Edge expansion of a graph: SDP-based computational strategies *

Akshay Gupte [1], Melanie Siebenhofer [2], and Angelika Wiegele [3]

[1] University of Edinburgh & Maxwell Institute for Mathematical Sciences, UK, akshay.gupte@ed.ac.uk

[2] Alpen-Adria-Universität Klagenfurt, Austria, melanie.siebenhofer@aau.at

[3] Alpen-Adria-Universität Klagenfurt, Austria & Universität zu Köln, Germany, angelika.wiegele@aau.at

April 25, 2024

Computing the edge expansion of a graph is a famously hard combinatorial problem for which there have been many approximation studies. We present two variants of exact algorithms using semidefinite programming (SDP) to compute this constant for any graph. The first variant uses the SDP relaxation first to reduce the search space considerably. One implementation of this variant applies then an SDP-based branch-and-bound algorithm, along with heuristic search. The second implementation transforms the problem into an instance of a max-cut problem and solves this using an SDP-based state-of-the-art solver. Our second variant to compute the edge expansion uses Dinkelbach's algorithm for fractional programming. This is, we have to solve a parametrized optimization problem and again we use semidefinite programming to obtain solutions of the parametrized problems. Numerical

*A proceedings article containing part of this work will appear in [15]. This article now contains full and complete proofs, an additional algorithm based on a Discrete Netwon-Dinkelbach method, and more computational results.

This research was funded in part by the Austrian Science Fund (FWF) [10.55776/DOC78]. For open access purposes, the authors have applied a CC BY public copyright license to any author-accepted manuscript version arising from this submission.

results demonstrate that with our algorithms one can compute the edge expansion on graphs up to 400 vertices in a routine way, including instances where standard branch-and-cut solvers fail. To the best of our knowledge, these are the first SDP-based solvers for computing the edge expansion of a graph.

Keywords: Edge expansion, Cheeger constant, bisection problems, semidefinite programming, parametric submodular minimization

1. Introduction

Let $G = (V, E)$ be a simple connected graph on $n \geq 3$ vertices and with m edges. A *cut* (S, S'), for any $\emptyset \neq S \subset V$ and $S' = V \setminus S$, in G is a partition of its vertices, and the *cut-set* ∂S is the edges between S and S'. The *(unweighted) edge expansion*, also called the *Cheeger constant* or *isoperimetric number* or *sparsest cut*, of G is a ratio that measures the relative number of edges across any vertex partition. It is defined as

$$\begin{aligned} h(G) &= \min_S \left\{ \frac{|\partial S|}{\min\{|S|, |S'|\}} : \emptyset \neq S \subset V \right\} \\ &= \min_S \left\{ \frac{|\partial S|}{|S|} : S \subset V,\, 1 \leq |S| \leq \frac{n}{2} \right\}, \end{aligned}$$

where $\partial S = \{(i, j) \in E : i \in S,\, j \in S'\}$ is the cut-set associated with any vertex subset $S \subset V$, and $S' = V \setminus S$. This constant is positive if and only if the graph is connected, and the exact value tells us that the number of edges across any cut in G is at least $h(G)$ times the number of vertices in the smaller partition. A weighted definition of edge expansion called the *conductance* of a graph, is

$$\begin{aligned} h_{\mathrm{vol}}(G) &= \min_S \left\{ \frac{|\partial S|}{\min\{\mathrm{vol}(S), \mathrm{vol}(S')\}} : \emptyset \neq S \subset V \right\} \\ &= \min_S \left\{ \frac{|\partial S|}{\mathrm{vol}(S)} : S \subset V,\, 1\,\mathrm{vol}(S) \leq m \right\}, \end{aligned}$$

where $\mathrm{vol}(S) = \sum_{v \in S} \deg(v)$, and the second equality is due to $\mathrm{vol}(S) + \mathrm{vol}(S') = 2m$.

Edge expansions arise in the study of expander graphs, for which there is a rich body of literature with applications in network science, coding theory, cryptography, complexity theory, cf. [13, 21, 39]. A graph with $h(G) \geq c$, for some constant $c > 0$, is called a c-expander. A graph with $h(G) < 1$ is said to have a bottleneck since there are not too many edges across it. A threshold for good expansion properties is having $h(G) \geq 1$, which is desirable in many of the above applications. The famous Mihail-Vazirani conjecture [11, 32] in polyhedral combinatorics claims that the graph (1-skeleton) of any 0/1-polytope has edge expansion at least 1. This has been proven to be true for several combinatorial polytopes [22, 32] and bases-exchange graphs of matroids [1], and a weaker form was established recently for random 0/1-polytopes [28].

Lattice polytopes were constructed in [14] with the property that in every dimension their graphs lie on the threshold of being good expanders (i.e., $h(G) = 1$).

Computing the edge expansion is related to the *uniform sparsest cut* problem which asks for computing a cut in the graph with the smallest sparsity, where sparsity is defined as the ratio of the size of the cut to the product of the sizes of the two partitions,

$$\begin{aligned}\phi(G) &= \min_S \left\{ \frac{|\partial S|}{|S||S'|} : \emptyset \neq S \subset V \right\} \\ &= \min_S \left\{ \frac{|\partial S|}{|S||S'|} : S \subset V,\, 1 \leq |S| \leq \frac{n}{2} \right\}.\end{aligned}$$

Since ${}^{n}\!/_{2} \leq |S'| \leq n$, it holds that $|S||S'| \leq n \cdot |S| \leq 2|S||S'|$, and hence $h(G) \leq n \cdot \phi(G) \leq 2h(G)$, which implies that the edge expansion problem is related to the sparsest cut problem up to a constant factor of 2. In particular, any cut (S, S') that is α-approx for $\phi(G)$ (resp. $h(G)$) is a 2α-approx for $h(G)$ (resp. $\phi(G)$), because $|\partial S|/|S| \leq n|\partial S|/(|S||S'|) \leq \alpha \cdot n \cdot \phi(G) \leq 2\alpha \cdot h(G)$.

There are polynomial reductions between $h(G), h_{\mathrm{vol}}(G)$ and $\phi(G)$ and they are all NP-hard to compute [27], in contrast to the minimum-cut of a graph which can be computed in polynomial time. Hence, almost all of the literature on edge expansion is devoted to finding good theoretical bounds. These are generally associated with the eigenvalues of the Laplacian matrix of the graph and form the basis for the field of spectral graph theory (see the monograph [9]). There have also been many approximation studies on this topic [3, 27, 34, 37], and semidefinite optimization (SDP) has been a popular tool in this regard. The best-known approximation for $\phi(G)$ is the famous $\mathcal{O}(\sqrt{\log n})$ factor by Arora et al. [3] which improved upon the earlier $\mathcal{O}(\log n)$-approximation [27]. The analysis is based on an SDP relaxation with triangle inequalities and uses metric embeddings and concentration of measure results. Meira and Miyazawa [31] developed a branch-and-bound algorithm for computing $\phi(G)$ using SDP relaxations and SDP-based heuristics. Recall that $\phi(G)$ is related to $h(G)$ in the approximate sense (up to a factor 2) but not in the exact sense. To the best of our knowledge, there is no exact solution algorithm for $h(G)$.

Contribution and outline We adopt mathematical programming approaches for numerical computation of $h(G)$. All our approaches make use of tight bounds obtained via semidefinite programming. The first algorithm works in two phases. In the first phase, we split the problem into subproblems and by computing lower and upper bounds for these subproblems, we can exclude a significant part of the search space. In the second phase, we either solve the remaining subproblems to optimality or until a subproblem can be pruned due to the bounds. For the second phase, we develop two versions. The first version implements a tailored branch-and-bound algorithm, in the second version we transform the subproblem into an instance of a max-cut problem and compute the maximum cut using an SDP-based solver.

The second algorithm we implement uses the idea of Dinkelbach's algorithm to solve fractional optimization problems. The main concept of this algorithm is to iteratively solve linearly constrained binary quadratic programs. We solve these problems again by transforming them into instances of max-cut and using an SDP-based solver to compute the maximum cut.

We perform numerical experiments on different types of instances which demonstrate the effectiveness of our approaches. To the best of our knowledge, no other algorithms are capable of computing the edge expansion for graphs with a few hundred vertices.

The rest of the paper is structured as follows. In § 2 we formulate the problem as a mixed-binary quadratic program and present an SDP relaxation. § 3 investigates a related problem, namely the k-bisection problem. We introduce a branch-and-bound algorithm and describe all the ingredients in this section. The first algorithm (relying on the k-bisection problem) for computing $h(G)$ is introduced in § 4, and another algorithm (following Dinkelbach's idea) in § 5. The performance of all algorithms is demonstrated in § 6, followed by conclusions in § 7.

Notation The set of $n \times n$ real symmetric matrices is denoted by $\mathcal{S}^n$. The positive semidefiniteness condition for $X \in \mathcal{S}^n$ is written as $X \succeq 0$. The trace of X is written as $\operatorname{tr}(X)$ and defined as the sum of its diagonal elements. The trace inner product for $X, Y \in \mathcal{S}^n$ is defined as $\langle X, Y \rangle = \operatorname{tr}(XY)$ and the operator $\operatorname{diag}(X)$ returns the main diagonal of matrix X as a vector. The vector of all ones is e and the matrix of all ones is $J = ee^\top$.

For a n-vertex graph $G = (V, E)$, the minimum and maximum vertex degrees are $\delta(G)$ and $\Delta(G)$, the adjacency matrix is a binary matrix $A \in \mathcal{S}^n$ having $A_{ij} = 1$ if and only if $(i, j) \in E$, and the degree matrix is a $n \times n$ positive diagonal matrix D having D_{ii} equal to the degree of vertex $i \in V$. The Laplacian matrix is $L = D - A$, and thus has its nonzero entries as $L_{ii} = \deg(i)$ and $L_{ij} = -1$ for $(i, j) \in E$. We denote by $\zeta(S) = |\partial S|$ the size of the cut-set defined by the partition (S, S') of the vertices. The minimum cut in G is defined as $\zeta_{\min}(G) = \min_{\emptyset \neq S \subset V} \zeta(S)$.

2. Formulations and SDP relaxations

To write an algebraic optimization formulation for cut problems in graphs, we represent a cut (S, S') in G by its incidence vector $\chi^S \in \{0,1\}^n$ which has $\chi_i^S = 1$ if and only if $i \in S$. The cut function is the size of a cut-set, also called the value of the cut, and is equal to

$$\zeta(S) = |\partial S| = \sum_{(i,j) \in E} \left(\chi_i^S - \chi_j^S\right)^2 = \left(\chi^S\right)^\top L \chi^S \ .$$

Any binary vector $x \in \{0,1\}^n$ represents a cut in this graph. Denote the set of all cuts with S containing at least one vertex and at most half of the vertices by

$$\mathcal{F} = \left\{ x \in \{0,1\}^n : 1 \leq e^\top x \leq \frac{n}{2} \right\}.$$

Using the common expression $x^\top L x$ for the cut function, the edge expansion problem is

$$
\begin{aligned}
h(G) &= \min_{x} \left\{ \frac{x^\top L x}{e^\top x} : x \in \mathcal{F} \right\} \\
&= \min_{x,y} \left\{ y : \frac{x^\top L x}{e^\top x} \le y,\, x \in \mathcal{F} \right\} \\
&= \min_{x,y} \left\{ y : x^\top L x - y\, e^\top x \le 0,\, x \in \mathcal{F} \right\}.
\end{aligned} \tag{1}
$$

The last formulation is a mixed-binary quadratically constrained problem (MIQCP). Standard branch-and-cut solvers may require a large computation time with these formulations even for instances of small to medium size, as we will report in § 6.

