a clique of size k can increase $|\hat{N}(S)|$ by at most k. Therefore, for all S, $|\hat{N}(S)| = (1 + o(1))n/2^s$ almost surely also in $G_{\max}$.

Since algorithm **FindClique** checks all possible subsets S of size s, in some step it will reach some S which is a subset of the planted clique Q. At this iteration we almost surely find the planted clique $Q = S \cup P_S$. Indeed, by the principle of deferred decisions, the subgraph induced on $\hat{N}(S)$ is a random graph $G_{|\hat{N}(S)|,1/2}$ with a planted clique of size $k - s$. By our choice of s, the planted clique size satisfies $k - s \geq c\sqrt{n}/2 \geq c'\sqrt{2n/2^s} \geq c'\sqrt{|\hat{N}(S)|}$. Thus, algorithm **BasicFind** will almost surely find the planted clique $Q \setminus S$ in $\hat{N}(S)$, and as a result we will find the planted clique of size k in $G_{\max}$.

With extremely high probability, for all subsets S of size s checked by the algorithm **FindClique**, the theta function of the subgraph induced on $\hat{N}(S)$ is at most $k - s$, i.e. $\vartheta_S \leq k - s$ for all S. Indeed, for each subset S, either $S \subset Q$ or S contains a vertex not from

Q. In the first case the subgraph induced on $\hat{N}(S)$ is a random graph with a planted clique of size at most $k-s$. In the second case, with extremely high probability $\hat{N}(S)$ contains only $(1+o(1))k/2 < k-s$ vertices of the planted clique Q. In either case, using the monotonicity properties, we may assume $\hat{N}(S)$ contains exactly $k-s$ vertices of the planted clique Q, and by Lemma 3.1, with extremely high probability the theta function of the induced subgraph is at most $k-s$. Using the union bound on polynomially many choices of S, the event that all these theta functions will be at most $k-s$ takes place with extremely high probability.

3.3.3 A sandwiched graph G^*

To show that algorithm **FindClique** is robust enough to withstand a monotone adversary of the sandwich model, we argue that an adversary cannot prevent the algorithm from succeeding. Thus, the proof for the strictly random model $G_{\max}$ extends to the semi-random model G^*.

For simplicity, consider first the basic case where $k > c'\sqrt{n}$ for a large enough constant c'. Let G^* be an arbitrary sandwiched graph, i.e. $G_{\min} \subset G^* \subset G_{\max}$. Then by the monotonicity properties of the theta function,

$$k \leq \vartheta(\overline{G_{\min}}) \leq \vartheta(\overline{G^*}) \leq \vartheta(\overline{G_{\max}}) \leq k$$

where the last inequality holds, almost surely, by our previous analysis. A similar phenomenon happens when we remove one vertex, i.e. also in $\vartheta(\overline{G_{\max} \setminus v})$, as in the observation of Section 3.3.1. This shows that an adversary cannot prevent the algorithm from finding the planted clique nor from certifying the optimality of its solution.

In the general case where $k \geq \Omega(\sqrt{n})$, we consider the possible effect of an adversary in any of the applications of the theta function in algorithm **FindClique**. These applications are equivalent to the removals of one vertex in the observation of Section 3.3.1 (which are used in the proof of algorithm **ImprovedFindClique**).

The algorithm **FindClique** only applies the theta function on induced subgraphs of the input graph G^*. Let H be any such induced subgraph of the corresponding $G_{\max}$. It follows from the principle of deferred decisions, that the induced subgraph H is a random graph (with edge probability 1/2) with a randomly planted clique of size $k' \leq k$. Consider the effect of an adversary on this application of the theta function, i.e. on $\vartheta(\overline{H})$. Recall that an adversary is only allowed to remove edges. Thus, its effect is limited to either removing some vertices from H (by reducing $\hat{N}(S)$), or removing some edges from H (by removing the same edges from $G_{\max}$).

On the one hand, these operations may only decrease the theta function, $\vartheta(\overline{H})$, by the aforementioned monotonicity properties. On the other hand, whenever the theta function is tight on H due to its planted clique, (i.e. $\vartheta(\overline{H}) = k'$), an adversary cannot affect the theta function because it cannot remove any of the edges forming the planted clique, and H must still contain a clique of size k'. Note that these are exactly the properties used in the analysis of algorithm **FindClique**.

Hence, an adversary cannot prevent the algorithm from finding the planted clique of size k, and cannot increase the upper bound ϑ, which thus must remain k. Overall, whenever algorithm **FindClique** succeeds on $G_{\max}$, it also succeeds on an arbitrary sandwiched G^*, regardless of the adversary operations, as claimed.

3.3.4 Extension to other edge probabilities

The approach that we present can be generalized to find a hidden clique in a random graph $G_{n,p}$ where the edge probability p is different than $1/2$.

For every fixed p, and thus fixed $q = 1 - p$, the main result holds, except, maybe, for a change in the (polynomial) running time of the algorithm. That is, a hidden clique of size $c\sqrt{n}$ can be found, for every fixed $c > 0$. Indeed, it is known from [Juh82] that with extremely high probability $\vartheta(\overline{G_{n,p}}) = \vartheta(G_{n,q}) = \Theta(\sqrt{np/q})$. To find in a $G_{n,p}$ graph a hidden clique of size $k \geq c'\sqrt{np/q}$, for a sufficiently large constant $c' > 0$ (and thus extend Lemma 3.1), one can take the matrix M of our proof in the spirit of [Juh82]. To finish the argument, observe that the idea from [AKS98] of guessing a constant number of vertices allows handling a hidden clique of size $c\sqrt{n}$ for arbitrary fixed $c > 0$, and that the sandwich properties of the theta function give robustness against a monotone adversary.

When $p = o(1)$ finding hidden cliques becomes easier. For example, if $p = 1/n^{\delta}$ for a fixed $\delta > 0$, it is possible to find a hidden clique of size $O(1/\delta) << n^{(1-\delta)/2} \simeq \sqrt{np/q}$ by exhaustively trying all subsets of this size in the graph. Indeed, with high probability the maximum clique in the random graph is of size $O(1/\delta)$, and can thus be found in polynomial time by exhaustive search. If a clique of size c'/δ is planted in the graph, for a sufficiently large constant $c' > 0$, then with high probability it will be the unique maximum clique in the graph, and can similarly be found in polynomial time.

When $q = o(1)$ finding a hidden clique becomes more difficult. For example, the idea of [AKS98] of guessing a constant number of vertices in the hidden clique has only a negligible effect in reducing the size of the graph. Nevertheless, we believe that our analysis of the algorithm based on the theta function can be extended to work for a large range of values of $q = o(1)$, finding cliques of size $c'\sqrt{n/q}$ for sufficiently large c'. We remark that the case of extremely small q, namely $q = \frac{c \log n}{n}$ for a sufficiently large $c > 0$, was handled in [FK01a] (in a model that is more adversarial than the one studied here), where it was shown how to find hidden cliques of linear size $k = \Omega(n)$. Note that this value of q is larger by a factor of $\log n$ then the one which a general bound of $c'\sqrt{n/q}$ would have required for a linear sized clique. It appears to us that the loss of the $\log n$ factor is unavoidable for the semi-random graph model when q is so small (for reasons that are explained in [FK01a]), but can perhaps be avoided in a random graph model that involves no adversary.

3.4 The matrix-cut operators of Lovász and Schrijver

In this section we describe several lift-and-project operators proposed by Lovász and Schrijver [LS91]. They called them matrix-cut operators. When given a convex set (e.g. a polytope) P, these operators consider it as a relaxation of the convex hull of its 0-1 vectors, and produce another relaxation that is tighter. In other words, these operators produce a convex set that is sandwiched (in terms of containment) between P and (the convex hull of) its 0-1 vectors. Furthermore, the produced relaxation is strictly tighter than P, (unless P is already tight).

For completeness, we review in this section the definitions of these operators, many of their properties (that we need) and relevant known results. We may repeat some known proofs and examples, partly in order to extend them to a more general setting that includes the $N_{\mathrm{FR}+}$ operator, and partly to aid readers who are unfamiliar with these operators. We mention properties that hold for general 0-1 optimization, and focus on the stable set problem. In Section 3.4.1 we describe the required framework and present the definitions of the matrix-cut operators. In Section 3.4.2 we collect some basic properties of these operators. In Section 3.4.3 we describe known techniques to evaluate the effectiveness of these operators, and known results. Lovász gives an alternative formulation of the matrix-cut operators in [Lov94].

Our notation mostly follows that of Lovász and Schrijver [LS91]. Throughout, let e_j be the jth unit vector, let $\mathbf{0}$ be the vector of all zeros, and let $\mathbf{1} = \sum_j e_j$ be the vector of all ones. The sizes (dimensions) of $\mathbf{0}, \mathbf{1}$ and e_j will be clear from the context.

Recall that a set is called a *cone* if it is closed under multiplication by a nonnegative number. A *convex cone* is thus a set that is closed under a nonnegative linear (a.k.a. *conic*) combination. (Throughout, we will consider convex cones rather than polytopes.) A *polyhedral cone* is a cone that is also a polyhedron; equivalently, a polyhedral cone is a set that can be defined by $\{x : Ax \geq 0\}$ for some matrix A.

3.4.1 Definitions

Homogenization. It will be convenient to deal with homogenous systems of inequalities. We therefore embed the n-dimensional space $\mathbb{R}^n$ in $\mathbb{R}^{n+1}$ as the hyperplane $x_0 = 1$ (throughout, the 0th variable plays a special role), and work with convex cones in $\mathbb{R}^{n+1}$, as follows.

Since we deal with 0-1 programming on n variables, our basic example is a polytope P that is contained in $[0,1]^n$ (the convex hull of the n-dimensional hypercube $\{0,1\}^n$). To homogenize P using the new variable x_0, first embed P in the hyperplane $x_0 = 1$ of $\mathbb{R}^{n+1}$, and then generate from it a convex cone. That is, if

$$P = \{x \in \mathbb{R}^n : Ax \leq b,\ \mathbf{0} \leq x \leq \mathbf{1}\}, \tag{3.5}$$

then the convex cone obtained by homogenization is

$$K := \left\{ \begin{pmatrix} x_0 \\ x \end{pmatrix} \in \mathbb{R}^{n+1} : Ax \leq x_0 b,\ 0 \leq x \leq x_0 \mathbf{1} \right\}. \tag{3.6}$$

Note that such K can be described as the intersection of finitely many linear constraints $u^t x \geq 0$ (here $x \in \mathbb{R}^{n+1}$), and hence it is a polyhedral cone.

We denote by $Q \subset \mathbb{R}^{n+1}$ the convex cone that is obtained from the polytope $[0,1]^n$ via the homogenization procedure (3.5)-(3.6). Namely,

$$Q := \left\{ (x_0, x_1, \ldots, x_n)^T : 0 \leq x_i \leq x_0 \text{ for all } 1 \leq i \leq n \right\}. \tag{3.7}$$

Note that Q is a polyhedral cone that can be described by $2n$ linear inequalities.

Throughout, let $K \subseteq Q$ be a (closed) convex cone. We denote by K_I the convex cone that is generated by all 0-1 vectors in K. Observe that within the hyperplane $x_0 = 1$, K_I is exactly the *integral hull* (i.e. convex hull of the integral vectors) of K. For example, $Q_I = Q$.

The *polar cone* of K, denoted K^*, is the convex cone defined by

$$K^* := \{u \in \mathbb{R}^{n+1} : x^T u \geq 0 \text{ for all } x \in K\}.$$

Observe that a vector $u \in K^*$ corresponds to a linear constraint $u^T x \geq 0$ that is *valid* for K (i.e. satisfied by all vectors $x \in K$). The polar cone K^* is thus the collection of valid linear constraints for K. For example, Q is defined in (3.7) by $2n$ linear constraints, and hence Q^* is spanned by the vectors e_i and $f_i = e_0 - e_i$, for $i = 1, \ldots, n$.

Fractional stable sets. We will be mostly intereted in the stable set problem. Let $G(V, E)$ be a graph with no isolated vertices and $|V| = n$. Then the stable sets of G correspond to the 0-1 solutions of the system of linear inequalities

$$x_i \geq 0 \quad \text{for all } i \in V \qquad \textit{(nonnegativity constraints)} \tag{3.8}$$

and

$$x_i + x_j \leq 1 \quad \text{for all } ij \in E \qquad \textit{(edge constraints)} \tag{3.9}$$

Let STAB$(G) \subset \mathbb{R}^n$ denote the convex hull of the 0-1 solutions of the system (3.8)-(3.9). Let FRAC$(G) \subset \mathbb{R}^n$ (for "fractional stable sets") denote the solution set of the system (3.8)-(3.9) (i.e. without integrality restriction). Clearly, STAB$(G) \subseteq$ FRAC(G).

