CS264: Beyond Worst-Case Analysis
Lectures #11 and 12: SDP Algorithms for Semi-Random Bisection and Clique*

Tim Roughgarden†

February 14 and 16, 2017

1 Preamble

Lectures #9 and #10 studied the planted bisection (a.k.a. community detection) and planted clique models, where we posited specific parameterized input distributions (parameterized by edge densities p and q within and between clusters, or by planted the clique size k), in which a "clearly optimal" solution is planted in an otherwise random graph. The goal was to identify necessary and sufficient conditions on the parameters ($p - q$, or k) of the distribution such that the planted solution can be recovered in polynomial time. We obtained state-of-the-art positive results for these problems, for $p - q = \Omega(\sqrt{\frac{\log n}{n}})$ and $k = \Omega(\sqrt{n})$. We obtained these results using spectral algorithms, which compute the second eigenvector of the adjacency matrix followed by some problem-specific postprocessing.

How should we feel about these results? Technically, the results are quite interesting. And in the end, the theory ends up advocating natural algorithms that are not overly tailored to the assumed input distributions, so there is hope that these algorithms could perform well much more generally. Indeed, at least for graph partitioning, spectral algorithms constitute one of the dominant paradigms in practice.

Still, one can't help but notice that our *analysis* of these spectral algorithms strongly exploited the assumed input distribution (e.g., through bounds on the eigenvalues of random mean-zero symmetric matrices). Can we do better? That is, can we obtain more robust versions of our recovery results, that assume much less about the underlying input distribution? The answer is yes, although we will have to up our game—spectral algorithms will no longer be sufficiently powerful, and we'll need to rely on semidefinite-programming (SDP)-based algorithms.

†Department of Computer Science, Stanford University, 474 Gates Building, 353 Serra Mall, Stanford, CA 94305. Email: tim@cs.stanford.edu.

2 Semi-Random Models

Semi-random graph modes are a very nice idea of Blum and Spencer [1]. The goal of these models is to have as robust a data model as possible, subject to preserving the existence of a "planted" or "clearly optimal" solution. Like several upcoming lectures (e.g., on smoothed analysis), this is a "hybrid model" that blends together aspects of worst-case and average-case analysis.

2.1 Definition and Discussion

We describe semi-random models in particular for the minimum bisection and maximum clique problems. Nature and an adversary conspire to produce a hard input as follows. In the first step, nature chooses a random instance from the corresponding planted model. Recall for planted bisection the model is (for vertex set V and parameters p, q):

1. Choose a partition (S, T) of V with $|S| = |T|$ uniformly at random.
2. Independently for each pair (i, j) of vertices inside the same cluster (S or T), include the edge (i, j) with probability p.
3. Independently for each pair (i, j) of vertices in different clusters, include the edge (i, j) with probability q (with $q < p$).

For planted clique the model is (for vertex set V and parameter k):

1. Choose a random subset $Q \subseteq V$ of k vertices.
2. Add an edge between each pair of vertices in Q.
3. Independently for each pair (i, j) of vertices with at least one of i, j not in Q, include the edge (i, j) with probability $\frac{1}{2}$.

In the second step, an adversary is allowed to modify the graph. If we want to recover the planted solution, then we need to restrict the power of the adversary, as otherwise the adversary could simply replace the random instance by a worst-case instance. The rules of the game are such that the adversary can make the sparse regions sparser (by removing edges) and the dense regions denser (by adding edges) in an arbitrary way—this is called a *monotone adversary*. So in the minimum bisection problem, the adversary can delete any edges between S and T and can add any edges within S or within T. In the maximum clique problem, the adversary can remove any edges that are not inside the planted clique Q.[1]

The planted solution survives arbitrary modifications of a monotone adversary—if it was an optimal solution in the original random planted instance, then it remains optimal after the adversary's actions. In fact, since a monotone adversary can intuitively only make the planted solution "more optimal," one might wonder if the semi-random model is really any

[1] See [4, 5] for recent proposals of semi-random graph models with even stronger adversaries.

harder than the planted one. It is not easy to come up with a provable separation between a planted problem and its semi-random counterpart (though it can be done, see [**?**]). But it is easy to see that there are specific algorithms that work well in a planted model but not in the corresponding semi-random one. For example, the "top k degrees" algorithm that recovers the planted maximum clique with $k = \Omega(\sqrt{n \log n})$ (see Lectures #9–10) fails miserably in the semirandom model, even when k is linear in n: the adversary can sparsify the edges between the clique vertices and the other vertices, lowering the degrees of the clique vertices to right around $n/2$ (Homework #6). It is less obvious but also true that our spectral algorithms from last week can fail in semi-random models. We will instead use optimization techniques—specifically, semidefinite programming, as in Lecture #8.

The benefit of the semirandom model over the planted one is that it encourages more "robust" solutions — algorithms that are not overfitted to a particular data model. In the planted model, an algorithm can take advantage of two properties of the input: (i) there is a clearly optimal solution (a plausible property of many "real instances"); and (ii) the input shares lots of non-trivial and useful properties with completely random instances, such as highly predictable vertex degrees (a questionable property of "real instances"). The first advantage was the motivating point behind the planted model, while the second was only a side effect of our modeling choices. The semirandom model offers an algorithm only the first advantage, and is thus more faithful to our original goals.

3 Semi-Random Graph Bisection

We begin with exact recovery for the semi-random Minimum Bisection problem. We need to retain last week's assumption about the edge density gap $p - q$:

$$p - q \geq c_1 \sqrt{\frac{\log n}{n}}, \tag{1}$$

where c_1 is a sufficiently large constant. For simplicity we also assume that $p, q = \Omega(1)$, although the results continue to hold (with the same arguments) for much smaller p and q.

Feige and Kilian [3] prove the following cool theorem.

Theorem 3.1 ([3]) *Under assumption* (1)*, there is a polynomial-time algorithm that, with high probability over nature's coin flips, computes the minimum bisection in the semi-random model.*

Their approach is to use semidefinite programming (SDP), which we touched on in Lecture #8 in the context of exact recovery in perturbation-stable Maximum Cut instances.[2]

Recall that one good way to think about SDPs is as relaxations of certain types of quadratic programs (where pairs of decision variables can be multiplied together) with ± 1 decision variables. It's easy to express an instance of Minimum Bisection as such a

[2]See also Lecture #20 of the instructor's lecture notes for CS261 for a more leisurely treatment of semidefinite programming.