Although the focus of this paper is on computing $h(G)$, let us also mention for the sake of completeness that analogous formulations can be derived for the graph conductance (weighted edge expansion) $h_{\text{vol}}(G)$ that was defined in § 1, by optimizing over the set

$$
\mathcal{F}_{\text{vol}} = \left\{ x \in \{0,1\}^n : 1 \le d^\top x \le m \right\},
$$

where $d = \operatorname{diag}(D)$ is the vector formed by the vertex degrees. For example, the same steps as in (1) yields the MIQCP

$$
h_{\text{vol}}(G) = \min_{x,y} \left\{ y : x^\top L x - y\, d^\top x \le 0,\, x \in \mathcal{F}_{\text{vol}} \right\}.
$$

2.1. Semidefinite relaxations

A well-known lower bound for the edge expansion is the *spectral bound*. It is based on the second smallest eigenvalue of the Laplacian matrix of the graph, namely $h(G) \ge {}^{\lambda_2(L)}/_2$. One way to derive this bound is by considering the following SDP relaxation

$$
\begin{array}{llllll}
h(G) \ge & \min_{\widetilde{X},k} & \frac{1}{k}\langle L, \widetilde{X} \rangle & = & \min_X & \langle L, X \rangle \\
& \text{s.t.} & \operatorname{tr}(\widetilde{X}) = k & & \text{s.t.} & \operatorname{tr}(X) = 1 \\
& & \langle J, \widetilde{X} \rangle = k^2 & & & 1 \le \langle J, X \rangle \le \frac{n}{2} \\
& & 1 \le k \le \frac{n}{2} & & & X \succeq 0, \\
& & \widetilde{X} \succeq 0 & & &
\end{array} \tag{2}
$$

where $\tilde{X}$ models $xx^\top$ and we scale $X = \frac{1}{k}\widetilde{X}$ to eliminate the variable k.

Proposition 2.1. *The optimal solution of the second SDP in* (2) *is* ${}^{\lambda_2(L)}/_2$.

Proof. First observe that $\frac{1}{n}I$ is a strictly feasible point of the primal SDP and the optimum value is finite, hence strong duality holds. The dual of the second SDP in (2) is

$$
\max_{v} \left\{ v_1 - \frac{n}{2} v_2 + v_3 : W = L - v_1 I + (v_2 - v_3) J \succeq 0,\, v_2, v_3 \ge 0 \right\}.
$$

The eigenvalue of W with respect to the eigenvector e is $-v_1 + n(v_2 - v_3)$. The other eigenvalues of W are then $\lambda_i(L) - v_1$ for $2 \le i \le n$. Therefore, we can write the dual as

$$
\max_{v} \left\{ v_1 - \frac{n}{2} v_2 + v_3 : n(v_2 - v_3) \ge v_1,\ \lambda_2(L) \ge v_1,\ v_2, v_3 \ge 0 \right\},
$$

which is a linear program with optimal solution $v_1 = \lambda_2(L)$, $v_2 = {}^{\lambda_2(L)}/_{n}$ and $v_3 = 0$ and optimal value ${}^{\lambda_2(L)}/_{2}$. □

To strengthen the SDP relaxation (2) we round down the upper bound to $\lfloor \frac{n}{2} \rfloor$ and add the following facet-inducing inequalities of the boolean quadric polytope [35] for $\widetilde{X}$

$$0 \leq \widetilde{X}_{ij} \leq \widetilde{X}_{ii} \tag{3a}$$
$$\widetilde{X}_{i\ell} + \widetilde{X}_{j\ell} - \widetilde{X}_{ij} \leq \widetilde{X}_{\ell\ell} \tag{3b}$$
$$\widetilde{X}_{ii} + \widetilde{X}_{jj} - \widetilde{X}_{ij} \leq 1 \tag{3c}$$
$$\widetilde{X}_{ii} + \widetilde{X}_{jj} + \widetilde{X}_{\ell\ell} - \widetilde{X}_{ij} - \widetilde{X}_{i\ell} - \widetilde{X}_{j\ell} \leq 1, \tag{3d}$$

resulting in the following valid inequalities for X

$$0 \leq X_{ij} \leq X_{ii} \tag{4a}$$
$$X_{i\ell} + X_{j\ell} - X_{ij} \leq X_{\ell\ell} \tag{4b}$$
$$X_{ii} + X_{jj} - X_{ij} \leq 1 \tag{4c}$$
$$X_{ii} + X_{jj} + X_{\ell\ell} - X_{ij} - X_{i\ell} - X_{j\ell} \leq 1 \tag{4d}$$

for all $1 \leq i,\ j,\ \ell \leq n$. Note, that in (4c) and (4d) we have to replace $\frac{1}{k}$ in the rhs by its upper bound 1 in order to obtain a formulation without k. Therefore, we cannot expect these inequalities to strengthen the SDP relaxation significantly.

2.2. Illustrative examples for motivation

We motivate our algorithm by considering the example of the graph of the grlex polytope, which is described in [14]. Table 1 compares different lower bounds on $h(G)$ for these graphs. The first column indicates the dimension of the polytope and the second column lists the number of vertices in the associated graph. The third column gives the edge expansion that is known to be one for these graphs in all dimensions [14]. The spectral bound is displayed in the fourth column. Column 5 lists the optimal value of the SDP relaxation (2) strengthened by inequalities (4) derived from the boolean quadric polytope. Column 6 displays a lower bound that is very easy to compute: the minimum cut of the graph divided by the largest possible size of the smaller set of the bipartition of the vertices, that is $\lfloor \frac{n}{2} \rfloor$. In the last column, the minimum of the lower bounds ℓ_k for $1 \leq k \leq \lfloor \frac{n}{2} \rfloor$ is listed with ℓ_k being a bound related to the solution of (2) for k fixed. The definition of ℓ_k follows in § 3.1.

The numbers in the table show that some of these bounds are very weak, in particular, if the number of vertices increases. Interestingly, if we divide the edge expansion problem into $\lfloor \frac{n}{2} \rfloor$ many subproblems with fixed denominator (as we did to obtain the numbers in column 6) the lower bound we obtain by taking the minimum over all SDP relaxations for the subproblems seems to be stronger than the other lower bounds presented in Table 1. We will, therefore, take this direction of computing the edge expansion, namely, we will compute upper and lower bounds on the problem with fixed k. Using these bounds will

d	n	$h(G)$	$\lambda_2/2$	(2) & (4)	$\zeta_{\min}(G)/\lfloor \frac{n}{2} \rfloor$	$\min_k \ell_k$
2	4	1	1	1	1	1
3	7	1	0.7929	0.8435	1	1
4	11	1	0.6662	0.7095	0.8	1
5	16	1	0.5811	0.6271	0.625	1
6	22	1	0.5231	0.5743	0.5455	1
7	29	1	0.4820	0.5395	0.5	1
8	37	1	0.4516	0.5164	0.4444	1

Table 1: Comparison of lower bounds for graphs from the grlex polytope in dimension d.

allow to exclude a (hopefully) large number of potential sizes k of the smaller partition. This will leave us with computing the maximum cut of a graph with fixed sizes of the partition k and $n-k$ for a few values of k only.

3. Fixing the size k: Bisection problem

If the size k of the smaller set of the partition of an optimum cut is known, the edge expansion problem would result in a scaled bisection problem. That is, we ask for a partition of the vertices into two parts, one of size k and one of size $n-k$, such that the number of edges joining these two sets is minimized. This problem is NP-hard [12] and has the following formulations for any $k \in \{1, 2, \ldots, \lfloor \frac{n}{2} \rfloor\}$,

$$h_k = \frac{1}{k} \begin{array}{ll} \min_x & x^\top L x \\ \text{s.t.} & e^\top x = k \\ & x \in \{0,1\}^n, \end{array} \tag{5}$$

but standard branch-and-cut solvers can solve these in reasonable time only for small-sized graphs.

Since SDP-based bounds have been shown to be very strong for partitioning problems, cf. [23, 30, 41, 42], we exploit these bounds by developing two kinds of solvers. In the first variant, we develop a tailored branch-and-bound algorithm based on semidefinite programming to solve the bisection problem. In the subsequent sections, we describe how to obtain lower and upper bounds on h_k (§ 3.1 and 3.2) as well as further ingredients of this exact solver (§ 3.3). The second variant is presented in § 3.4, where we transform the bisection problem into an instance of a max-cut problem which is then solved using the state-of-the-art solver BiqBin [17]. For completeness, a description of BiqBin is given in Appendix A.

3.1. SDP lower bounds for the bisection problem

After squaring the linear equality constraint in problem (5) and employing standard lifting and relaxation techniques, we obtain the following SDP relaxation that is generally computationally cheap to solve,

$$\begin{array}{lll} \ell_{\text{bisect}}(k) = & \min_{X,x} & \langle L, X\rangle \\ & \text{s.t.} & \operatorname{tr}(X) = k \\ & & \langle J, X\rangle = k^2 \\ & & \operatorname{diag}(X) = x \\ & & \begin{pmatrix} 1 & x^\top \\ x & X \end{pmatrix} \succeq 0. \end{array} \tag{6}$$

Since the bisection for a given simple unweighted graph has to be an integer, we get the following lower bound on the scaled bisection h_k,

$$h_k \geq \ell_k = \frac{\lceil \ell_{\text{bisect}}(k) \rceil}{k}. \tag{7}$$

There are several ways to strengthen the above relaxation of the bisection problem. In [42] a vector lifting SDP relaxation, tightened by non-negativity constraints, has been introduced. In our setting, this results in the following doubly non-negative programming (DNN) problem,

$$\begin{array}{ll} \min\limits_{X} & \langle L, X^{11} + X^{22}\rangle \\ \text{s.t.} & \operatorname{tr}(X^{11}) = k,\ \langle J, X^{11}\rangle = k^2 \\ & \operatorname{tr}(X^{22}) = n-k,\ \langle J, X^{22}\rangle = (n-k)^2 \\ & \operatorname{diag}(X^{12}) = 0,\ \operatorname{diag}(X^{21}) = 0,\ \langle J, X^{12} + X^{21}\rangle = 2k(n-k) \\ & X = \begin{pmatrix} 1 & (x^1)^\top & (x^2)^\top \\ x^1 & X^{11} & X^{12} \\ x^2 & X^{21} & X^{22} \end{pmatrix} \succeq 0, \quad x^i = \operatorname{diag}(X^{ii}), \quad i = 1,2 \\ & X \geq 0, \end{array} \tag{8}$$

where X is a matrix of size $(2n+1)\times(2n+1)$. This relaxation can be further strengthened by cutting planes from the Boolean Quadric Polytope. In particular, we want to add the inequalities

$$X_{i\ell} + X_{j\ell} \leq X_{\ell\ell} + X_{ij} \tag{9}$$

as Meijer et al. [30] demonstrated that these inequalities are the most promising ones to improve the bound.

The DNN relaxation (8) cannot be solved by standard methods due to the large number of constraints. The additional cutting-planes (9) make the SDP relaxation extremely difficult to solve already for medium-sized instances. Meijer et al. [30] apply facial reduction to the SDP relaxation which leads to a natural way of splitting the set of variables into two blocks in the following way. They present a $(2n+1)\times n$ matrix V such that $X = VRV^\top$ for $R \in \mathcal{S}^n$. Due to this facial reduction, we consider the sets $\mathcal{R}_{BP} = \{R \in \mathcal{S}^n : R \succeq 0\}$ and $\mathcal{X}_{BP}$ being the set of matrices in $\mathcal{S}^{2n+1}$ satisfying all polyhedral constraints. The facially reduced DNN relaxation of the bisection problem then reads

$$\min_{X,R} \left\{ \langle L_{BP}, X \rangle : X = VRV^\top,\ R \in \mathcal{R}_{BP},\ X \in \mathcal{X}_{BP} \right\}$$

with appropriate matrix L_{BP}. Using an alternating direction method of multipliers (ADMM) provides (approximate) solutions to this relaxation even for graphs with up to 1000 vertices. The steps to be performed in this ADMM algorithm result in projections onto the respective feasible sets. For projections onto polyhedra, Dykstra's projection algorithm is used. A careful selection of non-overlapping cuts, warm starts, and an intelligent separation routine are further ingredients of this algorithm in order to obtain an efficient solver for the SDP (8) enhanced with inequalities (9). A post-processing algorithm is also introduced to guarantee a valid lower bound. Using this algorithm, we can compute strong lower bounds for each k with reasonable computational effort.