Let FR$(G) \subset \mathbb{R}^{n+1}$ be the polyhedral cone that is obtained from the polytope FRAC(G) via the homogenization procedure (3.5)-(3.6). That is, FR(G) is the solution set of the following homogenous system of linear inequalities for the stable set problem:

$$x_i \geq 0 \quad \text{for each } i \in V \tag{3.10}$$

$$x_0 - x_i - x_j \geq 0 \quad \text{for each } ij \in E \tag{3.11}$$

Let ST(G) be the polyhedral cone that is obtained from the polytope STAB(G) via the homogenization procedure (3.5)-(3.6). It is straightforward that $(\text{FR}(G))_I = \text{ST}(G)$.

Throughout, we omit the graph G when it is clear from the context, denoting STAB(G) by STAB etc. It can be seen that its polar cone FR* is spanned by the vectors e_i for $i = 1, \ldots, n$ and the vectors $f_{ij} = e_0 - e_i - e_j$ for $ij \in E$. Note that FR $\subset Q$ and hence FR$^* \supseteq Q^*$.

Matrix-cut operators. Let $K_1, K_2 \subseteq Q$ be closed convex cones in $\mathbb{R}^{n+1}$ (e.g. $K_1 =$ FR(G) and $K_2 = Q$). Consider the cone $K_1 \cap K_2$. For each $u \in K_1^*$ the constraint $u^T x \geq 0$ is valid for K_1, and for each $v \in K_2^*$ the constraint $v^T x \geq 0$ is valid for K_2. It follows that the quadratic inequality $(u^T x)(x^T v) \geq 0$ is valid for $K_1 \cap K_2$. Furthermore,

$$K_1 \cap K_2 = \Big\{x : u^T x x^T v \geq 0 \text{ for all } u \in K_1^*, v \in K_2^*, x_0 \geq 0\Big\} \tag{3.12}$$

because any original inequality, say $u^T x \geq 0$ for K_1, can be recovered by adding the two quadratic inequalities obtained by $e_i, f_i \in Q^* \subseteq K_2^*$, giving $u^T x \cdot x_0 = u^T x x^T (e_i + f_i) \geq 0$.

Furthermore, all 0-1 vectors in $K_1 \cap K_2$ satisfy $x_i^2 = x_i$. Therefore, if x is a 0-1 vector in $K_1 \cap K_2$ and with $x_0 = 1$, then setting $Y = xx^T$ we have that

(a) Y is symmetric.

(b) $Ye_0 = diag(Y)$, i.e. $Y_{ii} = Y_{i0}$ for all $1 \leq i \leq n$.

(c) $u^T Y v \geq 0$ for all $u \in K_1^*$ and $v \in K_2^*$.

(d) Y is positive semidefinite.

Note that (c) can be written as

(c') $YK_2^* \subseteq K_1$

Lovász and Schrijver [LS91] proposed the following lift-and project procedure. Given K_1, K_2, consider the derived cones:

$$M(K_1, K_2) := \{Y \in \mathbb{R}^{(n+1)\times(n+1)} : Y \text{ satisifies (a)-(c)}\}$$

$$M_+(K_1, K_2) := \{Y \in \mathbb{R}^{(n+1)\times(n+1)} : Y \text{ satisifies (a)-(d)}\}$$

and define the projections of these liftings on $\mathbb{R}^{n+1}$:

$$N(K_1, K_2) := \{Ye_0 : Y \in M(K_1, K_2)\}$$

$$N_+(K_1, K_2) := \{Ye_0 : Y \in M_+(K_1, K_2)\}.$$

It follows from the above discussion that

$$(K_1 \cap K_2)_I \subseteq N_+(K_1, K_2) \subseteq N(K_1, K_2) \subseteq K_1 \cap K_2 \tag{3.13}$$

Relevant variants of the operators. We use shorter notation to easily handle two special cases. When $K_2 = Q$ we omit K_2, i.e. $N(K) := N(K, Q)$ and $N_+(K) := N_+(K, Q)$. In this case, we have that (c') is equivalent to:

(c") Every column of Y is in K_1; the difference of the first column and any other column of Y is in K_1.

Note that we have from (3.13) that

$$K_I \subseteq N_+(K) \subseteq N(K) \subseteq K \tag{3.14}$$

For the stable set problem, we may take $K_2 = \text{FR}$, which we denote in the subscript, i.e. $N_{\text{FR}}(K) := N(K, FR)$ and $N_{\text{FR}+}(K) := N_+(K, FR)$. In this case, we have that (c') is equivalent to:

(c"') $Ye_i \in K_1$ for all $i \geq 1$, and $Yf_{ij} \in K_1$ for all $ij \in E$.

We assume throughout that $K \subseteq \mathrm{FR}$, and then we have from (3.13) that

$$K_I \subseteq N_{\mathrm{FR}+}(K) \subseteq N_{\mathrm{FR}}(K) \subseteq K \tag{3.15}$$

It follows from the definition that using $K_2 = \mathrm{FR}$ is at least as strong as using $K_2 = Q$ in the same operator, i.e. $N_{\mathrm{FR}}(K) \subseteq N(K)$ and $N_{\mathrm{FR}_+}(K) \subseteq N_+(K)$. We therefore have that

$$K_I \subseteq N_{\mathrm{FR}+}(K) \subseteq N_{\mathrm{FR}}(K) \subseteq N(K) \subseteq K \tag{3.16}$$

$$K_I \subseteq N_{\mathrm{FR}+}(K) \subseteq N_+(K) \subseteq N(K) \subseteq K \tag{3.17}$$

The power of these operators is discussed in Section 3.4.3. As for the relation between $N_{\mathrm{FR}}(K)$ and $N_+(K)$, it can be seen that $N_{\mathrm{FR}}(K) \not\subseteq N_+(K)$ (e.g. by a clique on 5 vertices and $K = \mathrm{FR}$, see Section 3.4.3), but it is not clear (to us) whether $N_+(K) \subseteq N_{\mathrm{FR}}(K)$.

Iterated operators. Define the iterated operator $N^r(K)$ recursively by $N^0(K) = K$ and $N^r(K) = N(N^{r-1}(K))$ for $r \geq 1$. For other operators, the iterated operator is defined similarly.

The following Theorem of Lovász and Schrijver [LS91] proves that even without the positive semidefiniteness constraint (d), it suffices to apply n iterations in order to get from a convex cone $K \subseteq Q$ the cone K_I. It follows that applying the N operator on $K \neq K_I$ produces a relaxation of K_I that is strictly tighter than K.

Theorem 3.4 (Lovász and Schrijver [LS91]). *Let $K \subseteq Q$ be a convex cone in $\mathbb{R}^{n+1}$. Then $N^n(K) = K_I$.*

It is often easier to work in the original n-dimensional space (without homogenization), so in the case that K is the cone obtained from a polytope (or a convex set) P in $[0,1]^n$ via the homogenization procedure (3.5)-(3.6), define

$$N(P) := \left\{ x \in \mathbb{R}^n : \begin{pmatrix} 1 \\ x \end{pmatrix} \in N(K) \right\}$$

and similarly for the other operators (including the iterated ones).

For the stable set problem, K will be one of the cones obtained from $\mathrm{FR}(G)$ by an iterated operator, e.g. $N^r(\mathrm{FR}(G))$. Going back to the original n-dimensional space we shall abbreviate $N^r(G) := N^r(\mathrm{FRAC}(G))$ and similarly for the other operators. We then have from Theorem 3.4 that $N^n(G) = \mathrm{STAB}(G)$.

Ranks. The *N-rank of an inequality* $u^T x \geq 0$ that is valid for K_I, is the smallest nonnegative integer r such that $u^T x \geq 0$ is valid for $N^r(K)$. (Note that the rank is relative to K). For N_+,N_{FR} and $N_{\mathrm{FR}+}$ the rank is defined similarly. Theorem 3.4 states that these ranks are at most n (the dimension) for any valid inequality.

The *N-rank of a cone* K is the smallest nonnegative integer r such that $N^r(K) = K_I$, and similarly for the other operators. By Theorem 3.4, the N-rank of K is at most n (the dimension).

The N-rank of a graph G, is the N-rank of $\mathrm{FR}(G)$, and similarly for the other operators. For example, for a bipartite graph STAB = FRAC and hence the N-rank of a bipartite graph is 0. We will elaborate on bounds on the rank in Section 3.4.3.

Algorithmic aspects. Lovász and Schrijver [LS91] give sufficient conditions for efficient weak (i.e. up to arbitrary precision) optimization (of linear objective functions) over $N(K)$, $N_+(K)$, $N_{\mathrm{FR}}(K)$ and $N_{\mathrm{FR}+}(K)$. Technically, the matrix-cut operators have the following algorithmic property.

Theorem 3.5 (Lovász and Schrijver [LS91]). *A polynomial time weak separation oracle for K gives a polynomial time weak separation oracle for $N^r(K), N_+^r(K), N_{\mathrm{FR}}^r(K)$ and $N_{\mathrm{FR}+}^r(K)$ for any fixed constant r.*

By the equivalence between weak (i.e. up to arbitrary precision) optimization and weak separation (see [GLS93]), Theorem 3.5 implies a weak optimization of any linear objective function over these relaxations of K_I.

Lovász and Schrijver [LS91] suspect that Theorem 3.5 does not extend to $N(K, K)$. They remark, however, that it if K is given by an explicit system of polynomially many linear inequalities, then Theorem 3.5 does extend to $N(K, K)$.

For the stable set problem, the cone $K = \mathrm{FR}$ is given by an explicit linear program of polynomial size, so one can solve the separation problem for it in polynomial time. We thus obtain the following theorem.

Theorem 3.6. *For every fixed $r \geq 0$, the weak optimization problem for $N^r(G)$ can be solved in polynomial time, and similarly for $N_+^r, N_{\mathrm{FR}}^r, N_{\mathrm{FR}+}^r$.*

3.4.2 Basic properties

We collect some properties of the matrix-cut operators defined in Section 3.4.1. In particular, Corollary 3.7 and Lemma 3.16 will be used in Section 3.5.1.

Monotonicity. It is straightforward that the matrix-cut operators are monotone with respect to containment of K_1 and K_2, as follows.

Lemma 3.3. *Let $K_1' \subseteq K_1$ and $K_2 \subseteq K_2'$. Then $N(K_1', K_2') \subseteq N(K_1, K_2)$ and similarly for N_+.*

It follows that the matrix-cut operators are monotone with respect to adding/removing edges.

Corollary 3.4. *Let G' be a graph that is obtained from another graph G by adding edges. Then $N^r(G') \subseteq N^r(G)$, and similarly for N_+^r,N_{FR}^r,$N_{\mathrm{FR}+}^r$.*

Proof. Observe that $\mathrm{FR}(G') \subseteq \mathrm{FR}(G)$. The proof follows from Lemma 3.3. □

Down-monotonicity. Throughout, we use $x \geq y$, where x, y are two vectors, to denote $x_i \geq y_i$ for every coordinate i.

A non-empty convex set $P \subseteq [0,1]^n$ is called *down-monotone* (in $[0,1]^n$) if for every $x \in P$, every $y \in [0,1]^n$ with $y \leq x$ is also in P (see e.g. [GLS93, page 11]). Similarly, a convex cone $\{\mathbf{0}\} \neq K \subseteq Q$ is called *down-monotone* if for every $x \in K$, every $y \in Q$ with $y \leq x$ and $y_0 = x_0$ is also in K.

The next lemma shows that the matrix-cut operators preserve down-monotonicity. It extends a similar result for $N(\cdot)$ and $N_+(\cdot)$, that is given by Goemans and Tunçel [GT00, Theorem 5.1] (where a down-monotone polytope is called lower-comprehensive) and by Cook and Dash [CD00, Lemma 2.6] (where the polytope is said to be of an anti-blocking type).

Lemma 3.5. *Let $K_1, K_2 \subseteq Q$ be down-monotone convex cones. Then $N(K_1, K_2)$ is down-monotone, and similarly for N_+.*

Proof. Let $x \in N(K_1, K_2)$ and $0 \leq x' \leq x$ with $x'_0 = x_0$. It suffices to prove that $x' \in N(K_1, K_2)$ when x, x' differ only in a single coordinate, say $i = 1$, since we can repeat the same argument for each coordinate. Furthermore, for a single coordinate $i = 1$ it suffices to prove the case $x'_1 = 0$, since $N(K_1, K_2)$ is convex, and so convex combinations of x' and x give any desired value for coordinate $i = 1$.