quadratic program. We have one decision variable $z_i \in \{-1, +1\}$ for each vertex $i \in V$, indicating which side of the cut i belongs to. (We could use $\{0, 1\}$, but $\{-1, +1\}$ is more convenient to pass to an SDP relaxation.) To derive the appropriate objective function, first note that $z_i z_j$ is 1 if i and j are on the same side of the cut, and -1 otherwise. Thus, our objective is:[3]

$$\min \frac{1}{2} \sum_{(i,j) \in E} (1 - z_i z_j). \tag{2}$$

Unlike when we were discussing the MAXIMUM CUT problem, here we have to enforce the constraint that the two sides of the cut have the same size:

$$\left(\sum_{i \in V} z_i \right)^2 = \sum_{i,j \in V} z_i z_j = 0. \tag{3}$$

This ensures that there are an equal number of +1s and -1s. Why not just write $\sum_{i \in V} z_i = 0$? Because when we eventually pass to a semidefinite programming relaxation, it's going to be important that the z_i's only appear in pairs (multiplied together), and not individually. Optimizing (2) subject to (3) and $z_i \in \{-1, +1\}$ (or equivalently, $z_i^2 = 1$) is exactly the MINIMUM BISECTION problem, and hence is NP-hard (in the worst case).[4]

Following the pattern in Lecture #8, the next step is to look at a vector relaxation of the quadratic program. So rather than forcing each z_i to be either 1 or -1, we allow it to be an arbitrary unit vector in n-dimensional Euclidean space, where n is the number of decision variables (i.e., $|V|$). Every appearance of a z_i in the quadratic program is replaced by a a vector $v_i \in \mathbb{R}^n$. Wherever we previously had the product of two decision variables, we now take the inner product of the corresponding vectors. We arrive at the following vector program (all norms in this lecture are 2-norms):

$$\min \sum_{(i,j) \in E} \underbrace{\tfrac{1}{2}(1 - \langle v_i, v_j \rangle)}_{\in [0,1]} \tag{4}$$

subject to

$$v_i \in \mathbb{R}^n \tag{5}$$

and

$$\|v_i\|^2 = 1 \tag{6}$$

for every $i \in V$, and

$$\sum_{i,j \in V} \langle v_i, v_j \rangle = 0. \tag{7}$$

This vector program is a relaxation of the MINIMUM BISECTION problem, since for any bisection (A, B), one always has the option of assigning all vertices $i \in A$ to $\mathbf{e}_1$ and all $i \in B$

[3] In Lecture #8, we worked with the equivalent expression $\frac{1}{4} \sum_{(i,j) \in E} (z_i - z_j)^2$ (and were maximizing rather than minimizing).

[4] In Lecture #8 we also added "triangle inequality" constraints; we won't need them here.

to $-\mathbf{e}_1$, where $\mathbf{e}_1$ denotes the first standard basis vector. Geometrically, this relaxation is mapping each vertex to a point on the sphere (in $\mathbb{R}^n$), to make the endpoints of each edge as close to each other as possible (to make (4) small), subject to the "balance constraint" (7).

Let $b(G)$ denote the number of edges crossing the minimum bisection of G and let $h(G)$ denote the optimal solution to the vector program above (or rather, the equivalent SDP, see Section 5). Since the vector program is a relaxation of the MINIMUM BISECTION problem—every solution of the latter maps to one of the former, with equal objective function value—we always have $h(G) \leq b(G)$. That is, the optimal solution of the relaxation can only be better than that of the actual problem.

Since computing a minimum bisection is NP-hard and computing $h(G)$ can be done in polynomial time (see Section 5), we expect that $h(G) < b(G)$ for many graphs G. The main technical lemma in Feige and Kilian [3], which is inspired by Boppana [2], is that *the relaxation is exact (w.h.p.) in the planted bisection model.*

Lemma 3.2 ([2, 3]) *For a random graph in the planted (non-semirandom) bisection model, under assumption* (1)*, $h(G) = b(G)$ with high probability.*

We'll prove Lemma 3.2 over the course of the next few sections.

But Lemma 3.2 merely matches the recovery guarantee that we already proved last week. Moreover, solving a semidefinite program is significantly more computationally intensive (if still polynomial time, see Section 5) than computing the eigenvectors of a matrix (as in last week's spectral algorithms). So what use is this new, slower solution?

Unlike the known combinatorial and spectral algorithms for planted models, the semidefinite programming approach extends automatically to the semi-random model, and is therefore a "more robust algorithm" in some sense. Formally, we now show that Theorem 3.1 follows easily from Lemma 3.2.

Proof of Theorem 3.1: Recall that $b(G)$ denotes the value of the minimum bisection of G, $h(G)$ denotes the value of the optimal solution to the relaxation (4)–(7), and that $h(G) \leq b(G)$ for every graph G.

We claim that if the graph $\hat{G}$ is obtained by adding a single edge to another graph G, then

$$h(G) \leq h(\hat{G}) \leq h(G) + 1. \tag{8}$$

To see the first inequality, take an optimal solution to the vector relaxation for $\hat{G}$, and note that it is also a feasible solution to the relaxation for G (with only less objective function value). The optimal solution to the latter relaxation can of course only be better. The argument for the second inequality is similar: we can reinterpret the optimal solution to the vector relaxation for G as a feasible solution to the relaxation for $\hat{G}$, with objective function value at most 1 larger (since $\frac{1}{2}(1 - \langle v_i, v_j \rangle) \in [0, 1]$). The optimal solution to the latter relaxation can only be better.

Now suppose the graph G_0 is chosen by nature from the planted bisection model, with assumption (1). By Lemma 3.2, with high probability, $b(G) = h(G)$. Conditioned on this being the case, we now show that b and h stay equal no matter what the adversary does (and hence the relaxation remains exact).

The adversary adds or deletes edges one at a time, yielding a sequence $G_0, \ldots, G_t$. We claim that, by induction, $b(G_i) = h(G_i)$ for every i.

Let (A, B) denote the planted bisection. First, consider a step where the adversary adds an edge inside A or inside B to obtain G_i from G_{i-1}. The planted bisection has the same value and other bisections can only get worse, so $b(G_i) = b(G_{i-1})$. By (8), $h(G_i) \geq h(G_{i-1}) = b(G_{i-1}) = b(G_i)$. But since $h(G) \leq b(G)$ for every graph G, we must have $b(G_i) = h(G_i)$.

Second, consider a step in which the adversary removes an edge of the planted bisection. We then have $b(G_i) = b(G_{i-1}) - 1$ (the planted bisection gets better by 1, and no other bisection gets better by more than 1). By (8), $h(G_i) \geq h(G_{i-1}) - 1 = b(G_{i-1}) - 1 = b(G_i)$. But since $h(G) \leq b(G)$ for every graph G, we must have $b(G_i) = h(G_i)$. ■

The proof above shows how to compute the *value* $b(G)$ of the minimum bisection in polynomial time with high probability (by computing $h(G)$ instead). We leave as an exercise the task of using this subroutine to reconstruct the planted bisection itself in polynomial time (Homework #6).

Theorem 3.1 is one of the main results of this week: semidefinite programming extends the recovery guarantee we proved for a spectral algorithm from the random planted model to the semi-random model. But there a few issues that probably seem mysterious at this point:

1. What's the connection between the vector programs we've been looking at and these oft-mentioned semi-definite programs?

2. Why on earth can you solve vector (and semidefinite) programs in polynomial time?

3. Why is Lemma 3.2 true?

The next sections demystify each of these issues in turn. To answer the final question, we will need to discuss duality for semidefinite programs.

