CS264: Beyond Worst-Case Analysis Lecture #8: Exact Recovery in Perturbation-Stable Maximum Cut Instances[*]

Tim Roughgarden[†]

February 2, 2017

1 The Maximum Cut Problem

Today we continue our discussion of exact recovery under perturbation-stability assumptions. We'll look at a different cut problem than last lecture, and this will also give us an excuse to touch on two cool and fundamental topics, metric embeddings and semidefinite programming.

The Maximum Cut problem is a famous NP-hard problem. The input is an undirected graph $G = (V, E)$, where each edge $e \in E$ has a positive weight $w_e > 0$. The goal is to compute a cut (A, B) that maximizes the weight of the crossing edges (i.e., the edges with one endpoint in each of A, B). The Maximum Cut problem can arise in applications as a type of 2-clustering problem. For example, if weights represent dissimilarities (between images, genomes, etc.), then it makes sense to define clusters so that pairs of objects in different clusters tend to be quite dissimilar.

The fact that the Maximum Cut problem is NP-hard may be confusing, given that the Minimum s-t Cut problem is polynomial-time solvable (as proved last lecture or by a reduction to the maximum flow problem).[1] It's tempting to think that we can reduce the Maximum Cut problem to the Minimum Cut problem just by negating the weights of all of the edges. Such a reduction would yield an instance of Minimum Cut with negative weights (or capacities). But all of the polynomial-time algorithms for solving the Minimum Cut problem require nonnegative edge capacities. Indeed, it's not hard to prove that the maximum cut problem is NP-hard.

The goal for today, analogous to last lecture, is to recover the optimal solution in polynomial time in γ-perturbation-stable instances, where γ is as small as possible. The definition

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

[1] The maximum or minimum cut problem with no s or t reduces in polynomial time to the s-t maximum or minimum cut problem, by trying all possibilities for s and t.

of γ-perturbation-stable is the same as last lecture: if each of the edge weights is scaled down by an edge-dependent factor between 1 and γ, then the optimal solution should stay the same (and be unique).

2 A Linear Programming Relaxation

To argue that a linear program solves to integers under a perturbation-stability assumption, we need a linear program. Here's the one we used last lecture for the MINIMUM s-t CUT problem:

$$\min \sum_{e \in E} c_e x_e.$$

subject to:

$$\begin{array}{rcl} x_e & \in & [0,1] \\ x_e & \geq & d_u - d_v \\ x_e & \geq & d_v - d_u \end{array}$$

for every edge $e = (u,v)$, and also $d_s = 0$ and $d_t = 1$. We could try just flipping the "min" to a "max," but then nothing is stopping the linear program from just setting $x_e = 1$ for all edges e (no matter what the graph is). We could try also flipping the last two inequalities, so that they read $x_e \leq d_u - d_v$ and $x_e \leq d_v - d_u$, but this actually forces $x_e = 0$ (instead of $x_e \leq |d_u - d_v|$).[2]

So we need a different linear programming relaxation of the MAXIMUM CUT problem. We're not going to bother with the d-variables (which don't make much sense without s and t, anyway), but we'll keep a version of the x-variables (indicating how much each edge gets cut), and will impose some natural constraints on them.

There will be one decision variable x_{ij} for every unordered pair $i, j \in V$ of distinct vertices, whether or not $(i,j) \in E$. The intended semantics are for $x_{ij} = 1$ to indicate that i, j are on different sides of the cut, and $x_{ij} = 0$ to indicate that i, j are on the same side of the cut. Note that x_{ij} and x_{ji} are just two different names for this one decision variable. We impose the constraint

$$0 \leq x_{ij} \leq 1 \tag{1}$$

on each decision variable.

With these intended semantics, the objective function is clear:

$$\max \sum_{(i,j) \in E} w_{ij} x_{ij}. \tag{2}$$

Note that only pairs of vertices connected by an edge contribute to the objective function.