Since $x \in N(K_1, K_2)$, there exists a matrix $Y \in M(K_1, K_2)$ with $x = Ye_0$. Define the matrix Y' by

$$Y'_{ij} = \begin{cases} 0 & \text{if } i = 1 \text{ or } j = 1; \\ Y_{ij} & \text{otherwise.} \end{cases}$$

We claim that $Y' \in M(K_1, K_2)$. Indeed, Y' clearly satisfies (a) and (b). To prove (c), let $u \in K_1^*, v \in K_2^*$, and from Lemma 3.6 below we have that $u - u_1 x_1 \in K_1^*$ and $v - v_1 x_1 \in K_2^*$ and hence

$$u^T Y' v = (u - u_1 x_1)^T Y (v - v_1 x_1) \geq 0$$

Observe that $x' = Y' e_0$, and therefore $x' \in N(K_1, K_2)$, as required.

For the proof of N_+ we need to show that (d) also holds, and indeed from the Gram matrix representation of Y we can obtain a Gram matrix representation of Y' by replacing the vector that corresponds to coordinate $i = 1$ with the all zeros vector $\mathbf{0}$. ☐

Lemma 3.6. *Let $K \subseteq Q$ be down-monotone and let $v \in K^*$. Then $v - v_i e_i \in K^*$ for all $i \geq 1$.*

Proof. By the down-monotonicity of K, for every $x \in K$ we have that $x - x_i e_i \in K$, and hence $(v - v_i e_i)^T x = \sum_{j \neq i} v_j x_j = v^T (x - x_i e_i) \geq 0$. ☐

Observe that Q is down-monotone by its definition in (3.7), and that FRAC is down-monotone by its definition in (3.10)-(3.11). By Lemma 3.5 the matrix-cut operators preserve down-monotonicity and we obtain the following corollary for the iterated operators.

Corollary 3.7. *$N^r(G)$ is down-monotone, and similarly for N_+^r,N_{FR}^r,$N_{\mathrm{FR}+}^r$.*

Flipping and renaming coordinates. The operators $N, N_+, N_{\mathrm{FR}}, N_{\mathrm{FR}+}$ are invariant under various operations, including *renaming indices* (i.e. permuting the order of coordinates), and *flipping coordinates* $x_i \to (x_0 - x_i)$ for any subset of the indices $\{1, 2, \ldots, n\}$. More formally,

Lemma 3.8 (Lovász and Schrijver [LS91]). *Let A be a linear transformation mapping Q onto itself. Then $N(AK_1, AK_2) = AN(K_1, K_2)$ and similarly for N_+. Hence $N(AK) = AN(K)$ and similarly for N_+.*

By flipping coordinates, one can extend Lemma 3.5. For example, it follows that the N and N_+ operators preserve up-monotonicity (shown by Cook and Dash [CD00, Section 2] as the blocking property), and a "convex corner" property (shown by Goemans and Tunçel [GT00, Section 5]).

Intersection with faces. A *face* of Q is the intersection of Q with hyperplanes of the form $\{x : x_i = 0\}$ or $\{x : x_i = x_0\}$. The intersection of K with a face of Q consists of all $x \in K$ with one or more of their coordinates *fixed* to 0 or x_0 (recall that x_0 corresponds to 1 in the non-homogenous case).

The following lemma proves equivalence between fixing some coordinates before applying a matrix-cut operator (e.g. in K) and afterwards (e.g. in $N(K)$). It extends a similar result that is given by Goemans and Tunçel [GT00] for $N(\cdot)$ and $N_+(\cdot)$.

Lemma 3.9. *If F is a face of Q, then $N(K_1 \cap F, K_2) = N(K_1, K_2) \cap F$ and similarly for N_+.*

Proof. The direction "$\subseteq$" follows from Lemma 3.3, since $N(K_1 \cap F, K_2) \subseteq N(K_1, K_2)$ and $N(K_1 \cap F, K_2) \subseteq N(F, K_2) \subseteq F$, and similarly for N_+.

For the converse direction "$\supseteq$" with the N operator, let $x \in N(K_1, K_2) \cap F$. Then there exists a matrix $Y \in M(K_1, K_2)$ with $Ye_0 = x$. Let H be any one of the hyperplanes of the form $\{x : x_i = 0\}$ or $\{x : x_i = x_0\}$ that define F. Since $e_j, f_j \in Q^* \subseteq K_2^*$ for all j, we have that $Ye_j \in K_1 \subseteq Q$ and $Yf_j \in K_1 \subseteq Q$, while their sum satisfies $Ye_j + Yf_j = Ye_0 = x \in F \subset H$. Since H defines a face of Q then by definition of a face we have that Ye_j (and also Yf_j) must belong to H. [1] But every $v \in \mathbb{R}^{n+1}$ is a linear combination of $\{e_0, e_1, \ldots, e_n\}$ and $Ye_j \in H$ for all $j \geq 0$, and so $Yv \in H$ for every v, including all $v \in K_2^*$.

For every $v \in K_2^*$ we have that Yv belongs to $K_1 \subseteq Q$, by the definition of Y. We saw above that Yv also belongs to all hyperplanes H that define F, and we conclude that Yv belongs also to F. Hence, $Yv \in K_1 \cap F$ for all $v \in K_2^*$, implying that $Y \in M(K_1 \cap F, K_2)$ and $x \in N(K_1 \cap F, K_2)$. The proof for N_+ is similar, since Y is also known to be positive semidefinite. □

We remark that the above proof of Lemma 3.9 extends to the case where F is a face of K_1, as shown by Cook and Dash [CD00, Lemma 2.2] for $N(\cdot)$ and $N_+(\cdot)$. For the special cases $K_2 = Q$ and $K_2 = \mathrm{FR}$ we obtain the following.

Corollary 3.10. *If F is a face of Q (or a face of K), then $N(K \cap F) = N(K) \cap F$ and similarly for N_+,N_{FR},$N_{\mathrm{FR}+}$.*

[1] In other words, suppose that the hyperplane H is defined by the equality $u^T x = 0$ (i.e. $u = e_i$ or $u = f_i$) and that the inequality $u^T x \geq 0$ is valid for Q (i.e. Q is entirely contained in one side of H). We then have that $u^T(Ye_j), u^T(Yf_j) \geq 0$ while their sum is $u^T x = 0$, implying that $u^T(Ye_j) = u^T(Yf_j) = 0$.

Deleting fixed coordinates. Suppose that K is contained in a face of Q. Then some of the coordinates are fixed (i.e. $x_i = 0$ or $x_i = x_0$), and it may be desirable to delete these coordinates and reduce the dimension. Formally, a *deletion* operation of indices subset $I \subset \{1, \ldots, n\}$ is the function $f : \mathbb{R}^{n+1} \to \mathbb{R}^{n+1-|I|}$ where $f(x)$ is the vector x restricted to the coordinates not in I, i.e. $f(x) = (x_i)_{i \notin I}$.

In the following we show that deleting fixed coordinates of K before applying a matrix-cut operator (e.g. in K) is equivalent to deleting them afterwards (e.g. in $N(K)$). The following handles the basic case of one coordinate that is fixed to 0.

Lemma 3.11. *Let $F = Q \cap \{x : x_n = 0\}$ and let f be the deletion operation of coordinate $i = n$. If K_1, K_2 are convex cones that are contained in F then $f(N(K_1, K_2)) = N(f(K_1), f(K_2))$,* [2] *and similarly for N_+.*

Proof. The deletion operation f is a linear transformation from $\mathbb{R}^{n+1}$ to $\mathbb{R}^n$, and thus can be described as an $n \times (n+1)$ matrix A. Note that columns 0 to $n-1$ of A form an identity matrix and column n of A is all zeros. We first claim that $AK^* = (AK)^*$ for $K = K_1$ and for $K = K_2$. Indeed, by definition, $u \in AK^*$ if there exists $r \in \mathbb{R}$ with $\begin{pmatrix} u \\ r \end{pmatrix} \in K^*$. Note that $\begin{pmatrix} u \\ r \end{pmatrix} \in K^*$ holds either for all values of r or for no value of r, since $K \subset \{x : x_n = 0\}$. Therefore,

$$AK^* = \{u : \exists r \in \mathbb{R} \text{ with } \begin{pmatrix} u \\ r \end{pmatrix} \in K^*\} = \{u : \begin{pmatrix} u \\ 0 \end{pmatrix} \in K^*\}.$$

We also have that

$$(AK)^* = \{u : u^T(Ax) \geq 0 \quad \forall x \in K\} = \{u : A^T u \in K^*\}.$$

Since $A^T u = \begin{pmatrix} u \\ 0 \end{pmatrix}$, we obtain $AK^* = (AK)^*$.

Let us now prove that $M(AK_1, AK_2) = AM(K_1, K_2)A^T$. For the direction "$\subseteq$", let $Y \in M(AK_1, AK_2)$. Then by (c), for every $u \in K_1^*, v \in K_2^*$ we have that $u^T A^T Y A v \geq 0$. We therefore have that

$$\begin{pmatrix} Y & \mathbf{0} \\ \mathbf{0}^T & 0 \end{pmatrix} = A^T Y A \in M(K_1, K_2).$$

Multiplying by A from the left and by A^T from the right, we obtain (since AA^T is the identity matrix) that $Y \in AM(K_1, K_2)A^T$.

For the converse direction "$\supseteq$", let $Y \in AM(K_1, K_2)A^T$. Since $K_1 \subseteq \{x : x_n = 0\}$, every matrix in $M(K_1, K_2)$ has only zeros in row n, and by the symmetry (a) it has only zeros also in column n. Hence,

$$A^T Y A = \begin{pmatrix} Y & \mathbf{0} \\ \mathbf{0}^T & 0 \end{pmatrix} \in M(K_1, K_2).$$

By (c), for every $u \in K_1^*, v \in K_2^*$ it holds that $u^T A^T Y A v \geq 0$, and hence $Y \in M(AK_1, AK_2)$.

[2]Note that the application of N in the righthand side is in a smaller dimension than in the lefthand side.

Now since $A^T e_0$ is just e_0 (in a larger dimension), we conclude that

$$N(AK_1, AK_2) = AM(K_1, K_2)A^T e_0 = AM(K_1, K_2)e_0 = AN(K_1, K_2).$$

The proof for the N_+ operator is similar since Y is positive semidefinite if and only if $A^T Y A$ is (observe that Y has a Gram matrix representation if and only if $A^T Y A$ has such a representation). $\square$

We can extend Lemma 3.11 to an arbitrary face F and to an arbitrary K_2, as follows.

Lemma 3.12. *Let $F = Q \cap \{x : x_i = 0\ \forall i \in I_0\} \cap \{x : x_i = x_0\ \forall i \in I_1\}$ and let f be the deletion operation of the coordinates $I_0 \cup I_1$. If K_1 is a convex cone contained in F and K_2 is a convex cone contained in Q, then $f(N(K_1, K_2)) = N(f(K_1), f(K_2 \cap F))$, and similarly for N_+.*

Proof. K_1 and $K_2 \cap F$ are both contained in F, so we can repeatedly apply Lemma 3.11 on them, and delete the coordinates of $I_0 \cup I_1$. (Note that by using Lemma 3.8 we can extend Lemma 3.11 also to deleting coordinates that are fixed to x_0.) It follows that $f(N(K_1, K_2 \cap F)) = N(f(K_1), f(K_2 \cap F))$.

By Lemma 3.9 we have that $N(K_1, K_2 \cap F) = N(K_1, K_2) \cap F$, and since $N(K_1, K_2) \subseteq K_1 \subseteq F$, we have that $N(K_1, K_2 \cap F) = N(K_1, K_2)$. The proof follows. $\square$

For the stable set problem, it is straightforward to see that fixing and deleting a coordinate of $\text{FR}(G)$ has the following effect.

Lemma 3.13. *Let $F = Q \cap \{x : x_i = 0\}$, and let f be the deletion operation of coordinate i. Then $f(\text{FR}(G) \cap F) = \text{FR}(G - i)$.*

Lemma 3.14. *Let $F = Q \cap \{x : x_i = x_0\}$, and let f be the deletion operation of coordinate i. Then $f(\text{FR}(G) \cap F) = \text{FR}(G - i) \cap \{x : x_j = 0 \text{ for } j \in \Gamma(i)\}$.*

For the special cases $K_2 = Q$ and $K_2 = \text{FR}$ we obtain the following from Lemma 3.12.

Corollary 3.15. *Let $F = Q \cap \{x : x_i = 0\ \forall i \in I_0\} \cap \{x : x_i = x_0\ \forall i \in I_1\}$ and let f be the deletion operation of the coordinate $I_0 \cup I_1$. If K is a convex cone contained in F then $f(N(K)) = N(f(K))$,* [3] *and similarly for N_+, N_{FR} and $N_{\text{FR}+}$.*

Proof. For the N operator we have from Lemma 3.12 that

$$f(N(K)) = N(f(K), f(Q \cap F))$$

and $f(Q \cap F)$ is just Q in the smaller dimension, so $f(N(K)) = N(f(K))$. The proof for the N_+ operator is similar.