3.2. A heuristic for the bisection problem

The graph bisection problem can also be written as a quadratic assignment problem (QAP) [25]. To do so, we set the weight matrix W to be the Laplacian matrix L of the graph and the distance matrix $\widetilde{D}$ to be a matrix with a top left block of size k with all ones and the rest zero. The resulting QAP for this weight and distance matrix is

$$\min_{\pi \in \Pi_n} \sum_{i=1}^{n} \sum_{j=1}^{n} W_{i,j} \widetilde{D}_{\pi(i),\pi(j)} = \min_{\pi \in \Pi_n} \sum_{i=1}^{k} \sum_{j=1}^{k} L_{\pi^{-1}(i),\pi^{-1}(j)} = kh_k.$$

In this formulation, the vertices mapped to values between 1 and k by the permutation π are chosen to be in the set of size k in the partition. To compute an upper bound u_k on h_k, we can use any heuristic for the QAP and divide the solution by k.

Simulated annealing is a well-known heuristic to compute an upper bound for the QAP, we implement the algorithm introduced in [8]. We use a slightly different parameter setting that we determined via numerical experiments. That is, the initial temperature is set to ${}^{k^2}\!/\binom{n}{2} \cdot \operatorname{tr}(L)$, the number of transformation trials at constant temperature is initially set to n and increased by a factor of 1.15 after each cycle, and the cooling factor is set to 0.7. After every trial, we additionally perform a local search strategy to find the local minimum.

Other well-performing heuristics for the QAP are tabu search, genetic algorithms, or algorithms based on the solution of the SDP relaxation like the hyperplane rounding algorithm. Some of these heuristics have the potential to be superior to simulated annealing. However, as we will see later in the study of our experiments, the bounds we obtain with the simulated annealing heuristic are almost always optimal for the size of instances we are interested in.

3.3. A branch-and-bound algorithm for the bisection problem

The branch-and-bound algorithm is a helpful technique to solve optimization problems to optimality by dividing the feasible region. Existing exact algorithms for the graph bisection problem are presented in [19, 24] considering instances with up to 148 vertices, most of them being very sparse, and in [2] for large sparse graphs. Moreover, most of the computational results are presented for equipartition problems and not for general sizes of the partition. Implementations of these algorithms are not available and from the results in the papers these algorithms might not be successful on our problems.

We, therefore, implement an open-source solver to solve graph bisection problems of medium size to optimality using the ingredients described in the previous sections, namely, we implement an ADMM to obtain approximate solutions of the SDP bound as described in § 3.1 and we implement a simulated annealing procedure to obtain the upper bound as described in § 3.2.

Branching In the case of binary optimization, a natural way of branching is to fix a variable to 0 or 1. We base our branching decision on the solution of the relaxation of the subproblem. Namely, we branch on the node with the corresponding value in x^1 being closest to 0.5, i.e., most fractional. It turns out that we can write the subproblems again as problems of a similar form as the bisection problem but with fewer variables. In particular, if we set a variable x_i to be 0, we can write the problem as $\min\{\bar{x}^\top \bar{L}\bar{x} : e^\top \bar{x} = k\}$, where $\bar{x}$ is obtained from x by deleting x_i and we get $\bar{L}$ by deleting the i-th row and column of L. The subproblem where we set $x_i = 1$ can be written as $\min\{\bar{x}^\top \widetilde{L}\bar{x} + c : e^\top \bar{x} = k - 1\}$, with $\bar{x}$ again resulting from x by deleting entry x_i and $\widetilde{L}$ is obtained from L by adding the i-th row and column to the diagonal before deleting them and $c = L_{ii}$. Note that for both types of subproblems, although they are no bisection problems anymore, we can still use the methods discussed in the Sections 3.1 and 3.2 to compute lower and upper bounds.

Strategy on adding strengthening inequalities In our branch-and-bound algorithm, similar to the strategy in [17], in the root node we first compute the (DNN) bound (8) and then iteratively add violated triangle inequalities (9). The improvement attained by adding triangle inequalities is stored in `diff` and handed down to the child nodes. We use it in the child nodes for the decision of whether or not we add violated triangle inequalities to (DNN) since adding triangle inequalities is time-consuming. In case the solution of (DNN) plus `diff` does not suffice to prune the node, we refrain from adding triangle inequalities. In case we do continue adding violated triangle inequalities and are not able to prune the node, we update `diff` accordingly for the subsequent child nodes.

Lower bound verification/early stopping In preparation for § 4, we add the option to stop the branch-and-bound algorithm as soon as there is no open subproblem with a lower bound below a given threshold, that is, if the lower bound on the bisection problem is greater or equal than that threshold. This functionality can be used to verify a given lower bound without having to solve the bisection problem.

Before moving on to § 4, we present a second variant on how to solve the bisection problem.

3.4. Transformation to a max-cut problem

A different approach to solving the graph bisection problem is to transform it to a max-cut problem and use a max-cut solver, e.g. the open source parallel solver BiqBin [17], see also Appendix A. To do so, we first need to transform the bisection problem into a quadratic unconstrained binary problem (QUBO).

Lemma 3.1. *Let $\tilde{x} \in \{0,1\}^n$ such that $\mathrm{e}^\top \tilde{x} = k$, and choose μ_k such that $\mu_k > \tilde{x}^\top L \tilde{x}$. Then*

$$h_k = \frac{1}{k} \min_x \left\{ x^\top (L + \mu_k J)x - 2\mu_k k \, \mathrm{e}^\top x + \mu_k k^2 : x \in \{0,1\}^n \right\}.$$

Proof. First note that

$$x^\top (L + \mu_k ee^\top)x - 2\mu_k k \, \mathrm{e}^\top x + \mu_k k^2 = x^\top L x + \mu_k \|\mathrm{e}^\top x - k\|^2.$$

Let $x \in \{0,1\}^n$. Then we have

$$\begin{aligned} x^\top L x + \mu_k \|\mathrm{e}^\top x - k\|^2 &= x^\top L x && \text{if } \mathrm{e}^\top x = k, \\ x^\top L x + \mu_k \|\mathrm{e}^\top x - k\|^2 &\geq \mu_k && \text{if } \mathrm{e}^\top x \neq k. \end{aligned}$$

Note that $e^\top x - k$ is integer for $x \in \{0,1\}^n$. Hence, for any infeasible $x \in \{0,1\}^n$, the objective is greater than the given upper bound $\tilde{x}^\top L \tilde{x}$, and therefore the minimum can only be attained for $x \in \{0,1\}^n$ with $\mathrm{e}^\top x = k$. □

Barahona et al. [6] showed that any QUBO problem can be reduced to a max-cut problem by adding one additional binary variable. In our context, this means the following.

Corollary 3.2. *Let $G = (V, E)$ and let G' be the complete graph with vertex set $V \cup \{v_0\}$. Let the weights c_{uw} on the edges of G' be as follows.*

$$c_{uw} = \begin{cases} 4\mu_k(n - 2k) & \text{if } u \in V(G) \text{ and } w = v_0 \\ 4\mu_k - 1 & \text{if } uw \in E(G) \\ 4\mu_k & \text{if } uw \notin E(G) \end{cases}$$

Then the minimum bisection of G where one side of the cut has size k is equal to $4\mu_k(n-k)^2 - \text{max-cut}(G')$.

Since max-cut solvers can benefit from edge weights of the input graph being integer, a possible choice for μ_k is an upper bound on the bisection problem plus 1/4. Note that we choose μ_k to be as small as possible by doing so.

The formulation of the max-cut problem in ±1 variables additionally requires $x_v = 1$ to hold. Because of the symmetry of the cut, we can omit this constraint. Due to our choice of the penalty parameter, it holds that on one side of the maximum cut, there are exactly $k + 1$ vertices, including vertex v. These k vertices on the same side as v are the vertices in the set of size k in the optimum of the bisection problem.

To summarize, we can solve the bisection problem by solving a dense max-cut problem. With this, we now have all the tools needed for our new split & bound approach.

4. Split & bound

We now assemble the tools developed in § 3 to compute the edge expansion of a graph by splitting the problem into $\lfloor \frac{n}{2} \rfloor$ many bisection problems. Since the bisection problem is NP-hard as well, we want to reduce the number of bisection problems we have to solve exactly as much as possible. To do so, we start with a pre-elimination of the bisection problems. This procedure aims to exclude subproblems unnecessary for the computation of the edge expansion of the graph. Computing the edge expansion by considering the remaining values of k is summarized in Algorithm 2 below. We now explain the pre-elimination step and further ingredients of our algorithm.

4.1. Pre-elimination

The size k of the smaller set of the partition can theoretically be any value from 1 to $\lfloor \frac{n}{2} \rfloor$. However, it can be expected that for some candidates, one can quickly check that the optimal solution cannot be attained for that k. As a first quick check, we use the cheap lower bound ℓ_k obtained by solving the SDP (6) in combination with the upper bound introduced in the § 3.2. We do not need to further consider values of k where the lower bound ℓ_k of the scaled bisection problem is already above an upper bound on the edge expansion. A pseudo-code of this pre-elimination step is given in Algorithm 1.

Algorithm 1: Pre-eliminate certain values of k

```
for $k \in \{1, \dots, \lfloor \frac{n}{2} \rfloor\}$ do
    Compute an upper bound $u_k$ using the heuristic from § 3.2;
    Compute the lower bound $\ell_k$ from (7) by solving the cheap SDP (6);
Global upper bound $u^* \coloneqq \min \{u_k \colon 1 \le k \le \lfloor \frac{n}{2} \rfloor\}$;
if $\min_k \ell_k = u^*$ then
    $\mathcal{I} = \emptyset$, $h(G) = u^*$;
else
    $\mathcal{I} \coloneqq \{k \in \{1, \dots, \lfloor \frac{n}{2} \rfloor\} \colon \ell_k < u^*\}$;
return $\mathcal{I}$, $u_k$ for $k \in \mathcal{I}$, $u^*$
```

The hope is that many values of k can be excluded from computing the edge expansion. Clearly, this heavily depends on the instance itself, as in the worst case, it might happen that for many different values of k, the value of h_k is close to the optimum.

We can further reduce the number of candidates for k by computing a tighter lower bound $\tilde{\ell}_k$ by solving the DNN relaxation (8) with additional cutting planes. In our implementation we do not compute $\tilde{\ell}_k$ as part of the pre-elimination, since this bound is computed in the root node of the branch-and-bound tree in the algorithm from § 3.3.

Impact of pre-elimination on sample instances Figures 1 and 2 display the bounds associated with four different graphs. For the graph of the grevlex polytope in dimension 7, considering the bounds u_k and $\tilde{\ell}_k$ the only candidates for k where the optimal solution can be found are 12 and 14. For the grevlex polytope in dimension 8, the sizes 17 and 18 remain as the only candidates. Also for a graph associated to a randomly generated 0/1-polytope and to a network graph, about 2/3 of the potential values of k can be excluded already by considering the cheap lower bound ℓ_k.

4.2. Stopping exact computations early and updating u^*

All values of k that are not excluded in the pre-elimination step have to be further examined. For those values we run our branch-and-bound algorithm (either for k-bisection or max-cut, depending on the variant chosen by the user) to compute the scaled bisection h_k. We can stop the branch-and-bound algorithm as soon as the lower bound of all open nodes in the branch-and-bound tree is larger or equal to u^*. (Remember that u^* is an upper bound on the edge expansion of the graph but not necessarily an upper bound on h_k.) A simple way to implement this stopping criterion is to initialize the branch-and-bound algorithm for the k-bisection problem with $\lceil u^*k \rceil$ as an “artificial” upper bound.

In case $h_k < u^*$, we can update u^* which might lead to eliminating further values from $\mathcal{I}$. This fact is also part of the motivation for the order of choosing k for computing h_k, as described in the next section.

4.3. Order of selecting values k from $\mathcal{I}$

We consider the order of computing h_k in ascending order based on their upper bounds u_k. The motivation for this choice is as follows.

Remember that $u^* = \min_k u_k$ is a global upper bound on the edge expansion. The most promising values for k to even further improve this bound are those with small u_k. Therefore, before starting the branch-and-bound algorithm for the values k left after pre-elimination, we do another 30 trials of simulated annealing for each of these to hopefully further improve the upper bound.