[2] Such issues are to be expected, given that the MINIMUM s-t CUT problem is polynomial-time solvable while the MAXIMUM CUT problem is NP-hard. But it's instructive to see exactly how the previous arguments break down.

We clearly need some more constraints, since otherwise nothing is stopping the linear program from setting all the x_{ij}'s to 1. What would it mean if $x_{ij} = x_{ik} = x_{jk} = 1$? According to the intended semantics, this would mean that each of the pairs $(i, j), (i, k), (j, k)$ are on different sides of the cut. But this makes no sense: if i and j are on different sides and i and k are also on different sides, then j and k must be on the same side. In inequality-speak: for each triple $i, j, k \in V$ of distinct vertices, we add the constraint

$$x_{ij} + x_{jk} + x_{ik} \leq 2. \tag{3}$$

Similarly, if we know that i, j are on the same side of the cut and also i, k are on the same side, then by transitivity j, k are also on the same side:

$$x_{jk} \leq x_{ij} + x_{ik} \tag{4}$$

for every triple $i, j, k \in V$ of distinct vertices. This completes the description of the linear program.[3]

Our main result for this linear programming relaxation is the following.

Theorem 2.1 *There is a constant $c > 0$ such that: in every $(c \log n)$-perturbation-stable instance of the* Maximum Cut *problem with n vertices, the linear program* (1)–(4) *solves to integers.*

We'll improve the value of γ in Theorem 2.1 in Section 4 via semidefinite programming. But there are fundamental barriers to getting it down to a constant (as we could with the Minimum Multiway Cut problem last lecture).

3 Proof of Theorem 2.1

Our two results last lecture—exact recovery by linear programming in 1-perturbation-stable Minimum s-t Cut instances and 4-perturbation-stable Minimum Multiway Cut instances—were proved using very similar arguments. The only problem-specific part of these proofs is the design of a randomized rounding algorithm (used only in the analysis!) that outputs a (random) cut such that the probability of an edge being cut is approximately the same as the value of the corresponding decision variable. For the Minimum s-t Cut problem, we blew up a balloon around the source s with a random radius. We used a quite different iterative algorithm for the Minimum Multiway Cut problem, where each iteration a group and a threshold for membership in the group are chosen at random.

3.1 A Randomized Rounding Algorithm

For the Maximum Cut problem, the analogous problem-specific tool is the following.

[3] You might be wondering about the analogous fact that if i, j are on the same side of the cut and i, k are on different sides, then j, k are also on different sides. These constraints are already implied by (4).

Lemma 3.1 *Fix an instance of the* Maximum Cut *problem, with F^* the edges cut in the optimal cut, and $\hat{\mathbf{x}}$ the optimal solution to the linear programming relaxation* (1)–(4)*. Then there exists a randomized algorithm (outputting a random cut (A, B)) and a scaling parameter $\sigma > 0$ such that:*

(i) for every edge $e = (i, j) \notin F^*$,

$$\mathbf{Pr}[e \textit{ cut by } (A, B)] \geq \sigma \cdot \frac{\hat{x}_{ij}}{\alpha},$$

where $\alpha = \Theta(\log n)$;

(ii) for every edge $e = (i, j) \in F^*$,

$$\mathbf{Pr}[e \textit{ not cut by } (A, B)] \leq \sigma \cdot (1 - \hat{x}_{ij});$$

(iii) the rounding algorithm is deterministic (i.e., always outputs the same cut) if and only if $\hat{\mathbf{x}}$ is integral.

We leave the proof that Lemma 3.1 implies Theorem 2.1 to Homework #4; it follows the exact same proof template we used last lecture to prove exact recovery in 4-perturbation-stable instances of Minimum Multiway Cut, with only minor differences in the details. (The inequalities are flipped compared to last lecture, as we've switched from a minimization problem to a maximization problem.) The scaling parameter $\sigma > 0$ plays no real role in the argument; the reason for its existence will become clear in Theorem 3.3 below.