For the N_{FR} operator we have from Lemma 3.12 that

$$f(N_{\text{FR}}(K)) = N(f(K), f(\text{FR}(G) \cap F)),$$

[3]Note that the application of N in the righthand side is in a smaller dimension than in the lefthand side.

and it follows from Lemmas 3.13 and 3.14 that $f(\mathrm{FR}(G) \cap F) = \mathrm{FR}(G - I_0 - I_1) \cap H$, where $H = \{x : x_i = 0\ \forall i \in \Gamma(I_1) - I_0 - I_1\}$. We therefore have that

$$f(N_{\mathrm{FR}}(K)) = N(f(K), \mathrm{FR}(G - I_0 - I_1) \cap H).$$

Note that $f(K) \subset H$ since $K \subseteq F \cap \mathrm{FR}(G) \subseteq H$, and so by Lemma 3.9 we have that $f(N_{\mathrm{FR}}(K)) = N_{\mathrm{FR}}(f(K))$, as required. The proof for $N_{\mathrm{FR}+}(K)$ is similar. $\square$

Corollary 3.15 extends a similar result that is given by Cook and Dash [CD00] for $N(\cdot)$ and $N_+(\cdot)$. Technically, they define an embedding operation as one that introduces new coordinates that are fixed (to either 0 or x_0), and state that for every convex cone K' and an embedding operation g, $g(N(K')) = N(g(K'))$, and similarly for N_+ (see also [ST99]). The deletion operation is the inverse of embedding, so for $N(\cdot)$ and $N_+(\cdot)$ Corollary 3.15 is equivalent to the result of Cook and Dash [CD00].

Removing vertices from the graph. For the stable set problem, the properties collected so far, and in particular Corollary 3.15, give a useful characterization to whether $x \in N^r(G)$ in the case that x has a fixed coordinate (i.e. $x_i = 0$ or $x_i = x_0$).

Recall that $V = \{1, \ldots, n\}$. For a vector $x \in \mathbb{R}^n$ and a subset $W \subset V$, we denote by x_W the restriction of x to the coordinates of W.

Lemma 3.16. *Let $x \in \mathbb{R}^n$, and assume that for some i we have that $x_i = 1$ and that $x_j = 0$ for all $j \in \Gamma(i)$. Then for all $r \geq 0$, $x \in N^r(G)$ if and only if $x_{V-\Gamma(i)-i} \in N^r(G - \Gamma(i) - i)$, and similarly for N^r_+,N^r_{FR} and $N^r_{\mathrm{FR}+}$.*

Proof. It is clear that x belongs to the face F of Q that is defined by the hyperplanes $\{x : x_i = x_0\}$ and $\{x : x_j = 0\}$ for all $j \in \Gamma(i)$. Then $x \in N^r(G)$ if and only if $x \in N^r(G) \cap F$, which is equivalent, by Corollary 3.10, to $x \in N^r(\mathrm{FR}(G) \cap F)$. Let f be the deletion operation of the coordinates $\Gamma(i) \cup \{i\}$, and then we have equivalently that $f(x) \in f(N^r(\mathrm{FR}(G) \cap F))$. By Corollary 3.15, the latter is equivalent to $f(x) \in N^r(f(\mathrm{FR}(G) \cap F))$. By Lemmas 3.13 and 3.14, we have that $f(\mathrm{FR}(G) \cap F) = \mathrm{FR}(G - \Gamma(i) - i)$, and the proof follows. The proof for N^r_+,N^r_{FR} and $N^r_{\mathrm{FR}+}$ is similar. $\square$

Lemma 3.17. *Let $x \in \mathbb{R}^n$ be a vector and assume that $x_i = 0$ for some i. Then $x \in N^r(G)$ if and only if $x_{V-i} \in N^r(G - i)$, and similarly for N^r_+,N^r_{FR} and $N^r_{\mathrm{FR}+}$.*

Proof. It is clear that x belongs to the face F of Q that is defined by the hyperplane $x_i = 0$. Then $x \in N^r(G)$ if and only if $x \in N^r(G) \cap F$, which is equivalent, by Corollary 3.10, to $x \in N^r(\mathrm{FR}(G) \cap F)$. Let f be the deletion operation of the coordinate i, and then we have equivalently that $f(x) \in f(N^r(\mathrm{FR}(G) \cap F))$. By Corollary 3.15, the latter is equivalent to $f(x) \in N^r(f(\mathrm{FR}(G) \cap F))$. By Lemma 3.13 we have that $f(\mathrm{FR}(G) \cap F) = \mathrm{FR}(G - i)$, and the proof follows. The proof for N^r_+,N^r_{FR} and $N^r_{\mathrm{FR}+}$ is similar. $\square$

3.4.3 Bounds on the rank

We describe general methods to obtain upper and lower bounds on the N-rank and N_+-rank of valid inequalities, and extend them to N_{FR}-rank.We also illustrate the use of these methods on a few valid constraints for the stable set problem (see Table 3.4.3 on page 78).

Vertex deletion and contraction. Let $a^T x \leq b$ be an inequality valid for STAB(G). For a subset $W \subset V$, we denote by a_W the restriction of a to the coordinates of W. For every $i \in V$, if $a^T x \leq b$ is valid for STAB(G), then $a^T_{V-i} x \leq b$ is valid for STAB($G - i$) and $a^T_{V-\Gamma(v)-i} x \leq b - a_i$ is valid for STAB($G - \Gamma(i) - i$). Following the terminology of Lovász and Schrijver [LS91], we say that these inequalities arise from $a^T x \leq b$ by the *deletion* and *contraction* of vertex i, respectively. Note that if $a^T x \leq b$ is an inequality such that for some i, both the deletion and the contraction of i yield inequalities valid for the corresponding graphs, then $a^T x \leq b$ is valid for G.

The N-rank of an inequality valid for STAB(G) depends only on the subgraph induced by those vertices with a nonzero coefficient, and similarly for N_+,N_{FR} and $N_{\mathrm{FR}+}$. Indeed, if a vertex i has a zero coefficient, then the inequality being valid for $N^r(G)$ is equivalent, by Corollary 3.7, to the inequality being valid for $N^r(G) \cap \{x : x_i = 0\}$, which in turn is equivalent, by Lemma 3.17, to the inequality being valid for $N^r(G - i)$.

Upper bounds on the N-rank. Lovász and Schrijver [LS91] give an upper bound on $N(K)$, which allows to upper bound the N-rank of an inequality, as follows.

The *sum* of two sets $K', K'' \subseteq \mathbb{R}^{n+1}$ is defined as $K' + K'' := \{x' + x'' : x \in K', x'' \in K''\}$. Note that if K', K'' are convex cones in Q then $K' + K''$ is also a convex cone in Q. Furthermore, if K', K'' are obtained via the homogenization procedure (3.5)-(3.6) from polytopes $P', P'' \subseteq \mathbb{R}^n$, respectively, then $K' + K''$ corresponds to all convex combinations of a point from P' and a point from P'' (recall that coordinate 0 needs to be scaled to 1).

Lemma 3.18 (Lovász and Schrijver [LS91]). *For all $1 \leq i \leq n$,*

$$N(K) \subseteq \Big(K \cap \{x : x_i = 0\}\Big) + \Big(K \cap \{x : x_i = x_0\}\Big).$$

Proof. If $x \in N(K)$ then there exists $Y \in M(K)$ with $x = Ye_0 = Ye_i + Yf_i$ for any $i \leq i \leq n$. Clearly, $Ye_i \in K \cap \{x : x_i = x_0\}$ and $Yf_i \in K \cap \{x : x_i = 0\}$, and the proof follows. □

Corollary 3.19. *If an inequality is valid for both $K \cap \{x : x_i = 0\}$ and $K \cap \{x : x_i = x_0\}$, then it is valid for $N(K)$.*

Goemans and Tunçel [GT00] note that repeatedly using Lemma 3.18 and Corollary 3.10, gives that for all $I \subseteq \{1, \ldots, n\}$ with $|I| = r$,

$$N^r(K) \subseteq \sum_{I_0 \subseteq I} \Big(K \cap \{x : x_i = 0 \ \forall i \in I_0\} \cap \{x : x_i = x_0 \ \forall i \in I \setminus I_0\}\Big).$$

In particular, this shows that the N-rank of any cone K is at most n, proving Theorem 3.4.

For the stable set problem, Corollary 3.19 can be rephrased as follows (using Lemmas 3.16 and 3.17).

Lemma 3.20 (Lovász and Schrijver [LS91]). *Let P be a convex set with* STAB $\subseteq P \subseteq$ FRAC. *If $a^T x \leq b$ is an inequality such that for some $i \in V$, both the deletion and contraction of i give an inequality valid for P, then $a^T x \leq b$ is valid for $N(P)$.*

For example, if C induces a chordless odd cycle in G, the *odd hole constraint*

$$\sum_{i\in C} x_i \leq \frac{|C|-1}{2} \tag{3.18}$$

has N-rank at most (and actually exactly) 1, because both the contraction and the deletion of any vertex result in an inequality that is valid for FRAC. (In fact, Lovász and Schrijver [LS91] prove that $N(\text{FRAC})$ is exactly the relaxation that is obtained by adding to FRAC all the odd hole constraints.)

Lovász and Schrijver [LS91] also give the following upper bound on the N-rank of a graph. The proof follows by applying Lemma 3.20 repeatedly for $n-\alpha(G)-1$ vertices outside a maximum stable set in the graph, since the graph induced on the other vertices must be bipartite.

Corollary 3.21 (Lovász and Schrijver [LS91]). *The N-rank of a graph G of stability number $\alpha(G)$ is at most $n-\alpha(G)-1$.*

It follows that the N-rank of any graph G is at most $n-2$. Note that the N-rank of FR is at most $n-2$, while the N-rank of a general cone K is at most (and can actually be) n.

We next analyze the N-rank of a few more examples, due to Lovász and Schrijver [LS91]. By Corollary 3.21, if B is a clique in G, the *clique constraint*

$$\sum_{i\in B} x_i \leq 1 \tag{3.19}$$

has N-rank at most (and actually exactly) $|B|-2$. Note that the class of all clique constraints strengthens the class of all edge constraints (3.9).

If D induces a chordless odd cycle in $\overline{G}$ (the edge complement of G), the *odd antihole constraint*

$$\sum_{i\in D} x_i \leq 2 \tag{3.20}$$

has N-rank at most (and actually exactly) $(|D|-3)/2$, because the contraction of a vertex results in an inequality trivially valid for FRAC, and the deletion of a vertex results in an inequality that is the sum of two clique constraints, each of size $(|D|-1)/2$ and hence of N-rank $(|D|-5)/2$.

If W induces an odd wheel in G with center $i_0 \in W$, the *odd wheel constraint*

$$\sum_{i\in W\setminus\{i_0\}} x_i + \frac{|W|-2}{2} x_{i_0} \leq \frac{|W|-2}{2} \tag{3.21}$$

has N-rank at most (and actually exactly) 2, since the contraction of the center vertex results in a trivial inequality, and the deletion of the center vertex results with the odd hole constraint.

Upper bounds on the N_{FR}-rank. The methods for obtaining upper bounds on the N-rank can be extended (with modifications) to upper bounds on the N_{FR}-rank, as follows.

Lemma 3.22. *For all $ij \in E$,*

$$N(K) \subseteq \Big(K \cap \{x : x_i = x_j = 0\}\Big) + \Big(K \cap \{x : x_j = x_0\}\Big) + \Big(K \cap \{x : x_i = x_0\}\Big).$$

Proof. If $x \in N_{\mathrm{FR}}(K)$ then there exists $Y \in M(K)$ with $x = Ye_0 = Ye_i + Ye_j + Yf_{ij}$ for any $ij \in E$. Clearly, $Ye_i \in K \cap \{x : x_i = x_0\}$ and $Ye_j \in K \cap \{x : x_j = x_0\}$ and $Yf_{ij} \in K \cap \{x : x_i = x_j = 0\}$, and the proof follows. $\square$

Corollary 3.23. *If an inequality is valid for $K \cap \{x : x_i = x_0\}$, for $K \cap \{x : x_j = x_0\}$, and for $K \cap \{x : x_i = x_j = 0\}$, then it is valid for $N_{\mathrm{FR}}(K)$.*

Corollary 3.23 can be rephrased as follows (using Lemmas 3.16 and 3.17).