Moreover, we run the exact computation of h_k in this order since also during the branch-and-bound algorithm, the upper bound might drop further and this will improve the global upper bound u^* most likely for those candidates with small u_k.

An improvement of the upper bound u^* means that there is a possibility to further eliminate values k from $\mathcal{I}$. But even for values k that can not be eliminated, we obtain smaller artificial upper bounds and hence the computation of these bisection problems may be stopped earlier.

We summarize all the steps in Algorithm 2.

4.4. Algorithmic verification of lower bound

We close this section by addressing the important consideration that we are not interested in the exact value of the edge expansion in some applications, but want to check whether certain values are valid lower bounds on $h(G)$. A lower bound $c \leq h(G)$, for some constant $c > 0$, means that the graph is a c-expander. The value of this lower bound means that the graph expands by at least that much. This also arises in the context

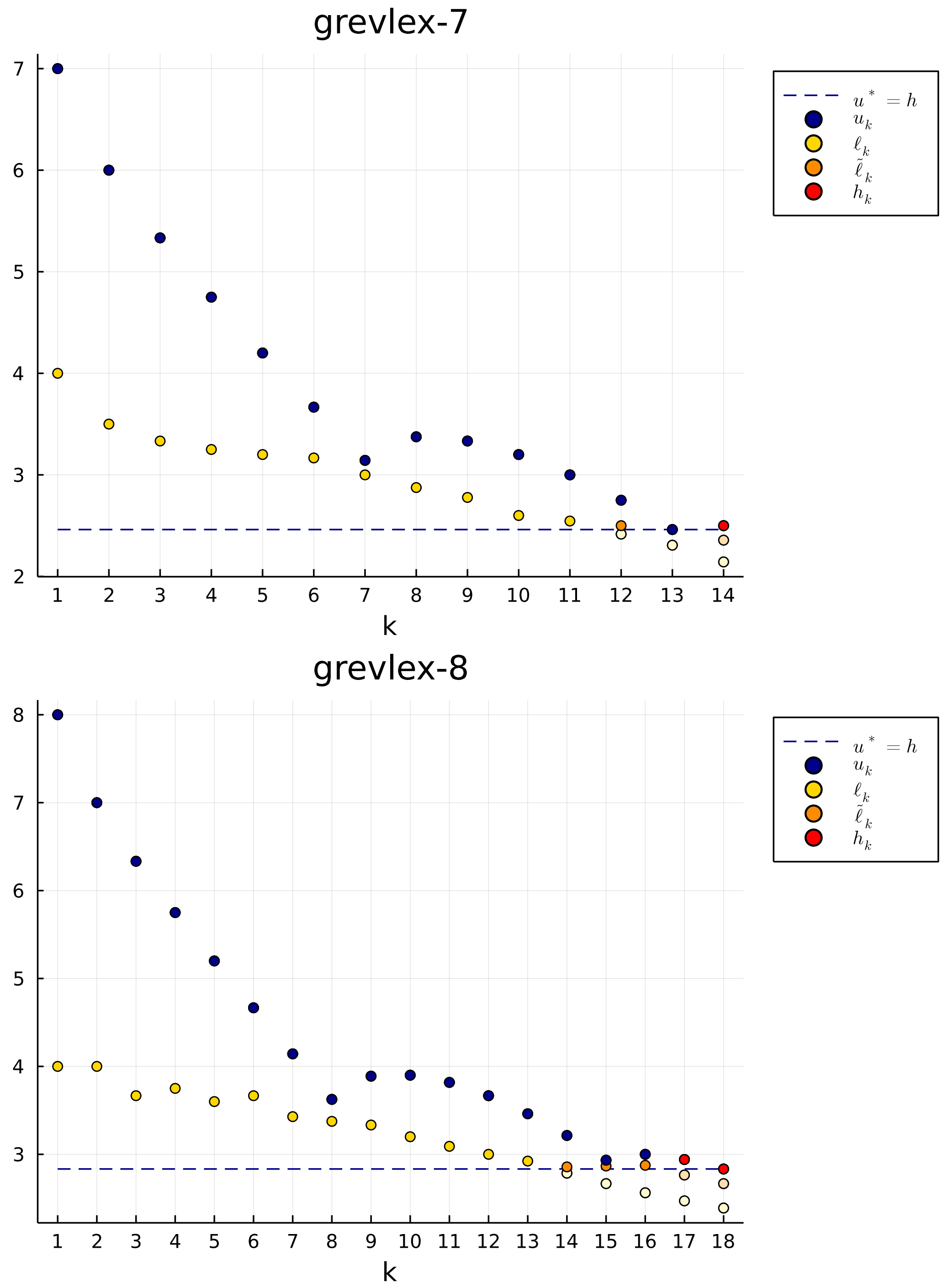

Figure 1: Lower and upper bounds for each k.

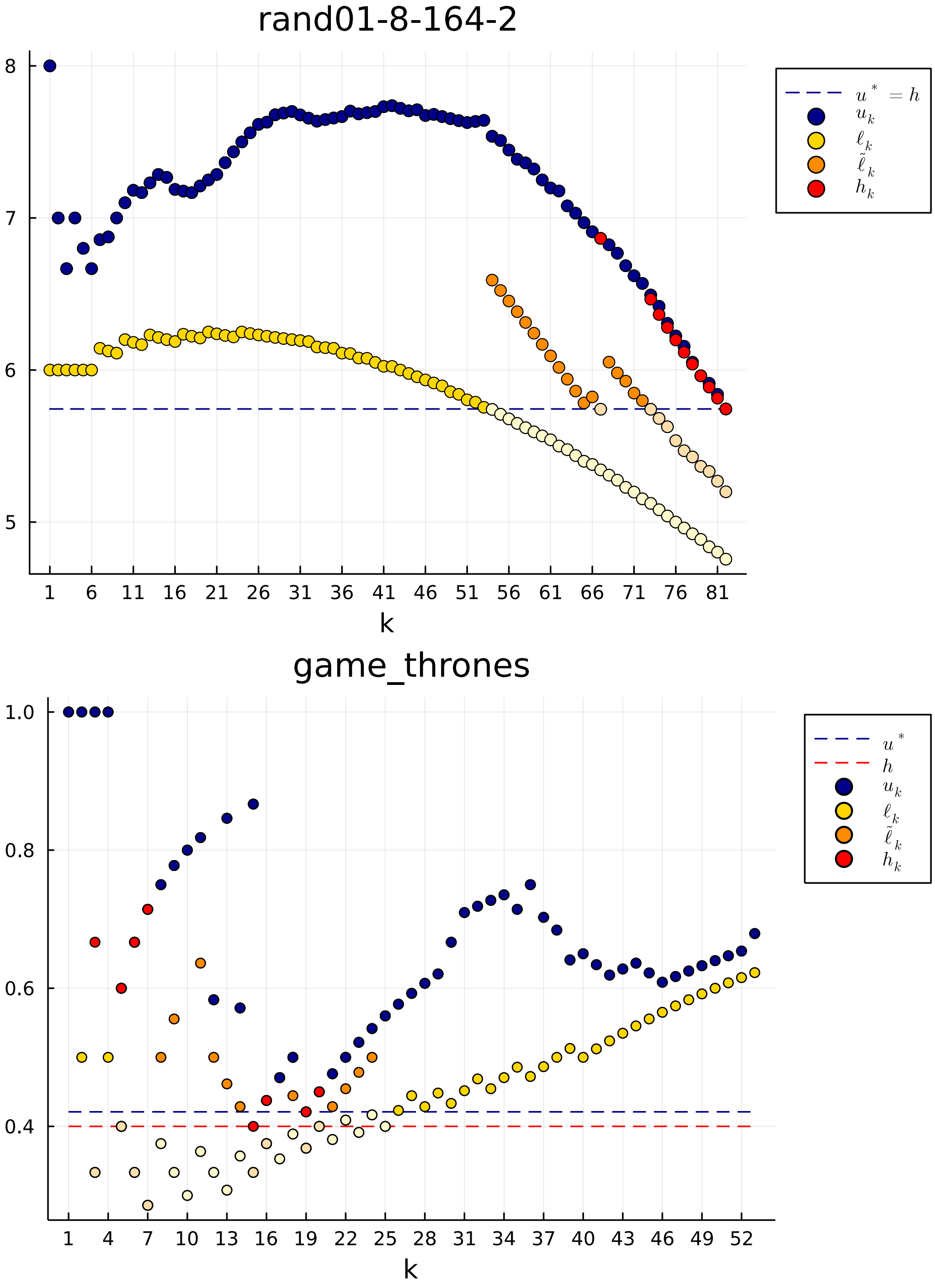

Figure 2: Lower and upper bounds for each k.

Algorithm 2: Split & bound

```
$\mathcal{I}$, $u_k$ for $k \in \mathcal{I}$, $u^* \leftarrow$ pre-elimination Algorithm 1;
for $k \in \mathcal{I}$ do
    Run heuristic from § 3.2 and update $u_k$;
    if $u_k < u^*$ then
        $u^* \leftarrow u_k$;
        update $\mathcal{I}$;
for $k \in \mathcal{I}$, consider $k$ in ascending order of $u_k$ do
    if variant $=$ $k$-bisection then
        compute $h_k$ using the $k$-bisection branch-and-bound solver, initialize the
          upper bound for $k$-bisection as $\lceil u^* k \rceil$;
    if variant $=$ max-cut then
        transform the instance to a max-cut instance;
        compute $h_k$ using the max-cut solver, initialize the lower bound for
          max-cut as offset $- \lceil u^* k \rceil$;
    if $h_k < u^*$ then
        $u^* \leftarrow h_k$;
        update $\mathcal{I}$
$h(G) = u^*$;
```

of the Mihail-Vazirani conjecture on 0/1-polytopes where one wants to check whether $h(G) \geq 1$ where G is the graph of a 0/1-polytope.

Our split & bound algorithm can also be used to verify a lower bound.

Proposition 4.1. *Let υ be a given scalar and suppose we initialise Algorithm 2 with $u^* = \upsilon$. Then υ is a valid lower bound on $h(G)$ if and only if the algorithm terminates without updating u^*.*

Proof. Assume we initialize $u^* = \upsilon$. If we find a better upper bound (or some computed value h_k is smaller than υ), this is a certificate that the given value υ is not a valid lower bound since we found a better solution. Otherwise, if the upper bound never gets updated, this means the provided bound is indeed a valid lower bound on the edge expansion of the graph. □

5. Parametric optimization

Another approach to compute $h(G)$ is following a discrete Newton-Dinkelbach algorithm. Dinkelbach [10] gave a general classical framework to solve (non)-linear fractional programs. The program one aims to solve is $\min_{x \in \mathcal{F}} f(x)$, where the objective f is a fraction of (non)-linear functions. In our case this is

$$f(x) = \frac{x^\top Lx}{e^\top x}, \quad \text{and} \quad \mathcal{F} = \Big\{x \in \{0,1\}^n : 1 \le e^\top x \le \frac{n}{2}\Big\}.$$

The main component of the algorithm is to form the following parametrized objective function

$$g_\gamma(x) = x^\top Lx - \gamma e^\top x,$$

and the corresponding parametrized optimization problem

$$P(\gamma) = \min_x \,\{g_\gamma(x) : x \in \mathcal{F}\}, \qquad \gamma \ge 0.$$

This problem then has the following useful properties.