We now turn to the proof of Lemma 3.1. First, a preliminary proposition.

Proposition 3.2 *Fix an instance of the* Maximum Cut *problem, a cut C, and a feasible solution $\hat{\mathbf{x}}$ to the linear programming relaxation* (1)–(4)*. For distinct $i, j \in V$, define*

$$\hat{y}_{ij} = \begin{cases} \hat{x}_{ij} & \textit{if } i, j \textit{ are on the same side of } C; \\ 1 - \hat{x}_{ij} & \textit{if } i, j \textit{ are on different sides of } C. \end{cases}$$

Then $\hat{\mathbf{y}}$ satisfies the triangle inequality:

$$\hat{y}_{jk} \leq \hat{y}_{ij} + \hat{y}_{ik}$$

for every $i, j, k \in V$.

Proposition 3.2 says that not only does $\hat{\mathbf{x}}$ define a metric space (due to the constraints (4)), but also $\hat{\mathbf{y}}$ as defined above is a metric (using also the constraints (3) on $\hat{\mathbf{x}}$).[4] The proof just checks a couple of cases, and the details are left to Homework #4.

[4] Throughout this lecture, we abuse the term "metric" slightly. Normally a metric space (X, d) is required to satisfy $d(x, y) = 0$ if and only if $x = y$. Here, we will allow distinct points to have zero pairwise distance. (These are sometimes called "semi-metrics.")

3.2 Bourgain's Theorem

The main tool used to prove Lemma 3.1 is a famous result known as Bourgain's theorem.

Theorem 3.3 (Bourgain's Theorem [2, 3]) *For every n-point metric space (X, d), there exists a randomized algorithm (outputting a random partition (A, B) of X) and a scaling parameter $\sigma > 0$ such that, for all distinct pairs $i, j \in X$,*

$$\mathbf{Pr}[i, j \textit{ on different sides of } (A, B)] \in \sigma \cdot \left[\frac{d(i,j)}{\alpha}, d(i,j) \right], \tag{5}$$

where $\alpha = \Theta(\log n)$.

It is clear that the scaling parameter $\sigma > 0$ is needed for Theorem 3.3: the left-hand side of (5) is always between 0 and 1, while the distances on the right-hand side could be anything.

Bourgain's theorem states the every n-point metric space admits a randomized partitioning algorithm so that the separation probabilities between pairs of points are proportional to the metric's distances, up to a $\Theta(\log n)$ factor.[5] The proof of Bourgain's theorem is pretty cool, and is outlined in a problem on Homework #4.[6] The $\Theta(\log n)$ approximation factor in Theorem 3.3 is the best possible for arbitrary metric spaces (up to the suppressed constant factor).[7]

3.3 Putting the Pieces Together

Fix an instance of Maximum Cut. Let C^* denote an optimal cut, cutting the edges F^*. Let $\hat{\mathbf{x}}$ be an optimal solution to the LP relaxation (1)–(4), and define $\hat{\mathbf{y}}$ as in Proposition 3.2 (with C^* playing the role of C). Since $\hat{\mathbf{y}}$ is a metric (Proposition 3.2), we can apply Theorem 3.3 to it and obtain a scaling parameter $\sigma > 0$ and a randomized algorithm that outputs a partition (A, B) such that

$$\mathbf{Pr}[i, j \text{ on different sides of } (A, B)] \in \sigma \cdot \left[\frac{\hat{y}_{ij}}{\alpha}, \hat{y}_{ij} \right],$$

where $\alpha = \Theta(\log n)$. By the definition of $\hat{\mathbf{y}}$, this means that

$$\mathbf{Pr}[i, j \text{ on different sides of } (A, B)] \in \sigma \cdot \left[\frac{\hat{x}_{ij}}{\alpha}, \hat{x}_{ij} \right], \tag{6}$$

[5] The field of "metric embeddings" has many results of this type, showing that a complex metric can be approximated by a simpler metric, with some distortion in the distances. The Johnson-Lindenstrauss (JL) lemma (see e.g. CS168) is another example. The JL lemma is about dimensionality reduction, and it states that the pairwise distances between n points in high-dimensional Euclidean space are well approximated by those between the scaled projections of these points to $\Theta(\log n)$-dimensional Euclidean space.