Lemma 3.24. *Let P be a convex set with* STAB $\subseteq P \subseteq$ FRAC*. If $a^T x \leq b$ is an inequality such that for some $ij \in E$, the contraction of i, the contraction of j and the deletion of $\{i, j\}$ give an inequality valid for P, then $a^T x \leq b$ is valid for $N(P)$.*

The following upper bound on the N_{FR}-rank of a graph follows by applying Lemma 3.24 repeatedly on edges, so that the removal of their endpoints results in a bipartite graph (e.g. a matching that is maximal with respect to containment).

Corollary 3.25. *Suppose that a graph G contains a set of β edges, whose endpoints removal results in a bipartite graph. Then the N_{FR}-rank of G is at most β.*

It follows that the N_{FR}-rank of a graph G is at most $(n-2)/2$ if n is even and $(n-1)/2$ if n is odd; in general it is at most $\lfloor (n-1)/2 \rfloor$. In particular, the N_{FR}-rank of the clique constraint (3.19) is at most $\lfloor (|B|-1)/2 \rfloor$.

We can apply these bounds on the other examples. The N_{FR}-rank of the odd hole constraint constraint (3.18) is at most (and thus exactly) 1, since the N_{FR} operator is at least as strong as N. The N_{FR}-rank of the odd antihole constraint (3.20) is at most $\lfloor (|D|+1)/4 \rfloor$, because the contraction of a vertex results in an inequality trivially valid for FRAC, and the deletion of two vertices results in an inequality that is the sum of two clique constraints, each of size at most $(|D|-1)/2$ and hence of N_{FR}-rank $\lfloor (|D|-3)/4 \rfloor$. (In fact, it can be shown by direct calculations that the N_{FR}-rank of the odd antihole constraint (3.20) with $|D| = 7$ is at most 1.) The N_{FR}-rank of the wheel constraint (3.21) is at most (and thus exactly) 1, since the contraction of the center vertex results in a trivial inequality, the contraction of a non-center vertex results in an inequality is valid for FRAC, and the deletion of these two vertices also results in an inequality is valid for FRAC.

Lower bounds on the N-rank. Lovász and Schrijver [LS91] show that certain uniform fractional stable sets belong to $N^r(G)$, regardless of the graph G. For example, for $r = 0$ it is straightforward that $(1/2)\mathbf{1} \in \mathrm{FRAC}(G)$. The following lemma gives an extension to larger r, with the uniform solution being smaller, depending on r.

Lemma 3.26 (Lovász and Schrijver [LS91]). *Assume that P is down-monotone and contains* $\mathrm{STAB}(G)$. *If* $(1/r)\mathbf{1} \in P$ *for* $r > 0$ *then* $1/(r+1)\mathbf{1} \in N(P)$.

Proof. Let K be the convex cone obtained from P via the homogenization procedure (3.5)-(3.6). Define the matrix $Y \in \mathbb{R}^{(n+1)\times(n+1)}$ by

$$Y_{ij} = \begin{cases} 1 & \text{if } i = j = 0; \\ 1/(r+1) & \text{if } (i = 0, j > 0) \text{ or } (i > 0, j = 0) \text{ or } (i = j > 0); \\ 0 & \text{otherwise.} \end{cases}$$

To see that $Y \in M(K, Q)$ observe that (a),(b) clearly hold, and let us now show that (c") holds.

$$Ye_i = \frac{1}{t+1}(e_0 + e_i) \in \mathrm{ST}(G) \subseteq K$$

and

$$Yf_i = \frac{r}{r+1}e_0 + \sum_{j\neq 0,i} \frac{1}{r+1}e_j = \frac{r}{r+1}\left(e_0 + \sum_{j\neq 0,i} \frac{1}{r}e_j\right).$$

By the induction hypothesis we have that

$$\sum_{j\neq 0,i} \frac{1}{r}e_j \leq \sum_{j\neq 0} \frac{1}{r}e_j \in P,$$

and the down-monotonicity of P implies that $Yf_i \in K$, and thus (c") holds. We conclude that $Ye_0 \in N(K)$, i.e. $1/(r+1)\mathbf{1} \in N(P)$. $\square$

Corollary 3.27 (Lovász and Schrijver [LS91]). $1/(r+2)\mathbf{1} \in N^r(G)$ *for all* $r \geq 0$.

Proof. Proceed by induction on r. We mentioned above that the case $r = 0$ is trivial. The inductive step follows from Lemma 3.26, since $N^r(\mathrm{FRAC}(G))$ clearly contains $\mathrm{STAB}(G)$ and is down-monotone by Corollary 3.7. $\square$

Corollary 3.28 (Lovász and Schrijver [LS91]). *The* N_{FR}*-rank of a graph* G *of stability number* $\alpha(G)$ *is at least* $n/\alpha(G) - 2$.

Proof. Let r be the N-rank of G, and hence $N^r(G) = \mathrm{STAB}(G)$. By Corollary 3.27 we have that $1/(r+2)\mathbf{1} \in N^r(G)$. The inequality $\mathbf{1}^T x \leq \alpha$ is valid for $\mathrm{STAB}(G) = N^r(G)$, and in particular for $1/(r+2)\mathbf{1}$, implying that $n/(r+2) \leq \alpha(G)$, and the proof follows. $\square$

For example, the stability number of a clique B is 1, so the N-rank of B is at least, and hence exactly, $|B| - 2$. In fact, the above proof shows that the N-rank of the clique constraint (3.19) is at least, and hence exactly, $|B| - 2$. The stability number of an an odd antihole D is 2, so the N-rank of D is at least $|D|/2 - 2$, and since $|D|$ is odd, it must be at least $(|D| - 3)/2$. In fact, this shows that the N-rank of the odd antihole constraint (3.20) is at least, and hence exactly, $(|D| - 3)/2$. Corollary 3.27 also yields a lower bound on the N-rank of the wheel constraint (3.21). Indeed, let r be the N-rank of this constraint. Then we have that this constraint is valid for $N^r(G)$ and, in particular, for $1/(r+2)\mathbf{1} \in N^r(G)$. Thus,

$$\frac{1}{r+2}\left(|W| - 1 + \frac{|W| - 2}{2}\right) \leq \frac{|W| - 2}{2}$$

which gives that $\frac{2(|W|-1)}{|W|-2} + 1 \leq r + 2$ and thus $r \geq 1 + \frac{2}{|W|-2}$. Since the N-rank of the wheel constraint is an integer, it must be at least, and hence exactly, 2.

Lower bounds on the N_{FR}-rank. The methods for obtaining lower bounds on the N-rank can be extended (with modifications) to lower bounds on the N_{FR}-rank, as follows.

Lemma 3.29. *Assume that P be down-monotone and contains* $\mathrm{STAB}(G)$*. If $(1/r)\mathbf{1} \in P$ for $r > 0$ then $1/(r+2)\mathbf{1} \in N_{\mathrm{FR}}(P)$.*

Proof. Define the matrix $Y \in \mathbb{R}^{(n+1)\times(n+1)}$ by

$$Y_{ij} = \begin{cases} 1 & \text{if } i = j = 0; \\ 1/(r+2) & \text{if } (i = 0, j > 0) \text{ or } (i > 0, j = 0) \text{ or } (i = j > 0); \\ 0 & \text{otherwise.} \end{cases}$$

To see that $Y \in M(K, FR)$ observe that (a),(b) clearly hold, and let us now show that (c") holds.

$$Ye_i = \frac{1}{r+2}(e_0 + e_i) \in \mathrm{ST}(G) \subseteq K$$

and for $ij \in E$

$$Yf_{ij} = \frac{r}{r+2}e_0 + \sum_{l \neq 0,i,j} \frac{1}{r+2}e_l = \frac{r}{r+2}\left(e_0 + \sum_{l \neq 0,i,j} \frac{1}{r}e_l\right)$$

By the induction hypothesis we have that

$$\sum_{l \neq 0,i,j} \frac{1}{r}e_l \leq \sum_{l \neq 0} \frac{1}{r}e_l \in P$$

and the down-monotonicity of P implies that $Yf_{ij} \in K$, and thus (c") holds. We conclude that $Ye_0 \in N_{\mathrm{FR}}(K)$, i.e. $1/(r+2)\mathbf{1} \in N_{\mathrm{FR}}(P)$. □

Corollary 3.30. $1/(2r+2)\mathbf{1} \in N^r_{\mathrm{FR}}(G)$ *for all $r \geq 0$.*

Proof. Proceed by induction on r. We mentioned above that the case $r = 0$ is trivial. The inductive step follows from Lemma 3.29, since $N^r_{\mathrm{FR}}(\mathrm{FRAC}(G))$ clearly contains $\mathrm{STAB}(G)$ and is down-monotone by Corollary 3.7. □

Corollary 3.31. *T N_{FR}-rank of a graph G of stability number $\alpha(G)$ is at least $n/(2\alpha(G))-1$.*

Proof. Let r be the N-rank of G, and hence $N^r(G) = \mathrm{STAB}(G)$. By Corollary 3.30 we have that $1/(r+2)\mathbf{1} \in N^r(G)$. The inequality $\mathbf{1}^T x \leq \alpha(G)$ is valid for $\mathrm{STAB}(G) = N^r(G)$, and in particular for $1/(r+2)\mathbf{1}$, implying that $n/(2r+2) \leq \alpha(G)$, and the proof follows. □

For example, the N_{FR}-rank of a clique B is at least $|B|/2 - 1$ (since the stability number of B is 1), and it must be an integer, so we have that it is at least $\lfloor(|B|-1)/2\rfloor$. In fact, the above proof shows that the N_{FR}-rank of the clique constraint (3.19) is at least, and hence exactly, $\lfloor(|B|-1)/2\rfloor$. The N_{FR}-rank of an odd antihole D is at least $|D|/4 - 1$ (since the stability number of D is 2), and it must be an integer (while $|D|$ is odd), so we have that it is at least $\lfloor|D|/4\rfloor$. In fact, this shows that the N-rank of the odd antihole constraint (3.20) is at least $\lfloor|D|/4\rfloor$. Corollary 3.27 also yields a lower bound on the N-rank of the wheel constraint (3.21). Indeed, let r be the N-rank of this constraint. Then we have that this constraint is valid for $N^r(G)$ and, in particular, for $1/(r+2)\mathbf{1} \in N^r(G)$. Thus,

$$\frac{1}{r+2}\left(|W|-1+\frac{|W|-2}{2}\right) \leq \frac{|W|-2}{2}$$

which gives that $\frac{2(|W|-1)}{|W|-2}+1 \leq r+2$ and thus $r \geq 1+\frac{2}{|W|-2}$. Since the N-rank of the wheel constraint is an integer, it must be at least, and hence exactly, 2.

Upper bounds on the N_+-rank. Lovász and Schrijver [LS91] give also a sufficient condition for an inequality to be valid for $N_+(K)$. The following lemma considers an inequality $u^Tx \geq 0$ with $u_0 \geq 0$ and $u_i \leq 0$ for $i \geq 1$. It can be extended to an arbitrary inequality $u^Tx \geq 0$ by flipping the relevant coordinates according to Lemma 3.8.

Lemma 3.32 (Lovász and Schrijver [LS91]). *If for all i with $u_i < 0$, $u^Tx \geq 0$ is valid for $K \cap \{x : x_i = x_0\}$, then $u^Tx \geq 0$ is valid for $N_+(K)$.*

For the stable set problem, Lemma 3.32 implies the following lemma, which is described in the original n-dimensional space, i.e. by inequalities $a^Tx \leq b$ (with $a \in \mathbb{R}^n$) that are valid for STAB(G). Observe that the only non-trivial case is $b > 0$ and $a \geq 0$, and then we can use Lemma 3.32.

Lemma 3.33 (Lovász and Schrijver [LS91]). *If $a^Tx \leq b$ is an inequality valid for* STAB(G) *such that for all $i \in V$ with $a_i > 0$ the contraction of i gives an inequality with N_+-rank at most r, then $a^Tx \leq b$ has N_+-rank at most $r+1$.*

For example, the clique, odd hole, odd wheel, and odd antihole constraints all have N_+-rank at most (and thus exactly) 1. Lovász and Schrijver [LS91] show also that the so-called orthogonality constraints (see [Lov79, GLS93] for definition) are valid for N_+(FRAC) by definition, and hence their N_+-rank is also 1.

One simple way to derive facet-defining valid inequalities from other facet-defining inequalities is *cloning* a clique at a vertex i. That is, replacing the vertex i by a clique and replacing every edge incident to i by corresponding edges that are incident to all the clique vertices, and substituting the variable of i in the inequality with the sum of the variables of the clique vertices. In general, it is not clear how cloning influences the N_+-rank of an inequality. However, Goemans and Tunçel [GT00] note that Lemma 3.33 implies that cloning at the center vertex of an odd wheel inequality still has N_+-rank 1, and that cloning at one or several vertices of an odd wheel, odd hole, or odd antihole inequality, the N_+-rank is at most 2. Indeed, fixing any variable (of the corresponding subgraph) to 1, the resulting inequality can be seen to be a linear combination of clique inequalities and hence valid for N_+(FRAC).