Proposition 5.1. *$P(0) = \zeta_{\text{min}}(G)$ and P is a strictly decreasing concave piecewise linear function over $\mathbb{R}_+$ whose unique root is equal to $h(G)$. Consequently,*

$$h(G) = \max_\gamma \big\{\gamma\colon g_\gamma(x) \ge 0\big\} = \min_\gamma \big\{\gamma\colon g_\gamma(x) \le 0\big\}.$$

Proof. We have

$$\begin{aligned} P(0) &= \min_x \big\{x^\top Lx : x \in \mathcal{F}\big\} \\ &= \min_S \big\{|\partial S| : \emptyset \ne S \subset V, |S| \le {}^{n}\!/_{2}\big\} \\ &= \min_S \big\{|\partial S| : \emptyset \ne S \subset V\big\} \\ &= \zeta_{\text{min}}(G) \end{aligned}$$

where the penultimate equality is from symmetry of the cut function $\zeta(S) = |\partial S| = \big|\partial S'\big|$. Finiteness of $\mathcal{F}$ and linearity of g_γ in γ tells us that P is the pointwise minimum of finitely many affine (in γ) functions, and so P is a concave piecewise linear function. The strictly decreasing property was shown in [10, Lemma 3] for general nonlinear fractional problems. □

This implies that the edge expansion of a graph can be computed using a root-finding algorithm for the function P. One evaluation of P for a given γ still means solving a binary quadratic problem with two linear inequalities. Hence, reducing this number of evaluations is crucial to compute $h(G)$ in reasonable time. There are several strategies to do so, such as binary search. Our approach is to evaluate P starting with γ_1 equal to some good upper bound on $h(G)$ (in our experiments, we used our heuristic from § 3.2). We are already done if we have found the optimum with our heuristic, that is when $P(\gamma_1) = 0$. Otherwise, there is some $x^1 \in \mathcal{F}$ such that $g_\gamma(x^1) < 0$ and therefore $f(x^1) < \gamma$. This means that $f(x^1)$ is a better upper bound than γ_1. Hence, we now set $\gamma_2 = f(x^1)$ and repeat until we find the optimum as described in Algorithm 3. Since $P(\gamma) < 0$ if and only if $\gamma > h(G)$, the stopping criterion is checking whether $P(\gamma) < 0$ at the current iterate.

The superlinear convergence rate of Dinkelbach's algorithm was established in [40]. We derive a similar convergence result for Algorithm 3.

Algorithm 3: Discrete Newton-Dinkelbach algorithm for edge expansion

```
Input: graph $G$, upper bound $\gamma_1 \geq h(G)$ from heuristic
Output: edge expansion $h(G)$
$i = 1$;
while $P(\gamma_i) < 0$ do
    $x_i \in \arg\min_{x \in \mathcal{F}} g_{\gamma_i}(x)$;
    $\gamma_{i+1} = f(x_i)$;
    $i = i + 1$;
$h(G) = \gamma_i$;
```

Theorem 5.2. *Algorithm 3 terminates with the optimal value after finitely many steps, the rate of convergence is superlinear.*

Proof. Let $x^* \in \mathcal{F}$ be the optimum of the edge expansion problem, i.e., $f(x^*) = h(G)$ and let $\gamma^* = f(x^*)$. From Proposition 5.1 we know that P is a strictly decreasing piecewise linear function and therefore the algorithm terminates after finitely many iterations with value γ^*.

Let further γ_i be the upper bound on $h(G)$ to check in the i-th iteration of Algorithm 3. From Lemma B.1, Appendix B, we get that

$$\gamma_{i+1} - \gamma^* \leq (\gamma_i - \gamma^*)\left(1 - \frac{e^\top x^*}{e^\top x_i}\right)$$

holds, since $P(\gamma^*) = 0$. The sequence

$$\left(1 - \frac{e^\top x^*}{e^\top x_i}\right)$$

is strictly decreasing (and converging to 0) as proved in Lemma B.2, Appendix B. Therefore, the convergence rate of Algorithm 3 is superlinear. □

5.1. Solving the parametrized optimization problem

Evaluating $P(\gamma)$ requires solving a binary quadratic problem with two linear inequality constraints which is in general NP-hard. We again use BiqBin [17], a solver that is tailor-made for binary quadratic programs.

Most solvers for binary quadratic programs benefit from input data given as integer as this aids the performance of the underlying branch-and-bound algorithm. Since we only consider rational values for γ, we introduce the following parametric optimization problem

$$Q(\gamma) = \min_x \{\gamma_d x^\top L x - \gamma_n e^\top x : x \in \mathcal{F}\}$$

for $\gamma = {}^{\gamma_n}/_{\gamma_d}$ with integers $\gamma_n \geq 0$ and $\gamma_d > 0$. Observe that $Q(\gamma) = \gamma_d P(\gamma)$ and all considerations from above apply to this new formulation as well.

We solve $Q(\gamma)$ by transforming it into a max-cut problem. The first step towards achieving this is to obtain an exact formulation as a QUBO using binary encoding of the slack variables and the penalty parameter suggested in [18, Thm. 15]. To aid our derivation, let us denote the integer n_s and vector $v^{n_s} \in \mathbb{R}^{n_s+1}$ by

$$n_s = \left\lceil \log_2 \left\lfloor \frac{n}{2} \right\rfloor \right\rceil - 1, \qquad v_i^{n_s} = 2^{i-1} \text{ for all } i.$$

Proposition 5.3. *Let $\mathcal{F} = \{x \in \{0,1\}^n : 1 \leq e^\top x \leq \frac{n}{2}\}$ and $v^{n_s} \in \mathbb{R}^{n_s+1}$ with $v_i^{n_s} = 2^{i-1}$ and $n_s = \lceil \log_2(\lfloor \frac{n}{2} \rfloor) \rceil - 1$. Then*

$$\mathcal{F} = \left\{ x \in \{0,1\}^n : e^\top x - \alpha^\top v^{n_s} = 1,\ e^\top x + \beta^\top v^{n_s} = \left\lfloor \frac{n}{2} \right\rfloor,\ \alpha, \beta \in \{0,1\}^{n_s+1} \right\}.$$

Proof. For any $x \in \mathcal{F}$ it holds that $e^\top x = 1 + s = \lfloor \frac{n}{2} \rfloor - t$ for some slack variables s and t with $0 \leq s, t \leq \lfloor \frac{n}{2} \rfloor - 1$. In fact, any upper bound on s and t greater or equal than $\lfloor \frac{n}{2} \rfloor - 1$ is fine, since from $e^\top x = 1 + s$ and $s \geq 0$ it follows that $e^\top x \geq 1$ and from $e^\top x = \lfloor \frac{n}{2} \rfloor - t$ and $t \geq 0$ it follows that $e^\top x \leq \lfloor \frac{n}{2} \rfloor$. The smallest possible value for n_s is $\lceil \log_2(\lfloor \frac{n}{2} \rfloor) \rceil - 1$, since this gives an upper bound of $2^{n_s+1} - 1$ on s and t. □

Proposition 5.4. *Let $x' \in \mathcal{F}$ with $\frac{x'^\top L x'}{e^\top x'} = \frac{\gamma_n}{\gamma_d} = \gamma$ and $\gamma_d > 0$. The problem $Q(\gamma)$ can then be equivalently formulated as the following QUBO,*

$$\min_{x,\alpha,\beta} \left\{ \gamma_d x^\top L x - \gamma_n e^\top x + \sigma \left\| \begin{pmatrix} e^\top x - \alpha^\top v^{n_s} - 1 \\ e^\top x + \beta^\top v^{n_s} - \lfloor \frac{n}{2} \rfloor \end{pmatrix} \right\|^2 : \right. \left. x \in \{0,1\}^n,\ \alpha, \beta \in \{0,1\}^{n_s+1} \right\} \tag{11}$$

with $\sigma > \gamma_n n$.

Proof. Let $g(x, \alpha, \beta)$ denote the objective function of (11). For any feasible vector $x \in \mathcal{F}$ there exist uniquely defined α_x, β_x such that $g(x, \alpha_x, \beta_x) = \gamma_d x^\top L x - \gamma_n e^\top x$. Thus, the objective function of $Q(\gamma)$ and (11) coincide for $x \in \mathcal{F}$. Moreover, for x' it holds that $g(x', \alpha_{x'}, \beta_{x'}) = 0$.

For $x \in \{0,1\}^n \setminus \mathcal{F}$ there do not exist α, $\beta \in \{0,1\}^{n_s+1}$ such that both equalities $e^\top x - \alpha^\top v^{n_s} = 1$ and $e^\top x + \beta^\top v^{n_s} = \lfloor \frac{n}{2} \rfloor$ are satisfied, as one of the slack variables has to be negative in order to fulfill the constraints. Additionally, since L is positive semidefinite we can conclude that

$$g(x, \alpha, \beta) \geq -\gamma_n n + \sigma > 0 = g(x', \alpha_{x'}, \beta_{x'}).$$

Therefore, $Q(\gamma)$ is an equivalent formulation of (11) □

The unconstrained binary quadratic program (11) can again be transformed to a max-cut problem, as explained in [6] for example. Applied to our problem we obtain the following result.

Corollary 5.5. *Let $G = (V, E)$ and let G'' be the graph with vertices from V plus the vertices $v_0, v_{\alpha_0}, \dots, v_{\alpha_{n_s}}, v_{\beta_0}, \dots, v_{\beta_{n_s}}$ for the variable vectors α and β.*

Let the weights c_{uw} on the edges of G'' be as follows.

$$c_{uv_0} = \begin{cases} 2\sigma(n - 1 - \lfloor \frac{n}{2} \rfloor) - \gamma_n & \text{if } u \in V(G) \\ 2\sigma(2^{n_s} - \frac{n-1}{2})2^i & \text{if } u = v_{\alpha_i} \\ 2\sigma(2^{n_s} - \lfloor \frac{n}{2} \rfloor + \frac{n-1}{2})2^i & \text{if } u = v_{\beta_i} \end{cases}$$

$$c_{uw} = \begin{cases} -2^i\sigma & \text{if } u = v_{\alpha_i} \text{ and } w \in V(G) \\ 2^i\sigma & \text{if } u = v_{\beta_i} \text{ and } w \in V(G) \\ 2^{i+j}\sigma & \text{if } u = v_{\alpha_i} \text{ and } w = v_{\alpha_j} \\ 2^{i+j}\sigma & \text{if } u = v_{\beta_i} \text{ and } w = v_{\beta_j} \end{cases}$$

For $u \in V(G)$ and $w \in V(G)$, we have

$$c_{uw} = \begin{cases} 2\sigma - \gamma_d & \text{if } uw \in E(G) \\ 2\sigma & \text{if } uw \notin E(G) \end{cases}$$

Edges not specified above have weight zero. Let the penalty parameter be $\sigma = \gamma n + 1$. Then all weights are integers and it holds that $Q(\gamma) = \texttt{offset} - \text{max-cut}(G'')$ where

$$\texttt{offset} = -\gamma_n n + \sigma \cdot \Bigg[2^{n_s+2}\Big(2 \cdot 2^{n_s} - \Big\lfloor \frac{n}{2} \Big\rfloor - 1\Big) + 2n^2 - 2n + 1 + \Big\lfloor \frac{n}{2} \Big\rfloor \cdot \Big(\Big\lfloor \frac{n}{2} \Big\rfloor - 2n + 2 \Big) \Bigg].$$

6. Numerical results

All of our algorithms were written[1] in Julia [7] version 1.9.2. That is, the branch-and-bound Algorithm 2 including pre-elimination and solving the k-bisection problem (using the ADMM for obtaining a lower bound and the simulated annealing for an upper bound as well as the transformation from k-bisection to max-cut). Also, Algorithm 3 we implemented in Julia. The SDPs to compute our cheap lower bounds ℓ_k from the bisection problem in (7) are solved with MOSEK 10.0 [33] using JuMP [29]. We also use JuMP to solve MIQCPs with Gurobi [16] version 11.0. The solver BiqBin [17] for binary quadratic problems was used to solve the parametrized problems in Dinkelbach's method, and we extended the C code of this solver by adding the option to provide an initial lower bound on the maximization problem. The corresponding changes are tracked in the git repository `https://gitlab.aau.at/BiqBin/biqbin`. All computations were carried out on an AMD EPYC 7532 with 32 cores with 3.30GHz and 1024GB RAM, operated under Debian GNU/Linux 11.

[1] The code is available on the arXiv page of this paper and on `https://github.com/melaniesi/EdgeExpansion.jl`

6.1. Benchmark instances

Randomly generated 0/1-polytopes The first class of graphs are the graphs of random 0/1-polytopes. The polytopes are generated by randomly selecting n_d vertices of the polytope in dimension d, i.e., n_d different 0/1-vectors in dimension d. To obtain the graph, we then solve a linear programming feasibility problem to check whether there is an edge for a given pair of vertices. For any pair (d, n_d) with $d \in \{8, 9, 10\}$ and $n_8 \in \{164, 189\}$, $n_9 \in \{153, 178, 203, 228, 253, 278\}$, and $n_{10} \in \{256, 281\}$, we generated 3 random 0/1-polytopes.