[6] Bourgain's theorem is usually stated in terms of approximately distance-preserving embeddings of general n-point metric spaces into ℓ_1 metrics, meaning metrics that arise as the ℓ_1 distances between points in $\mathbb{R}^d$ (for some d). Randomized partitioning algorithms are essentially equivalent to ℓ_1 metrics; Homework #4 spells out the details of this.

[7] It turns out that you can obtain a matching lower bound by considering the metric of shortest-path distances of a constant-degree expander graph.

for i, j on the same side of C^* (and in particular if (i, j) is an edge not in F^*), while

$$\mathbf{Pr}[i, j \text{ on different sides of } (A, B)] \in \sigma \cdot \left[\frac{1 - \hat{x}_{ij}}{\alpha}, 1 - \hat{x}_{ij}\right] \tag{7}$$

for i, j on different sides of C^* (and in particular if $(i, j) \in F^*$). The lower bound in (6) gives us Lemma 3.1(i); the upper bound in (7) yields Lemma 3.1(ii).[8]

To recap: Theorem 2.1 reduces to Lemma 3.1 by the same reasoning as in last lecture. Lemma 3.1 reduces to Bourgain's theorem. The simple trick in Proposition 3.2 of applying Bourgain's theorem to $\hat{\mathbf{y}}$ rather than to the LP solution $\hat{\mathbf{x}}$ ensures that the separation probabilities in the resulting randomized algorithm well approximate $\hat{x}_{ij}$ for edges (i, j) not cut by the optimal cut and $1 - \hat{x}_{ij}$ for the other edges. Homework #4 outlines all of the missing details.

4 A Semidefinite Programming Relaxation

Can we do better than exact recovery in $\Theta(\log n)$-perturbation-stable instances of Maximum Cut (Theorem 2.1)? Not using the LP relaxation in (1)–(4), in turns out. To get exact recovery for smaller values of γ, we're going to have to up our game and use a *semidefinite programming (SDP)* relaxation.

4.1 A Quadratic Programming Formulation of Maximum Cut

Semidefinite programming is even more powerful than linear programming, and can still be done in polynomial time.[9] We'll give a proper treatment of SDPs in Lectures 11–12; for now, this lecture will just give you a glimpse of it, in the running context of exact recovery in Maximum Cut instances. Here's the plan: (i) replace the LP relaxation (1)–(4) with a more powerful SDP relaxation that imposes more structure on the resulting metric; and (ii) use an analog of Bourgain's theorem that exploits this additional structure to achieve a better approximation.

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 Maximum Cut as such a 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 0 if i and j are on the same side of the cut, and ± 2 otherwise. Thus $(z_i - z_j)^2$—and this is the quadratic part—is 0 if i, j are on the same side of the cut and 4 otherwise. Thus, the total weight of

[8] Lemma 3.1(iii) follows directly from the description of the randomized algorithm in the proof of Bourgain's theorem, see Homework #4.

[9] SDPs can be used in practice, but usually can only deal with medium-size instance (e.g., graphs with ≤ 1000 vertices), while modern LP solvers can handle problem sizes 1–2 orders of magnitude larger.

a cut (A, B), with $z_i = -1$ for all $i \in A$ and $z_i = +1$ for all $i \in B$, is precisely

$$\max \frac{1}{4} \sum_{(i,j)\in E} w_{ij}(z_i - z_j)^2. \tag{8}$$

Optimizing (8) subject to $z_i \in \{-1, +1\}$ is exactly the Maximum Cut problem, and hence is NP-hard.