Corollary 3.34 (Lovász and Schrijver [LS91]). *If $G - \Gamma(i) - i$ has N_+-rank at most r for every $i \in V$, then the N_+-rank of G is at most $r+1$.*

It follows for example, that the N_+-rank of a clique, an odd antihole or an odd wheel, is at most (and hence exactly) 1.

Constraint\Rank	N	N_{FR}	N_+	$N_{\mathrm{FR}+}$
odd hole (3.18)	1	1	1	1
clique (3.19)	$\|B\|-2$	$\lfloor(\|B\|-1)/2\rfloor$	1	1
antihole (3.20)	$(\|D\|-3)/2$	$\lfloor \|D\|/4\rfloor \leq \text{rank} \leq \lfloor(\|D\|+1)/4\rfloor$	1	1
wheel (3.21)	2	1	1	1

Table 3.1: The ranks of some example constraints

Corollary 3.35 (Lovász and Schrijver [LS91]). *The N_+-rank of a graph G is at most its stability number $\alpha(G)$.*

Note that Corollary 3.35 is tight for a clique.

Lower bounds on the N_+-rank. Lovász and Schrijver [LS91] give no general method to lower bound the N_+-rank. The approach taken by Stephen and Tunçel [ST99], Goemans and Tunçel [GT00], and Cook and Dash [CD00] is to obtain an analog of Corollary 3.27 that holds for a specific cone K. That is, they show that $N_+^r(K)$ contains a "uniform" solution that does not belong to K_I, and thus obtain that the N_+-rank of K must be larger than r. Our analysis in Section 3.5 also follows this approach.

We note that Goemans and Tunçel [GT00] give a sufficient condition for $N_+(K) = N(K)$ to hold, but this condition appears to be not applicable to the stable set problem.

The ranks of the example constraints are listed in Table 3.4.3.

3.5 The Lovász-Schrijver relaxations in a random graph

In this section we show that the N_+-rank of a random graph $G_{n,1/2}$ is $\Theta(\log n)$, almost surely. In particular, we analyze the asymptotic behavior of $\max\{\mathbf{1}^T x : x \in N_+^r(G)\}$ for $r = o(logn)$. Loosely speaking, we show that the value of this relaxation is "roughly" $\sqrt{n/2^r}$, almost surely.

Below are the precise formulations of our lower bound and upper bound on $\max\{\mathbf{1}^T x : x \in N_+^r(G)\}$. Our proofs extend the proof of Juhász [Juh82] which shows that the theta function of a random graph is almost surely $\Theta(\sqrt{n})$.

Theorem 3.7. *For any fixed $c > \sqrt{2}$ there exists a fixed $\epsilon' > 0$, such that if $0 \leq r \leq \epsilon' \log n$, then almost surely $\max\{\mathbf{1}^T x : x \in N_+^r(G_{n,1/2})\} \geq \sqrt{n}/c^{r+1}$.*

The proof of Theorem 3.7 appears in Section 3.5.1. Technically, we show that $N_+^r(G_{n,1/2})$ contains, almost surely, the "uniform" solution $(1/c^{r+1}\sqrt{n})\mathbf{1}$, and hence obtain a lower bound on the value of the relaxation.

To show that the above lower bound is nearly tight, we give in the next thereom an upper bound on the value of the relaxation. Its proof appears in Section 3.5.2.

Theorem 3.8. *For any fixed $d < \sqrt{2}$ there exists a fixed $\epsilon' > 0$, such that if $1 \leq r \leq \epsilon' \log n$, then almost surely $\max\{\mathbf{1}^T x : x \in N_+^r(G_{n,1/2})\} \leq 4\sqrt{n}/d^{r+1}$.*

It is straightforward that Theorem 3.2 follows from Theorems 3.7 and 3.8 by taking $c = \sqrt{2+\delta}$ and $d = \sqrt{2-\delta}$.

The N_+-rank of a random graph $G_{n,1/2}$. We can now use Theorem 3.7 and Corollary 3.35 to show that the N_+-rank of a random graph is almost surely logarithmic in n, proving Theorem 3.3. In comparison, the N-rank of a random graph is almost surely at least $\Omega(n/\log n)$ by Corollary 3.28, and at most $n - O(\log n)$ by Corollary 3.21.

Proof of Theorem 3.3. Let G be a random graph from the distribution $G_{n,1/2}$, and let us first show a lower bound on the N_+-rank. It is well known that, almost surely, the maximum size of a stable set in G is roughly $2\log_2 n$, i.e.

$$\max\{\mathbf{1}^T x : x \in \mathrm{STAB}\} \leq O(\log n)$$

We have from Theorem 3.7 with $r = \epsilon' \log n$ that, almost surely,

$$\max\{\mathbf{1}^T x : x \in N_+^r(\mathrm{FRAC})\} \geq n^{\Omega(1)}$$

It follows that $N_+^r(\mathrm{FRAC}) \neq \mathrm{STAB}$, and hence the N_+-rank of FRAC (and therefore of G), is larger than $r = \epsilon' \log n = \Omega(\log n)$.

The upper bound on N_+-rank of G follows from Corollary 3.35. Indeed, the stability number of a random graph $G_{n,1/2}$ is, almost surely, roughly $2\log_2 n$, and hence the N_+-rank of G is, almost surely, $O(\log n)$, as claimed. □

3.5.1 Lower bound on the value of $N_+^r(G_{n,1/2})$

We prove Theorem 3.7 by showing that $N_+^r(G_{n,1/2})$ contains, almost surely, the "uniform" solution $(1/c^{r+1}\sqrt{n})\mathbf{1}$. First we exhibit in Lemma 3.36 certain conditions that are sufficient for such a uniform solution to be feasible in $N_+^r(G_{n,1/2})$. We then show in Lemma 3.37 that these conditions are almost surely satisfied by a random graph $G_{n,1/2}$.

Notation. We will say that two vertices are *non-adjacent* if they are not adjacent and they are not equal (i.e. they are adjacent in the complement graph). We make no attempt to optimize constants.

Lemma 3.36. *Let G be a graph on n vertices, let $c = \sqrt{2}(1+\epsilon)^{10}$ for $0 < \epsilon < 1/5$ and let $r \geq 0$. Assume that for every $S \subset V$ with $|S| \leq r$, the graph $G' = G - S - \Gamma(S)$ satisfies (let n' denote the number of vertices in G'):*

(i) All eigenvalues of the adjacency matrix of $\overline{G'}$ are at least $-(1+\epsilon)\sqrt{n'}$.

(ii) The degree of every vertex in $\overline{G'}$ is between $\frac{1}{1+\epsilon}\frac{n'}{2}$ and $(1+\epsilon)\frac{n'}{2}$.

If $c^{r+1} \leq \epsilon\sqrt{n}$ then $(1/c^{r+1}\sqrt{n})\mathbf{1} \in N_+^r(G)$.

Proof. Proceed by induction on r. For the base case $r = 0$, observe that $(1/c^{r+1}\sqrt{n})\mathbf{1}$ satisfies the nonnegativity and edge constraints and therefore is in $\mathrm{FR}(G)$ by definition.

For the inductive step, assume it holds for $r \geq 0$, and let us show that it holds for $r+1$. Let G be a graph with (i),(ii) holding for any $|S| \leq r+1$, and $c^{r+2} \leq \epsilon\sqrt{n}$. We can choose, in particular, $|S| = 0$ and have that (i),(ii) hold for the graph G itself. To ease notation, define

$$\mu := (1+\epsilon)^5 (c^{r+1}/\sqrt{2})\sqrt{n} \tag{3.22}$$

Let A be the $n \times n$ adjacency matrix of $\overline{G}$, i.e. $A_{ij} = 0$ whenever $(i,j) \in E$ or $i = j$ and $A_{ij} = 1$ otherwise. We know from (i) that all eigenvalues of A are at least $-(1+\epsilon)\sqrt{n} \geq -\mu$. Hence, the matrix $B = A + \mu I$ is positive semidefinite, and there exist vectors $z_1, \ldots, z_n$ such that $B_{ij} = z_i^T z_j$. Therefore

$$\|z_i\|^2 = B_{ii} = \mu, \quad \forall i \geq 1. \tag{3.23}$$

Let $z_0 = \sum_{i=1}^n z_i$. Then

$$\|z_0\|^2 = (\sum_{i>0} z_i)^T (\sum_{j>0} z_j) = \sum_{i,j>0} B_{ij}. = \sum_{i>0}\sum_{j>0} B_{ij}$$

To estimate $\sum_{j>0} B_{ij} = \sum_{j>0} A_{ij} + \mu$ for $i > 0$, observe that we have from (ii) that

$$\frac{1}{1+\epsilon}\frac{n}{2} \leq \sum_{j>0} A_{ij} \leq (1+\epsilon)\frac{n}{2}$$

while $\mu \leq (c^{r+2}/2)\sqrt{n} \leq \epsilon n/2$. Hence,

$$\frac{1}{1+\epsilon}\frac{n}{2} \leq \sum_{j>0} B_{ij} \leq (1+\epsilon)^2 \frac{n}{2}, \tag{3.24}$$

and we conclude that

$$\frac{1}{1+\epsilon}\frac{n^2}{2} \leq \|z_0\|^2 \leq (1+\epsilon)^2 \frac{n^2}{2} \tag{3.25}$$

For every $i \geq 0$ let v_i be the unit length vectors in the direction of the vector z_i, i.e. $v_i = z_i/\|z_i\|$, and let $x_i = (v_i^T v_0)^2$. Observe that $x_0 = (v_0^T v_0)^2 = 1$.

We claim that $x = (x_1, \ldots, x_n)^T$ is in $N_+^{r+1}(G)$. Let us first show how the proof of Lemma 3.36 follows from this claim. Indeed, from (ii) we have that

$$v_i^T v_0 = (\frac{z_i}{\|z_i\|})^T (\frac{\sum_{j>0} z_j}{\|z_0\|}) = \frac{\sum_{j>0} B_{ij}}{\sqrt{\mu}\|z_0\|}$$

Together with (3.24) and (3.25) we can estimate $x_i = (v_i^T v_0)^2$ by

$$\frac{1}{(1+\epsilon)^4} \cdot \frac{1}{2\mu} \leq x_i \leq (1+\epsilon)^5 \frac{1}{2\mu} \tag{3.26}$$

and from (3.22) we have that

$$x_i \geq \frac{1}{2(1+\epsilon)^4} \cdot \frac{\sqrt{2}}{(1+\epsilon)^5 c^{r+1}\sqrt{n}} \geq \frac{1}{c^{r+2}\sqrt{n}}$$

and thus $(1/c^{r+2}\sqrt{n})\mathbf{1} \le x \in N_+^{r+1}(G)$. By the monotonicity guaranteed in Corollary 3.7 we have $(1/c^{r+2}\sqrt{n})\mathbf{1} \in N_+^{r+1}(G)$, which indeed proves the inductive step.

We now prove the claim $x \in N_+^{r+1}(G)$, by presenting a matrix $Y \in M_+(N_+^r(G))$ whose 0th column corresponds to x. Indeed, let Y be the $(n+1)\times(n+1)$ matrix defined by $Y_{ij} = (v_i^T v_j)\sqrt{x_i x_j}$ for all $i, j \ge 0$. By definition, $Y_{i0} = (v_i^T v_0)\sqrt{x_i} = x_i$ for $i \ge 0$, and in particular $Y_{00} = x_0 = 1$. We will show that Y satisfies (a),(b),(c") and (d). Three of them are straightforward:

(a) Y is symmetric by definition.

(b) $Y_{ii} = \|v_i\|^2 x_i = x_i$ and hence Hence $Y_{ii} = x_i = Y_{i0}$.

(d) Y is positive semidefinite because it can be represented by the vectors $\{\sqrt{x_i}v_i\}$, i.e. $Y_{ij} = (\sqrt{x_i}v_i)^T(\sqrt{x_j}v_j)$ for all $i, j \ge 0$.