Grlex and grevlex graphs Another class of graphs we consider are the graphs of grlex and grevlex polytopes introduced and characterised by Gupte and Poznanović [14]. The grevlex-d and grlex-d instances of our benchmark set are the corresponding graphs of the polytopes in dimension d.

DIMACS and Network graphs The last category of graphs originates from the graph partitioning and clustering application. The set of DIMACS instances are the graphs of the 10th DIMACS challenge on graph partitioning and graph clustering [5] with at most 500 vertices. Additionally, we consider some more network graphs obtained from the online network repository [36].

6.2. Discussion of the experiments

We compare different algorithms for computing the edge expansion of a graph, namely

1. Split & bound Algorithm 2 using the k-bisection solver,
2. Split & bound Algorithm 2 using the max-cut solver,
3. Fractional programming using Discrete Newton-Dinkelbach's method in Algorithm 3,
4. Gurobi for solving the MIQCP.

k-bisection solver vs. max-cut solver In our preliminary numerical experiments, the max-cut variant demonstrated superior performance compared to the branch-and-bound algorithm for k-bisection. For example for the instance `chesapeake` from the DIMACS set, it took 2 seconds to compute the edge expansion using the max-cut variant compared to 9.8 seconds with the k-bisection branch-and-bound algorithm. Therefore we only report the results for the max-cut variant to solve the scaled bisection problems. Additionally, the fact that the max-cut solver BiqBin is parallelized further motivates this selection.

Algorithm 2 vs. Algorithm 3 vs. Gurobi The detailed results of our experiments are given in Tables 2 to 6. In each of the tables, the first column gives the name of the instance followed by the number of vertices and edges. Column 4 reports the optimal solution, i.e., the edge expansion of the graph.

In the split & bound section of the table, the first two columns give the global lower and upper bound after the pre-elimination Algorithm 1. The number of candidates for k after the pre-elimination is given in column 3. In column 4 we report the number of indices $k \in \mathcal{I}$ we were able to eliminate after solving the root node of the branch-and-bound tree. Column 5 lists the total number of branch-and-bound nodes in the max-cut algorithm for all values of k considered. The last two columns display the time spent in the pre-elimination and the total time (including pre-elimination) of the algorithm.

In the section for Dinkelbach's algorithm, the first column gives the first guess for the edge expansion, i.e., the first trial for γ. As described before, we take the upper bound from the heuristic for this initialization. Note, that this first guess may differ from u^*, since in the pre-elimination step of split & bound we perform 30 additional rounds of simulated annealing for all indices $k \in \mathcal{I}$. Column 2 indicates how many parametrized problems P_{γ_i} have been solved, and column 3 gives the total number of branch-and-bound nodes for solving all parametrized problems. The fourth column of the results of Dinkelbach's algorithm displays the total time, including running the heuristic to obtain the first guess.

The final column of the tables holds information about computing the edge expansion using Gurobi. For the graphs from the randomly generated polytopes, Gurobi did not succeed to solve any of these instances within a time limit of 3 hours, we therefore report the gap after this time limit in Table 2. In Tables 3 to 6 we report the time for computing the edge expansion.

We highlight in the tables, which of our two algorithms performs better.

Algorithm 2 (Split & bound) As can be seen in all tables, the pre-elimination phase of split & bound only leaves a comparably small number of candidates for k to be further investigated. Remember that the number of potential candidates is $\lfloor \frac{n}{2} \rfloor$, whereas $|\mathcal{I}|$ is the number of candidates that remain after the pre-elimination. For the randomly generated 0/1-polytopes on average only 12% of the candidates have to be further examined, for the other instance classes we are left with approximately 20% on average. This indicates that already the cheap SDP bound is of good quality.

We also observe that the SDP bound in the root-node of the branch-and-bound tree is of high quality: in 48 out of the 67 instances the gap is closed within the root node for all candidates to be considered.

The heuristic for computing upper bounds also performs extremely well: for almost all instances the upper bound found is the edge expansion of the graph, see columns titled $h(G)$ and u^*. In fact, only for 3 instances the heuristic fails to find the optimal solution.

Overall, we solve almost all of the considered instances within a few minutes, for very few instances the branch-and-bound tree grows rather large and therefore computation times exceed several hours.

Algorithm 3 (Discrete Newton-Dinkelbach) Whenever the heuristic already returns the value of the optimal solution, we only have to solve one parametrized problem to certify the optimality of this value. For most of the instances tested, this certificate is already obtained in the root node of the branch-and-bound tree. However, there are many instances where BiqBin terminates because of numerical problems even for the first parametrized problem, see Table 2. This in particular arises when γ_d and γ_n (see Section § 5.1) are large.

Solving the MIQCP with Gurobi To compute the edge expansion using Gurobi, we input the last formulation of (1) adding the redundant constraint $y \geq 0$. Without this constraint, Gurobi terminated only after 1.65 hours/3 548 work units (resp. more than 24 hours/59 000 work units) on a graph with 29 vertices and 119 edges (resp. 37 vertices and 176 edges) corresponding to the grevlex polytope in dimension 7 (resp. 8).

Adding the redundant constraint, Gurobi is very efficient for graphs with an expansion less than one, see Tables 5 and 6, but as soon as the expansion (and also the number of vertices of the graph) gets larger, Gurobi cannot solve the instance within a few hours, see Tables 2 and 4.

Conclusion To summarize the results, we give a performance profile in Figure 3. Gurobi solves the MIQCP very efficiently for several instances, but fails to yield results for others within a time limit of 3 hours. It is the clear winner for instances with very small edge expansion. Comparing split & bound with the algorithm following the Discrete Newton-Dinkelbach method, we observe the following behavior. For the grlex instances, Dinkelbach performs extremely well compared to the split & bound approach, see Table 3. Whereas for the grevlex instances in Table 4, we observe that, except for dimension 13, the split & bound algorithm by far outperforms Algorithm 3. In addition to the already mentioned, there are some other instances where the difference in the runtimes between the two algorithms is significant. For example, on the instances `rand01-9-153-0` and `malaria_genes_HVR1` split & bound clearly dominates Algorithm 3, whereas the latter is significantly better on the instances `rand01-9-2781` and `sp-office`.

The conclusion is that in general for graphs with larger edge expansion, the split & bound algorithm is best, and for graphs with small edge expansion Algorithm 3 has a better performance than split & bound, but there are a few exceptions, and the difference in the total time of solving an instance can be quite large.

Overall, we conclude that with our algorithms we can compute the edge expansion of various graphs of size up to around 400 vertices and no other algorithm can achieve this. The time for solving an instance varies, it can be a few seconds for very structured instances and it can exceed one hour, in particular for the instances coming from 0/1-polytopes with rather large expansion. For standard branch-and-cut solvers like Gurobi these instances are out of reach.

7. Summary and future research

We developed a split & bound algorithm as well as an algorithm applying Dinkelbach's idea for fractional programming to compute the edge expansion of a graph. The split-

				Algorithm 2 (split & bound)							Algorithm 3 (Dinkelbach)				Gurobi
Instance	n	m	$h(G)$	min ℓ_k	u^*	$\lvert\mathcal{I}\rvert$	solved in root	B&B nodes	Alg. 1 time (s)	total time (s)	first guess	# of steps	B&B nodes	total time (s)	relative gap
rand01-9-153-0	153	4081	18.7500	17.7763	18.7500	5	5	5	43.2	129.4	18.7500	1	147	3397.6	0.900
rand01-9-153-1	153	4044	18.4868	17.5789	18.4868	5	5	5	39.9	111.9	18.4868	-	-	-	0.898
rand01-9-153-2	153	4107	19.0000	17.8421	19.0000	6	6	6	45.2	220.4	19.0000	1	83	1371.1	0.899
rand01-8-164-0	164	1868	5.7683	4.8659	5.7683	17	11	123	62.0	2037.7	5.7683	1	27	1610.4	0.809
rand01-8-164-1	164	1837	5.3537	4.7073	5.3537	15	12	27	56.9	774.7	5.3537	1	61	1786.1	0.785
rand01-8-164-2	164	1808	5.7439	4.7561	5.7439	29	5	251	85.3	5347	5.7561	2	78	2743.5	0.833
rand01-9-178-0	178	4590	17.0787	16.0899	17.0787	6	4	18	92.6	320.4	17.0787	-	-	-	0.919
rand01-9-178-1	178	4467	16.7079	15.3933	16.7079	9	8	11	87.9	506.8	16.7079	-	-	-	0.899
rand01-9-178-2	178	4537	16.7528	15.6517	16.7528	7	7	7	70.0	219.1	16.7528	-	-	-	0.920
rand01-8-189-0	189	1768	4.2234	3.4681	4.2234	23	11	633	99.2	5581.6	4.2340	2	14	1036.0	0.817
rand01-8-189-1	189	1745	4.0426	3.3723	4.0426	26	20	128	103.8	2634.7	4.0426	1	33	1603.1	0.795
rand01-8-189-2	189	1719	4.0638	3.3511	4.0745	28	19	100	97.9	2669.6	4.0851	2	38	2014.7	0.788
rand01-9-203-0	203	4900	15.1386	14.0198	15.1386	9	4	41	109.7	892.1	15.1386	-	-	-	0.930
rand01-9-203-1	203	4781	14.8416	13.5545	14.8416	12	2	388	117.2	3591.5	14.8515	-	-	-	0.925
rand01-9-203-2	203	4720	14.3762	13.3861	14.3762	9	9	9	105.8	412.1	14.3762	-	-	-	0.945
rand01-9-228-0	228	5065	13.2368	12.0439	13.2368	13	7	129	166.0	2083.8	13.2368	-	-	-	0.943
rand01-9-228-1	228	4927	9.0000	9.0000	9.0000	0	0	0	135.6	135.6	9.0000	1	23	586.7	0.857
rand01-9-228-2	228	4984	12.8246	11.8070	12.8246	11	11	11	174.3	619.9	12.8246	-	-	-	0.937
rand01-9-253-0	253	5258	11.8730	10.6825	11.8730	16	8	684	234.5	10547.7	11.9760	-	-	-	0.955
rand01-9-253-1	253	5053	9.0000	9.0000	9.0000	0	0	0	186.9	186.9	9.0000	1	1	121.3	0.910
rand01-9-253-2	253	5072	11.2222	10.1190	11.2222	16	3	402	232.7	8709.2	11.2302	-	-	-	0.970
rand01-10-256-0	256	11056	30.4766	29.4219	30.4766	5	5	5	228.8	547.7	30.4766	-	-	-	0.967
rand01-10-256-1	256	10611	28.8438	27.7031	28.8438	6	3	18	233.5	926.9	28.8438	1	1	308.6	0.969
rand01-10-256-2	256	10746	29.3750	28.1563	29.3750	6	3	20	240.6	769.7	29.3750	1	7	607.2	0.966
rand01-9-278-0	278	5224	10.0719	8.9065	10.0719	20	3	1292	326.8	17542.8	10.0719	-	-	-	0.955
rand01-9-278-1	278	5007	9.0000	8.3237	9.0000	15	3	387	336.6	8153.3	9.0000	1	1	103.7	0.954
rand01-9-278-2	278	5132	9.9209	8.6906	9.9209	22	6	2238	338.1	31125.4	9.9209	-	-	-	0.974
rand01-10-281-0	281	11828	28.9000	27.7357	28.9000	7	3	75	311.7	1807.9	28.9000	-	-	-	0.975
rand01-10-281-1	281	11490	27.7929	26.5214	27.7929	8	5	30	321.2	1776.4	27.8071	-	-	-	0.973
rand01-10-281-2	281	11454	27.7500	26.4571	27.7500	8	4	66	316.9	2435.7	27.7500	1	11	1103.5	0.972

Table 2: Comparison of Algorithm 2 (split & bound), Algorithm 3 (Dinkelbach) and Gurobi for graphs of random 0/1-polytopes. The last column displays the gap reported by Gurobi after a time limit of 3 hours.