4 Linear Algorithm Review (Part 2)

To make sure we're all on the same page, we review the basics of *(symmetric) positive semidefinite (psd) matrices*. Let's start from the spectral decomposition for symmetric matrices, which we reviewed last week. If M is an $n \times n$ symmetric matrix, then it can be written as

$$\mathbf{M} = \underbrace{\begin{pmatrix} | & | & & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_n \\ | & | & & | \end{pmatrix}}_{\mathbf{Q}} \cdot \underbrace{\begin{pmatrix} \lambda_1 & & 0 \\ & \lambda_2 & \\ & & \ddots \\ 0 & & & \lambda_n \end{pmatrix}}_{\mathbf{D}} \cdot \underbrace{\begin{pmatrix} — & \mathbf{u}_1 & — \\ — & \mathbf{u}_2 & — \\ & \vdots & \\ — & \mathbf{u}_m & — \end{pmatrix}}_{\mathbf{Q}^\top}. \tag{9}$$

The $\mathbf{u}_i$'s are the eigenvectors of $\mathbf{M}$ (scaled to have unit norm), which form an orthonormal basis of $\mathbb{R}^n$, and the λ_i's are the corresponding (real-valued) eigenvalues. We can assume

that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$. Recall that $\mathbf{Q}^\top$ re-represents a vector $\mathbf{v}$ in terms of the basis of $\mathbf{u}_i$'s, the diagonal matrix $\mathbf{D}$ then scales and possibly flips independently along the directions of the $\mathbf{u}_i$'s, and $\mathbf{Q}$ translates the result back into the standard basis. Thus the operator (from $\mathbb{R}^n$ to itself) corresponding to a symmetric matrix is always super-simple, really the same as a diagonal matrix (after a suitable change of basis).

One definition of a psd matrix is that all of its eigenvalues are nonnegative, or equivalently that the smallest eigenvalue λ_n is nonnegative. Geometrically, then, the operator defined by a psd matrix can only scale (and not flip) along the directions of its eigenvectors.

There's zillions of equivalent definitions of psd matrices, and its worth knowing a few of them. For example, a symmetric matrix $\mathbf{M}$ is psd if and only if the corresponding "quadratic form" is nonnegative, meaning

$$\mathbf{x}^\top \mathbf{M}\mathbf{x} \geq 0 \text{ for all } \mathbf{x} \in \mathbb{R}^n. \tag{10}$$

For the proof of equivalence, first assume that all of M's eigenvalues are nonnegative. Then for an arbitrary $\mathbf{x}$, express it as a linear combination $\mathbf{x} = \sum_{i=1}^n \alpha_i \mathbf{u}_i$ of $\mathbf{M}$'s eigenvectors and note that $\mathbf{x}^\top \mathbf{M}\mathbf{x} = \sum_{i=1}^n \lambda_i \alpha_i^2 \geq 0$. Conversely, if $\lambda_n < 0$, then $\mathbf{u}_n^\top \mathbf{M}\mathbf{u}_n = \lambda_n < 0$.

Another characterization is: an $n \times n$ symmetric matrix $\mathbf{M}$ is psd if and only if it has a "square root," meaning there is a matrix $\mathbf{V} \in \mathbb{R}^{n \times n}$ that satisfies

$$\mathbf{M} = \mathbf{V}^\top \mathbf{V}. \tag{11}$$

If (11) holds, then certainly (10) holds as well, since for every $\mathbf{x} \in \mathbb{R}^n$ we have $\mathbf{x}^\top \mathbf{M}\mathbf{x} = \mathbf{x}^\top \mathbf{V}^\top \mathbf{V}\mathbf{x} = \|\mathbf{V}\mathbf{x}\|^2 \geq 0$. In the other direction, suppose $\mathbf{M}$ is psd according to original definition (all nonnegative eigenvalues). Then it is clear from the decomposition in (9) how to define the square root, as $\sqrt{\mathbf{D}}\mathbf{Q}^\top$ (where $\sqrt{\mathbf{D}}$ just means taking entrywise square roots of $\mathbf{D}$, which makes sense since its entries—$\mathbf{M}$'s eigenvalues—are all nonnegative).

What the condition (11) really asserts is that the entries of $\mathbf{M}$ arise as all pairwise inner projects of the columns of $\mathbf{V}$. (Matrix multiplication says that M_{ij} is the inner product of the ith row of $\mathbf{V}^\top$ and the jth column of $\mathbf{V}$.) That is, $\mathbf{M}$ is a psd matrix if and only if it has the form

$$\begin{pmatrix} \langle \mathbf{v}_1, \mathbf{v}_1 \rangle & \langle \mathbf{v}_1, \mathbf{v}_2 \rangle & \cdots & \langle \mathbf{v}_1, \mathbf{v}_n \rangle \\ \langle \mathbf{v}_2, \mathbf{v}_1 \rangle & \langle \mathbf{v}_2, \mathbf{v}_2 \rangle & \cdots & \langle \mathbf{v}_2, \mathbf{v}_n \rangle \\ \vdots & \vdots & \vdots & \vdots \\ \langle \mathbf{v}_n, \mathbf{v}_1 \rangle & \langle \mathbf{v}_n, \mathbf{v}_2 \rangle & \cdots & \langle \mathbf{v}_n, \mathbf{v}_n \rangle \end{pmatrix}, \tag{12}$$

where $\mathbf{v}_i$ corresponds to the ith column of $\mathbf{V}$. It is this view of psd matrices that suggests a connection between vector programs and semidefinite programs, developed in the next section.

5 Semidefinite Programs

An $n \times n$ matrix is psd if and only if its entries arise as all pairwise inner products of n vectors $\mathbf{v}_i \in \mathbb{R}^n$ (the columns of $\mathbf{V}$ in the previous section). Thus, if we have a vector

program where the vectors only appear as inner products with each other (like (4)–(7)), we can recast the decision variables as these inner products, subject to a psd constraint. For example, the vector program in (4)–(7) is equivalent to the following:

$$\min \frac{1}{2} \sum_{(i,j) \in E} (1 - x_{ij}) \tag{13}$$

subject to

$$\sum_{i,j \in V} x_{ij} = 0, \tag{14}$$

$$x_{ii} = 1 \tag{15}$$

for every vertex $i \in V$, and the constraint that

$$\mathbf{X} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} \end{pmatrix}, \tag{16}$$

is a (symmetric) psd matrix. In effect, we have "linearized" the inner products in (4)–(7), replacing each one with a new decision variable ($\langle v_i, v_j \rangle$ by x_{ij}), but then adding the psd constraint to keep the x_{ij}'s honest (i.e., they really should correspond to the inner products of n vectors in $\mathbb{R}^n$, as intended). "Integer solutions" to this SDP correspond to psd matrices that have the form $\mathbf{X} = \mathbf{x}\mathbf{x}^\top$ for $\mathbf{x} \in \{\pm 1\}^n$. In general, linear programs supplemented by psd constraints (like (13)–(16)) are called *semidefinite programs*, or *SDPs*.