Before proving (c"), observe that for $i, j > 0$ we have

$$Y_{ij} = \left(\frac{z_i}{\|z_i\|}\right)^T\left(\frac{z_j}{\|z_j\|}\right)\sqrt{x_i x_j} = (1/\mu)B_{ij}\sqrt{x_i x_j}$$

and B_{ij} is either μ, 0 or 1. So for $i, j > 0$ we have

$$Y_{ij} = \begin{cases} x_i & \text{if } i = j \\ 0 & \text{if } i \ne j \text{ and } ij \in E \\ (1/\mu)\sqrt{x_i x_j} & \text{if } i \ne j \text{ and } ij \notin E \end{cases}$$

and the estimate of (3.26) gives that $x_i \sim 1/2\mu$ and $\sqrt{x_i x_j} \sim 1/2\mu$. Hence,

$$Y = \begin{bmatrix} 1 & x_1 & \cdots & x_n \\ x_1 & x_1 & 0 \,\Big|\, \frac{\sqrt{x_i x_j}}{\mu} & \\ \vdots & & \ddots & \\ x_n & 0 \,\Big|\, \frac{\sqrt{x_i x_j}}{\mu} & & x_n \end{bmatrix} \sim \begin{bmatrix} 1 & \frac{1}{2\mu} & \cdots & \frac{1}{2\mu} \\ \frac{1}{2\mu} & \frac{1}{2\mu} & 0 \,\Big|\, \frac{1}{2\mu^2} & \\ \vdots & & \ddots & \\ \frac{1}{2\mu} & 0 \,\Big|\, \frac{1}{2\mu^2} & & \frac{1}{2\mu} \end{bmatrix}$$

Consider Ye_i, the ith column of Y, for $i > 0$, and scale it by a factor of $1/x_i$ so that its 0th entry will be 1. We get a fractional solution where vertex i has value 1, its adjacent vertices have value 0, and its non-adjacent vertices j have value $(1/\mu)\sqrt{x_j/x_i} \sim 1/\mu$. Let G' be the subgraph of G induced on the latter vertices (i.e. those non-adjacent to i), and let n' denote the number of vertices in G'. Then by Lemma 3.16, we have that the fractional solution Ye_i is in $N_+^r(G)$ if and only if its restriction to G' is in $N_+^r(G')$. Each coordinate in the fractional solution restricted to G' is bounded by

$$\frac{1}{\mu}\sqrt{\frac{x_j}{x_i}} \le \frac{1}{\mu}(1+\epsilon)^{9/2} \le \frac{\sqrt{2}}{c^{r+1}\sqrt{n(1+\epsilon)}} \le \frac{1}{c^{r+1}\sqrt{n'}}$$

where the first inequality is due to (3.26), the second is due to (3.22), and the third follows from $n' \le (1+\epsilon)\frac{n}{2}$ which we have from (ii). The fractional solution restricted to G' is thus dominated by the uniform solution $(1/c^{r+1}\sqrt{n'})\mathbf{1}$, which belongs to $N_+^r(G')$ by applying the induction hypothesis to G'. (Note that G' satisfies (i),(ii) for any $0 \leq |S| \leq r$ by definition, and that we have $c^{r+1} \leq \epsilon\sqrt{n}/c \leq \epsilon\sqrt{n'}$.) From the monotonicity guaranteed by Corollary 3.7, we conclude that also the fractional solution restricted to G' is in $N_+^r(G')$, and therefore $Ye_i \in N_+^r(G)$.

Consider Yf_i, the difference between column 0 and column i of Y, for $i > 0$. Its 0th entry is $1 - x_i \sim 1 - 1/2\mu$, its ith entry is 0, and any other jth entry is at most roughly $1/2\mu$. Observe that

$$x_i \leq \frac{(1+\epsilon)^5}{2\mu} \leq \frac{1}{\sqrt{2n}} \leq 1 - \frac{1}{\sqrt{2}} \tag{3.27}$$

where the first inequality is due to (3.26), the second is due to (3.22) and the third is due to $\sqrt{n} \geq 5\epsilon\sqrt{n} \geq 5c^{r+2} > 10$. Scaling the vector Yf_i by a factor $1/(1-x_i)$ so that its 0th entry is 1, we obtain a fractional solution in which the value of the jth entry is at most

$$\frac{x_j}{1-x_i} \leq \frac{(1+\epsilon)^5/2\mu}{1/\sqrt{2}} = \frac{1}{c^{r+1}\sqrt{n}}.$$

The fractional solution is thus dominated by $(1/c^{r+1}\sqrt{n})\mathbf{1}$, which by the induction hypothesis belongs to $N_+^r(G)$. (Note that G satisfies the requirements for r). From the monotonicity guaranteed by Corollary 3.7, (as all entries of Yf_i are nonnegative) we conclude that $Yf_i \in N_+^r(G)$.

We therefore have that (c") holds, which completes the proof of the inductive step and of Lemma 3.36.

□

The proof of Lemma 3.36 extends also to $N_{\mathrm{FR}+}^r(G)$. Indeed, we need to consider also Yf_{ij} for $ij \in E$. The 0th entry of this vector is $1 - x_i - x_j \sim 1 - 2/2\mu$, the ith and jth entries are 0, and any other kth entry is either roughly $1/2\mu$ if k is adjacent to both i, j, or roughly $1/2\mu - 2/2\mu^2 \sim 1/2\mu$ if k is non-adjacent to both i, j, or roughly $1/2\mu - 1/2\mu^2 \sim 1/2\mu$ if k is adjacent to exactly one of i, j. Similar to (3.27) we have that

$$x_i + x_j \leq 2 \cdot \frac{1}{\sqrt{2n}} \leq 1 - \frac{1}{\sqrt{2}}.$$

Scaling this vector (by a small factor) so that the 0th entry is 1, we obtain a fractional solution in which the value of the kth entry is at most

$$\frac{x_k}{1 - x_i - x_j} \leq \frac{(1+\epsilon)^5/2\mu}{1/\sqrt{2}} = \frac{1}{c^{r+1}\sqrt{n}}.$$

The fractional solutions is thus dominated by $(1/c^{r+1}\sqrt{n})\mathbf{1}$, which by the induction hypothesis belongs to $N_+^r(G)$. From the monotonicity guaranteed by Corollary 3.7, (as all entries of Yf_{ij} are nonnegative) we conclude that $Yf_{ij} \in N_+^r(G)$.

Lemma 3.37. *Let $\epsilon > 0$ be fixed. Then there exists a fixed $\epsilon' > 0$ that depends on ϵ, such that for any $r \leq \epsilon' \log n$, a random graph $G_{n,1/2}$ almost surely satisfies the requirements of Lemma 3.36.*

Proof. Observe that a sufficiently small $\epsilon' > 0$ that depends on ϵ guarantees that $c^{r+1} \leq \epsilon\sqrt{n}$ (we can assume, without loss of generality, that $\epsilon < 1/5$).

Consider a particular choice of S of size $s \leq r$, and its corresponding graph $G'(V', E')$ (the subgraph of G induced on the vertices that are non-adjacent to all the vertices of S). The number of vertices in G', which we denote by $n' = |V'|$, has binomial distribution $B(n-s, 1/2^s)$. Since $s \leq \log n \leq n/4$, we have by Chernoff bound that

$$\Pr\left[n' \leq n/2^{s+1}\right] \leq 2^{-\delta_1 n/2^s} \tag{3.28}$$

for some fixed $\delta_1 > 0$.

G' is a random graph (with edge probability $1/2$) on n' vertices. Therefore, the adjacency matrix of $\overline{G'}$ is a random symmetric matrix and we can use results on the concentration of its eigenvalues. In particular, we have from Krivelevich and Vu [KV00] (which improve the concentration shown by Füredi and Kómlos [FK81], see also [AKV01]) that

$$\Pr\left[G' \text{ does not satisfy (i)}\right] \leq 2^{-\delta_2 n'} \tag{3.29}$$

for some $\delta_2 > 0$ that depends on ϵ.

Since G' is a random graph, the degree of a particular vertex in $\overline{G'}$ has binomial distribution $B(n'-1, 1/2)$. By Chernoff bound and the union bound on the n' vertices we have that

$$\Pr\left[G' \text{ does not satisfy (ii)}\right] \leq n' 2^{-\delta_3 n'} \tag{3.30}$$

for some fixed $\delta_3 > 0$ that depends on ϵ.

Using the union bound on the events of (3.29) and (3.30) we can bound the probability that G' does not satisfy (i) or (ii). In order to obtain a bound in terms of n (rather than n'), we add to the union bound also the event of (3.28) and have that for some fixed $\delta > 0$ that depends on ϵ,

$$\Pr\left[G' \text{ does not satisfy (i) or (ii)}\right] \leq n 2^{-\delta n/2^s}$$

Taking the union bound on all possible sets S of size at most r, the probability that the requirements of Lemma 3.36 do not hold is at most

$$\sum_{s=0}^{r} \binom{n}{s} n 2^{-\delta n/2^s} \leq r n^{r+1} 2^{-\delta n/2^r} \leq n^{r+2} 2^{-\delta n/2^r} \ll 1$$

when $r \leq \epsilon' \log n$ for a sufficiently small fixed $\epsilon' > 0$ that depends on ϵ, and hence these requirements hold almost surely. □

The proof of Theorem 3.7 follows from Lemma 3.36 and Lemma 3.37.

3.5.2 Upper bound on the value of $N_+^r(G_{n,1/2})$

To prove Theorem 3.8 we first exhibit in Lemma 3.38 certain conditions that are sufficient for the inequality $\mathbf{1}^T x \leq 4\sqrt{n}/d^{r+1}$ to be valid for $N_+^r(G)$. We then show in Lemma 3.39 that these conditions are almost surely satisfied by a random graph $G_{n,1/2}$.

The Lovász theta function of a graph is defined as $\vartheta(G) = \max\{\mathbf{1}^T x : x \in TH(G)\}$, where $TH(G)$ is the solution set of the nonnegativity constraints (3.8) and the so-called orthogonality constraints (see [Lov79, GLS93] for definition). Lovász and Schrijver [LS91] show that the orthogonality constraints have N_+-rank at most 1, and hence $N_+(G) \subseteq TH(G)$.

Lemma 3.38. *Let G be a graph on n vertices, let $d = \sqrt{2}(1-\epsilon)$ for $0 < \epsilon < 1$ and let $r \geq 1$. Assume that for every $S \subset V$ with $|S| \leq r$, the graph $G' = G - S - \Gamma(S)$ satisfies (let n' denote the number of vertices in G'):*

(i) $\vartheta(G') \leq 2(1+\epsilon)\sqrt{n'}$.

(ii) The degree of every vertex in $\overline{G'}$ is between $\frac{1}{1+\epsilon}\frac{n'}{2}$ and $(1+\epsilon)\frac{n'}{2}$.

If $d^{r+1} \leq \epsilon^2\sqrt{n}$ then $\max\{\mathbf{1}^T x : x \in N_+^r(G)\} \leq 4\sqrt{n}/d^{r+1}$.

Proof. Proceed by induction on r. For the base case $r = 1$, we can choose $|S| = 0$ and then (i) and (ii) hold for the graph G itself. In particular, we have that

$$\max\{\mathbf{1}^T x : x \in N_+(G)\} \leq \vartheta(G) \leq 2(1+\epsilon)\sqrt{n} < 4\sqrt{n}/d^2$$

For the inductive step, assume it holds for $r \geq 1$ and let us show that it holds for $r+1$. In other words, given a graph G with (i),(ii) holding for any $|S| \leq r+1$, we will prove that the inequality $\mathbf{1}^T x \leq 4\sqrt{n}/d^{r+2}$ is valid for $N_+^{r+1}(G)$. By Lemma 3.33 we know that it suffices to prove that for every vertex v, the inequality that arises from the contraction of v, i.e. $\mathbf{1}^T x \leq 4\sqrt{n}/d^{r+2} - 1$, is valid for $N_+^r(G - \Gamma(v) - v)$.

By the induction hypothesis for $G' = G - \Gamma(v) - v$ we have that $\max\{\mathbf{1}^T x : x \in N_+^r(G')\} \leq 4\sqrt{n'}/d^{r+1}$, i.e. the inequality $\mathbf{1}^T x \leq 4\sqrt{n'}/d^{r+1}$ is valid for $N_+^r(G')$. Since (ii) holds also for G itself, we have that $n' \leq (1+\epsilon)\frac{n}{2}$, and hence

$$\frac{4\sqrt{n'}}{d^{r+1}} \leq \frac{4\sqrt{n}}{d^{r+1}}\frac{\sqrt{1+\epsilon}}{\sqrt{2}} = \frac{4\sqrt{n}}{d^{r+2}}\sqrt{1+\epsilon}(1-\epsilon) \leq \frac{4\sqrt{n}(1-\epsilon^2)}{d^{r+2}} \leq \frac{4\sqrt{n}}{d^{r+2}} - 1$$

where the last inequality follows from $d^{r+2} \leq 4\epsilon^2\sqrt{n}$. Therefore we have that for $N_+^r(G')$ the inequality $\mathbf{1}^T x \leq 4\sqrt{n'}/d^{r+1} \leq 4\sqrt{n}/d^{r+2} - 1$ holds, which completes the proof of the inductive step. $\square$

Lemma 3.39. *Let $\epsilon > 0$ be fixed. Then there exists a fixed $\epsilon' > 0$ that depends on ϵ, such that for any $r \leq \epsilon' \log n$, a random graph $G_{n,1/2}$ almost surely satisfies the requirements of Lemma 3.38.*

Proof. The proof is similar to the proof of Lemma 3.37, but with the different requirement (i). Juhász [Juh82] shows that $\vartheta(G')$ is at most $(2 + o(1))\sqrt{n'}$, almost surely, by using the result of Füredi and Kómlos [FK81] on the concentration of eigenvalues of random symmetric matrices. By using the stronger concentration result of Krivelevich and Vu [KV00] (see also [AKV01]), we have that (3.29) holds also here, and the proof follows. $\square$

The proof of Theorem 3.8 follows from Lemma 3.38 and Lemma 3.39.