4.2 A Vector Relaxation

Here's the crazy idea for a computationally tractable relaxation: rather than forcing each z_i to be either 1 or -1, let's allow it to be *any unit vector in n-dimensional Euclidean space*, where n is the number of decision variables. Thus we replace each scalar z_i with 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. Thus the quadratic program becomes the following *vector program* (all norms in this lecture are 2-norms):

$$\max \frac{1}{4} \sum_{(i,j)\in E} w_{ij}\|v_i - v_j\|^2 \tag{9}$$

subject to

$$v_i \in \mathbb{R}^n \tag{10}$$

and

$$\|v_i\|^2 = 1 \tag{11}$$

for every $i \in V$.

Several comments. First, remember that $\langle u, u\rangle = \|u\|^2$ for any vector u; this is where the $\|v_i - v_j\|^2$ in the objective function comes from. Second, it probably seems obscure to write $\|v_i\|^2 = 1$ rather than $\|v_i\| = 1$, but this is important for computational tractability, as we'll see in a future lecture. We'll also see later that such vector programs are equivalent to semidefinite programs, which optimize over the space of positive semidefinite matrices. The connection is that the vectors v_i correspond to the columns of the "square root" of the psd matrix (more on this later, see Lectures 11–12).

The vector program (9)–(11) is a relaxation of the Maximum Cut problem, since one always has the option of assigning $\pm e_1$ to each vector v_i, where e_1 denotes the first standard basis vector. Geometrically, it's easy to understand what this relaxation is doing: it maps each vertex to a point on the sphere (in $\mathbb{R}^n$), to make the endpoints of each edge as close to antipodal as possible (to make (9) large). This relaxation is already interesting in its own right, and leads to the best-known approximation algorithm for the Maximum Cut problem (see e.g. CS261, Lecture #20).

For our purposes of exact recovery, however, we want our SDP to generalize our previous LP relaxation (1)–(4), and in particular to have analogs of the constraints (3) and (4). Looking at the objective function (9), we see that the term $\frac{1}{4}\|v_i - v_j\|^2$ is playing the same role as the variable x_{ij} in the LP objective (2). (Note that since v_i, v_j are unit vectors, $\frac{1}{4}\|v_i - v_j\|^2 \in [0,1]$.) This suggests adding the constraints (after scaling by 4)

$$\|v_i - v_j\|^2 + \|v_j - v_k\|^2 + \|v_i - v_k\|^2 \leq 8 \tag{12}$$

and

$$\|v_j - v_k\|^2 \leq \|v_i - v_j\|^2 + \|v_i - v_k\|^2 \tag{13}$$

for every triple $i, j, k \in V$ of distinct vertices. Recall that the constraints (12) enforce the semantics that if i, j are on different sides of the cut and also j, k are on different sides, then i, k are on the same side. Similarly, the constraints (12) enforce the semantics that if i, j are on the same side of the cut and also j, k are on the same side, then i, k are on the same side. Since bona fide cuts (with all v_i's mapped to $\pm e_1$, according to their side of the cut) satisfy these constraints, the vector program (9)–(13) is still a relaxation of the Maximum Cut problem, and its optimal objective function value is at least that of the maximum cut.

Here's the crazy part: the vector programming relaxation (9)–(13) can actually be solved in polynomial time![10] You might well find this counterintuitive, given that the inner products in the objective function seem hopelessly quadratic. The moral reason for computational tractability is *convexity*. Linear programs have a convex feasible region (convex combinations of feasible solutions are again feasible) and a linear objective function (a special case of a convex function). More generally, a convex function can be minimized over a convex set in polynomial time (under minor conditions, which hold in our applications).