Back in Lecture #8, we stated without proof or intuition that vector programs can be solved in polynomial time. We alluded to the fact that convexity is what underlies the computational tractability. But where's the convexity in a vector relaxation? After all, if you take the average of two points on the unit sphere, you don't get another point on the unit sphere.

By reformulating vector programs as semidefinite programs, we now see where the convexity comes from. Specifically, it is clear from the characterization of psd matrices in (10) that the set of psd matrices of a given size is convex.[5] Since everything else about our semidefinite programs are linear, the optimization problem boils down to optimizing a linear function over a convex feasible region. This can be done in polynomial time (under minor technical conditions, which hold in our applications), using the ellipsoid method or interior-point methods (see e.g. Lecture #20 of CS261 for more discussion).[6,7] If the corresponding vectors (the $\mathbf{v}_i$'s) are desired, they can be recovered as the columns of the "square root" of the computed psd matrix.

[5] Using linearity, if $\mathbf{x}^\top \mathbf{M}_1 \mathbf{x} \geq 0$ and $\mathbf{x}^\top \mathbf{M}_2 \mathbf{x} \geq 0$ for all $\mathbf{x}$, then $\mathbf{x}^\top(\alpha \mathbf{M}_1 + (1-\alpha)\mathbf{M}_2)\mathbf{x} = \alpha \mathbf{x}^\top \mathbf{M}_1 \mathbf{x} + (1-\alpha)\mathbf{x}^\top \mathbf{M}_2 \mathbf{x} \geq 0$ for all $\mathbf{x}$.

[6] Strictly speaking, since the optimal solution might be irrational, we can only solve it up to arbitrarily small error.

[7] It is also important that psd matrices can be recognized in polynomial time, but this just boils down to an eigenvalue computation.

6 Duality

Our high-level plan for proving Lemma 3.2 is: (i) let $\hat{\mathbf{x}}$ denote the SDP solution corresponding to the planted bisection (S, T) (with $\hat{x}_{ij} = 1$ if i, j are on the same side of the bisection and $\hat{x}_{ij} = -1$ otherwise); and (ii) certify the optimality of the "integral solution" $\hat{\mathbf{x}}$ to (13)–(16) by exhibiting a feasible solution to the dual SDP that has the same objective function value. Clearly, to implement this, we need to understand what the dual of an SDP is. Fortunately, SDP duality can be understood fairly easily by analogy with linear programming duality (which hopefully you have at least passing familiarity with, otherwise see below and CS261).[8] This "dual certificate" approach is ubiquitous in the literature on exact recovery (across many different problems), and in particular works well for both the planted bisection and planted clique problems.

6.1 Linear Programming Duality

First, a brief review of linear programming duality, to serve as a model for the subsequent description of SDP duality. One canonical way to write a linear program is as

$$\min \mathbf{c}^\top \mathbf{x} \tag{17}$$

subject to

$$\begin{aligned} \mathbf{A}\mathbf{x} &= \mathbf{b} \\ \mathbf{x} &\geq 0, \end{aligned} \tag{18}$$

where $\mathbf{c}$ and $\mathbf{x}$ are (column) n-vectors, $\mathbf{b}$ is a (column) m-vector, $\mathbf{A}$ is an $m \times n$ matrix and the equations and inequalities are componentwise. It will be convenient to write the equality constraints in a more verbose form:

$$\mathbf{a}_i \mathbf{x} = b_i \tag{19}$$

for $i = 1, 2, \ldots, m$, where $\mathbf{a}_i$ is the ith row of $\mathbf{A}$ (and so $\mathbf{a}_i \mathbf{x}$ is a scalar). Let's call this the primal linear program.

The corresponding dual linear program is, by definition,

$$\max \mathbf{y}^\top \mathbf{b} \tag{20}$$

subject to

$$\sum_{j=1}^{m} y_j \cdot \mathbf{a}_j^\top \leq \mathbf{c}, \tag{21}$$

where the y_j's are scalars and the $\mathbf{a}_j^\top$'s and $\mathbf{c}$ are column n-vectors. The inequalities are componentwise (for $i = 1, 2, \ldots, n$).

In general, there is a purely syntactic way of passing from an arbitrary linear program to its dual and back (the dual of the dual is equivalent to the original linear program).

[8] Our treatment is inspired by Gupta [**?**], which in turn is inspired primarily by Lovász [**?**].

This can be very useful, but don't forget the true meaning of the dual (which holds in all cases): *feasible dual solutions correspond to bounds on the best-possible primal objective function value (derived from taking linear combinations of the constraints), and the optimal dual solution is the tightest-possible such bound.*

If you remember the meaning of duals, then "weak duality" becomes tautological.

Theorem 6.1 (Weak Duality) *For every minimization linear program (P) and corresponding dual linear program (D),*

$$\textit{OPT value for (P)} \geq \textit{OPT value for (D)}.$$

The proof, in matrix-vector notation, simply takes primal and dual feasible solutions $\mathbf{x}$ and $\mathbf{y}$ and notes that $\mathbf{c}^\top \mathbf{x} \geq \mathbf{y}^\top \mathbf{A}\mathbf{x} = \mathbf{y}^\top \mathbf{b}$.

The following corollary (and its extension to SDPs) is highly relevant for exact recovery.

Corollary 6.2 *Let (P),(D) be a primal-dual pair of linear programs. If $\mathbf{x}, \mathbf{y}$ are feasible for (P),(D) and $\mathbf{c}^T\mathbf{x} = \mathbf{y}^T\mathbf{b}$, then $\mathbf{x}$ and $\mathbf{y}$ are both optimal.*

Corollary 6.2 follows from the fact that there is no feasible primal solution with objective function value strictly smaller than that of any feasible dual solution. So if we ever find a primal-dual pair with matching objective function values, there can be no better (i.e., smaller) primal solution, and there can be no better (i.e., larger) dual solution.

You may also know about strong linear programming duality, which states that whenever both the primal and dual LPs have optimal solutions (guaranteed if, e.g., their feasible regions are non-empty and bounded), these optimal solutions have equal objective function values. (In particular, providing the converse to Corollary 6.2.) We won't need strong duality in these lectures.

6.2 Semidefinite Programming Duality

In the semidefinite programming analog of linear programming duality, we replace n-vectors with $n \times n$ symmetric matrices, inner products of vectors with the corresponding product

$$\mathbf{A} \bullet \mathbf{B} = \sum_{i,j} a_{ij} b_{ij} \tag{22}$$

of matrices (i.e., the inner product treating the matrices as vectors of length of n^2), and nonnegativity constraints by psd constraints. With these replacements, the primal linear program (17)–(19) becomes the semidefinite program with objective function

$$\min \mathbf{C} \bullet \mathbf{X} \tag{23}$$

and constraints

$$\mathbf{A}_i \bullet \mathbf{X} = b_i \tag{24}$$

for $i = 1, 2, \ldots, m$ and

$$\mathbf{X} \text{ psd.} \tag{25}$$

Here, $\mathbf{C}$, $\mathbf{X}$, and the $\mathbf{A}_i$'s are $n \times n$ matrices. The b_i's are scalars.