Bibliography

[ABSS97] S. Arora, L. Babai, J. Stern, and Z. Sweedyk. The hardness of approximate optima in lattices, codes, and systems of linear equations. *J. Comput. System Sci.*, 54(2, part 2):317–331, 1997.

[AKK99] S. Arora, D. Karger, and M. Karpinski. Polynomial time approximation schemes for dense instances of NP-hard problems. *J. Comput. System Sci.*, 58(1):193–210, 1999.

[AKS98] N. Alon, M. Krivelevich, and B. Sudakov. Finding a large hidden clique in a random graph. *Random Structures Algorithms*, 13(3-4):457–466, 1998.

[AKV01] N. Alon, M. Krivelevich, and V. H. Vu. On the concentration of eigenvalues of random symmetric matrices. Manuscript, January 2001.

[Ali95] Farid Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM J. Optim.*, 5(1):13–51, 1995.

[ALM$^+$98] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.

[AR98] Y. Aumann and Y. Rabani. An $O(\log k)$ approximate min-cut max-flow theorem and approximation algorithm. *SIAM J. Comput.*, 27(1):291–301, 1998.

[Aro98] S. Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, 1998.

[AS92] N. Alon and J. H. Spencer. *The probabilistic method.* John Wiley & Sons, Inc., New York, 1992.

[AS98] S. Arora and S. Safra. Probabilistic checking of proofs: a new characterization of NP. *J. ACM*, 45(1):70–122, 1998.

[BCLS87] T. N. Bui, S. Chaudhuri, F. T. Leighton, and M. Sipser. Graph bisection algorithms with good average case behavior. *Combinatorica*, 7(2):171–191, 1987.

[BH92] R. Boppana and M. M. Halldórsson. Approximating maximum independent sets by excluding subgraphs. *BIT*, 32:180–196, 1992.

[BJ92] T. N. Bui and C. Jones. Finding good approximate vertex and edge partitions is NP-hard. *Inform. Process. Lett.*, 42(3):153–159, 1992.

[Bop87] R. B. Boppana. Eigenvalues and graph bisection: An average-case analysis. In *28th Annual Symposium on Foundations of Computer Science*, pages 280–285, October 1987.

[BS95] A. Blum and J. Spencer. Coloring random and semi-random k-colorable graphs. *J. Algorithms*, 19(2):204–234, September 1995.

[CD00] W. Cook and S. Dash. On the matrix-cut rank of polyhedra. Manuscript, August 2000.

[CK95] P. Crescenzi and V. Kann. A compendium of NP optimization problems. Technical Report SI/RR-95/02, Dipartimento di Scienze dell'Informazione, Università di Roma La Sapienza, 1995. This list is updated continuously and can be found at `http://www.nada.kth.se/~viggo/problemlist`.

[CK99] A. Condon and R. M. Karp. Algorithms for graph partitioning on the planted partition model. In *Randomization, approximation, and combinatorial optimization*, pages 221–232. Springer, Berlin, 1999.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press/McGraw-Hill, Cambridge, Massachusetts, 1990.

[DF89] M. E. Dyer and A. M. Frieze. The solution of some random NP-hard problems in polynomial expected time. *J. Algorithms*, 10(4):451–489, 1989.

[DF99] R. G. Downey and M. R. Fellows. *Parameterized complexity.* Springer-Verlag, New York, 1999.

[DKRS99] I. Dinur, G. Kindler, R. Raz, and S. Safra. An improved lower bound for approximating CVP. Manuscript, 1999.

[DS99] I. Dinur and S. Safra. On the hardness of approximating label cover. Technical Report TR99-015, ECCC, 1999.

[ENRS97] G. Even, J. Naor, S. Rao, and B. Schieber. Fast approximate graph partitioning algorithms. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 639–648. ACM, New York, 1997.

[ENRS99] G. Even, J. Naor, S. Rao, and B. Schieber. Fast approximate graph partitioning algorithms. *SIAM J. Comput.*, 28(6):2187–2214, 1999.

[EP00] M. Elkin and D. Peleg. Strong inapproximability of the basic k-spanner problem. In *27th International Colloquium on Automata, Languages and Programming*, pages 636–647. Springer, 2000.

[FdlVL81] W. Fernandez de la Vega and G. S. Lueker. Bin packing can be solved within $1 + \varepsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981.

[Fei97] U. Feige. Randomized graph products, chromatic numbers, and the Lovász ϑ-function. *Combinatorica*, 17(1):79–90, 1997.

[Fei98] U. Feige. A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[Fei00] U. Feige. Approximating the bandwidth via volume respecting embeddings. *J. Comput. System Sci.*, 60(3):510–539, 2000.

[FGL$^+$96] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques. *J. ACM*, 43(2):268–292, March 1996.

[FK81] Z. Füredi and J. Komlós. The eigenvalues of random symmetric matrices. *Combinatorica*, 1(3):233–241, 1981.

[FK98] U. Feige and J. Kilian. Zero knowledge and the chromatic number. *J. Comput. System Sci.*, 57(2):187–199, 1998.

[FK00a] U. Feige and R. Krauthgamer. Finding and certifying a large hidden clique in a semirandom graph. *Random Structures Algorithms*, 16(2):195–208, 2000.

[FK00b] U. Feige and R. Krauthgamer. A polylogarithmic approximation of the minimum bisection. In *41st Annual IEEE Symposium on Foundations of Computer Science*, pages 105–115, November 2000.

[FK01a] U. Feige and J. Kilian. Heuristics for semirandom graph problems. A preliminary version appeared in FOCS'98, pp. 674–683. To appear in J. Comput. System Sci., 2001.

[FK01b] U. Feige and R. Krauthgamer. The probable value of the Lovász-Schrijver relaxations for maximum independent set. Manuscript, April 2001.

[FKN00] U. Feige, R. Krauthgamer, and K. Nissim. Approximating the minimum bisection size. In *32nd Annual ACM Symposium on Theory of Computing*, pages 530–536, May 2000.

[FM97] A. Frieze and C. McDiarmid. Algorithmic theory of random graphs. *Random Structures Algorithms*, 10(1-2):5–42, 1997.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* W.H. Freeman and Company, 1979.

[GJS76] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoret. Comput. Sci.*, 1(3):237–267, 1976.

[GKR$^+$99] V. Guruswami, S. Khanna, R. Rajaraman, B. Shepherd, and M. Yannakakis. Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. In *31st Annual ACM Symposium on Theory of Computing*, pages 19–28. ACM, 1999.

[GLS93] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric algorithms and combinatorial optimization.* Springer-Verlag, Berlin, second edition, 1993.

[GSV99] N. Garg, H. Saran, and V. V. Vazirani. Finding separator cuts in planar graphs within twice the optimal. *SIAM J. Comput.*, 29(1):159–179, 1999.

[GT00] M. X. Goemans and L. Tunçel. When does the positive semidefiniteness constraint help in lifting procedures. Manuscript, January 2000.

[Hal93] M. M. Halldórsson. A still better performance guarantee for approximate graph coloring. *Inform. Process. Lett.*, 45(1):19–23, 1993.

[Hås97] J. Håstad. Some optimal inapproximability results. In *29th Annual ACM Symposium on Theory of Computing*, pages 1–10, El Paso, Texas, 4–6 May 1997.

[Hås99] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Math.*, 182(1):105–142, 1999.

[HJ85] R. A. Horn and C. R. Johnson. *Matrix analysis.* Cambridge University Press, Cambridge-New York, 1985.

[Hoc97] D. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Company, 1997.

[IK75] O. H. Ibarra and C. E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM*, 22(4):463–468, 1975.

[Jer92] M. Jerrum. Large cliques elude the Metropolis process. *Random Structures Algorithms*, 3(4):347–359, 1992.

[Joh74] D. S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. System Sci.*, 9:256–278, 1974.

[JP00] A. Juels and M. Peinado. Hiding cliques for cryptographic security. *Des. Codes Cryptogr.*, 20(3):269–280, 2000.

[JS98] M. Jerrum and G. B. Sorkin. The Metropolis algorithm for graph bisection. *Discrete Appl. Math.*, 82(1-3):155–175, 1998.

[JT96] D. S. Johnson and M. A. Trick, editors. *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, 1993*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science.* American Mathematical Society, 1996.

[Juh82] F. Juhász. The asymptotic behaviour of Lovász' θ function for random graphs. *Combinatorica*, 2(2):153–155, 1982.

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations (Proc. Sympos.)*, pages 85–103. Plenum Press, 1972.

[Kar76] R. M. Karp. The probabilistic analysis of some combinatorial search algorithms. In *Algorithms and complexity (Proc. Sympos.)*, pages 1–19. Academic Press, 1976.

[KMR97] D. Karger, R. Motwani, and G. D. S. Ramkumar. On approximating the longest path in a graph. *Algorithmica*, 18(1):82–98, 1997.

[Knu94] D. E. Knuth. The sandwich theorem. *Electron. J. Combin.*, 1:Article 1, approx. 48 pp. (electronic), 1994.

[KPR93] P. Klein, S. A. Plotkin, and S. Rao. Excluded minors, network decomposition, and multicommodity flow. In *25th Annual ACM Symposium on Theory of Computing*, pages 682–690, May 1993.

[KS96] D. R. Karger and C. Stein. A new approach to the minimum cut problem. *J. ACM*, 43(4):601–640, 1996.

[Kuč95] L. Kučera. Expected complexity of graph partitioning problems. *Discrete Appl. Math.*, 57(2-3):193–212, 1995.

[KV00] M. Krivelevich and V. H. Vu. Approximating the independence number and the chromatic number in expected polynomial time. In *27th International Colloquium on Automata, Languages and Programming*, pages 13–24. Springer, 2000.

[Lev86] L. A. Levin. Average case complete problems. *SIAM J. Comput.*, 15(1):285–286, 1986.

[LLKS85] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D . B. Shmoys, editors. *The Traveling Salesman Problem.* Wiley-Interscience series in discrete mathematics, 1985.

[LLR95] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995.

[Lov79] L. Lovász. On the Shannon capacity of a graph. *IEEE Trans. Inform. Theory*, 25(1):1–7, 1979.

[Lov94] L. Lovász. Stable sets and polynomials. *Discrete Math.*, 124(1-3):137–153, 1994.

[LR88] F. T. Leighton and S. Rao. An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms. In *29th Annual Symposium on Foundations of Computer Science*, pages 422–431, October 1988.

[LR99] T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, 1999.

[LS91] L. Lovász and A. Schrijver. Cones of matrices and set-functions and 0-1 optimization. *SIAM J. Optim.*, 1(2):166–190, 1991.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[PY91] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. System Sci.*, 43(3):425–440, December 1991.

[Rei94] G. Reinelt. *The Traveling Salesman*, volume 840 of *Lecture Notes in Computer Science.* Springer-Verlag, Berlin, 1994.

[SFVM98] C. R. Subramanian, M. Fürer, and C. E. Veni Madhavan. Algorithms for coloring semi-random graphs. *Random Structures Algorithms*, 13(2):125–158, 1998.

[Shm97] D.B. Shmoys. Cut problems and their applications to divide-and-conquer. In D. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Company, 1997.

[ST97] H. D. Simon and S. Teng. How good is recursive bisection? *SIAM J. Sci. Comput.*, 18(5):1436–1445, 1997.

[ST99] T. Stephen and L. Tunçel. On a representation of the matching polytope via semidefinite liftings. *Math. Oper. Res.*, 24(1):1–7, 1999.

[Sub99] C. R. Subramanian. Minimum coloring k-colorable graphs in polynomial average time. *J. Algorithms*, 33(1):112–123, 1999.

[SV95] H. Saran and V. V. Vazirani. Finding k cuts within twice the optimal. *SIAM J. Comput.*, 24(1):101–108, 1995.

[Vaz01] V. Vazirani. *Approximation algorithms.* Springer Verlag, 2001. To appear.