				Algorithm 2 (split & bound)							Algorithm 3 (Dinkelbach)				Gurobi
Instance	n	m	$h(G)$	min ℓ_k	u^*	$\|\mathcal{I}\|$	solved in root	B&B nodes	Alg. 1 time (s)	total time (s)	first guess	# of steps	B&B nodes	total time (s)	total time (s)
grlex-7	29	119	1	1.0000	1	0	0	0	0.3	0.3	1	1	1	0.3	0.3
grlex-8	37	176	1	1.0000	1	0	0	0	0.6	0.6	1	1	1	0.9	0.6
grlex-9	46	249	1	1.0000	1	0	0	0	1.5	1.5	1	1	1	0.8	0.8
grlex-10	56	340	1	0.8571	1	7	7	7	2.7	22.7	1	1	1	2.4	1.2
grlex-11	67	451	1	0.8333	1	12	12	12	3.6	148	1	1	1	4.4	2.0
grlex-12	79	584	1	0.8000	1	15	15	15	5.8	280.4	1	1	1	4.6	2.0
grlex-13	92	741	1	0.8000	1	18	10	1788	8.5	14037.2	1	1	1	4.1	2.3

Table 3: Comparison of Algorithm 2 (split & bound) and Algorithm 3 (Dinkelbach) for grlex instances.

				Algorithm 2 (split & bound)							Algorithm 3 (Dinkelbach)				Gurobi
Instance	n	m	$h(G)$	min ℓ_k	u^*	$\|\mathcal{I}\|$	solved in root	B&B nodes	Alg. 1 time (s)	total time (s)	first guess	# of steps	B&B nodes	total time (s)	total time (s)
grevlex-7	29	119	2.4615	2.1429	2.4615	3	3	3	0.4	1.0	2.4615	1	41	33.3	1.1
grevlex-8	37	176	2.8333	2.3889	2.8333	5	5	5	1.0	5.8	2.8333	1	105	188.6	3.7
grevlex-9	46	249	2.9565	2.5652	2.9565	5	5	5	1.5	20.7	2.9565	1	89	194.1	39.9
grevlex-10	56	340	3.2222	2.7857	3.2222	6	6	6	2.9	33.8	3.2222	1	161	316.5	70.3
grevlex-11	67	451	3.6667	3.0909	3.6667	8	7	20	3.5	193.9	3.7188	2	1478	10412.3	1460.7
grevlex-12	79	584	3.9231	3.3333	3.9231	9	2	241	6.9	1315.5	3.9231	1	1293	11861.7	8624.9
grevlex-13	92	741	4.0000	3.5435	4.0000	7	1	475	9.4	2246.3	4.0000	1	1	29.4	-

Table 4: Comparison of Algorithm 2 (split & bound), Algorithm 3 (Dinkelbach) and Gurobi for grevlex instances.

				Algorithm 2 (split & bound)							Algorithm 3 (Dinkelbach)				Gurobi
Instance	n	m	$h(G)$	min ℓ_k	u^*	$\lvert\mathcal{I}\rvert$	solved in root	B&B nodes	Alg. 1 time (s)	total time (s)	first guess	# of steps	B&B nodes	total time (s)	total time (s)
karate	34	78	0.5882	0.5000	0.5882	4	4	4	0.7	2.3	0.5882	1	1	1.0	0.2
chesapeake	39	170	2.1667	2.0000	2.1667	8	8	8	1.0	2.0	2.1667	1	1	2.4	0.7
dolphins	62	159	0.2857	0.2000	0.2857	16	16	16	4.0	13.2	0.2857	1	1	2.3	0.7
lesmis	77	254	0.3000	0.2500	0.3000	2	2	2	4.7	14.7	0.3000	1	1	8.0	0.7
polbooks	105	441	0.3654	0.3269	0.3654	37	37	37	18.0	540	0.3654	1	11	128.7	3.3
adjnoun	112	425	1.0000	1.0000	1.0000	0	0	0	16.9	16.9	1.0000	1	1	8.6	4.2
football	115	613	1.0702	0.9825	1.0702	5	4	55	15.2	399.9	1.0702	1	1	25.8	31.2
jazz	198	2742	1.0000	1.0000	1.0000	0	0	0	118.4	118.4	1.0000	1	1	56.9	12.9
celegansneural	297	2148	1.0000	1.0000	1.0000	0	0	0	389.3	389.3	1.0000	1	1	80.6	22.0
celegans_metabolic	453	2025	0.4000	0.3333	0.5000	20	19	24	1475.6	2383.3	0.5000	3	3	828.2	5.1

Table 5: Comparison of Algorithm 2 (split & bound), Algorithm 3 (Dinkelbach) and Gurobi for DIMACS instances.

				Algorithm 2 (split & bound)							Algorithm 3 (Dinkelbach)				Gurobi
Instance	n	m	$h(G)$	min ℓ_k	u^*	$\lvert\mathcal{I}\rvert$	solved in root	B&B nodes	Alg. 1 time (s)	total time (s)	first guess	# of steps	B&B nodes	total time (s)	total time (s)
moviegalaxies-567	52	146	0.3810	0.3636	0.3810	3	3	3	2.3	3.5	0.3810	1	1	2.7	0.4
moviegalaxies-52	59	119	0.5385	0.4000	0.5385	27	27	27	3.9	16.3	0.5385	1	1	9.1	0.6
terrorists-911	62	152	0.2174	0.2000	0.2174	6	6	6	3.2	10.7	0.2174	1	1	7.5	0.5
train_terrorists	64	243	0.6000	0.4000	0.6000	20	20	20	5.2	44.9	0.6000	1	1	2.5	1.1
highschool	70	274	0.9143	0.7059	0.9143	26	26	26	5.5	131.2	0.9143	1	9	92.1	1.7
blumenau_drug	75	181	0.5000	0.5000	0.5000	0	0	0	5.1	5.1	0.5000	1	1	4.6	1.7
sp_office	92	755	3.3696	3.1739	3.3696	5	5	5	9.9	19.3	3.3696	1	77	858.7	522.2
swingers	96	232	0.3333	0.3333	0.3333	0	0	0	10.2	10.2	0.5000	3	3	26.2	1.4
game_thrones	107	352	0.4000	0.2857	0.4211	22	22	22	13.0	290.6	0.4375	2	2	28.6	2.7
revolution	141	160	0.0962	0.0770	0.0962	33	28	111	39.4	1595.6	0.1000	2	98	639.0	1.3
foodweb_little_rock	183	2434	1.0000	1.0000	1.0000	0	0	0	99.2	99.2	1.0000	1	1	21.4	8.9
cintestinalis	205	2575	1.0000	1.0000	1.0000	0	0	0	117.9	117.9	1.0000	1	1	25.2	20.1
malaria_genes_HVR1	307	2812	0.2377	0.2105	0.2377	120	91	1890	503.1	62943.4	0.2377	1	5	425.8	7.8

Table 6: Comparison of Algorithm 2 (split & bound), Algorithm 3 (Dinkelbach) and Gurobi for network instances.

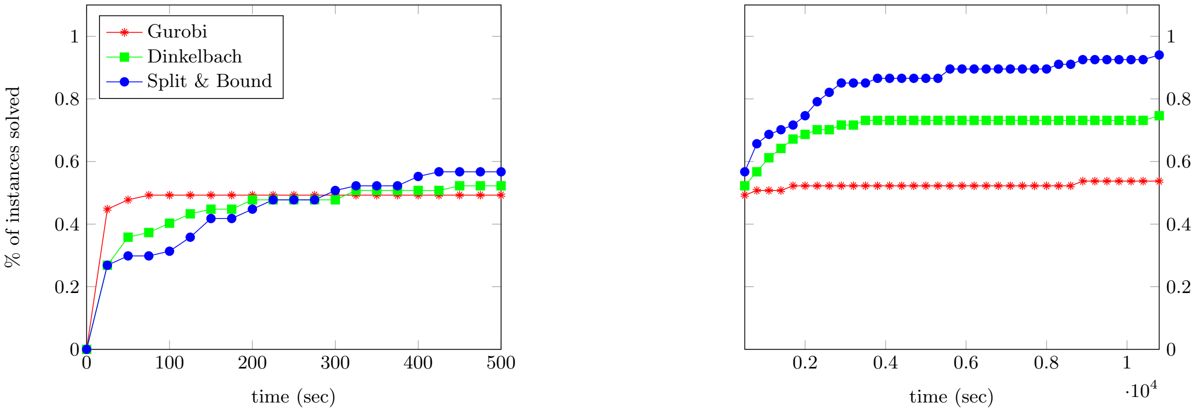

Figure 3: Performance comparison of the exact algorithms. Note the different scale on the x-axis: the plot on the left displays the time range from 0 to 500 seconds, the plot on the right from 500 seconds to 3 hours.

ting refers to the fact, that we consider the different values of k (k being the size of the smaller partition) separately. We used semidefinite programming in both phases of our algorithm: on the one hand, SDP-based bounds are used to eliminate several values for k and we use SDP-based bounds in a branch-and-bound algorithm that solves the problem for k fixed. I.e., we also developed a branch-and-bound solver for the k-bisection problem using bounds from semidefinite programming. Also, the algorithm following the Dinkelbach framework uses semidefinite programming in order to solve the underlying parametrized problems. Through numerical results on various graph classes, we demonstrate that our split-and-bound algorithm is a robust method for computing the edge expansion while Dinkelbach's algorithm and Gurobi are very sensitive with respect to the edge expansion of the graph.

In some applications, one is not interested in the exact value of the edge expansion but wants to check whether a certain value is a lower bound on the edge expansion, e.g., to check the Mihail-Vazirani conjecture on 0/1-polytopes. We implemented an option in our algorithm that enables this feature of verifying a given lower bound.

As a heuristic, we use a simulated annealing approach that works very well for the problem sizes we are interested in. However, if one wants to obtain high-quality solutions for larger instances, a more sophisticated heuristic will be needed. Tabu-search, genetic algorithms, or a heuristic in the spirit of the Goemans-Williamson rounding could be potential candidates. In future research, we will also investigate convexification techniques by using recent results on fractional programming [20] and on exploiting submodularity of the cut function as has been done for mixed-binary conic optimization [4].

References

[1] Nima Anari, Kuikui Liu, Shayan Oveis Gharan, and Cynthia Vinzant. "Log-Concave Polynomials II: High-Dimensional Walks and an FPRAS for Counting

Bases of a Matroid". In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing.* STOC 2019. New York, NY, USA: Association for Computing Machinery, 2019, pp. 1–12.

[2] Michael Armbruster, Marzena Fügenschuh, Christoph Helmberg, and Alexander Martin. "LP and SDP branch-and-cut algorithms for the minimum graph bisection problem: a computational comparison". In: *Mathematical Programming Computation* 4.3 (2012), pp. 275–306.

[3] Sanjeev Arora, Satish Rao, and Umesh Vazirani. "Expander flows, geometric embeddings and graph partitioning". In: *Journal of the ACM (JACM)* 56.2, 5 (2009), pp. 1–37.

[4] Alper Atamtürk and Andrés Gómez. "Submodularity in conic quadratic mixed 0–1 optimization". In: *Operations Research* 68.2 (2020), pp. 609–630.

[5] David A. Bader, Henning Meyerhenke, Peter Sanders, and Dorothea Wagner, eds. *Graph Partitioning and Graph Clustering, 10th DIMACS Implementation Challenge Workshop, Georgia Institute of Technology, Atlanta, GA, USA, February 13-14, 2012. Proceedings.* Vol. 588. Contemporary Mathematics. American Mathematical Society, 2013.

[6] Francisco Barahona, Michael Jünger, and Gerhard Reinelt. "Experiments in quadratic 0–1 programming". In: *Mathematical Programming* 44.1-3 (1989), pp. 127–137.

[7] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. "Julia: A fresh approach to numerical computing". In: *SIAM review* 59.1 (2017), pp. 65–98.

[8] Rainer E. Burkard and Franz Rendl. "A thermodynamically motivated simulation procedure for combinatorial optimization problems". In: *European Journal of Operational Research* 17 (1984), pp. 169–174.