The dual program (20)–(21) becomes the semidefinite program with objective function

$$\max \mathbf{y}^\top \mathbf{b} \tag{26}$$

and constraint

$$\mathbf{C} - \sum_{j=1}^{m} y_j \mathbf{A}_j \text{ is psd.} \tag{27}$$

As a sanity check, let's verify weak duality—that the (dual) objective function of every dual feasible solution is at most the (primal) objective function value of every primal feasible solution. (This should hold as long as we came up with the "correct" definition of a dual SDP.) As with weak linear programming duality, we'll just follow our nose, applying whatever equations or inequalities apply at each step of the derivation.

Theorem 6.3 (Weak SDP Duality) *For every semidefinite program in the form* (23)–(25) *and corresponding dual semidefinite program* (26)–(27)*,*

$$\textit{optimal value of primal} \geq \textit{optimal value of dual}.$$

We'll need the following lemma, which you'll prove on Homework #6. It is a simple consequence of the fact (which you'll prove) that the entrywise product of psd matrices is again psd.[9]

Lemma 6.4 *If $\mathbf{A}, \mathbf{B}$ are $n \times n$ psd matrices, then $\mathbf{A} \bullet \mathbf{B} \geq 0$.*

We then have

$$\mathbf{C} \bullet \mathbf{X} = \left(\mathbf{C} - \sum_{j=1}^{m} y_j \mathbf{A}_j + \sum_{j=1}^{m} y_j \mathbf{A}_j \right) \bullet \mathbf{X} = \mathbf{y} \top \mathbf{b} + \underbrace{\left(\mathbf{C} - \sum_{j=1}^{m} y_j \mathbf{A}_j \right)}_{\text{psd}} \bullet \underbrace{\mathbf{X}}_{\text{psd}} \geq \mathbf{y}^\top \mathbf{b},$$

with the final inequality following from Lemma 6.4. This concludes the proof of Theorem 6.3.[10]

[9] This is one of many senses in which psd matrices behave analogously to nonnegative real numbers.

[10] There is also a version of strong duality for SDPs, but it requires more conditions than in the case of LPs. The most common sufficient condition requires that both the primal and dual SDPs have "strictly feasible" solutions.

7 Proof of Lemma 3.2

7.1 The Set-Up

To prove Lemma 3.2, we first instantiate the theory of SDP duals to the SDP that we actually care about, given in (13)–(16). Let's rewrite that SDP in the notation used in (23)–(25). We can write the objective function (13) as

$$\min \frac{|E|}{2} - \frac{1}{4} \cdot \mathbf{B} \bullet \mathbf{X},$$

where $\mathbf{B}$ denotes the adjacency matrix of the input graph (the $\frac{1}{4}$ because each edge is counted twice in $\mathbf{B} \bullet \mathbf{X}$). In the notation of (24), the constraint (14) translates to a psd matrix $\mathbf{A}_0 = \mathbf{J}$, where $\mathbf{J}$ is the all-ones matrix, and the n constraints in (15) to matrices $\mathbf{A}_1, \ldots, \mathbf{A}_n$, where $\mathbf{A}_j$ has a 1 in entry (j, j) and 0 everywhere else, and $b_j = 1$.

The corresponding dual SDP, in the form of (26)–(27), is then

$$\max \frac{|E|}{2} + \sum_{j=1}^{n} y_j \tag{28}$$

subject to

$$\left(-\frac{1}{4}\mathbf{B} - y_o\mathbf{J} - \sum_{j=1}^{n} y_j \mathbf{A}_j \right) \text{ is psd.} \tag{29}$$

The harmless constant $\frac{|E|}{2}$ is just a shift of the objective function and hence gets carried over to the dual. Note that y_0 does not appear in the dual objective function, because $b_0 = 0$.

To write the dual more succinctly, write $\mathbf{Y} = \text{diag}(y_1, \ldots, y_n)$ for the $n \times n$ diagonal matrix that has the y_j's on the diagonal; note that this is precisely $\sum_{j=1}^{n} y_j \mathbf{A}_j$. Also, for convenience, let's multiply the constraint (29) by 4 (which doesn't affect it) and replace each y_j by $y_j/4$. The result is the SDP

$$\max \frac{|E|}{2} + \frac{1}{4} \sum_{j=1}^{n} y_j \tag{30}$$

subject to

$$-\mathbf{B} - y_0\mathbf{J} - \mathbf{Y} \text{ is psd.} \tag{31}$$

7.2 The Analysis

Lemma 3.2 asserts that the feasible solution to the primal SDP above (with $x_{ij} = 1$ if i, j are on the same side of the planted bisection, and -1 otherwise) is optimal. By weak duality (Theorem 6.3), one way to prove this is to exhibit a feasible solution to the dual SDP (30)–(31) that has the same objective function value (i.e., the number of edges crossing the planted bisection). We proceed with this plan.

Defining a dual solution boils down to specifying values for $y_0, y_1, \ldots, y_n$. The first, and easier, step is to define a dual solution with the right objective function value. For this, we only need to discuss values for $y_1, \ldots, y_n$ (since y_0 does not participate in (31)). We want these values to satisfy

$$\begin{aligned}\frac{1}{4}\sum_{j=1}^{n} y_j &= (\#\text{ of edges cut by } (S,T)) - \frac{|E|}{2} \\ &= \frac{1}{2}\cdot(\#\text{ of edges cut by } (S,T) - \#\text{ of edges not cut by } (S,T)), \end{aligned} \tag{32}$$

where (S,T) denotes the planted bisection. A natural solution is to ascribe to each y_j its share of the "imbalance" in (32). That is, if we set

$$y_j = \#\text{ of } j\text{'s neighbors on other side of } (S,T) - \#\text{ of } j\text{'s neighbors on same side}, \tag{33}$$

then indeed, $\frac{1}{4}\sum_{j=1}^{n} y_j$ equals the number of edges crossing the planted bisection.

The second and harder step is to show that, for a suitable choice of y_0, the dual solution $y_0, y_1, \ldots, y_n$ is feasible, meaning that the matrix $-\mathbf{B} - y_0 \cdot \mathbf{J} - \mathbf{Y}$ is psd. Set $y_0 = -1$, so that we are trying to prove that the matrix $\mathbf{M} := -\mathbf{B} + \mathbf{J} - \mathbf{Y}$ is psd. It's not hard to check that $\mathbf{M}$ is singular (Homework #6) and hence has a zero eigenvalue. Since a symmetric matrix is psd if and only if all of its eigenvalues are nonnegative (Section 4), it suffices to show that the second-smallest eigenvalue $\lambda_{n-1}(\mathbf{M})$ of $\mathbf{M}$ is strictly positive. Happily, we can reuse much of the work we did last week to prove this without too much trouble. Last week, this linear-algebraic machinery drove the (spectral) algorithms themselves, but here we use it purely in the analysis (to prove exactness of an SDP).