OK. . . but where's the convexity in the vector relaxation (9)–(13)? (After all, if you take the average of two points on the unit sphere, you don't get another point on the unit sphere.) It comes from the equivalence alluded to earlier between the vectors in the vector program and positive semidefinite (psd) matrices in the equivalent semidefinite program. The space of psd matrices is convex, and so can be optimized over efficiently. We'll say more about all this later, but let's take it on faith for now that the vector program (9)–(13) can be solved in polynomial time.

4.3 ℓ_2^2 Metrics

Fix an instance of the Maximum Cut problem and let $\hat{\mathbf{v}}$ denote an optimal solution to (9)–(13). As shorthand, write

$$\hat{x}_{ij} = \frac{1}{4}\|\hat{v}_i - \hat{v}_j\|^2 \tag{14}$$

for every pair $i, j \in V$ of distinct vertices. Note that all of the $\hat{x}_{ij}$'s lie in $[0,1]$.

From the constraints (13), we know that the $\hat{x}_{ij}$'s, viewed as pairwise distances, form a metric. But we also know more: these distances arise as the squared Euclidean distances between pairs of points in $\mathbb{R}^d$ for some d. (For us, these points are the $\hat{v}_i$'s computed by the vector relaxation.) Such metrics are called *ℓ_2^2 metrics*.[11] It is not automatic that the squared

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

[11] Pronounced "ell-too squared."

Euclidean distances of points in $\mathbb{R}^n$ form a metric (even though their "normal" Euclidean distances do).[12] Not every metric is an ℓ_2^2 metric (can you find an example?). The fact that ℓ_2^2 metrics are not as general as arbitrary metrics raises the hope that Bourgain's theorem (Theorem 3.3) can be improved for this subclass of metrics. This is indeed true, as proved roughly 10 years ago.

Theorem 4.1 ([1]) *For every n-point ℓ_2^2 metric space (X, d), there exists a randomized algorithm (outputting a random partition (A, B) of X) and a scaling parameter $\sigma > 0$ such that, for all distinct pairs $i, j \in X$,*

$$\mathbf{Pr}[i, j \text{ on different sides of } (A, B)] \in \sigma \cdot \left[\frac{d(i,j)}{\alpha}, d(i,j)\right],$$

where $\alpha = O(\sqrt{\log n}\log\log n)$.

The proof of Theorem 4.1 is outside the scope of this course. Most people believe that the annoying $\log\log n$ factor should be removable, and that the optimal guarantee has $\alpha = \Theta(\sqrt{\log n})$.[13]

We're not looking to apply Theorem 4.1 directly to the metric defined by the $\hat{x}_{ij}$'s, since this only implies part (i) of the analog of Lemma 3.1. So we need an analog of Proposition 3.2.

Proposition 4.2 *Fix an instance of the* Maximum Cut *problem, a cut $C = (A, B)$, and a feasible solution $\hat{\mathbf{v}}$ to the vector programming relaxation* (9)–(13) *with induced ℓ_2^2 metric $\hat{\mathbf{x}}$ defined as in* (14)*. For distinct $i, j \in V$, define*

$$\hat{y}_{ij} = \begin{cases} \hat{x}_{ij} & \text{if } i, j \text{ are on the same side of } C; \\ 1 - \hat{x}_{ij} & \text{if } i, j \text{ are on different sides of } C. \end{cases}$$

Then $\hat{\mathbf{y}}$ is also an ℓ_2^2 metric.

Proof: As in the proof of Proposition 4.2, the constraints (12) and (13) imply that $\hat{\mathbf{y}}$ is a metric. To prove that it is in fact an ℓ_2^2 metric, we need to exhibit an embedding of the vertices V into $\mathbb{R}^d$ (for some d) such that the $\hat{y}_{ij}$'s arise as squared Euclidean distances. To this end, define

$$\hat{u}_i = \begin{cases} \hat{v}_i & \text{if } i \in A; \\ -\hat{v}_i & \text{if } i \in B. \end{cases}$$

We leave it to you to check that

$$\hat{y}_{ij} = \frac{1}{4}\|\hat{u}_i - \hat{u}_j\|^2$$

for every distinct $i, j \in V$, which proves that $\hat{\mathbf{y}}$ indeed defines an ℓ_2^2 metric. ■

[12] For example, consider the points $x = 1$, $y = 2$, and $z = 3$ in $\mathbb{R}$. The squared Euclidean distance between x and y is 1; the same for y and z. But the squared Euclidean distance between x and z is 4, a violation of the triangle inequality.