[9] Fan RK Chung. *Spectral graph theory.* Vol. 92. CBMS Regional Conference Series in Mathematics. American Mathematical Society, 1997.

[10] Werner Dinkelbach. "On nonlinear fractional programming". In: *Management Science* 13.7 (1967), pp. 492–498.

[11] Tomás Feder and Milena Mihail. "Balanced Matroids". In: *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing.* Ed. by S.R. Kosaraju, T. Fellows, Avi Wigderson, and J.A. Ellis. STOC '92. Victoria, British Columbia, Canada: Association for Computing Machinery, 1992, pp. 26–38.

[12] Michael R. Garey, David S. Johnson, and Larry Stockmeyer. "Some simplified NP-complete graph problems". In: *Theoret. Comput. Sci.* 1.3 (1976), pp. 237–267.

[13] Oded Goldreich. "Basic facts about expander graphs". In: *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation.* Ed. by Oded Goldreich et al. Vol. 6650. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2011, pp. 451–464.

[14] Akshay Gupte and Svetlana Poznanović. "On Dantzig figures from graded lexicographic orders". In: *Discrete Mathematics* 341.6 (2018), pp. 1534–1554.

[15] Akshay Gupte, Melanie Siebenhofer, and Angelika Wiegele. “Computing the Edge Expansion of a Graph using SDP”. In: *Combinatorial optimization*. Lecture Notes in Comput. Sci. Springer, [Cham], 2024, 13 pages.

[16] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2023.

[17] Nicolò Gusmeroli, Timotej Hrga, Borut Lužar, Janez Povh, Melanie Siebenhofer, and Angelika Wiegele. “BiqBin: A Parallel Branch-and-Bound Solver for Binary Quadratic Problems with Linear Constraints”. In: *ACM Transactions on Mathematical Software* 48.2 (2022), 15:1–15:31.

[18] Nicolò Gusmeroli and Angelika Wiegele. “EXPEDIS: An exact penalty method over discrete sets”. In: *Discrete Optimization* 44.2 (2022), p. 100622.

[19] William W. Hager, Dzung T. Phan, and Hongchao Zhang. “An exact algorithm for graph partitioning”. In: *Mathematical Programming* 137.1-2 (2013), pp. 531–556.

[20] Taotao He, Siyue Liu, and Mohit Tawarmalani. *Convexification Techniques for Fractional Programs*. 2023. arXiv: 2310.08424 [math.OC].

[21] Shlomo Hoory, Nathan Linial, and Avi Wigderson. “Expander graphs and their applications”. In: *Bulletin of the AMS* 43.4 (2006), pp. 439–561.

[22] Volker Kaibel. “On the expansion of graphs of 0/1-polytopes”. In: *The Sharpest Cut. The Impact of Manfred Padberg and His Work*. Ed. by Martin Grötschel. Vol. 4. MOS-SIAM Optimization Series. SIAM, 2004. Chap. 13, pp. 199–216.

[23] Stefan E. Karisch and Franz Rendl. “Semidefinite programming and graph equipartition”. In: *Topics in semidefinite and interior-point methods (Toronto, ON, 1996)*. Vol. 18. Fields Inst. Commun. Amer. Math. Soc., Providence, RI, 1998, pp. 77–95.

[24] Stefan E. Karisch, Franz Rendl, and Jens Clausen. “Solving graph bisection problems with semidefinite programming”. In: *INFORMS Journal on Computing* 12.3 (2000), pp. 177–191.

[25] Etienne de Klerk, Dmitrii Pasechnik, Renata Sotirov, and Cristian Dobre. “On semidefinite programming relaxations of maximum k-section”. In: *Math. Program.* 136.2, Ser. B (2012), pp. 253–278.

[26] Nathan Krislock, Jérôme Malick, and Frédéric Roupin. “Biqcrunch: A semidefinite branch-and-bound method for solving binary quadratic problems”. In: *ACM Trans. Math. Software (TOMS)* 43.4 (2017), pp. 1–23.

[27] Tom Leighton and Satish Rao. “Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms”. In: *J. ACM* 46.6 (1999), pp. 787–832.

[28] Brett Leroux and Luis Rademacher. “Expansion of random 0/1 polytopes”. In: *Random Structures & Algorithms* 64 (2024), pp. 609–619.

[29] Miles Lubin, Oscar Dowson, Joaquim Dias Garcia, Joey Huchette, Benoit Legat, and Juan Pablo Vielma. “JuMP 1.0: Recent improvements to a modeling language for mathematical optimization”. In: *Mathematical Programming Computation* (2023). DOI: 10.1007/s12532-023-00239-3.

[30] Frank de Meijer, Renata Sotirov, Angelika Wiegele, and Shudian Zhao. "Partitioning through projections: strong SDP bounds for large graph partition problems". In: *Computers & Operations Research* 151 (2023). Id/No 106088, p. 20.

[31] Luis A. A. Meira and Flávio K. Miyazawa. "Semidefinite Programming Based Algorithms for the Sparsest Cut Problem". In: *RAIRO - Operations Research* 45.2 (2011), pp. 75–100.

[32] Milena Mihail. "On the expansion of combinatorial polytopes". In: *Mathematical Foundations of Computer Science 1992. MFCS 1992.* Ed. by Ivan M. Havel and Václav Koubek. Vol. 629. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 1992, pp. 37–49.

[33] MOSEK ApS. *MOSEK Optimizer API for C 10.0.47.* 2023.

[34] Asaf Nachmias and Asaf Shapira. "Testing the expansion of a graph". In: *Information and Computation* 208.4 (2010), pp. 309–314.

[35] Manfred Padberg. "The boolean quadric polytope: some characteristics, facets and relatives". In: *Mathematical Programming* 45 (1989), pp. 139–172.

[36] Tiago P. Peixoto. *The Netzschleuder network catalogue and repository.* 2020.

[37] Prasad Raghavendra and David Steurer. "Graph expansion and the Unique Games Conjecture". In: *Proceedings of the 42nd ACM Symposium on Theory of Computing.* STOC '10. Association for Computing Machinery, 2010, pp. 755–764.

[38] Franz Rendl, Giovanni Rinaldi, and Angelika Wiegele. "Solving max-cut to optimality by intersecting semidefinite and polyhedral relaxations". In: *Math. Program. Ser. A* 121.2 (2010), pp. 307–335.

[39] Peter Sarnak. "WHAT IS... an Expander?" In: *Notices of the AMS* 51.7 (2004), pp. 762–763.

[40] Siegfried Schaible. "Fractional programming. II, On Dinkelbach's algorithm". In: *Management Science* 22.8 (1976), pp. 868–873.

[41] Angelika Wiegele and Shudian Zhao. "SDP-based bounds for graph partition via extended ADMM". In: *Comput. Optim. Appl.* 82.1 (2022), pp. 251–291.

[42] Henry Wolkowicz and Qing Zhao. "Semidefinite programming relaxations for the graph partitioning problem". In: *Discrete Appl. Math.* 96/97 (1999), pp. 461–479.

A. Computing the maximum cut of a graph

Some of our algorithms for computing $h(G)$ rely on finding the maximum cut in a graph. For computing the value of the max-cut, we will use the SDP-based solver BiqBin [17]. Note that the software BiqBin can not only compute the max-cut in a graph and solve quadratic unconstrained binary problems (QUBOs) but it is also applicable to linearly constrained binary problems with a quadratic objective function. However, we only need the max-cut solver in this work, and briefly describe the main ingredients in this section.

BiqBin is a branch-and-bound algorithm that uses a tight SDP relaxation as upper bound and the celebrated Goemans-Williamson rounding procedure to generate a high-quality lower bound on the value of the maximum cut in a graph. To be more precise, the SDP

$$\max_X \left\{ \frac{1}{4}\langle L, X\rangle : \operatorname{diag}(X) = e,\ \mathcal{A}(X) = b,\ X \succeq 0 \right\} \tag{12}$$

where $\mathcal{A}(X) \leq b$ models a set of triangle-, pentagonal- and heptagonal-inequalities is approximately solved using a bundle method. To do so, only the inequality constraints are dualized yielding the nonsmooth convex partial dual function

$$\begin{aligned} f(\gamma) = \max_X \left\{ \frac{1}{4}\langle L, X\rangle - \gamma^\top(\mathcal{A}(X) - b) : \operatorname{diag}(X) \right. &= \left. e,\ X \succeq 0 \right\} \\ = b^\top\gamma + \max_X \left\{ \left\langle \frac{1}{4}L - \mathcal{A}^\top(\gamma), X \right\rangle : \operatorname{diag}(X) = e,\ X \succeq 0 \right\} \end{aligned}$$

where γ are the nonnegative dual variables associated with the constraints $\mathcal{A}(X) \leq b$. Evaluating the dual function $f(\gamma)$ and computing the subgradient amounts to solving an SDP that can be efficiently computed using an interior-point method tailored for this problem. It provides us with the matching pair (X_γ, γ) such that $f(\gamma) = b^\top\gamma + \langle 1/4 L - \mathcal{A}^\top(\gamma), X_\gamma\rangle$. Moreover, the subgradient of f at γ is given by $\partial f(\gamma) = b - \mathcal{A}(X_\gamma)$. For obtaining an approximate minimizer of problem

$$\min_\gamma \{f(\gamma) : \gamma \geq 0\},$$

the bundle method is used. We refer to [38] for more details.

Note that interior-point methods are far from computing a solution of (12) already for small graphs due to the number of constraints being too large and therefore already forming the system matrix is an expensive task or even impossible due to memory requirements.

BiqBin dominates all max-cut solvers based on linear programming and is comparable to the SDP-based solver BiqCrunch [26]. Moreover, BiqBin is available as a parallelized version.

B. Two lemmas for the proof of convergence of Algorithm 3

Lemma B.1. *Let γ', $\gamma'' \in \mathbb{R}$ and x', $x'' \in \mathcal{F}$ be the optimal solutions of $P(\gamma')$ and $P(\gamma'')$, then*

$$f(x') - f(x'') \leq \left(P(\gamma'') - (\gamma' - \gamma'')e^\top x''\right)\left(\frac{1}{e^\top x'} - \frac{1}{e^\top x''}\right).$$

Proof. By the optimality of x' for $P(\gamma')$ it holds that

$$x'^\top L x' - \gamma' e^\top x' \leq x''^\top L x'' - \gamma' e^\top x''.$$

Dividing both sides by $e^\top x''$ and rearranging yields

$$f(x') \le \frac{x''^\top L x''}{e^\top x'} + \gamma' \Big(1 - \frac{e^\top x''}{e^\top x'}\Big).$$

Hence, we get that

$$\begin{aligned} f(x') - f(x'') &\le \frac{x''^\top L x''}{e^\top x'} + \gamma' \Big(1 - \frac{e^\top x''}{e^\top x'}\Big) - \frac{x''^\top L x''}{e^\top x''} \\ &= (x''^\top L x'' - \gamma' e^\top x'') \Big(\frac{1}{e^\top x'} - \frac{1}{e^\top x''}\Big) \\ &= (x''^\top L x'' - \gamma'' e^\top x'' + \gamma'' e^\top x'' - \gamma' e^\top x'') \Big(\frac{1}{e^\top x'} - \frac{1}{e^\top x''}\Big) \\ &= \big(P(\gamma'') - (\gamma' - \gamma'') e^\top x''\big) \Big(\frac{1}{e^\top x'} - \frac{1}{e^\top x''}\Big). \end{aligned}$$

□

Lemma B.2. *Let x' and x'' be optimal solutions of $P(\gamma')$ and $P(\gamma'')$, then for $\gamma'' < \gamma'$ it holds that $e^\top x'' \le e^\top x'$.*

Proof. From the optimality of x' and x'' it follows that

$$\begin{aligned} x'^\top L x' - \gamma' e^\top x' &\le x''^\top L x'' - \gamma' e^\top x'' \quad \text{and} \\ x''^\top L x'' - \gamma'' e^\top x'' &\le x'^\top L x' - \gamma'' e^\top x'. \end{aligned}$$

Adding the above two inequalities yields

$$(\gamma'' - \gamma') e^\top x' \le (\gamma'' - \gamma') e^\top x''$$

and hence the above claim holds. □