To track the value of λ_{n-1} as we successively add the components of $\mathbf{M}$ to each other, we recall a linear algebra lemma from last week (proved on Homework #5, using the variational characterization of eigenvalues).

Lemma 7.1 *Let $\mathbf{X}, \mathbf{Y}$ be $n \times n$ symmetric matrices with eigenvalues $\lambda_1 \geq \cdots \geq \lambda_n$ and $\mu_1 \geq \cdots \geq \mu_n$. For every $i = 1, 2, \ldots, n$, the ith eigenvalue ν_i of $\mathbf{X} + \mathbf{Y}$ satisfies*

$$\lambda_i + \mu_n \leq \nu_i \leq \lambda_i + \mu_1.$$

That is, adding one symmetric matrix $\mathbf{Y}$ to another $\mathbf{X}$ shifts each eigenvalue by a number somewhere between the smallest and largest eigenvalues of $\mathbf{Y}$ (with different eigenvalues possibly shifting by different amounts).

Since $\mathbf{M} = -\mathbf{B} + \mathbf{J} - \mathbf{Y}$, we can also write $\mathbf{M}$ as $\overline{\mathbf{B}} + \mathbf{I} - \mathbf{Y}$, where $\mathbf{I}$ is the identity matrix and $\overline{\mathbf{B}}$ is the adjacency matrix of the complement graph $\overline{G}$ of G (formed by complementing G's edge set). As in Lectures #9–10, we can further decompose $\overline{\mathbf{B}}$ as $\widehat{\mathbf{M}} + \mathbf{R}$, where $\widehat{\mathbf{M}}$ is

the "expected adjacency matrix" of $\overline{G}$:

$$\widehat{\mathbf{M}} = \left(\begin{array}{cccc|cccc} 1-p & 1-p & \cdots & 1-p & 1-q & 1-q & \cdots & 1-q \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1-p & 1-p & \cdots & 1-p & 1-q & 1-q & \cdots & 1-q \\ \hline 1-q & 1-q & \cdots & 1-q & 1-p & 1-p & \cdots & 1-p \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1-q & 1-q & \cdots & 1-q & 1-p & 1-p & \cdots & 1-p \end{array}\right), \tag{34}$$

where we have permuted the rows and columns so that those corresponding to vertices in S come first; and $\mathbf{R}$ is the random matrix

$$\mathbf{R} = \left(\begin{array}{c|c} -(1-p)/p & -(1-q)/q \\ \hline -(1-q)/q & -(1-p)/p \end{array}\right), \tag{35}$$

where "$-(1-p)/p$" means that an entry is either $1-p$ (with probability p) or p (with probability $1-p$), and similarly for "$-(1-q)/q$." Like last week, for convenience we're cheating a bit on the diagonal entries, which are actually all zeros in $\widehat{\mathbf{M}}$ and $\mathbf{R}$. This cheat only changes the eigenvalues of $\widehat{\mathbf{M}}$ by $1-p$, which is negligible in our context.

We already computed the eigenvalues of the matrix $\widehat{\mathbf{M}}$ in Lectures #9–10. The matrix has rank 2, and thus has only two non-zero eigenvalues, and these eigenvalues are $(2-p-q)n$ and $(q-p)n$. Since the former number is positive and the latter negative, we have $\lambda_1(\widehat{\mathbf{M}}) = (2-p-q)n$, $\lambda_2(\widehat{\mathbf{M}}) = \cdots = \lambda_{n-1}(\widehat{\mathbf{M}}) = 0$, and $\lambda_n(\widehat{\mathbf{M}}) = (q-p)n$.

We also gave a high-probability bound on the eigenvalues of symmetric random matrices such as $\mathbf{R}$. Here is the relevant statement from last week.[11]

Theorem 7.2 ([?]) *Let $\mathbf{P}$ be a random matrix with the following properties:*

(i) With probability 1, $\mathbf{P}$ is symmetric, with only zeroes on the diagonal.

(ii) Each random variable P_{ij} has expectation 0, and is bounded between -1 and 1.

(iii) The random variables $\{P_{ij}\}_{1 \le i < j \le n}$ are independent.

Then there is a constant $c_2 > 0$ such that, with probability approaching 1 as $n \to \infty$,

$$\|\mathbf{P}\| \le c_2 \sqrt{n}.$$

[11] We sketched a proof in lecture of the weaker bound of $O(\sqrt{n \log n})$, the proof of Theorem 7.2 on Homework #5. With more work, one can prove a bound with $c_2 = 1$ [**?**].

Recall that the operator norm $\|\mathbf{M}\|$ of a matrix $\mathbf{M}$ is the largest magnitude of one of its eigenvalues. So Theorem 7.2 says that all of the eigenvalues of a random symmetric matrix have magnitude at most $c_2\sqrt{n}$ (with high probability).

Finally, recall that the eigenvalues of a diagonal matrix (like $\mathbf{I}$ or $-\mathbf{Y}$) are precisely the diagonal entries. The matrix $-\mathbf{Y}$ is random, with y_j defined as in (33). Recalling the random graph model, the expected number of neighbors on the same side is $p(\frac{n}{2}-1)$ and on the opposite side is $q\frac{n}{2}$. In light of assumption (1), we have

$$\mathbf{E}[-y_j] \approx (p-q)\tfrac{n}{2} \geq \tfrac{c_1}{2} \cdot \sqrt{n}.$$

A straightforward application of the Hoeffding bound (see Homework #5) shows that, with high probability,

$$-y_j \geq \frac{c_1}{4}\sqrt{n}$$

for every $j = 1, 2, \ldots, n$.

Now the dominoes fall. Recalling that

$$\mathbf{M} = \widehat{\mathbf{M}} + \mathbf{R} + \mathbf{I} - \mathbf{Y}$$

and applying Lemma 7.1 repeatedly (with initial matrix $\widehat{\mathbf{M}}$ and successive perturbations by $\mathbf{R}$, $\mathbf{I}$, and $-\mathbf{Y}$) yields

$$\lambda_{n-1}(\mathbf{M}) \geq \underbrace{\lambda_{n-1}(\widehat{\mathbf{M}})}_{=0} + \underbrace{\lambda_n(\mathbf{R})}_{\geq -c_2\sqrt{n}\text{ (w.h.p.)}} + \underbrace{\lambda_n(\mathbf{I})}_{=1} + \underbrace{\lambda_n(-\mathbf{Y})}_{\geq \frac{c_1}{4}\sqrt{n}\text{ (w.h.p.)}}.$$

We conclude that, provided $c_1 \geq 4c_2$, with high probability, $\lambda_{n-1}(\widehat{\mathbf{M}}) > 0$. (Recall that c_2 comes from Theorem 7.2 and is not under our control, but c_1 is in the assumption (1) and we can choose it as needed.) In this case, $\mathbf{M}$ is psd, and so the dual solution defined by $y_0, y_1, \ldots, y_n$ is feasible, and so the primal solution corresponding to the planted bisection is in fact optimal. This completes the proof of Lemma 3.2.