[13] Very recently, a lower bound of $\Omega(\sqrt{\log n})$ for a worst-case ℓ_2^2 metric was proved in [5].

4.4 Better Exact Recovery

Applying Theorem 4.1 to the ℓ_2^2 metric defined by $\hat{\mathbf{y}}$ in Proposition 4.2, we obtain an analog of Lemma 3.1.

Lemma 4.3 *Fix an instance of the* Maximum Cut *problem, with F^* the edges cut in the optimal cut, and $\hat{\mathbf{v}}$ the optimal solution to the vector programming relaxation* (9)–(13)*. Then there exists a randomized algorithm (outputting a random cut (A, B)) and a scaling parameter $\sigma > 0$ such that:*

(i) for every edge $e = (i, j) \notin F^*$,

$$\mathbf{Pr}[e \text{ cut by } (A, B)] \geq \sigma \cdot \frac{\frac{1}{4}\|v_i - v_j\|^2}{\alpha},$$

where $\alpha = \Theta(\sqrt{\log n}\log\log n)$;

(ii) for every edge $e = (i, j) \in F^*$,

$$\mathbf{Pr}[e \text{ not cut by } (A, B)] \leq \sigma \cdot (1 - \tfrac{1}{4}\|v_i - v_j\|^2);$$

(iii) the rounding algorithm is deterministic (i.e., always outputs the same cut) if and only if $\hat{\mathbf{v}}$ is integral.

Part (iii) follows from inspection of the randomized algorithm in the proof of Theorem 4.1. By $\hat{\mathbf{v}}$ being "integral," we mean that there exists antipodal unit vectors $w, -w$ such that $\hat{v}_i \in \{w, -w\}$ for every $i \in V$, in which case $\hat{\mathbf{v}}$ can be immediately interpreted as a cut with the same objective function value.

Just as Lemma 3.1 implies Theorem 2.1, Lemma 4.3 implies the following.

Theorem 4.4 ([4]) *There is a constant $c > 0$ such that: in every $(c\sqrt{\log n}\log\log n)$-perturbation-stable instance of the* Maximum Cut *problem with n vertices, every optimal solution of the vector program* (9)–(13) *is integral.*

We should again ask, can we do better? Not by much, it would seem. There are examples of $O(\sqrt{\log n})$-perturbation-stable instances of Maximum Cut for which the optimal solution to the vector program (9)–(13) is not integral [4]. There is even evidence that *no* polynomial-time algorithm can always recover an optimal solution in γ-perturbation-stable instances, for arbitrarily large constants γ (see Homework #4).

References

[1] S. Arora, J. R. Lee, and A. Naor. Euclidean distortion and the sparsest cut. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 553–562, 2005.

[2] J. Bourgain. On Lipschitz embeddings of finite metric spaces in Hilbert space. *Israel Journal of Mathematics*, 52(1-2):46–52, 1985.

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

[4] K. Makarychev, Y. Makarychev, and A. Vijayaraghavan. Bilu-Linial stable instances of max cut and minimum multiway cut. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 890–906, 2014.

[5] A. Naor and R. Young. The integrality gap of the Goemans-Linial SDP relaxation for sparsest cut is at least a constant multiple of $\sqrt{\log n}$. In *Proceedings of the 49th Annual ACM Symposium on Theory of Computing (STOC)*, 2017. To appear.