8 Semi-Random Planted Clique

The techniques that we've been studying are quite flexible. To illustrate this point, we now show that the same type of SDP-based algorithm and analysis can be used to extend our recovery results for the planted clique problem (Lecture #10) to the semi-random model with a monotone adversary.

8.1 The Main Result

Recall the model:

Semi-Random Planted Clique

Step 1 (Nature): For a fixed vertex set V and parameter k, form a graph $G = (V, E)$ as follows:

1. Choose a random subset $Q \subseteq V$ of k vertices.
2. Add an edge between each pair of vertices in Q.
3. Independently for each pair (i, j) of vertices with at least one of i, j not in Q, include the edge (i, j) with probability $\frac{1}{2}$.

Step 2 (Monotone adversary): An arbitrary subset of non-clique edges (i.e., at least one endpoint is outside the clique) is removed from G.

The main result of this section is the following, also due to Feige and Kilian [3].

Theorem 8.1 ([3]) *There exists a constant $c_3 > 0$ such that, for the planted clique model with $k \geq c_3\sqrt{n}$, the optimal value of a SDP relaxation is exactly k (with high probability over G).*

We are stating Theorem 8.1 only for the planted model of Lecture #10, not for the semi-random model. But just as with the planted bisection problem, monotonicity properties of SDPs mean that exact recovery guarantees for the classical planted model translate easily to the same guarantees for the semi-random model with a monotone adversary (see Homework #6). The spectral algorithm from last week is not robust enough to recover planted cliques in this semi-random model.

Another complaint is that the guarantee in Theorem 8.1 is only about the optimal objective function value, and the more natural goal is to actually recover the planted clique. But a solution of the former type naturally leads to one of the latter type; see Homework #6.

8.2 An SDP Relaxation and Its Dual

We need an SDP relaxation for the clique problem. It's more convenient to consider the independent set problem (i.e., given a graph, find the largest subset of mutually non-adjacent vertices), which is an equivalent problem (just complement all edges). In this transformation, the planted clique becomes a planted independent set (of the same size), and a monotone adversary is now allowed to add any edges that it wants, except with edges in the planted independent set forbidden.

The SDP relaxation (for independent set) will not be as immediately natural as the ones we've seen for cut problems, but it will still be easy to work with. First, it's easy to come up with a quadratic programming formulation. Introduce a variable z_i that is meant to be 1 if i is in the independent set and 0 if not. (The integrality constraint corresponds to the

quadratic constraint $z_i^2 = z_i$.) The objective is then

$$\max \sum_{i \in V} z_i. \tag{36}$$

One way to write the constraint that an independent set cannot contain two adjacent vertices is

$$z_i \cdot z_j = 0 \quad \text{for all } (i,j) \in E. \tag{37}$$

Here E denotes the edge set of the independent set instance, which is the complement of the edge set of the clique instance that we started with. By definition, this is an exact formulation of the maximum independent set problem.

There are two obstacles to relaxing this quadratic program to an SDP. The first is the objective function (36). Variables in an SDP correspond to products of pairs of variables, and in (36) each z_i appears solo. One way to address this is to change the objective to

$$\max \left(\sum_{i \in V} z_i \right)^2, \tag{38}$$

which gives a collection of quadratic terms. For this objective function to correspond to the maximum independent set problem, we need to change the semantics of the z_i's slightly. An independent set S of G now translates to a solution where $z_i = 1/\sqrt{|S|}$ for every $i \in S$ and $z_i = 0$ otherwise. To enforce these semantics, we replace the old 0-1 constraint (that $z_i^2 = z_i$ for each i) by

$$\sum_{i \in V} z_i^2 = 1. \tag{39}$$

The optimal objective function value (38) subject to the constraints (37) and (39) is precisely the size of largest independent set of G. (For one direction, take an independent set S and set $z_i = 1/\sqrt{|S|}$ for all $i \in S$ and 0 otherwise; for the other direction, see Homework #6.)

Since the z_i's only occur in pairs in (37)–(39), we can consider the corresponding vector program relaxation (with each $z_i z_j$ replaced by $\langle \mathbf{v}_i, \mathbf{v}_j \rangle$ for vectors $\mathbf{v}_i, \mathbf{v}_j \in \mathbb{R}^n$) and equivalent SDP relaxation (with x_{ij} standing in for $\langle \mathbf{v}_i, \mathbf{v}_j \rangle$). Skipping straight to the latter, our SDP relaxation is

$$\max \sum_{i,j \in V} x_{ij} \tag{40}$$

subject to

$$x_{ij} = 0 \quad \text{for all } (i,j) \in E \tag{41}$$

$$\sum_{i \in V} x_{ii} = 1 \tag{42}$$

$$\mathbf{X} \text{ is psd.} \tag{43}$$

This optimal value of this relaxation for the maximum independent set problem is actually a famous quantity, known as the *Lovász theta-function*. It has many equivalent definitions, see [**?**].

Next we derive the dual SDP. In the notation of Section 6.2, the coefficient matrix in the objective function (40) is the all-ones matrix $\mathbf{J}$, the coefficient matrix of a constraint of the form (41) for edge $e = (i, j) \in E$ is a matrix $\mathbf{A}_e$ with all zeroes except for a "1" in entries (i, j) and (j, i), and the coefficient matrix of the constraint (42) is the identity matrix $\mathbf{I}$.

The recipe for taking SDP duals in Section 6.2 assumes that the primal problem is a minimization problem. To apply the recipe here, we change the objective function (40) to

$$\min \sum_{i,j \in V} -x_{ij},$$

derive the dual SDP as in Section (6.2), and then negate the objective function of this dual SDP to obtain a minimization problem. The end result (as you should check) is:

$$\min -t \tag{44}$$

subject to

$$-\mathbf{J} - t\mathbf{I} - \mathbf{Y} \text{ is psd}, \tag{45}$$

where $\mathbf{Y}$ denote the matrix that has value y_e in the two entries corresponding to the edge e.

This dual simplifies to something very clean. In (45), note that the subtraction of $t\mathbf{I}$ decreases the eigenvalues of $-\mathbf{J} - \mathbf{Y}$ by exactly t. Thus for a given choice of $\mathbf{Y}$, t is feasible if and only if

$$t \leq \lambda_n(-\mathbf{J} - \mathbf{Y}),$$

or equivalently,

$$-t \geq \lambda_1(\mathbf{J} + \mathbf{Y}),$$

since negating a matrix negates all its eigenvalues. Looking at the objective (44), we see that at an optimal solution we will have $-t = \lambda_1(\mathbf{J} + \mathbf{Y})$. What are our possible choices of for the matrix $\mathbf{M} := \mathbf{J} + \mathbf{Y}$? Since $\mathbf{Y}$ has zeros in entries (i, j) that do not correspond to edges, and arbitrary values in entries that do correspond to edges, we can take $\mathbf{M}$ to be any matrix that has a 1 in every entry corresponding to a non-edge.

Summarizing, the dual SDP is equivalent to the following eigenvalue minimization problem:

$$\min \lambda_1(\mathbf{M}) \tag{46}$$

subject to

$$M_{ij} = 1 \qquad \text{for every } (i, j) \notin E. \tag{47}$$

8.3 Proof of Theorem 8.1

To prove Theorem 8.1, we follow the same plan as for Lemma 3.2: exhibit a feasible solution to the dual SDP (46)–(47) with the desired objective function value (in this case, the planted independent set size k). The hard part will again be verifying feasibility of the dual solution (with high probability over the input graph G).

So what's the easiest way to define a dual solution $\mathbf{M}$ so that $\lambda_1(\mathbf{M}) = k$? If we visualize $\mathbf{M}$ with the rows and columns corresponding to the planted independent set Q coming first, then we can write

$$\left(\begin{array}{c|c} \mathbf{J}_k & \approx 50\% \text{ 1's} \\ \hline \approx 50\% \text{ 1's} & \approx 50\% \text{ 1's} \end{array}\right),$$

where $\mathbf{J}_k$ denotes the $k \times k$ all-ones matrix. Note that the top and left "halves" of the matrix only contain $k \ll \frac{n}{2}$ rows and columns. The 1's correspond to non-edges of the planted independent set instance, or equivalently to edges of the original planted clique instance. In the rest of the entries, we are free to fill in whatever values we want. How can we fill them in so that $\lambda_1(\mathbf{M}) = k$?

First let make sure that there is some eigenvector (not necessarily the largest) with eigenvalue k. The obvious candidate for such an eigenvector is the vector $\mathbf{v}$ that puts a constant value on clique values and zero on non-clique vertices:

$$\mathbf{v} = (\underbrace{1, 1, \ldots, 1}_{k \text{ coordinates}}, \underbrace{0, 0, \ldots, 0}_{n-k \text{ coordinates}}).$$

For $\mathbf{v}$ to actually be an eigenvector of $\mathbf{M}$ with eigenvalue k, we need $\mathbf{Mv} = (k, k, \ldots, k, 0, 0, \ldots, 0)$. Note that $\mathbf{Mv}$ is sum of $\mathbf{M}$'s first k columns. So the first k coordinates of $\mathbf{Mv}$ are automatically k. To make sure that the other coordinates are 0, we can just assign the unconstrained entries in the lower-left part of the matrix accordingly. That is, in each row $i \in \{k+1, \ldots, n\}$, if ℓ of the first k entries of the row are forced to be 1, then we cancel this out by evenly spreading negative mass on the other $k - \ell$ entries in the row (from among the first k). Thus each such entry gets assigned the value $-\frac{\ell}{k-ell}$. For example, if ℓ is exactly its expectation ($\frac{k}{2}$, right?), then the unconstrained entries all get the value -1. If ℓ is $2k/3$ then they each get value -2, and if ℓ is $k/3$ they each get the value $-\frac{1}{2}$. To complete the description of $\mathbf{M}$, we set every unconstrained entry in the lower-right submatrix to -1. By construction, the matrix $\mathbf{M}$ has k as an eigenvalue (with eigenvector $\mathbf{v}$ as above).[12] We still need to show that all of $\mathbf{M}$'s other eigenvalues are smaller (with high probability).

We will again use Lemma 7.1 to track the eigenvalues of $\mathbf{M}$ as we build it up from its

[12]One detail: $\mathbf{M}$ is not well defined if $\ell = k$ in some row. But this happens only with extremely small probability.

constituent parts. So write

$$\mathbf{M} = \underbrace{\left(\begin{array}{c|c} \mathbf{J}_k & \mathbf{0} \\ \hline \mathbf{0} & \mathbf{0} \end{array}\right)}_{=\mathbf{M}_1} + \underbrace{\left(\begin{array}{c|c} \mathbf{0} & 1/-\frac{k-\ell}{k} \\ \hline 1/-\frac{k-\ell}{k} & \mathbf{0} \end{array}\right)}_{=\mathbf{M}_2} + \underbrace{\left(\begin{array}{c|c} \mathbf{0} & \mathbf{0} \\ \hline \mathbf{0} & \pm 1 \end{array}\right)}_{=\mathbf{M}_3},$$

where in $\mathbf{M}_3$, each non-zero entry is equally likely to be 1 or -1. By Lemma 7.1,

$$\lambda_2(\mathbf{M}) \leq \underbrace{\lambda_2(\mathbf{M}_1)}_{=0} + \underbrace{\lambda_1(\mathbf{M}_2)}_{\leq c_2\sqrt{n} \text{ w.h.p.}} + \underbrace{\lambda_1(\mathbf{M}_3)}_{\leq c_2\sqrt{n-k} \text{ w.h.p.}} . \tag{48}$$

Since $\mathbf{M}_1$ is a rank-1 matrix, $\lambda_2(\mathbf{M}_1) = 0$. Theorem 7.2 directly implies that $\lambda_1(\mathbf{M}_3) \leq c_1\sqrt{n-k}$ with high probability, where c_2 is the constant in Theorem 7.2. Theorem 7.2 does not apply directly to $\mathbf{M}_2$, since its nonzero entries are i.i.d. But intuitively, $\mathbf{M}_2$ is "enough like" a random symmetric matrix that the bound in Theorem 7.2 should still apply. This is in fact the case, although it requires a non-trivial, basically reworking one of the proofs of Theorem 7.2 so that it applies to $\mathbf{M}_2$ (see [3] for details).

From (48), with $k \geq c_3\sqrt{n}$ as in Theorem 8.1, we have $\lambda_2(\mathbf{M}) < k$ with high probability provided $c_3 > 2c_2$. When this is the case, we have $\lambda_1(\mathbf{M}) = k$, as desired. This completes the proof of Theorem 8.1.

References

[1] A. Blum and J. H. Spencer. Coloring random and semi-random k-colorable graphs. *Journal of Algorithms*, 19(2):204–234, 1995.

[2] R. B. Boppana. Eigenvalues and graph bisection: An average-case analysis. In *Proceedings of the 28th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 280–285, 1987.

[3] U. Feige and J. Kilian. Heuristics for semirandom graph problems. *Journal of Computer and System Sciences*, 63(4):639–671, 2001. Preliminary version in *FOCS '98*.

[4] K. Makarychev, Y. Makarychev, and A. Vijayaraghavan. Approximation algorithms for semi-random partitioning problems. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing (STOC)*, pages 367–384, 2012.

[5] K. Makarychev, Y. Makarychev, and A. Vijayaraghavan. Constant factor approximation for balanced cut in the PIE model. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*, pages 41–49, 2014.